thyroid\_prediction

## Importation of modules and Processed Dataset[¶](#X05bb4afad28ec89c1f3d609f15e3678b94161b6)

In [2]:

import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split,GridSearchCV  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import confusion\_matrix,accuracy\_score,precision\_score,f1\_score,recall\_score,ConfusionMatrixDisplay  
from sklearn.preprocessing import StandardScaler  
from sklearn.metrics import roc\_auc\_score, classification\_report, confusion\_matrix, roc\_curve, f1\_score  
from sklearn import tree  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.model\_selection import cross\_val\_predict, KFold

In [3]:

data=pd.read\_csv("/content/drive/MyDrive/dataset/processed\_thyroid\_data.csv")  
data

Out[3]:

|  |  |  |
| --- | --- | --- |
|  |  |  |

3774 rows × 29 columns

## Examining of dataset and sclaing splitting of dataset for training and testing[¶](#X5fbd8b50b9fff0971368457ccda1f086a57dd26)

In [26]:

data = data.dropna()  
data.tail()

Out[26]:

5 rows × 29 columns

In [5]:

#data = data.reset\_index(drop=True)  
data.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 3772 entries, 0 to 3771  
Data columns (total 29 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 age 3772 non-null float64  
 1 sex 3772 non-null float64  
 2 on thyroxine 3772 non-null float64  
 3 query on thyroxine 3772 non-null float64  
 4 on antithyroid medication 3772 non-null float64  
 5 sick 3772 non-null float64  
 6 pregnant 3772 non-null float64  
 7 thyroid surgery 3772 non-null float64  
 8 I131 treatment 3772 non-null float64  
 9 query hypothyroid 3772 non-null float64  
 10 query hyperthyroid 3772 non-null float64  
 11 lithium 3772 non-null float64  
 12 goitre 3772 non-null float64  
 13 tumor 3772 non-null float64  
 14 hypopituitary 3772 non-null float64  
 15 psych 3772 non-null float64  
 16 TSH measured 3772 non-null float64  
 17 TSH 3772 non-null float64  
 18 T3 measured 3772 non-null float64  
 19 T3 3772 non-null float64  
 20 TT4 measured 3772 non-null float64  
 21 TT4 3772 non-null float64  
 22 T4U measured 3772 non-null float64  
 23 T4U 3772 non-null float64  
 24 FTI measured 3772 non-null float64  
 25 FTI 3772 non-null float64  
 26 TBG measured 3772 non-null float64  
 27 referral source 3772 non-null float64  
 28 binaryClass 3772 non-null float64  
dtypes: float64(29)  
memory usage: 884.1 KB

In [6]:

data.isnull().sum()

Out[6]:

|  |  |
| --- | --- |
|  | 0 |
| 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 measured | 0 |
| TSH | 0 |
| T3 measured | 0 |
| T3 | 0 |
| TT4 measured | 0 |
| TT4 | 0 |
| T4U measured | 0 |
| T4U | 0 |
| FTI measured | 0 |
| FTI | 0 |
| TBG measured | 0 |
| referral source | 0 |
| binaryClass | 0 |

**dtype:** int64

In [7]:

standardScaler = StandardScaler()  
columns\_to\_scale = ['age', 'TT4', 'TSH', 'T3','T4U','FTI']  
data[columns\_to\_scale] = standardScaler.fit\_transform(data[columns\_to\_scale])

<ipython-input-7-26316fcf818a>:3: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 data[columns\_to\_scale] = standardScaler.fit\_transform(data[columns\_to\_scale])

In [8]:

y = data["binaryClass"]  
X = data.drop('binaryClass', axis=1)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=5)

## Model 1 (Random Forest Classifier)[¶](#Model-1-(Random-Forest-Classifier))

In [9]:

from sklearn.ensemble import RandomForestClassifier  
model1=RandomForestClassifier()  
model1.fit(X\_train,y\_train)

Out[9]:

RandomForestClassifier()

 RandomForestClassifier[?Documentation for RandomForestClassifier](https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html)iFitted

RandomForestClassifier()

In [10]:

y\_predict1=model1.predict(X\_test)

In [11]:

v=confusion\_matrix(y\_test,y\_predict1)  
cm=ConfusionMatrixDisplay(confusion\_matrix=v)  
cm.plot()

Out[11]:

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7dbfee0a92d0>

![No description has been provided for this image](data:image/png;base64;base64,)

