EVALUATION OF MACHINE LEARNING TECHNIQUES FOR PCOS PREDICTION

Submitted in fulfillment for the J Component of ITA6004 – Soft Computing

in

MCA

by

Kamran Ansari (22MCA0223)

Bhooshan Birje (22MCA0233)

Under the guidance of

Dr. Anitha A
Associate Professor Sr.
SITE

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

1. ABSTRACT :

Polycystic Ovary Syndrome (PCOS) is a hormonal imbalance brought on by the ovaries producing too many male hormones, which affects a woman's reproductive system during her childbearing age. PCOS has been identified as of the major factors of infertility in women. It also raises the likelihood of developing other illnesses including diabetes, high blood pressure, sleep difficulties, depression, and anxiety, among others. The goal of this research was to address this issue by implementing machine learning approaches for early PCOS identification based on several medical markers. The model used a dataset of 541 women from 10 hospitals in Kerala. This paper examined trends and correlation within the medical parameters and determined the attributes which are most important in the diagnosis of PCOS. SMOTE was performed to balance the dataset. 15 important features were found most relevant to the target using Extra-Tree classifier. This paper experimented with various Classification models and compared them using accuracy and sensitivity. Hyperparameter tuning was performed to model to improve their performances. Out of all models, Random Forest gave the best results with 94% accuracy and 93% sensitivity. Furthermore, Number of follicles in the ovaries, skin darkening and age came out to be the most influential features for the prediction of PCOS.

Keywords: PCOS, dataset, feature selection, machine learning, classification models, hyperparameter tuning, feature importance

2. INTRODUCTION WITH RELATED WORK :

Polycystic Ovary Syndrome (PCOS) is a prevalent endocrine disease that affects a significant number of women of child-bearing age. PCOS is characterized by various symptoms, including irregular menstrual cycles, skin infections, weight gain, high blood pressure, and an increased risk of diabetes. Additionally, PCOS can lead to anxiety and depression, making early detection and diagnosis crucial for managing the condition and preventing further complications. However, diagnosing PCOS can be challenging due to the vagueness of its symptoms and the lack of awareness among both patients and healthcare providers. Moreover, the coordination between different physicians and the availability and cost of medical tests can further delay the diagnosis process. To address these challenges, machine learning techniques have emerged as valuable tools in the healthcare sector.Machine learning offers the ability to process large volumes of patient data and derive clinical insights that aid in the diagnosis of diseases. By analyzing medical parameters and patterns within the data, machine learning models can predict the probability of PCOS in a patient. This approach reduces the potential for human error, enables early diagnosis, and facilitates cost-effective and easily accessible diagnostic methods. To highlight the research conducted in this domain, several studies have been reviewed. Prapty and Shitu (2020) conducted an analysis of decision tree models for PCOS using a dataset of 542 data points. Bharati et al. (2020) focused on the diagnosis of PCOS using machine learning algorithms, utilizing hybrid random forest and logistic regression models with 40-fold cross-validation. Abu Adla et al. (2021) employed a dataset of 541 data points and implemented SVM with K-Fold Cross Validation for automated PCOS detection. Chauhan et al. (2021) compared various machine learning algorithms and

found that the decision tree classifier performed best in terms of accuracy, precision, and specificity. Nabi et al. (2021) developed a machine learning-based approach for detecting PCOS in Bangladeshi women, achieving high accuracy using techniques such as SVM, XGBoost, Gaussian Naive Bayes, logistic regression, decision trees, and gradient boosting classifiers. The combination of machine learning and PCOS diagnosis has the potential to revolutionize the early detection and management of PCOS. By leveraging machine learning models and analyzing relevant medical parameters, clinicians can make more informed decisions and provide timely interventions. This project aims to develop a diagnostic model using machine learning techniques, enabling accurate PCOS prediction and reducing the complexity and cost of diagnosis. Ultimately, this approach will contribute to improving the quality of healthcare services and enhancing the well-being of individuals affected by PCOS.

3. BACKGROUND FUNDAMENTALS :

a. Polycystic Ovary Syndrome (PCOS) :

Polycystic Ovary Syndrome (PCOS) is a hormonal disorder that affects women of reproductive age. It is characterized by the presence of multiple small cysts on the ovaries, menstrual irregularities, and elevated androgen levels. PCOS can lead to various symptoms such as infertility, weight gain, excessive hair growth, acne, and mood swings. It is associated with long-term health risks, including insulin resistance, type 2 diabetes, cardiovascular disease, and mental health issues. Diagnosis involves evaluating symptoms, hormone testing, ultrasound imaging, and ruling out other conditions. Management may include lifestyle changes, medication to regulate hormones, and fertility treatments, depending on individual needs.

b. Machine Learning in Healthcare :

Machine Learning in healthcare refers to the application of computational algorithms that can automatically learn patterns from data to make predictions, diagnose diseases, and improve patient care. It involves training models using large datasets to identify patterns, relationships, and anomalies in medical data. Machine Learning algorithms can assist in disease diagnosis, risk assessment, treatment planning, and patient monitoring. By leveraging machine learning, healthcare systems can improve accuracy, efficiency, and personalized healthcare delivery, leading to better outcomes for patients.

c. Extra Tree Classifier for feature selection :

The Extra Trees Classifier is an ensemble learning algorithm that combines decision trees and random forests for classification tasks. However, it can also be leveraged as a powerful tool for feature extraction in machine learning. During the feature extraction process, the algorithm constructs multiple decision trees using random subsets of features and training data. These trees are grown to their full depth without pruning, resulting in an ensemble of diverse and

highly expressive trees. As the trees are built, the Extra Trees Classifier calculates the importance of each feature based on its contribution to the purity of the splits. By measuring feature importance, the algorithm generates a ranking that identifies the most informative and discriminative features for the classification task at hand. Features with higher importance scores are considered more influential and can be selected for further analysis or used as input for downstream machine learning models. The Extra Trees Classifier's ability to handle high-dimensional datasets and capture complex feature interactions makes it an effective and efficient technique for extracting relevant features. By selecting the most important features, the algorithm simplifies the data representation, enhances computational efficiency, and potentially improves the generalization ability of the model.

d. SMOTE :

SMOTE, which stands for Synthetic Minority Over-sampling Technique, is a popular algorithm used in the field of imbalanced learning. It addresses the challenge of imbalanced datasets, where one class is significantly underrepresented compared to the other. SMOTE works by synthesizing new minority class samples based on the existing minority class instances. It does this by selecting a random minority class sample and generating synthetic samples along the line segments connecting it to its nearest neighbors. This oversampling technique helps to balance the class distribution and mitigate the bias towards the majority class. By creating synthetic samples, SMOTE increases the diversity of the minority class, making it easier for classifiers to learn from these instances. However, it is important to note that SMOTE should be applied carefully, considering the potential impact on the model's generalization ability and the specific characteristics of the dataset.

e. Hyperparameter tuning :

