THYROID DISEASE PREDICTION

ABSTRACT

Thyroid disease (TD) is one of the most progressive endocrine disorders in the human population today. Prediction of the endocrine disease is a critical task in the field of clinical data analysis. Machine Learning (ML) has shown effective results in the decision making and predictions from the enormous data generated by healthcare domain. In this project we have proposed three models based on the primary dataset collected from 3772 patients. In these models, I have compared top three ML algorithms (Support Vector Machine, Random Forest, Logistic Regression) with three Over-sampling methods (Random over sampling, SMOTE, ADASYN) for effective classification.

KEYWORDS: Thyroid disease, Machine Learning, ML algorithms, Over-sampling

methods

INTRODUCTION

Millions of individuals worldwide suffer from a thyroid condition, a prevalent glandular system ailment. For effective treatment and management, it is important to find and diagnose thyroid disease as soon as possible. But thyroid disease can be hard to identify because its symptoms aren't always clear and can be like those of other illnesses. Laboratory tests and medical scans are often used to help figure out what's wrong with the thyroid, but these methods can take a long time and cost a lot. Thyroid disease detection and diagnosis have been greatly aided by the advent of machine learning algorithms and artificial intelligence (AI) techniques in recent years. These algorithms can look at a huge amount of patient data, like lab tests, medical images, and clinical symptoms, to find patterns and connections that may not be obvious to human experts. Using machine learning algorithms to find thyroid disease has several benefits, such as finding the disease early, getting a correct diagnosis, and making personalized treatment plans.

In this study, we aim to provide an overview of the current state of the art in using machine learning algorithms to detect thyroid diseases. We will discuss the limitations of traditional diagnostic methods, the advantages of using machine learning algorithms, and the various techniques and algorithms employed to identify thyroid disease.

We evaluated the performance of different models such as Support Vector Machine, RandomForest Classifier, Logistic Regression using different sampling methods like Random Over-sampling, SMOTE (Synthetic Minority Over-sampling Technique), ADASYN (Adaptive Synthetic Sampling Technique).

METHODOLOGY

The dataset used for this study consists of information collected from approximately 3772 individuals, including both females and males, with ages ranging from 1 to 100 years. The dataset includes individuals with hypothyroidism and hyperthyroidism, as well as healthy adults without thyroid disease. The data was obtained from Kaggle, and the goal was to use machine

learning techniques to classify thyroid diseases.

Data Collection:

1. **age**	age of the patient	Object
2. **sex**	sex patient identifies	Object
3. **on_thyroxine**	whether patient is on thyroxine	Object
4. **query on thyroxine**	whether patient is on thyroxine	Object
5. **on antithyroid meds**	whether patient is on antithyroid medication	Object
6. **sick**	whether patient is sick	Object
7. **pregnant**	whether patient is pregnant	Object
8. **thyroid_surgery**	whether patient has undergone thyroid surgery	Object
9. **I131_treatment**	whether patient is undergoing I131 treatment	Object
10. **query_hypothyroid**	whether patient believes they have hypothyroid	Object
11. **query_hyperthyroid**	whether patient believes they have hyperthyroid	Object
12. **lithium**	whether patient has lithium	Object
13. **goitre**	whether patient has goitre	Object
14. **tumor**	whether patient has tumor	Object
15. **hypopituitary**	whether patient has hyperpituitary gland	Object

16. **psych**	whether patient has psych	Object
17. **TSH_measured**	whether TSH was measured in the blood	Object
18. **TSH**	TSH level in blood from lab work	Object
19. **T3_measured**	whether T3 was measured in the blood	Object
20. **T3**	T3 level in blood from lab work	Object
21. **TT4_measured**	whether TT4 was measured in the blood	Object
22. **TT4**	TT4 level in blood from lab work	Object
23. **T4U_measured**	whether T4U was measured in the blood	Object

24. **T4U**	T4U level in blood from lab work	Object
25. **FTI_measured**	whether FTI was measured in the blood	Object
26. **FTI**	FTI level in blood from lab work	Object
27. **TBG_measured**	whether TBG was measured in the blood	Object
28. **TBG**	TBG level in blood from lab work	Object
29. **binaryClass**	whether the patients has thyroid or not	Object