In [12]:

accuracy = accuracy\_score(y\_test,y\_predict1)  
print(f'Accuracy: {accuracy \* 100:.2f}%')  
precision = precision\_score(y\_test,y\_predict1)  
print(f'Precision: {precision \* 100:.2f}%')  
recall=recall\_score(y\_test,y\_predict1)  
print(f'Recall: {recall \* 100:.2f}%')  
f1 = f1\_score(y\_test, y\_predict1, average='binary')  
print(f'F1 Score: {f1\*100:.2f}')  
  
tn, fp, fn, tp = confusion\_matrix(y\_test, y\_predict1).ravel()  
  
  
specificity = tn / (tn + fp)  
print(f'Specificity: {specificity\*100:.2f}')  
  
fpr = fp / (fp + tn)  
print(f'False Positive Rate: {fpr\*100:.2f}')  
  
fnr = fn / (fn + tp)  
print(f'False Negative Rate: {fnr\*100:.2f}')

Accuracy: 99.74%  
Precision: 99.85%  
Recall: 99.85%  
F1 Score: 99.85  
Specificity: 98.59  
False Positive Rate: 1.41  
False Negative Rate: 0.15

## Model 2 (SVM)[¶](#Model-2-(SVM))

In [13]:

from sklearn.svm import SVC  
  
model2 = SVC(C=30 ,kernel= 'rbf')

In [14]:

model2.fit(X\_train,y\_train)  
y\_predict2=model2.predict(X\_test)

In [15]:

v=confusion\_matrix(y\_test,y\_predict2)  
cm=ConfusionMatrixDisplay(confusion\_matrix=v)  
cm.plot()

Out[15]:

![No description has been provided for this image](data:image/png;base64;base64,)

In [16]:

accuracy = accuracy\_score(y\_test,y\_predict2)  
print(f'Accuracy: {accuracy \* 100:.2f}%')  
precision = precision\_score(y\_test,y\_predict2)  
print(f'Precision: {precision \* 100:.2f}%')  
recall=recall\_score(y\_test,y\_predict2)  
print(f'Recall: {recall \* 100:.2f}%')  
f1 = f1\_score(y\_test, y\_predict2, average='binary')  
print(f'F1 Score: {f1\*100:.2f}')  
  
tn, fp, fn, tp = confusion\_matrix(y\_test, y\_predict2).ravel()  
  
specificity = tn / (tn + fp)  
print(f'Specificity: {specificity\*100:.2f}')  
  
fpr = fp / (fp + tn)  
print(f'False Positive Rate: {fpr\*100:.2f}')  
  
fnr = fn / (fn + tp)  
print(f'False Negative Rate: {fnr\*100:.2f}')

Accuracy: 97.22%  
Precision: 97.42%  
Recall: 99.56%  
F1 Score: 98.48  
Specificity: 74.65  
False Positive Rate: 25.35  
False Negative Rate: 0.44

In [16]:

## Model 3 (Logistic Regression)[¶](#Model-3-(Logistic-Regression))

In [17]:

model3=LogisticRegression(C= 20, penalty= 'l1', solver= 'liblinear')  
model3.fit(X\_train,y\_train)  
y\_predict3=model3.predict(X\_test)

In [18]:

v=confusion\_matrix(y\_test,y\_predict3)  
cm=ConfusionMatrixDisplay(confusion\_matrix=v)  
cm.plot()

Out[18]:

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7dbfecb70a60>

![No description has been provided for this image](data:image/png;base64;base64,)

In [19]:

accuracy = accuracy\_score(y\_test,y\_predict3)  
print(f'Accuracy: {accuracy \* 100:.2f}%')  
precision = precision\_score(y\_test,y\_predict3)  
print(f'Precision: {precision \* 100:.2f}%')  
recall=recall\_score(y\_test,y\_predict3)  
print(f'Recall: {recall \* 100:.2f}%')  
f1 = f1\_score(y\_test, y\_predict3, average='binary')  
print(f'F1 Score: {f1\*100:.2f}')  
  
tn, fp, fn, tp = confusion\_matrix(y\_test, y\_predict3).ravel()  
  
# Calculate specificity  
specificity = tn / (tn + fp)  
print(f'Specificity: {specificity\*100:.2f}')  
  
# Calculate false positive rate (FPR)  
fpr = fp / (fp + tn)  
print(f'False Positive Rate: {fpr\*100:.2f}')  
  
# Calculate false negative rate (FNR)  
fnr = fn / (fn + tp)  
print(f'False Negative Rate: {fnr\*100:.2f}')

Accuracy: 96.29%  
Precision: 96.33%  
Recall: 99.71%  
F1 Score: 97.99  
Specificity: 63.38  
False Positive Rate: 36.62  
False Negative Rate: 0.29

In [19]:

## Model 4 (Decision Tree)[¶](#Model-4-(Decision-Tree))

In [20]:

model4 = DecisionTreeClassifier()  
model4.fit(X\_train, y\_train)

Out[20]:

DecisionTreeClassifier()

DecisionTreeClassifier[?Documentation for DecisionTreeClassifier](https://scikit-learn.org/1.5/modules/generated/sklearn.tree.DecisionTreeClassifier.html)iFitted