Hyperparameter tuning is the process of finding the optimal values for the hyperparameters of a machine learning algorithm. Hyperparameters are settings that are not learned from the data but are set before the learning process begins. Tuning involves systematically exploring different combinations of hyperparameter values to find the configuration that yields the best model performance. This is typically done using techniques like grid search, random search, or Bayesian optimization. Hyperparameter tuning helps improve the model's performance, generalization ability, and robustness by finding the best set of hyperparameters for a given dataset and learning task.

f. Algorithms :

i. XGBRF (Extreme Gradient Boosting with Random Forest) : XGBRF is an ensemble learning algorithm that combines the concepts of gradient boosting and random forests. It is particularly effective for handling tabular data with complex relationships and achieving high predictive

accuracy. XGBRF combines multiple decision trees to make predictions and utilizes boosting techniques to iteratively improve the model's performance.

ii. Logistic Regression : Logistic regression is a statistical algorithm used for binary classification tasks. It models the relationship between a set of independent variables and a binary outcome using a logistic function. Logistic regression is often used when the outcome variable represents the probability of a certain event occurring.

iii. K-Nearest Neighbors (KNN) : KNN is a simple and intuitive algorithm used for both classification and regression tasks. It operates by assigning a new data point to the class or category that is most common among its k nearest neighbors in the feature space. KNN is a non-parametric algorithm that doesn't make any assumptions about the underlying data distribution.

iv. SVM (Support Vector Machines): SVM is a powerful algorithm for both classification and regression tasks. It finds an optimal hyperplane that separates different classes or predicts continuous values by maximizing the margin between the classes. SVM can handle linear and non-linear data by using different kernel functions.

v. Decision Tree : Decision tree algorithms build a tree-like model of decisions and their possible consequences. Each internal node represents a test on a particular feature, each branch represents the outcome of the test, and each leaf node represents a class label or a predicted value. Decision trees are interpretable and can handle both categorical and numerical data.

vi. Random Forest : Random forest is an ensemble learning method that combines multiple decision trees. It improves the predictive performance and reduces overfitting by aggregating the predictions of individual trees. Each tree is trained on a random subset of the training data, and the final prediction is obtained by majority voting or averaging the predictions of the individual trees.

vii. Cat Boost Classifier : Cat Boost is a gradient boosting algorithm designed to handle categorical variables efficiently. It uses an optimized algorithm to handle categorical features and can automatically handle missing data. Cat Boost can be effective in handling complex datasets and achieving high predictive accuracy.

4. PROPOSED METHODOLOGY :

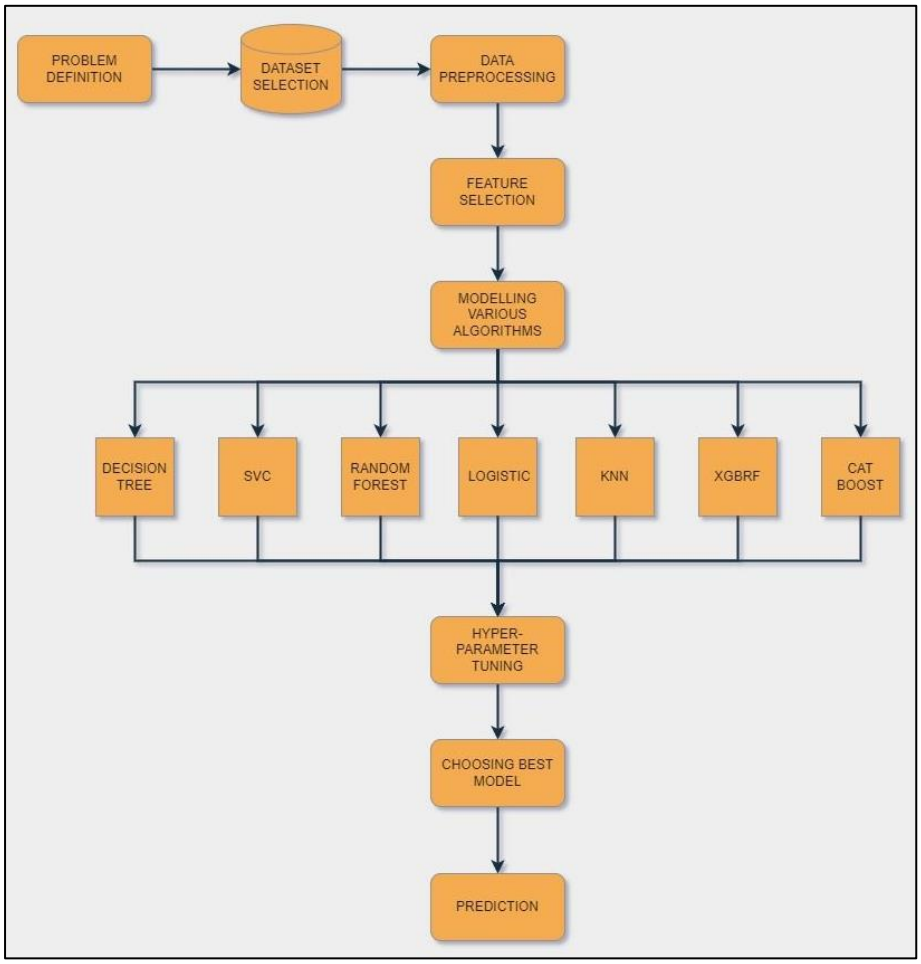

Figure 1. Methodology

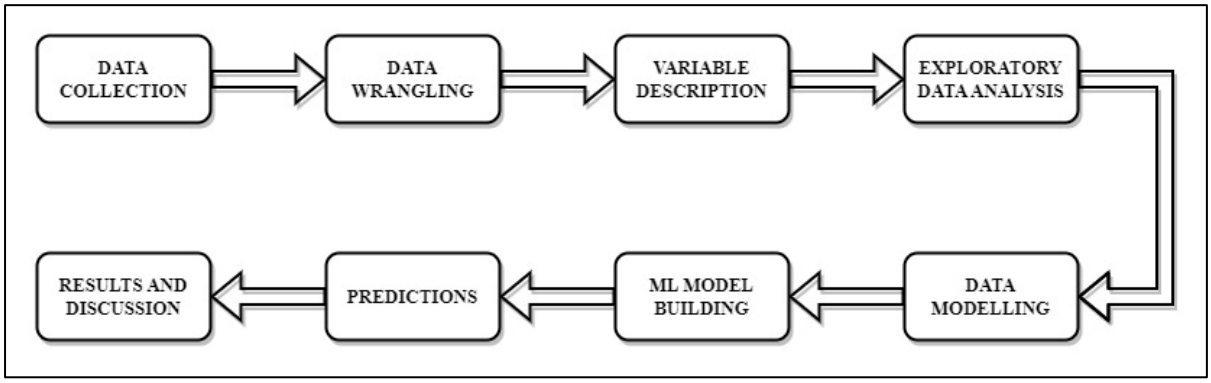

Figure 2. Flow Chart

1. **Data Wrangling -**
    a. Dropped a column ‘Unnamed:44’ which had empty cells
    b. Also dropped all the duplicate columns

c. Removed descriptive columns which that were not needed for processing such as “Serial No.” and “Patient file no.”
d. Some columns had values encoded as objects. Standardized all of them to numeric values
e. Filled the cells with null values with the median of the column values
f. Removed leading and trailing spaces in column names

2. **Variable Description-** This includes:
   a. The name of the variable and a brief description of what it represents
   b. The type of the variable (e.g., categorical, numerical, continuous, ordinal, etc.)
   c. The range of values that the variable can take on (e.g., for numerical variables, the minimum and maximum values; for categorical variables, the possible categories)

3. **Feature Selection –**

   Feature Selection was done using Extra Tree Classifier. Feature Importance of all the variables was calculated and top 15 features were selected for futher processing.