Data Preprocessing:

Before the datasets are loaded into the machine learning model, a variety of techniques are

applied to enhance its efficiency. Data normalization, encoding, handling missing values, and other pre-processing techniques are a few of them. Data preprocessing techniques such as Ordinal encoding and feature standardization were applied to prepare the dataset for analysis. Ordinal encoding converts categorical data into a format that can be provided to ML algorithms to do a better job in prediction. Feature standardization involves scaling the features so that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one. KNNImputer is utilized to address missing values in a dataset by predicting them based on the average of nearby data points within the k-nearest neighbors, offering an effective imputation method.

Data Machine Learning Techniques:

We compared several algorithms with different sampling methods to understand which was the one that best labeled each instance of our dataset.

Logistic Regression

Logistic regression is a statistical model used for binary classification. It is used to predict the probability of a binary outcome based on one or more predictor variables. In logistic regression, the dependent variable is binary, meaning it can take only two possible values, typically 0 and 1. The model uses the logistic function (also known as the sigmoid function) to model the relationship between the dependent variable and the independent variables.

Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

Support Vector Machine

SVM, or Support Vector Machine, is a supervised machine learning algorithm that can be used for both classification and regression tasks. It is particularly effective in solving complex classification problems and is widely used in various fields, including image recognition, text classification, and bioinformatics. SVMs are based on the concept of finding the optimal hyperplane that best separates data points belonging to different classes.

Random Over-Sampling

In random over-sampling, the minority class is augmented by creating exact copies of minority class observations. This method helps balance the class distribution by increasing the number of instances in the minority class.

SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE generates synthetic observations for the minority class by interpolating between existing minority class instances. It creates synthetic samples along the line connecting a minority

instance and its k-nearest neighbors. The number of synthetic observations and the number of nearest neighbors are key parameters in SMOTE.

ADASYN (Adaptive Synthetic Sampling)

ADASYN is an extension of SMOTE that focuses on generating more synthetic data for minority class instances that are harder to learn. ADASYN generates more synthetic observations for minority instances with more majority class neighbors in their vicinity. It aims to address the challenge of learning from difficult-to-classify minority instances.

RESULTS AND DISCUSSION:

1.IMPORTING LIBRARIES

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import KNNImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import ADASYN, RandomOverSampler, SMOTE
from sklearn.metrics import accuracy_score, confusion_matrix,precision_score, recall_score, f1_score
```

Loading Dataset:

```
td=pd.read_csv("/content/drive/MyDrive/thyroid data.csv")
td.head(10)
```

	age	sex	on thyroxine	query on thyroxine	on antithyroid medication	sick	pregnant	thyroid surgery	I131 treatment	query hypothyroid	...	TT4 measured	TT4	T4U measured	T4U	FTI measured	FTI	TBG measured	TBG	referral source	binaryclass
0	41	F	f	f	f	f	f	f	f	f	...	t	125	t	1.14	t	109	f	?	SVHC	P
1	23	F	f	f	f	f	f	f	f	f	...	t	102	f	?	t	?	f	?	other	P
2	46	M	f	f	f	f	f	f	f	f	...	t	109	t	0.91	t	120	f	?	other	P
3	70	F	t	f	f	f	f	f	f	f	...	t	175	f	?	f	?	f	?	other	P
4	70	F	f	f	f	f	f	f	f	f	...	t	61	t	0.87	t	70	f	?	SVI	P
5	18	F	t	f	f	f	f	f	f	f	...	t	183	t	1.3	t	141	f	?	other	P
6	59	F	f	f	f	f	f	f	f	f	...	t	72	t	0.92	t	78	f	?	other	P
7	80	F	f	f	f	f	f	f	f	f	...	t	80	t	0.7	t	115	f	?	SVI	P
8	66	F	f	f	f	f	f	f	f	f	...	t	123	t	0.93	t	132	f	?	SVI	P
9	68	M	f	f	f	f	f	f	f	f	...	t	83	t	0.89	t	93	f	?	SVI	P

10 rows × 30 columns

2.DATA PREPROCESSING

#Drop the irrelevant features from the dataset

td.drop(['TBG measured','TBG','T3 measured','TSH measured','TT4 measured','T4U measured','FTI measured','referral source'],axis=1,inplace=True)

td

