EXPERIMENT NO: 5

AIM: Use SMOTE technique to generate synthetic data. (To solve the problem of class Imbalance).

THEORY:

SMOTE (Synthetic Minority Over-sampling Technique) is a popular algorithm used to address class imbalance in datasets. Here's the theory behind each step of the process:

Understanding Class Imbalance:

In many real-world datasets, the distribution of classes is often imbalanced, meaning that one class (majority class) significantly outnumbers the other class or classes (minority class or classes). Class imbalance can lead to biased models that favor the majority class, resulting in poor performance on the minority class.

Synthetic Minority Over-sampling Technique (SMOTE):

SMOTE is a popular technique used to address class imbalance by generating synthetic samples for the minority class. It works by creating synthetic instances along the line segments joining k nearest neighbors of each minority class instance. This helps in increasing the representation of the minority class without duplicating existing samples.

Checking Imbalance Ratio:

Before applying any balancing technique, it's essential to check the imbalance ratio of the dataset to understand the extent of class imbalance. This can be done by counting the occurrences of each class in the dataset.

Random Generation for Balancing:

Random generation involves randomly undersampling the majority class or oversampling the minority class to balance the dataset. However, random undersampling may lead to loss of information, and oversampling may result in overfitting. Nonetheless, it provides a simple baseline for comparison with more advanced techniques like SMOTE.

Balancing with SMOTE:

SMOTE addresses the limitations of random generation by creating synthetic samples for the minority class. By generating new instances based on the characteristics of existing minority class instances, SMOTE effectively expands the minority class and balances the dataset.

Training and Evaluating Models:

After balancing the dataset using both random generation and SMOTE, models are trained on the balanced datasets. Typically, a machine learning model such as logistic regression or a decision tree classifier is trained. The performance of the models is evaluated using various evaluation metrics such as accuracy, precision, recall, and F1-score.

Comparison of Model Performance:

The performance of the models trained on randomly balanced data and SMOTE-balanced data is compared to assess the effectiveness of SMOTE in improving the model's ability to correctly classify instances from the minority class and overall classification accuracy. Evaluation metrics such as accuracy, precision, recall, and F1-score are used for comparison.

By following these steps, SMOTE can effectively address class imbalance in datasets and improve the performance of machine learning models, particularly in scenarios where the minority class is underrepresented.

CODE:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, precision_score,

recall_score, f1_score, confusion_matrix

from imblearn.over_sampling import SMOTE
```

```
# Step 1: Create a sample dataset with imbalanced classes
X, y = make_classification(
n_classes=2,
class_sep=0.5,
weights=[0.95, 0.05], # Imbalanced classes
n_informative=2,
n_redundant=0,
flip_y=0,
n_features=2,
n_clusters_per_class=1,
n_samples=1000,
random_state=42
)

# Step 2: Check class imbalance ratio
print("Class imbalance ratio:", np.bincount(y))

# Step 3: Balance the dataset using random generation
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

# Step 4: Balance the dataset using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Step 5: Train a model on both balanced datasets
logreg = LogisticRegression()
```

```
logreg.fit(X_train, y_train) # Model trained on randomly balanced data
logreg_smote = LogisticRegression()
logreg_smote.fit(X_resampled, y_resampled) # Model trained on SMOTE
balanced data

# Step 6: Compare the performance of models
def evaluate_model(model, X, y):
y_pred = model.predict(X)
accuracy = accuracy_score(y, y_pred)
precision = precision_score(y, y_pred)
recall = recall_score(y, y_pred)
f1 = f1_score(y, y_pred)
conf_mat = confusion_matrix(y, y_pred)
return accuracy, precision, recall, f1, conf_mat

# Evaluate models on test data
accuracy_random, precision_random, recall_random, f1_random,
conf_mat_random = evaluate_model(logreg, X_test, y_test)
accuracy_smote, precision_smote, recall_smote, f1_smote,
conf_mat_smote = evaluate_model(logreg_smote, X_test, y_test)
# Print evaluation metrics
print("\nEvaluation Metrics for Model trained on Randomly Balanced Data:")
print("Accuracy:", accuracy_random)
print("Precision:", precision_random)
print("Recall:", recall_random)
print("F1 Score:", f1_random)
print("Confusion Matrix:\n", conf_mat_random)
```

```
print("\nEvaluation Metrics for Model trained on SMOTE Balanced Data:")
print("Accuracy:", accuracy_smote)
print("Precision:", precision_smote)
print("Recall:", recall_smote)
print("F1 Score:", f1_smote)
print("Confusion Matrix:\n", conf_mat_smote)
```

OUTPUT:

```
Class imbalance ratio: [950  50]

Evaluation Metrics for Model trained on Randomly Balanced Data:
Accuracy: 0.96
Precision: 1.0
Recall: 0.2
F1 Score: 0.33333333333333337
Confusion Matrix:
 [[190   0]
 [  8   2]]

Evaluation Metrics for Model trained on SMOTE Balanced Data:
Accuracy: 0.78
Precision: 0.11363636363636363
Recall: 0.5
F1 Score: 0.18518518518518517
Confusion Matrix:
 [[151  39]
 [  5   5]]
```

CONCLUSION:

In this experiment, we explored SMOTE, a method to tackle class imbalance in datasets by generating synthetic samples for the minority class. Through Python implementation, we witnessed SMOTE's ability to enhance model performance compared to random generation, making it a crucial tool for improving accuracy in machine learning tasks with imbalanced data.