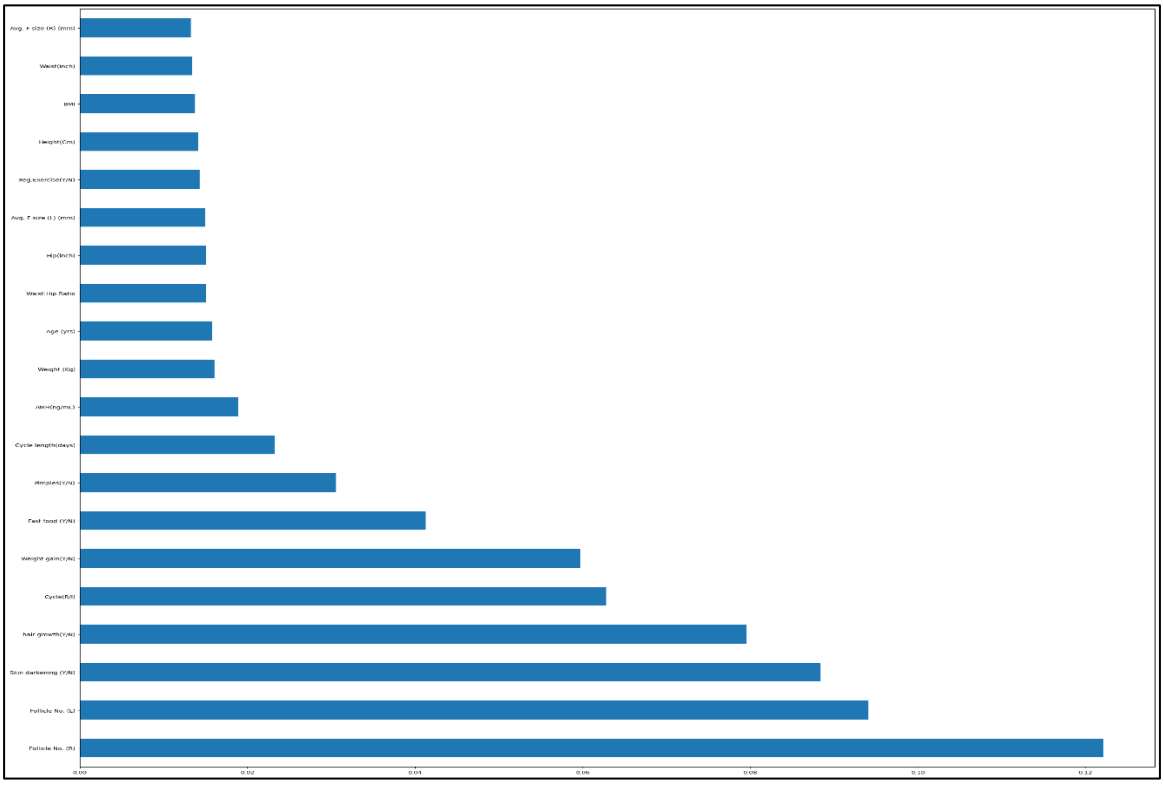

Figure 3. Feature Importance

1. Follicle No. (R)	2. Follicle No. (L)
3. Skin darkening (Y/N)	4. hair growth(Y/N)
5. Weight gain(Y/N)	6. Cycle(R/I)
7. Fast food (Y/N)	8. Pimples(Y/N)
9. AMH (ng/mL)	10. Avg. F size (L) (mm)
11. Hip (inch)	12. Cycle length(days)
13. Age (years)	14. Waist (inch)
15. Avg. F size (R) (mm)	

4. **Exploratory Data Analysis-** Here, we explored all the features selected using various graphical components. Association of variables with the target and also with each other.

   Inferences:

   i. Menstrual Cycle length seemed to be shorter and also the length increased as the patient age was greater. While the negative target patients had same cycle length throughout the age groups.
   ii. BMI of the positive patients increased drastically as the age increased while the negative patients had stable BMI.
   iii. Also, patients with PCOS had higher irregularity in the menstruation as compared to negative patients.
   iv. Number of follicles in both left and right ovaries of the PCOS patients were significantly higher than that of negative ones.
   v. Surprisingly, a greater number of younger patients had PCOS than the elder ones. But this could be because of the unawareness and readiness to treat the disorder.
   vi. Weight of patients with PCOS was significantly higher than that of the non-PCOS patients.
   vii. PCOS patients had slightly lower Hb levels.

5. **Data Modelling-**
   a. **SMOTE –** SMOTE was applied to balance the number of output labels. After SMOTE, both ‘yes’ and ‘no’ labels had 346 tuples each.
   b. **Splitting Data-** Data was split into 2 groups for training and testing the models. 70% of the data was considered for training while remaining 30% for testing. Assigning 15 independent variables to X and target variable to Y.

6. **Model Building-**
   a. **List of Models employed in this system**:
      i. XGBRF
      ii. Logistic Regression
      iii. K-Nearest Neighbors
      iv. SVM
      v. Decision Tree
      vi. Random Forest

vii. Cat Boost Classifier

We built above models using 'sklearn' library in python. Metrics were calculated for all the models for further evaluation. Models used default hyperparameters for building.

b. **Hyperparameter tuning-** We have used Grid Search CV in our system. Range of values for each parameter were given. Scoring parameter for the CV was set to 'accuracy'. Grid Search returned with the best parameter for all the models which helps the model achieve highest accuracy.
After finding best parameters. Again, models were trained and tested but with their best hyperparameters.

5. EXPERIMENTAL ANALYSIS WITH CODE :

IMPORTING LIBRARIES :

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
import lightgbm as lgb
import xgboost as xgb
from imblearn.over_sampling import SMOTEN
import pandas as pd

from collections import Counter
from mlxtend.plotting import plot_confusion_matrix
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
import numpy as np
!pip install catboost
```

```
from catboost import CatBoostClassifier


import warnings
warnings.filterwarnings('ignore')
```

DATA LOADING :

```
# ! gdown --id 1p3Senw-yrE6xSqXM6A5rJzaRGbALBgIq, --id is deprecated
import os
if os.path.isfile("/content/PCOS_data_without_infertility.xlsx") == False:
! gdown 1p3Senw-yrE6xSqXM6A5rJzaRGbALBgIq


data = pd.read_excel('PCOS_data_without_infertility.xlsx',sheet_name= 'Full_new')
data.columns
data.head()
```

DATA CLEANING :

```
data =data.drop(['Unnamed: 44'],axis = 1)
data =data.drop(['Sl. No', 'Patient File No.'],axis = 1)
data.head()
data.columns


data.info()
data["AMH(ng/mL)"].head()
data["II    beta-HCG(mIU/mL)"].head()
data["AMH(ng/mL)"] = pd.to_numeric(data["AMH(ng/mL)"], errors='coerce')
data["II    beta-HCG(mIU/mL)"] = pd.to_numeric(data["II    beta-HCG(mIU/mL)"], errors='coerce').

data['Marraige Status (Yrs)'].fillna(data['Marraige Status (Yrs)'].median(),inplace=True)
data['II    beta-HCG(mIU/mL)'].fillna(data['II    beta-HCG(mIU/mL)'].median(),inplace=True)
data['AMH(ng/mL)'].fillna(data['AMH(ng/mL)'].median(),inplace=True)
data['Fast food (Y/N)'].fillna(data['Fast food (Y/N)'].median(),inplace=True)
```

```
data.columns = [col.strip() for col in data.columns]

#Dropping the outliers.
data = data[(data["BP _Diastolic (mmHg)"]>20)]
data = data[(data["AMH(ng/mL)"]<40)]
data = data[(data["BP _Systolic (mmHg)"]>20)]
data = data[(data["Endometrium (mm)"]>0)]
data = data[(data["Avg. F size (L) (mm)"]>0)]
data = data[(data["Avg. F size (R) (mm)"]>0)]
data = data[(data["RBS(mg/dl)"]<200)]
data = data[(data["PRG(ng/mL)"]<20)]
data = data[(data["Pulse rate(bpm)"]>20)]
data = data[(data["FSH(mIU/mL)"]<4000)]
data = data[(data["LH(mIU/mL)"]<1500)]
data = data[(data["Cycle(R/I)"]<4.5)]

data.shape
X = data.drop(['PCOS (Y/N)'], axis = 1)
y = data['PCOS (Y/N)']

print(X.columns)
```

FEATURE SELECTION :

```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt

model = ExtraTreesClassifier()
model.fit(X,y)

print(model.feature_importances_)

feat_importances = pd.Series(model.feature_importances_, index=X.columns)
```

```
feat_importances.nlargest(15).plot(kind='barh')
plt.rcParams["figure.figsize"] = (30,30)
plt.show()

fimp = feat_importances.nlargest(15)
df_final = data[fimp.index]
df_final.columns
```

DATA MODELLING :

```
sm = SMOTEN(random_state=42)
X_sm, y_sm = sm.fit_resample(df_final, y)

X_sm = pd.DataFrame(X_sm,
columns=df_final.columns)
y_sm = pd.DataFrame(y_sm, columns=['PCOS (Y/N)'])

print('New balance of 1 and 0 classes (%):')
y_sm.value_counts()

#Splitting the data into test and training sets
x_train,x_test, y_train, y_test = train_test_split(X_sm,y_sm, test_size=0.3, random_state=12)

y_train = np.array(y_train)
y_test = np.array(y_test)
```

MODEL BUILDING :

```
##**Untuned Models**

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix,accuracy_score,f1_score
accuracy_score11= 0.0
accuracy_score1 = 0.0
```

```
u_train_acc = []
u_test_acc = []

u_sens = []
u_spec = []
u_prec = []
u_f1 = []

#OUTPUT FUNCTION
def output(model):

        y_pred1 = model.predict(x_train)

        #model testing
        y_pred2 = model.predict(x_test)

        print("\nClassification Report - Training")
        print(classification_report(y_train,y_pred1))

        print("\nClassification Report - Testing")
        print(classification_report(y_test,y_pred2))

        cm11 = confusion_matrix(y_train,y_pred1)
        print("\nConfusion Matrix - Training")
        print(cm11)

        cm1 = confusion_matrix(y_test,y_pred2)
        print("\nConfusion Matrix - Testing")
        print(cm1)

        cm = confusion_matrix(y_test, y_pred2)
```

```
        tn, fp, fn, tp = cm.ravel()
        print(tp,fp,fn,tn)

        sensitivity = tp / (tp + fn)
        u_sens.append(sensitivity)

        specificity = tn / (tn + fp)
        u_spec.append(specificity)

        precision = tp/(tp + fn)
        u_prec.append(precision)

        f1score = f1_score(y_test, y_pred2)
        u_f1.append(f1score)

        accuracy_score11 = accuracy_score(y_train,y_pred1)
        print('\nAccuracy Score - Training')
        print(accuracy_score11)
        u_train_acc.append(accuracy_score11)

        accuracy_score1 = accuracy_score(y_test,y_pred2)
        print('\nAccuracy Score - Testing')
        print(accuracy_score1)
        u_test_acc.append(accuracy_score1)


#XGBOOST
from xgboost import XGBClassifier
model1 = XGBClassifier()
model1.fit(x_train, y_train)
y_pred1 = model1.predict(x_test)
output(model1)
```

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
model2 = LogisticRegression(solver='lbfgs',max_iter=1000)
model2.fit(x_train, y_train.ravel())
y_pred2 = model2.predict(x_test)
output(model2)

#KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
model3 = KNeighborsClassifier()
model3.fit(x_train,y_train.ravel())
y_pred3 = model3.predict(x_test)
output(model3)

#SVM
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
model4 = SVC()
model4.fit(x_train,y_train.ravel())
y_pred4 = model4.predict(x_test)
output(model4)

#Decision Tree
from sklearn.tree import DecisionTreeClassifier
model5 = DecisionTreeClassifier()
model5.fit(x_train,y_train)
y_pred5 = model5.predict(x_test)
output(model5)

#Random Forest
from sklearn.ensemble import RandomForestClassifier
```

```
model6 = RandomForestClassifier()
model6.fit(x_train,y_train.ravel())
y_pred6 = model6.predict(x_test)
output(model6)

#Catboost
model7 = CatBoostClassifier()
model7.fit(x_train, y_train, verbose = False)
y_pred7 = model7.predict(x_test)
output(model7)
```

RESULTS :

```
#ROC Curves
from sklearn.metrics import roc_curve,auc

xgboost_fpr, xgboost_tpr, threshold = roc_curve(y_test,y_pred1)
logistic_fpr, logistic_tpr, threshold = roc_curve(y_test,y_pred2)
knn_fpr, knn_tpr, threshold = roc_curve(y_test,y_pred3)
svm_fpr, svm_tpr, threshold = roc_curve(y_test,y_pred4)
decisiontree_fpr, decisiontree_tpr, threshold = roc_curve(y_test,y_pred5)
randomforest_fpr, randomforest_tpr, threshold = roc_curve(y_test,y_pred6)
catboost_fpr, catboost_tpr, threshold = roc_curve(y_test, y_pred7)

#AUC values
auc_xgboost = auc(xgboost_fpr,xgboost_tpr)
auc_logistic = auc(logistic_fpr,logistic_tpr)
auc_knn = auc(knn_fpr,knn_tpr)
auc_svm = auc(svm_fpr,svm_tpr)
auc_decisiontree = auc(decisiontree_fpr,decisiontree_tpr)
auc_randomforest = auc(randomforest_fpr,randomforest_tpr)
auc_catboost = auc(catboost_fpr, catboost_tpr)
```

```
plt.figure(figsize=(5,5),dpi=100)

plt.plot(xgboost_fpr, xgboost_tpr, marker='.',label='xgboost(auc=%0.3f)'%auc_xgboost)

plt.plot(logistic_fpr, logistic_tpr, marker='.', label='logistic(auc=%0.3f)'%auc_logistic)

plt.plot(knn_fpr, knn_tpr, marker='.', label='knn(auc=%0.3f)'%auc_knn)

plt.plot(svm_fpr, svm_tpr, marker='.', label='svm(auc=%0.3f)'%auc_svm)

plt.plot(decisiontree_fpr, decisiontree_tpr, marker='.',
label='decisiontree(auc=%0.3f)'%auc_decisiontree)

plt.plot(randomforest_fpr, randomforest_tpr, marker='.',
label='randomforest(auc=%0.3f)'%auc_randomforest)

plt.plot(catboost_fpr, catboost_tpr, marker = '.', label = 'catboost(auc=%0.3f)'%auc_catboost)

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend(fontsize = 10)

plt.show()
```

#TRAINING AND TESTING ACCURACIES

```
model_list = ['XGBRF', 'Logistic Regression', 'KNearestNeighbours', 'SVM', 'Decision
Tree','RandomForest','CatBoostClassifier']

final_train_acc = u_train_acc

final_test_acc = u_test_acc


Models = ['XGBRF', 'Logistic', 'KNN', 'SVM','DecisionTree', 'RandomForest', 'CatBoost']

final_train_acc = [round(x,2) for x in final_train_acc]

final_test_acc = [round(x,2) for x in final_test_acc]

u_prec = [round(x,2) for x in u_prec]

u_sens = [round(x,2) for x in u_sens]

u_spec = [round(x,2) for x in u_spec]

u_f1 = [round(x,2) for x in u_f1]

u_auc = [auc_xgboost, auc_logistic, auc_knn, auc_svm, auc_decisiontree, auc_randomforest,
auc_catboost]

u_auc = [round(x,2) for x in u_auc]


u_cmp_acc = pd.DataFrame({'Models' : Models,'Training' : final_train_acc, 'Testing' : final_test_acc})
```

```
u_cmp_acc = u_cmp_acc.sort_values(by=['Testing'], ascending = False)
display(u_cmp_acc)
```

ALL METRICS COMPARISON

```
metr_df = pd.DataFrame({'Models' : Models, 'Accuracy' : final_test_acc, 'Precision' : u_prec,
'Sensitivity' : u_sens, 'Specificity' : u_spec, 'F1-Score' : u_f1,
'AUC' : u_auc})
metr_df.sort_values(by=['Accuracy','Sensitivity'], inplace = True, ascending = False)
metr_df
```

HYPERPARAMETER TUNING :

```
random_state = 0
classifier = [XGBClassifier(random_state = random_state),
LogisticRegression(random_state = random_state,solver='lbfgs',max_iter=1000),
KNeighborsClassifier(),
SVC(random_state = random_state),
DecisionTreeClassifier(random_state = random_state),
RandomForestClassifier(random_state = random_state),
CatBoostClassifier(random_state = random_state)]

# Decision Tree
dtg = {"min_samples_split" : range(10,500,20),
"max_depth": range(1,20,2)}

# SVM
svmg = {"kernel" : ["rbf"],
"gamma": [0.001, 0.01, 0.1, 1],
"C": [1,10,50,100,200,300,1000]}

# Random Forest
rfg = {"max_features": ['sqrt', 'log2'],
"n_estimators":[300,500],
"criterion":["gini"],
```

```
'max_depth' : [4,5,6,7,8,9,10,12],}

# Logistic Regression
lrg = {"C":np.logspace(-3,3,7),
"penalty": ["l1","l2"]}

# KNN
knng = {"n_neighbors": np.linspace(1,19,10, dtype = int).tolist(),
"weights": ["uniform","distance"],
"metric":["euclidean","manhattan"]}

#Catboost
catg =  {'max_depth': [3,4,5],'n_estimators':[100, 200, 300]}

#XGBRF
xgbrfg = {
'min_child_weight': [1, 5, 10],
'gamma': [0.5, 1, 1.5, 2, 5],
'subsample': [0.6, 0.8, 1.0],
'colsample_bytree': [0.6, 0.8, 1.0],
'max_depth': [3, 4, 5]
}

classifier_param = [xgbrfg, lrg, knng, svmg, dtg, rfg, catg]
```

GRID SEARCH CV

```
cv_result = []
best_estimators = []
best_params = []
for i in range(len(classifier)):
clf = GridSearchCV(classifier[i], param_grid=classifier_param[i], cv = StratifiedKFold(n_splits = 10),
n_jobs = -1, verbose = 1)
```

```
if(classifier_param[i] in [rfg, svmg, lrg, knng]):
clf.fit(x_train,y_train.ravel())
else:
clf.fit(x_train,y_train)
cv_result.append(round(clf.best_score_*100,2))
best_estimators.append(clf.best_estimator_)
best_params.append(clf.best_params_)
print(cv_result[i])
```

MODEL BUILDING :

```
train_acc = []
test_acc = []
sens = []
spec = []
prec = []
f1 = []

#OUTPUT FUNCTION
def output(model):
        y_pred1 = model.predict(x_train)
        y_pred2 = model.predict(x_test)

        print("\nClassification Report - Training")
        print(classification_report(y_train,y_pred1))

        print("\nClassification Report - Testing")
        print(classification_report(y_test,y_pred2))

        cm11 = confusion_matrix(y_train,y_pred1)
        print("\nConfusion Matrix - Training")
        print(cm11)
```

```
cm1 = confusion_matrix(y_test,y_pred2)
print("\nConfusion Matrix - Testing")
print(cm1)

cm = confusion_matrix(y_test, y_pred2)
tn, fp, fn, tp = cm.ravel()
print(tp,fp,fn,tn)

sensitivity = tp / (tp + fn)
sens.append(sensitivity)

specificity = tn / (tn + fp)
spec.append(specificity)

precision = tp/(tp + fn)
prec.append(precision)

f1score = f1_score(y_test, y_pred2)
f1.append(f1score)

accuracy_score11 = accuracy_score(y_train,y_pred1)
print('\nAccuracy Score - Training')
print(accuracy_score11)
train_acc.append(accuracy_score11)

accuracy_score1 = accuracy_score(y_test,y_pred2)
print('\nAccuracy Score - Testing')
print(accuracy_score1)
test_acc.append(accuracy_score1)
```

```
#XGBOOST
from xgboost import XGBClassifier
```

```
be = best_params[0]
model1 = XGBClassifier(**be)
model1.fit(x_train, y_train)
y_pred1 = model1.predict(x_test)
output(model1)

#Logistic Regression
from sklearn.linear_model import LogisticRegression
be = best_params[1]
model2 = LogisticRegression(**be,solver='lbfgs',max_iter=1000)
model2.fit(x_train, y_train.ravel())
y_pred2 = model2.predict(x_test)
output(model2)

#KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
be = best_params[2]
model3 = KNeighborsClassifier(**be)
model3.fit(x_train,y_train.ravel())
y_pred3 = model3.predict(x_test)
output(model3)

#SVM
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
be = best_params[3]
model4 = SVC(**be)
model4.fit(x_train,y_train.ravel())
y_pred4 = model4.predict(x_test)
output(model4)

#Decision Tree
```

```
from sklearn.tree import DecisionTreeClassifier
be = best_params[4]
model5 = DecisionTreeClassifier(**be)
model5.fit(x_train,y_train)
y_pred5 = model5.predict(x_test)
output(model5)

#Random Forest
from sklearn.ensemble import RandomForestClassifier
be = best_params[5]
model6 = RandomForestClassifier(**be)
model6.fit(x_train,y_train.ravel())
y_pred6 = model6.predict(x_test)
output(model6)

#Catboost
be = best_params[6]
model7 = CatBoostClassifier(**be)
model7.fit(x_train, y_train, verbose = False)
y_pred7 = model7.predict(x_test)
output(model7)
```

RESULTS :

```
#ROC Curves
from sklearn.metrics import roc_curve,auc

xgboost_fpr, xgboost_tpr, threshold = roc_curve(y_test,y_pred1)
logistic_fpr, logistic_tpr, threshold = roc_curve(y_test,y_pred2)
knn_fpr, knn_tpr, threshold = roc_curve(y_test,y_pred3)
svm_fpr, svm_tpr, threshold = roc_curve(y_test,y_pred4)
decisiontree_fpr, decisiontree_tpr, threshold = roc_curve(y_test,y_pred5)
randomforest_fpr, randomforest_tpr, threshold = roc_curve(y_test,y_pred6)
```

```
catboost_fpr, catboost_tpr, threshold = roc_curve(y_test, y_pred7)


#AUC values
auc_xgboost = auc(xgboost_fpr,xgboost_tpr)
auc_logistic = auc(logistic_fpr,logistic_tpr)
auc_knn = auc(knn_fpr,knn_tpr)
auc_svm = auc(svm_fpr,svm_tpr)
auc_decisiontree = auc(decisiontree_fpr,decisiontree_tpr)
auc_randomforest = auc(randomforest_fpr,randomforest_tpr)
auc_catboost = auc(catboost_fpr, catboost_tpr)


plt.figure(figsize=(5,5),dpi=100)
plt.plot(xgboost_fpr, xgboost_tpr, marker='.',label='xgboost(auc=%0.3f)'%auc_xgboost)
plt.plot(logistic_fpr, logistic_tpr, marker='.', label='logistic(auc=%0.3f)'%auc_logistic)
plt.plot(knn_fpr, knn_tpr, marker='.', label='knn(auc=%0.3f)'%auc_knn)
plt.plot(svm_fpr, svm_tpr, marker='.', label='svm(auc=%0.3f)'%auc_svm)
plt.plot(decisiontree_fpr, decisiontree_tpr, marker='.',
label='decisiontree(auc=%0.3f)'%auc_decisiontree)
plt.plot(randomforest_fpr, randomforest_tpr, marker='.',
label='randomforest(auc=%0.3f)'%auc_randomforest)
plt.plot(catboost_fpr, catboost_tpr, marker = '.', label = 'catboost(auc=%0.3f)'%auc_catboost)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(fontsize = 10)
plt.show()


#TRAINING-TESTING
model_list = ['XGBRF', 'Logistic Regression', 'KNearestNeighbours', 'SVM', 'Decision
Tree','RandomForest','CatBoostClassifier']
final_train_acc = train_acc
final_test_acc = test_acc


Models = ['XGBRF', 'Logistic', 'KNN', 'SVM','DecisionTree', 'RandomForest', 'CatBoost']
final_train_acc = [round(x,2) for x in final_train_acc]
```

```
final_test_acc = [round(x,2) for x in final_test_acc]

prec = [round(x,2) for x in prec]

sens = [round(x,2) for x in sens]

spec = [round(x,2) for x in spec]

f1 = [round(x,2) for x in f1]

auc = [auc_xgboost, auc_logistic, auc_knn, auc_svm, auc_decisiontree, auc_randomforest,
auc_catboost]

auc = [round(x,2) for x in auc]

cmp_acc = pd.DataFrame({'Models' : Models,'Training' : final_train_acc, 'Testing' : final_test_acc})

cmp_acc = cmp_acc.sort_values(by=['Testing'], ascending = False)

display(cmp_acc)
```

ALL METRICS COMPARISON

```
metr_df = pd.DataFrame({'Models' : Models, 'Accuracy' : final_test_acc, 'Precision' : prec, 'Sensitivity'
: sens, 'Specificity' : spec, 'F1-Score' : f1,

'AUC' : auc})

metr_df.sort_values(by=['Accuracy','Sensitivity'], inplace = True, ascending = False)

metr_df
```

*"""*Best Model is Random Forest*"""*

PLOTTING ACCURACIES :

```
accuracies = pd.DataFrame({'Models': Models, 'Accuracies': final_test_acc})

fig, ax1 = plt.subplots(figsize=(10, 5))

g = sns.barplot(x=Models,y = final_test_acc, ax=ax1, palette='Paired')

g.set_title('COMPARISON BY ACCURACIES')

ticks = np.linspace(0,1.0,11)

g.set_yticks(ticks)

for i in g.containers:

g.bar_label(i,)

sns.despine(fig)
```

Confusion Matrix and AUC of Random Forest

```
plt.figure(figsize=(5,5),dpi=100)
plt.plot(randomforest_fpr, randomforest_tpr, marker='.', label='rfc(auc=%0.3f)'%auc_randomforest)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.fill_between(randomforest_fpr, randomforest_tpr, facecolor='darkgray', alpha=0.7)
plt.legend(fontsize = 10)
plt.show()

cm = confusion_matrix(y_test, y_pred6, labels=[1,0])
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True)
plt.title("RF Confusion Matrix")
plt.xticks(range(2), ["PCOS","No PCOS"], fontsize=16)
plt.yticks(range(2), ["PCOS","No PCOS"], fontsize=16)
plt.show()
```

PREDICTION :

SAVING AND RETRIEVING MODEL

```
import pickle
filename = 'rf_pcos_model.sav'
pickle.dump(model6, open(filename, 'wb'))
final_model = pickle.load(open(filename, 'rb'))
```

PREDICTION 1 :

```
follicle_no_right = 15
hair_growth = 0
follicle_no_left = 13
skin_darkening = 0
cycle = 2 #Regular
weight_gain = 0
fast_food = 1
```

```
pimples = 1
cycle_length = 5
amh = 6.63
regular_exercise = 0
age = 33
hip_inches = 40
average_follicle_size_right = 20
average_follicle_size_left = 18

test_list = [[follicle_no_right, hair_growth, follicle_no_left, skin_darkening, cycle,
weight_gain, fast_food, pimples, cycle_length, amh, marriage_status_years,
regular_exercise, age, hip_inches, average_follicle_size_right]]

res = final_model.predict(test_list)

if res:
print('Patient has a risk of PCOS')
else:
print('Patient does not have a risk of PCOS')
```

PREDICTION 2 :

```
follicle_no_right = 4
hair_growth = 0
follicle_no_left = 3
skin_darkening = 0
cycle = 2 #Regular
weight_gain = 0
fast_food = 0
pimples = 0
cycle_length = 5
amh = 2.26
marriage_status_years = 2
```

```
regular_exercise = 0
age = 25
hip_inches = 37
average_follicle_size_right = 14

test_list = [[follicle_no_right, hair_growth, follicle_no_left, skin_darkening, cycle,
weight_gain, fast_food, pimples, cycle_length, amh, marriage_status_years,
regular_exercise, age, hip_inches, average_follicle_size_right]]

res = final_model.predict(test_list)

if res:
print('Patient has a risk of PCOS')
else:
print('Patient does not have a risk of PCOS')
```

6. COMPARATIVE ANALYSIS :

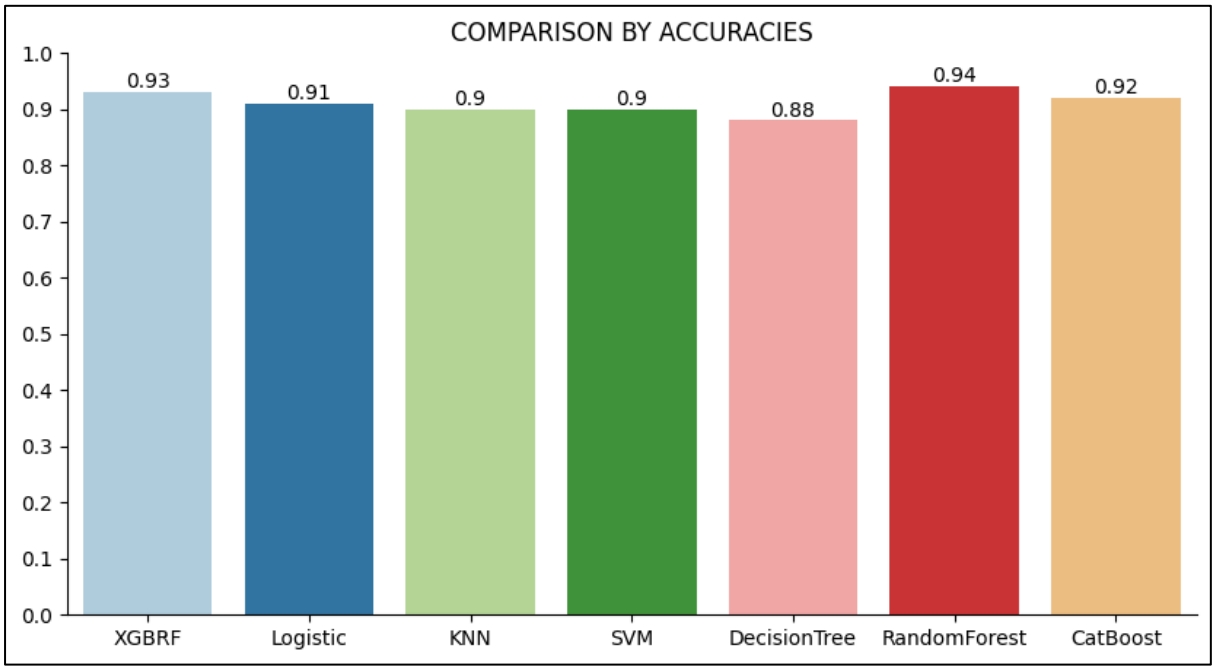

Figure 4. Comparison of models by accuracies

Table 1. Models and their metric values sorted by first highest **Accuracy** and **Sensitivity**

Models	Accuracy	Precision	Sensitivity	Specificity	F1-Score	AUC
RandomForest	0.94	0.93	0.93	0.94	0.94	0.94
XGBRF	0.93	0.91	0.91	0.94	0.93	0.93
CatBoost	0.92	0.9	0.9	0.94	0.92	0.92
Logistic	0.91	0.9	0.9	0.92	0.91	0.91
KNN	0.9	0.89	0.89	0.92	0.9	0.9
SVM	0.9	0.88	0.88	0.92	0.9	0.9
DecisionTree	0.88	0.89	0.89	0.86	0.88	0.87

Random Forest performs better than all the other models for our system with an accuracy of 94% and a sensitivity of 93%

Accuracy and Sensitivity are used as important metrics for this study. This is because, we hope to achieve highest possible accuracy for our prediction model. Higher the accuracy, greater the chances of accurate predictions. Moreover, in medical inspections, it is intended to miss as few positive cases as possible and higher the sensitivity values, lower the chances of positive cases missed.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad Sensitivity = \frac{TP}{TP + FN}$$

Where, TP, TN, FP, FN stand for True Positive, True Negative, False Positive, and False Negative respectively.

Random Forest Results :

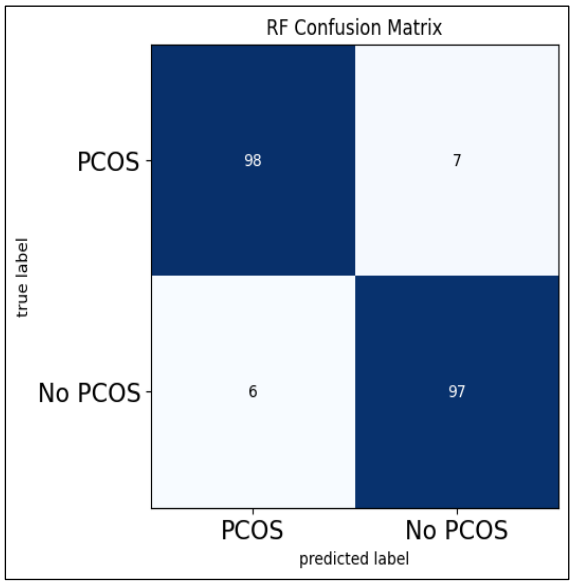

Figure 5. Confusion Matrix

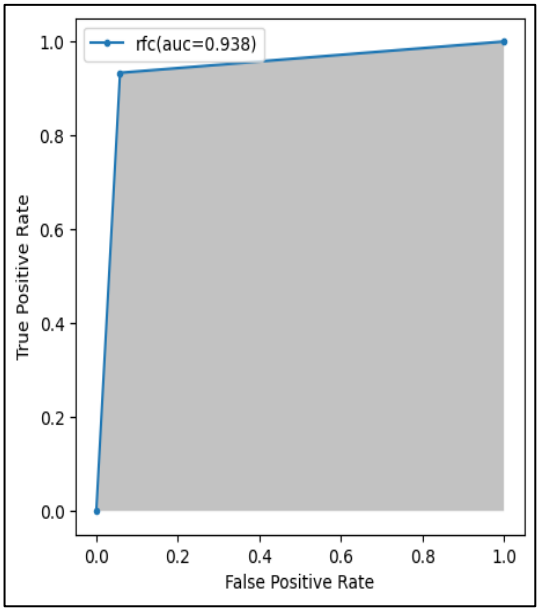

Figure 6. ROC-AUC

The Area Under the Curve (Figure 6) for RF is 0.938 which is greatest amongst all the models Confusion matrix (Figure 5) gives the following inferences:

1) **True Positive:** RFpredicted *98* PCOS patients who actually had it.
2) **False Positive:** RF predicted *6* patients to have PCOS, but they did not.
3) **False Negative:** *7* patients were predicted not to have PCOS by RF, but they actually had it.
4) **True Negative:** RF predicted *97* patients to be unaffected by PCOS who were in fact unaffected.

Feature Importance :

For a given model, feature importance refers to the method of calculating a score that represents the importance of each input feature. If a feature has a high score, it has a greater impact on the model that is being used to predict a certain variable.

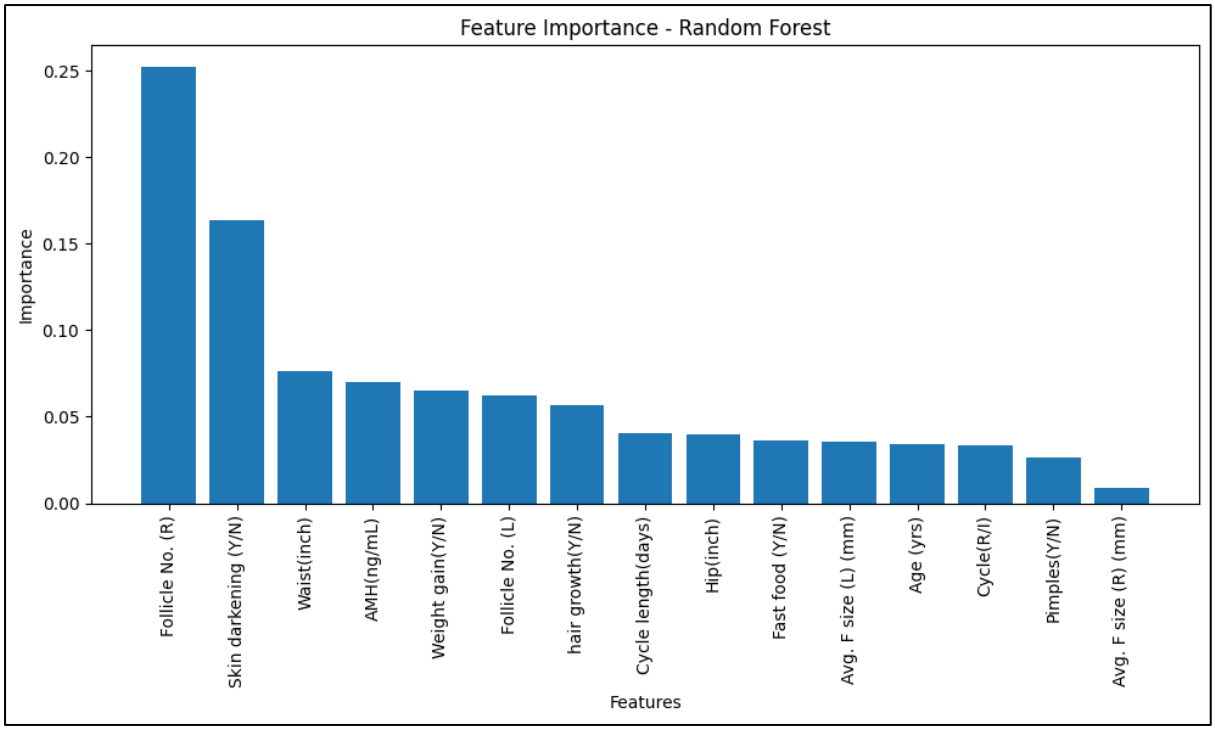

Figure 7. Feature Importance of variables

Plotting Permutation importance graph (Figure 7), we get the top 4 features which Random Forest found most influential in prediction of PCOS are

1. Follicle No. (R)
2. Skin darkening (Y/N)
3. Waist(inch)
4. AMH(ng/mL)

From the tuned Random Forest we can still get false results in that case to avoid misleading results, we can take the help of feature importance from which we will see the 4 most important features and compare them with the ideal parameters and if they are violating it, we can tell the patient that he/she is at the risk of having PCOS.

7. CONCLUSION WITH FUTURE ENHANCEMENT :

Today, millions of patients are affected by PCOS but the tragedy is that most of them are unaware of it. Patients suffer significant health consequences as a result. Our main goal was to develop a system that would allow a patient to determine their risk of PCOS by entering a few basic health information. In this study, we explored various medical parameters, how they are related to each other and how they vary for a person with PCOS. Then we explored various machine learning models. Tuned their hyperparameters to get accurate results. Out of all, Random Forest performed better than others with an accuracy of 0.94 and Sensitivity of 0.93. Further, Number of follicles in the ovaries, Age and darkening of skin came out to be the most

important features. The dataset used for investigation had a smaller number of samples and it was concentrated on a single geographical location. For improving the diagnosis model large number of datasets have to be collected and awareness about PCOS should be increased. Many more machine learning models can be tested on the dataset. Hyperparameter tuning can be performed extensively to get more accurate results.

In addition to the findings of this project, there are several potential future enhancements that can be considered to further improve the diagnosis of Polycystic Ovary Syndrome (PCOS) using machine learning algorithms. Firstly, integrating more advanced feature engineering techniques could enhance the performance of the models. Techniques such as feature scaling, dimensionality reduction, and feature selection algorithms can help identify the most informative features for PCOS diagnosis, leading to more accurate predictions. Secondly, exploring ensemble methods could be beneficial. Ensemble techniques, such as stacking, bagging, or boosting, involve combining multiple models to make predictions. This can help improve the overall predictive accuracy and robustness of the models. Incorporating domain-specific knowledge by collaborating with medical experts is another avenue for future enhancement. Medical professionals can provide valuable insights and expertise, which can be integrated into the model development process. This ensures that the models align with the medical field standards and are clinically relevant. Developing a user-friendly interface or application for real-time prediction is another promising future enhancement. This would allow healthcare professionals or individuals to input their medical parameters and obtain immediate PCOS risk predictions. Providing such a tool can facilitate early detection and timely intervention, leading to improved patient outcomes. Lastly, external validation using datasets from different demographics or healthcare settings would further validate the performance and generalizability of the models. This would provide additional evidence of the models' effectiveness and reliability in various real-world scenarios. By considering these future enhancements, we can continue to refine and advance the use of machine learning algorithms in PCOS diagnosis. This will ultimately contribute to improved healthcare decision-making, early intervention, and better management of PCOS for affected individuals.