```
td.drop(['TBG measured','TBG','T3 measured','TSH measured','TT4 measured','T4U measured','FTI measured','referral source'],axis=1,inplace=True)
td
```

	age	sex	on thyroxine	query on thyroxine	on antithyroid medication	sick	pregnant	thyroid surgery	I131 treatment	query hypothyroid	...	goitre	tumor	hypopituitary	psych	TSH	T3	TT4	T4U	FTI	binaryClass
0	41	F	t	f	f	f	f	f	f	f	...	f	f	f	f	1.3	2.5	125	1.14	109	P
1	23	F	t	f	f	f	f	f	f	f	...	f	f	f	f	4.1	2	102	?	?	P
2	46	M	t	f	f	f	f	f	f	f	...	f	f	f	f	0.98	?	109	0.91	120	P
3	70	F	t	f	f	f	f	f	f	f	...	f	f	f	f	0.16	1.9	175	?	?	P
4	70	F	t	f	f	f	f	f	f	f	...	f	f	f	f	0.72	1.2	61	0.87	70	P
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3767	30	F	t	f	f	f	f	f	f	f	...	f	f	f	f	?	?	?	?	?	P
3768	68	F	t	f	f	f	f	f	f	f	...	f	f	f	f	1	2.1	124	1.08	114	P
3769	74	F	t	f	f	f	f	f	f	f	...	f	f	f	f	5.1	1.8	112	1.07	105	P
3770	72	M	t	f	f	f	f	f	f	f	...	f	f	f	f	0.7	2	82	0.94	87	P
3771	64	F	t	f	f	f	f	f	f	f	...	f	f	f	f	1	2.2	89	1.07	82	P

3772 rows × 22 columns

td.shape

output
(3772, 30)

#It shows the number of non-null values, data types, and memory usage for each column.

td.info()

#Generate descriptive statistics for each numerical column in the dataset

td.describe()
td = td.replace(['?'],np.nan) # Replacing ‘?’ into NaN value

```
td.isnull().sum()
```

```
age                            1
sex                          150
on thyroxine                   0
query on thyroxine             0
on antithyroid medication      0
sick                           0
pregnant                       0
thyroid surgery                0
I131 treatment                 0
query hypothyroid              0
query hyperthyroid             0
lithium                        0
goitre                         0
tumor                          0
hypopituitary                  0
psych                          0
TSH                          369
T3                           769
TT4                          231
T4U                          387
FTI                          385
binaryClass                    0
dtype: int64
```

#Filling the null values for age, sex columns

#KNNimputer is used to fill the missing values for TSH,TT4,T4U,FTI columns

```
column="age"
td[column]=pd.to_numeric(td[column],errors="coerce").fillna(-1).astype(int)
```

```
td.sex.replace({'F':2,'M':1},inplace=True)
round_Values = round(td.sex.mean())
td.sex.fillna(round_Values,inplace=True)
```

```
#KNNImputer- used to fill the missing values for certain features

knnimp=KNNImputer(n_neighbors=3)

cols= ['TSH','T3','TT4','T4U','FTI']
for i in cols:
    td[i]=knnimp.fit_transform(td[[i]])
```

#After filling the null values , we got this

```
td.isnull().sum()
```

```
age                          0
sex                          0
on thyroxine                 0
query on thyroxine           0
on antithyroid medication    0
sick                         0
pregnant                     0
thyroid surgery              0
I131 treatment               0
query hypothyroid            0
query hyperthyroid           0
lithium                      0
goitre                       0
tumor                        0
hypopituitary                0
psych                        0
TSH                          0
T3                           0
TT4                          0
T4U                          0
FTI                          0
binaryClass                  0
dtype: int64
```

3.EDA:

```
[ ] #To check the multicollinearity in the dataset variance inflation factor is used
    x=td.iloc[:,[16,17,18,19,20]]

    vif_data=pd.DataFrame()
    vif_data["feature"]=x.columns
```

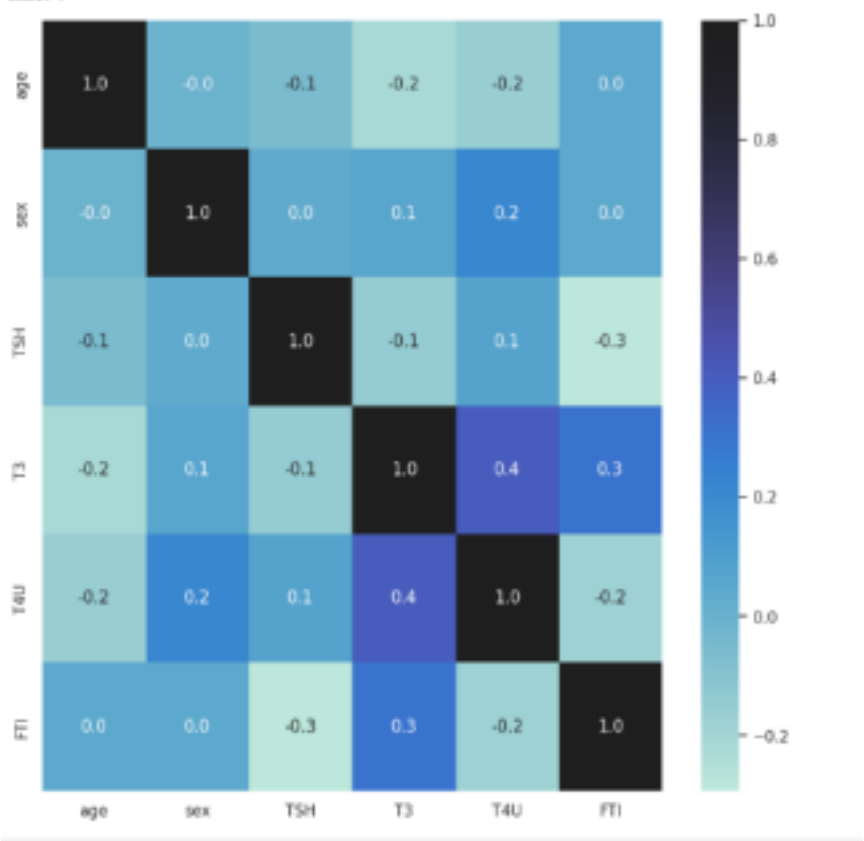
```
[ ] from statsmodels.stats.outliers_influence import variance_inflation_factor

    vif_data["VIF"]=[variance_inflation_factor(x.values,i)
                     for i in range(len(x.columns))]

    print(vif_data)
```

```
  feature        VIF
0     TSH   1.186165
1      T3  12.262473
2     TT4  46.126858
3     T4U  18.629518
4     FTI  27.443861
```

```
[ ] #The variable "TT4" has a high VIF of about 46.13, indicating significant multicollinearity.
    #Droping highly correlated variables from the dataset
    td.drop(["TT4"],axis=1,inplace=True)
```

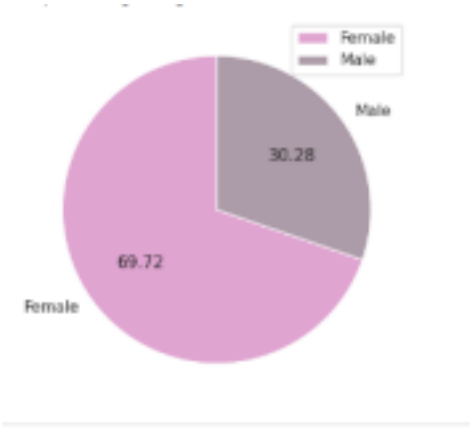
Variance inflation factor is used to identify the column which was highly correlated among the dataset. We identify that TT4 has high VIF value (46.12) so we dropped that column to increase our model performance.

```
plt.figure(figsize=(10,10))
sns.heatmap(td.corr(), cbar=True, fmt='.1f', annot=True, center=1)
```



Pie Charts:

```
plt.figure(figsize=(5,5))
plt.pie(x=td.sex.value_counts(),
        labels=['Female','Male'],
        startangle = 90,
        colors=["#E0A4D1","#AC9BA8"],
        autopct='%.2f'
       )
plt.legend()
```



This chart shows that Female patients who has disease is greater than male

patients. 2. plt.figure(figsize=(5,5))

plt.pie(x=td.sick.value_counts(),

labels=['Sick','Well'],

startangle = 20,

colors=['lightSkyBlue','lightSeaGreen'],

autopct='%.1f',

explode=[0,0.2]

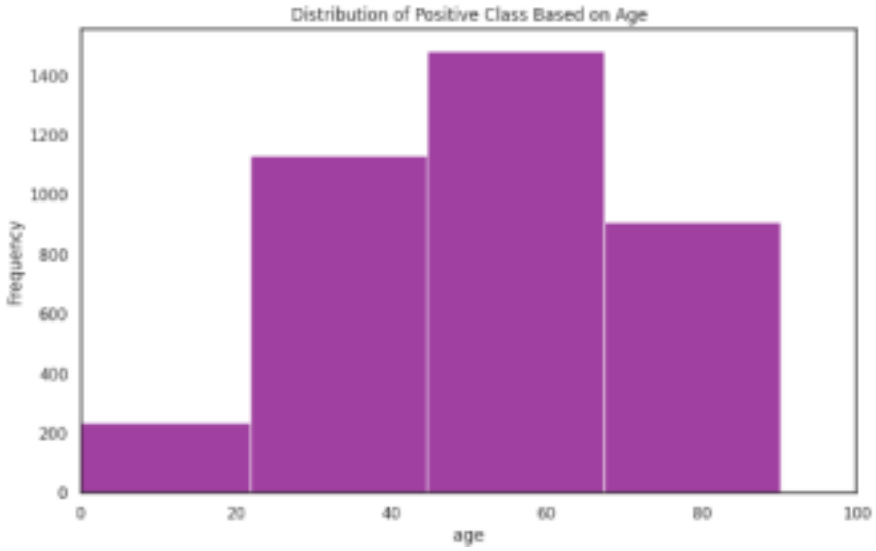
)

According to the dataset, a vast majority of individuals (96%) were diagnosed with thyroid disease.

Histplot:

```
plt.figure(figsize=(10, 6))
# Set x-axis limits to show ages from 0 to 100
plt.xlim(0, 100)
```

```
sns.histplot(x='age', data=td, color='purple', bins=20) # You can adjust the number of bins
plt.title("Distribution of Positive Class Based on Age")
plt.xlabel('age', fontsize=12)
plt.ylabel('Frequency', fontsize=12)

plt.show()
```



This plot shows that the majority of individuals aged between 50 and 70 have a positive result for thyroid disease.

4.ENCODING:

#OrdinalEncoder is used to transform the categorical features into numerical

categorical_columns = ['sick', 'on thyroxine','query on thyroxine','on antithyroid medication','pregnant','thyroid surgery','I131 treatment',

'query hypothyroid','lithium','query hyperthyroid','goitre','tumor','hypopituitary','psych','binaryClass']

ordinal_encoder = OrdinalEncoder()

td[categorical_columns] =

ordinal_encoder.fit_transform(td[categorical_columns]) print(td.head())

```
   age  sex  on thyroxine  query on thyroxine  on antithyroid medication  \
0   41  2.0           0.0                 0.0                        0.0
1   23  2.0           0.0                 0.0                        0.0
2   46  1.0           0.0                 0.0                        0.0
3   70  2.0           1.0                 0.0                        0.0
4   70  2.0           0.0                 0.0                        0.0

   sick  pregnant  thyroid surgery  I131 treatment  query hypothyroid  ...  \
0   0.0       0.0              0.0             0.0                0.0  ...
1   0.0       0.0              0.0             0.0                0.0  ...
2   0.0       0.0              0.0             0.0                0.0  ...
3   0.0       0.0              0.0             0.0                0.0  ...
4   0.0       0.0              0.0             0.0                0.0  ...

   lithium  goitre  tumor  hypopituitary  psych   TSH      T3    T4U  \
0      0.0     0.0    0.0            0.0    0.0  1.30  2.5000  1.140
1      0.0     0.0    0.0            0.0    0.0  4.10  2.0000  0.995
2      0.0     0.0    0.0            0.0    0.0  0.98  2.0135  0.910
3      0.0     0.0    0.0            0.0    0.0  0.16  1.9000  0.995
4      0.0     0.0    0.0            0.0    0.0  0.72  1.2000  0.870

          FTI  binaryClass
0  109.000000          1.0
1  110.469649          1.0
2  120.000000          1.0
3  110.469649          1.0
4   70.000000          1.0
```

#Converting every features in the dataset as int datatype

```
#converting every features types as int

columns_to_convert = [col for col in td.columns]
td[columns_to_convert] = td[columns_to_convert].astype(int)

print(td.head())
```

5.BUILDING MODEL:

#Splitting The Dataset For Train And Test

```
X=td.drop('binaryClass',axis='columns')
y=td.binaryClass
```

```
#Spliting the dataset to train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.20,
                                                    stratify=y,
                                                    random_state=13)
```

Standardizing the features:

#Defining a function called compute_results that takes in 6 arguments: the training and testing data matrices, the corresponding labels, a sampling strategy, and a classifier.

```
    def compute_results(X_train, y_train, X_test, y_test, sampler,
classifier): X_train_res, y_train_res = sampler.fit_resample(X_train,
                                        y_train)
    classifier.fit(X_train_res, y_train_res)
    y_pred = classifier.predict(X_test)
#To store evaluation metrics for a single classifier-sampler combination.
    metrics = dict()
    metrics['Accuracy'] = accuracy_score(y_test, y_pred)
    metrics['Precision'] = precision_score(y_test, y_pred)
    metrics['Recall'] = recall_score(y_test, y_pred)
    metrics['F-score'] = f1_score(y_test, y_pred)

#Error matrix is created using the confusion matrix generated by comparing the true labels with the predicted labels
    error_matrix = pd.DataFrame(
        confusion_matrix(y_test, y_pred, labels=[1, 0]),
        columns=['a(x) = 1', 'a(x) = 0'],
        index=['y = 1', 'y = 0'],
    )
```

```
    return metrics, error_matrix
#multiple over sampling are used
samplers = {
    'Random Over Sampling' : RandomOverSampler(random_state=13),
    'SMOTE' : SMOTE(sampling_strategy=0.4, random_state=13,
    k_neighbors=5), 'ADASYN' : ADASYN(random_state=13, n_neighbors=5)
}
#'classifiers' containing three different classifiers
classifiers = {
    'RandomForestClassifier': RandomForestClassifier(n_estimators=50, class_weight='balanced',
random_state=13),
    'SVC': SVC(kernel='linear', class_weight='balanced', random_state=13),
     'LogisticRegression': LogisticRegression(class_weight='balanced',
random_state=13) }
#To aggregate and compare evaluation metrics across multiple classifier-sampler combinations.
sampler_metrics = dict()
for classifier_name in classifiers: # Iterates over each classifier in the
    "classifiers" for sampler in samplers: # Iterates over each sampler in the
    "samplers"
        classifier = classifiers[classifier_name]
        #Calls compute_results function to evaluate the classifier with the current sampler.
        sampler_metric, error_matrix = compute_results(X_train, y_train, X_test, y_test,
samplers[sampler], classifier)
        sampler_metrics[(sampler, classifier_name)] = {
            'Accuracy': sampler_metric['Accuracy'],
            'Precision': sampler_metric['Precision'],
            'Recall': sampler_metric['Recall'],
            'F1_score': sampler_metric['F-score']
```

}

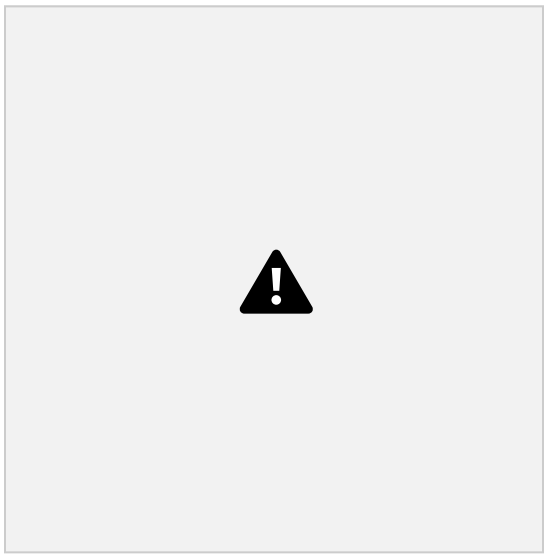
6.FINAL RESULTS:

```
metrics_df = pd.DataFrame(sampler_metrics)

metrics_df.style.highlight_max(color='lightblue',axis=1)
```

The "ADASYN" method resulted in the highest accuracy of 0.993377 when using the RandomForestClassifier, compared to other oversampling methods such as "Random Over Sampling" and "SMOTE" with accuracies of 0.990728 and 0.992053, respectively.

Output of confusion matrix:

```
conf_matrix = error_matrix.to_numpy()

plt.figure(figsize=(5,5))

sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', xticklabels=error_matrix.columns, yticklabels=error_matrix.index, cbar=False)

plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix')

plt.show()
```



ROC AUC Score:

```
from sklearn.metrics import roc_curve,auc,roc_auc_score

y_pred_prob=classifier.predict_proba(X_test)

fpr4,tpr4,threshold=roc_curve(y_test,y_pred_prob[:,1])

roc_auc4=auc(fpr4,tpr4)

print(roc_auc4)
```

0.973754514421412

The ROC AUC score of 0.973754514421412 indicates that the RandomForest classifier has a high performance on the given test set.

CONCLUSION AND FUTURE ENHANCEMENTS:

Thyroid disease is one of the diseases that afflict the world's population, and the number of cases of this disease is increasing. Because of medical reports that show serious imbalances in thyroid diseases, our study deals with the classification of thyroid disease between hyperthyroidism and hypothyroidism. This disease was classified using algorithms. RandomForestClassifier has the highest accuracy (0.993377) using ADASYN.SVC and Logistic Regression has the lowest accuracy (0.964238) with Random Over sampling.

In order to generalize our findings more accurately, it is essential to broaden the dataset and attributes considered in the future. As the amount of data increases, the training process is likely to generate more efficient classifiers and provide a more reliable assessment of the observed

performance. Another area for investigation could be examining the presence of any secondary thyroid diseases in patients, as it is common for patients to have more than one thyroid disease simultaneously. Understanding if a specific additional thyroid disease influences hypothyroidism could be beneficial.

REFERENCES

1.Aversano, L. B. (2021). Thyroid disease treatment prediction with machine learning approaches. . *Procedia Computer Science,*, 192, 1031-1040.

2.Singh, T. S. (2022). Treatment of thyroid disease through machine learning predictive model. *International Journal of Health Sciences, 6(S8)*, 3176–3188.

3.Sonuç, E. (2021, July). Thyroid disease classification using machine learning algorithms. *In Journal of Physics: Conference Series* , Vol. 1963, No. 1, p. 012140.

4.Tyagi, A. M. (2018, December). Interactive thyroid disease prediction system using machine learning technique. . *In 2018 Fifth international conference on parallel, distributed and grid computing (PDGC)* , (pp. 689-693). .

5.Viswanatha, V. (2023). Thyroid Disease Detection Using Machine Learning approach.