DecisionTreeClassifier()

In [21]:

y\_predict4 = model4.predict(X\_test)  
accuracy = accuracy\_score(y\_test, y\_predict4)  
print(f'Accuracy: {accuracy \* 100:.2f}%')  
precision = precision\_score(y\_test,y\_predict4)  
print(f'Precision: {precision \* 100:.2f}%')  
recall=recall\_score(y\_test,y\_predict4)  
print(f'Recall: {recall \* 100:.2f}%')  
f1 = f1\_score(y\_test, y\_predict4, average='binary')  
print(f'F1 Score: {f1\*100:.2f}')  
  
tn, fp, fn, tp = confusion\_matrix(y\_test, y\_predict4).ravel()  
  
# Calculate specificity  
specificity = tn / (tn + fp)  
print(f'Specificity: {specificity\*100:.2f}')  
  
# Calculate false positive rate (FPR)  
fpr = fp / (fp + tn)  
print(f'False Positive Rate: {fpr\*100:.2f}')  
  
# Calculate false negative rate (FNR)  
fnr = fn / (fn + tp)  
print(f'False Negative Rate: {fnr\*100:.2f}')

Accuracy: 99.74%  
Precision: 99.85%  
Recall: 99.85%  
F1 Score: 99.85  
Specificity: 98.59  
False Positive Rate: 1.41  
False Negative Rate: 0.15

In [21]:

## Model 5 (KNN)[¶](#Model-5-(KNN))

In [22]:

from sklearn.neighbors import KNeighborsClassifier

In [23]:

model5 = KNeighborsClassifier(n\_neighbors=3)  
model5.fit(X\_train, y\_train)

Out[23]:

KNeighborsClassifier(n\_neighbors=3)

KNeighborsClassifier[?Documentation for KNeighborsClassifier](https://scikit-learn.org/1.5/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)iFitted

KNeighborsClassifier(n\_neighbors=3)

In [24]:

y\_predict5 = model5.predict(X\_test)  
accuracy = accuracy\_score(y\_test, y\_predict5)  
print(f'Accuracy: {accuracy \* 100:.2f}%')  
precision = precision\_score(y\_test,y\_predict5)  
print(f'Precision: {precision \* 100:.2f}%')  
recall=recall\_score(y\_test,y\_predict5)  
print(f'Recall: {recall \* 100:.2f}%')  
f1 = f1\_score(y\_test, y\_predict5, average='binary')  
print(f'F1 Score: {f1\*100:.2f}')  
  
tn, fp, fn, tp = confusion\_matrix(y\_test, y\_predict5).ravel()  
  
# Calculate specificity  
specificity = tn / (tn + fp)  
print(f'Specificity: {specificity\*100:.2f}')  
  
# Calculate false positive rate (FPR)  
fpr = fp / (fp + tn)  
print(f'False Positive Rate: {fpr\*100:.2f}')  
  
# Calculate false negative rate (FNR)  
fnr = fn / (fn + tp)  
print(f'False Negative Rate: {fnr\*100:.2f}')

Accuracy: 93.64%  
Precision: 93.68%  
Recall: 99.71%  
F1 Score: 96.60  
Specificity: 35.21  
False Positive Rate: 64.79  
False Negative Rate: 0.29

In [24]:

## Visualizing the clustering[¶](#Visualizing-the-clustering)

In [34]:

from sklearn.decomposition import PCA  
labels\_rfc = model1.predict(X\_train)  
labels\_svm = model2.predict(X\_train)  
labels\_lr = model3.predict(X\_train)  
labels\_dtc = model4.predict(X\_train)  
labels\_knn = model5.predict(X\_train)  
  
model\_labels = [labels\_rfc, labels\_svm, labels\_lr, labels\_dtc, labels\_knn]  
model\_names = ['RandomForestClassifier', 'SVM', 'Logistic Regression', 'Decision Tree Classifier','KNN']  
  
plt.figure(figsize=(12, 8))  
pca = PCA(n\_components=28)  
pca\_result = pca.fit\_transform(X\_train)  
markers = ['o', 's', 'D', '^', 'v']  
colors = ['b', 'g', 'r', 'c', 'm']  
for i, (name, labels, marker, color) in enumerate(zip(model\_names, model\_labels, markers, colors)):  
 sns.scatterplot(x=pca\_result[:, 0], y=pca\_result[:, 1], hue=labels, palette='viridis', legend=None, alpha=0.6, marker=marker, edgecolor='w', label=name)  
plt.title('Clustering Results of All Models')  
plt.xlabel('Principal Component 1')  
plt.ylabel('Principal Component 2')  
plt.legend(title='Models')  
plt.show()

![No description has been provided for this image](data:image/png;base64;base64,)

In [25]: