

# Early Stage Diabetes Detection

## About the data

This dataset contains the sign and symptom data of newly diabetic or would be diabetic patient. This has been collected using direct questionnaires from the patients of Sylhet Diabetes Hospital in Sylhet, Bangladesh and approved by a doctor.

1. **Age:** Age in years ranging from (20years to 65 years)
2. **Gender:** Male / Female
3. **Polyuria:** Yes / No
4. **Polydipsia:** Yes/ No
5. **Sudden weight loss:** Yes/ No
6. **Weakness:** Yes/ No
7. **Polyphagia:** Yes/ No
8. **Genital Thrush:** Yes/ No
9. **Visual blurring:** Yes/ No
10. **Itching:** Yes/ No
11. **Irritability:** Yes/No
12. **Delayed healing:** Yes/ No
13. **Partial Paresis:** Yes/ No
14. **Muscle stiffness:** yes/ No
15. **Alopecia:** Yes/ No
16. **Obesity:** Yes/ No

**Class:** Positive / Negative

## Importing Neccessary Packages

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, precision_score, confusion_matrix,
recall_score, roc_auc_score
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import Image
```

## Data Collection and Exploring the data

In [ ]:

```
df = pd.read_csv('../input/predicting-the-diabetes-at-an-early-stage/diabetes_data_upload.csv')
```

In [ ]:

```
df.head()
```

```
In [ ]:
```

```
# Checking Missing Values
```

```
df.isnull().sum()
```

```
# out data is clean and dont contain any missing values
```

```
In [ ]:
```

```
df.describe() #for only numerical values
```

```
In [ ]:
```

```
df.info()
```

```
In [ ]:
```

```
import seaborn as sns
```

```
sns.countplot(df['class'], data=df)
```

```
In [ ]:
```

```
#Distribution of the target variable
```

```
df['class'].value_counts()
```

```
In [ ]:
```

```
# plotting to create pie chart and bar plot as subplots
```

```
plt.figure(figsize=(14,7))
```

```
plt.subplot(121)
```

```
df["class"].value_counts().plot.pie(autopct = "%1.0f%%", colors = sns.color_palette("prism",7), startangle = 60, labels=["Positive", "Negative"],  
wedgeprops={"linewidth":2, "edgecolor":"k"}, explode=[.1,0], shadow =True)  
plt.title("Distribution of Target variable")
```

```
plt.subplot(122)
```

```
ax = df["class"].value_counts().plot(kind="barh")
```

```
for i,j in enumerate(df["class"].value_counts().values):  
    ax.text(.7,i,j, weight = "bold", fontsize=20)
```

```
plt.title("Count of Target variable")
```

```
plt.show()
```

```
In [ ]:
```

```
# Distribution of gender
```

```
import seaborn as sns
```

```
sns.countplot(df['Gender'], hue=df['class'], data=df)
```

## Conclusions:

**From our observations we came to know that**

***Female are more prone for getting diabetes.***

```
In [ ]:
```

```
plot_criteria= ['Gender', 'class']
```

```
cm = sns.light_palette("red", as_cmap=True)
```

```
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100,  
2)).style.background_gradient(cmap = cm)
```

## Distribution of Polyuria

# POLYURIA

Polyuria is defined as a urine volume in excess of 3 L/day.

## Causes

- Excess fluid intake
- Hyperglycaemia
- Cranial diabetes insipidus
- Nephrogenic diabetes insipidus
- Lithium
- Diuretics
- Interstitial nephritis
- Hypokalaemia
- Hypercalcaemia



**Polyuria is defined as the frequent passage of large volumes of urine – more than 3 litres a day compared to the normal daily urine output in adults of about 1 to 2 litres.**

## Causes:

**The most common cause of polyuria in both adults and children is uncontrolled diabetes mellitus, which causes osmotic diuresis, when glucose levels are so high that glucose is excreted in the urine. Water follows the glucose concentration passively, leading to abnormally high urine output.**

**In the absence of diabetes mellitus, the most common causes are decreased secretion of aldosterone due to adrenal cortical tumor, primary polydipsia (excessive fluid drinking),**

In [ ]:

```
plot_criteria= ['Polyuria', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap = cm)
```

## Distribution of Polydipsia



**Polydipsia is the term given to excessive thirst and is one of the initial symptoms of diabetes. It is also usually accompanied by temporary or prolonged dryness of the mouth.**

**However, if you feel thirsty all the time or your thirst is stronger than usual and continues even after you drink, it can be a sign that not all is well inside your body.**

**Excessive thirst can be caused by high blood sugar (hyperglycemia), and is also one of the 'Big 3' signs of diabetes mellitus i.e**

1. Polyuria

## 2. Polydipsia

3. **Polyphagi** ### Generally, increased thirst (polydipsia) and an increased need to urinate (polyuria) will often come as a pair.

In [ ]:

```
plot_criteria= ['Polydipsia', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap = cm)
```

In [ ]:

```
# Distribution of sudden weight loss
plot_criteria= ['sudden weight loss', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap = cm)
```

In [ ]:

```
# Distribution of weakness
plot_criteria= ['weakness', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap = cm)
```

## Distribution of Polyphagia

□

**Polyphagia**, also known as **hyperphagia**, is the medical term for excessive or extreme hunger.

**It's different than having an increased appetite after exercise or other physical activity.**

**While your hunger level will return to normal after eating in those cases, polyphagia won't go away if you eat more food.**

In [ ]:

```
plot_criteria= ['Polyphagia', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap = cm)
```

## Distribution of genital thrush

**Thrush (or candidiasis)** is a common condition caused by a type of yeast called **Candida**. It mainly affects the vagina, though may affect the penis too, and can be irritating and painful.

**Many types of yeast and bacteria naturally live in the vagina and rarely cause problems. Candida is a yeast-like fungus that lives in warm, moist places such as the mouth, bowel, vagina and the foreskin of the penis. Thrush is caused when there is an overgrowth of Candida.**

In [ ]:

```
plot_criteria= ['Genital thrush', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap = cm)
```

```
,2)).style.background_gradient(cmap = cm)
```

In [ ]:

```
plot_criteria= ['visual blurring', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100
,2)).style.background_gradient(cmap = cm)
```

In [ ]:

```
# Distribution of Visual Blurring
plot_criteria= ['visual blurring', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100
,2)).style.background_gradient(cmap = cm)
```

In [ ]:

```
# Itching
plot_criteria= ['Itching', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100
,2)).style.background_gradient(cmap = cm)
```

In [ ]:

```
#Irritability
plot_criteria= ['Irritability', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100
,2)).style.background_gradient(cmap = cm)
```

In [ ]:

```
# Delayed Healing
plot_criteria= ['delayed healing', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100
,2)).style.background_gradient(cmap = cm)
```

## Partial Paresis

**Paresis involves the weakening of a muscle or group of muscles. It may also be referred to as partial or mild paralysis. Unlike paralysis, people with paresis can still move their muscles. These movements are just weaker than normal.**

In [ ]:

```
plot_criteria= ['partial paresis', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100
,2)).style.background_gradient(cmap = cm)
```

In [ ]:

```
# Muscle Stiffness
plot_criteria= ['muscle stiffness', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100
,2)).style.background_gradient(cmap = cm)
```

## Alopecia





**Sudden hair loss that starts with one or more circular bald patches that may overlap. Alopecia areata occurs when the immune system attacks hair follicles and may be brought on by severe stress. The main symptom is hair loss.**

In [ ]:

```
plot_criteria= ['Alopecia', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap = cm)
```

In [ ]:

```
# Obesity
plot_criteria= ['Obesity', 'class']
cm = sns.light_palette("red", as_cmap=True)
(round(pd.crosstab(df[plot_criteria[0]], df[plot_criteria[1]], normalize='columns') * 100, 2)).style.background_gradient(cmap = cm)
```

## Data pre-processing

In [ ]:

```
df['class'] = df['class'].apply(lambda x: 0 if x=='Negative' else 1)
```

In [ ]:

```
X= df.drop(['class'],axis=1)
y=df['class']
```

In [ ]:

```
objList = X.select_dtypes(include = "object").columns
print (objList)
```

In [ ]:

```
#Label Encoding for object to numeric conversion
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for feat in objList:
    X[feat] = le.fit_transform(X[feat].astype(str))

print (X.info())
```

In [ ]:

```
X.head()
```

In [ ]:

```
X.corrwith(y)
```

In [ ]:

```
#Correlation with Response Variable class
```

```
X.corrwith(y).plot.bar(
    figsize = (16, 6), title = "Correlation with Diabetes", fontsize = 15,
    rot = 90, grid = True)
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15, stratify=y, r
andom_state = 1999)
```

In [ ]:

```
## checking distribution of target variable in train test split
print('Distribution of target variable in training set')
print(y_train.value_counts())

print('Distribution of target variable in test set')
print(y_test.value_counts())
```

In [ ]:

In [ ]:

```
# Data Normalization
minmax = MinMaxScaler()

X_train[['Age']] = minmax.fit_transform(X_train[['Age']])
X_test[['Age']] = minmax.transform(X_test[['Age']])
```

In [ ]:

```
X_train.head()
```

# Model Building

## 1. Logistic Regression (Base model)

In [ ]:

```
logi = LogisticRegression(random_state = 0, penalty = 'l2')
logi.fit(X_train, y_train)
```

In [ ]:

```
from sklearn import model_selection
kfold = model_selection.KFold(n_splits=10)
scoring = 'accuracy'

acc_logi = cross_val_score(estimator = logi, X = X_train, y = y_train, cv = kfold, scorin
g=scoring)
acc_logi.mean()
```

In [ ]:

```
y_predict_logi = logi.predict(X_test)
acc= accuracy_score(y_test, y_predict_logi)
roc=roc_auc_score(y_test, y_predict_logi)
prec = precision_score(y_test, y_predict_logi)
rec = recall_score(y_test, y_predict_logi)
f1 = f1_score(y_test, y_predict_logi)

results = pd.DataFrame([['Logistic Regression', acc, acc_logi.mean(), prec, rec, f1, roc]],
    columns = ['Model', 'Accuracy', 'Cross Val Accuracy', 'Precision', 'Recall', 'F1 Score', 'ROC'])
results
```

```
In [ ]:
```

```
cm_logi = confusion_matrix(y_test, y_predict_logi)
plt.title('Confusion matrix of the Logistic classifier')
sns.heatmap(cm_logi, annot=True, fmt="d")
plt.show()
```

```
In [ ]:
```

```
TP = cm_logi[1,1] # true positive
TN = cm_logi[0,0] # true negatives
FP = cm_logi[0,1] # false positives
FN = cm_logi[1,0] # false negatives
# Let us calculate specificity
TN / float(TN+FP)
```

```
In [ ]:
```

```
feature_importance = abs(logi.coef_[0])
feature_importance = 100.0 * (feature_importance / feature_importance.max())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .3

featfig = plt.figure(figsize=(12,8))
featax = featfig.add_subplot(1, 1, 1)
featax.barh(pos, feature_importance[sorted_idx], align='center')
featax.set_yticks(pos)
featax.set_yticklabels(np.array(X_train.columns)[sorted_idx], fontsize=8)
featax.set_xlabel('Relative Feature Importance')

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

## 2. Random forest (Before Tuning)

```
In [ ]:
```

```
rf = RandomForestClassifier(criterion='gini',n_estimators=100)
rf.fit(X_train,y_train)
```

```
In [ ]:
```

```
kfold = model_selection.KFold(n_splits=10)
scoring = 'accuracy'

acc_rf = cross_val_score(estimator = rf, X = X_train, y = y_train, cv = kfold,scoring=scoring)
acc_rf.mean()
```

```
In [ ]:
```

```
y_predict_r = rf.predict(X_test)
roc=roc_auc_score(y_test, y_predict_r)
acc = accuracy_score(y_test, y_predict_r)
prec = precision_score(y_test, y_predict_r)
rec = recall_score(y_test, y_predict_r)
f1 = f1_score(y_test, y_predict_r)

model_results = pd.DataFrame([[ 'Random Forest (Untuned)',acc, acc_rf.mean(),prec,rec, f1, roc]],
                               columns = ['Model', 'Accuracy','Cross Val Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results = results.append(model_results, ignore_index = True)
results
```

```
In [ ]:
```

```
cm_rf = confusion_matrix(y_test, y_predict_r)
```



```
plt.title('Confusion matrix of the Random Forest classifier')
sns.heatmap(cm_rf, annot=True, fmt="d")
plt.show()
```

In [ ]:

```
TP = cm_rf[1,1] # true positive
TN = cm_rf[0,0] # true negatives
FP = cm_rf[0,1] # false positives
FN = cm_rf[1,0] # false negatives
# Let us calculate specificity
TN / float(TN+FP)
```

In [ ]:

```
feat_importances = pd.Series(rf.feature_importances_, index=X_train.columns)
feat_importances.nlargest(20).plot(kind='barh')
```

In [ ]:

```
# Feature Selection - Top 10 Features
X.columns
```

In [ ]:

```
X.head()
```

In [ ]:

```
# filtering unwanted features
X_fs = X[['Polyuria', 'Polydipsia', 'Age', 'Gender', 'partial paresis', 'sudden weight loss', 'Irritability', 'delayed healing', 'Alopecia', 'Itching']]
```

In [ ]:

```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X_fs, y, test_size = 0.15, stratify=y, random_state = 1999)
```

In [ ]:

```
X_train.head()
```

In [ ]:

```
# Data Normalization
minmax = MinMaxScaler()
X_train[['Age']] = minmax.fit_transform(X_train[['Age']])
X_test[['Age']] = minmax.transform(X_test[['Age']])
```

## Model building

### Regression - Post Feature Selection

In [ ]:

```
logi = LogisticRegression(random_state = 0, penalty = 'l2')
logi.fit(X_train, y_train)
```

In [ ]:

```
# Cross validation
kfold = model_selection.KFold(n_splits=10)
scoring = 'accuracy'

acc_logi = cross_val_score(estimator = logi, X = X_train, y = y_train, cv = kfold, scoring=scoring)
```

```
acc_logi.mean()
```

```
In [ ]:
```

```
acc= accuracy_score(y_test, y_predict_logi)
roc=roc_auc_score(y_test, y_predict_logi)
prec = precision_score(y_test, y_predict_logi)
rec = recall_score(y_test, y_predict_logi)
f1 = f1_score(y_test, y_predict_logi)

model_results = pd.DataFrame([['Logistic Regression-Post FS',acc, acc_logi.mean(),prec,rec,
f1,roc]],
                             columns = ['Model', 'Accuracy','Cross Val Accuracy', 'Precision', 'Recall',
'F1 Score','ROC'])
results = results.append(model_results, ignore_index = True)
results
```

## Random Forest - Post Feature selection

```
In [ ]:
```

```
rf = RandomForestClassifier(criterion='gini',n_estimators=100)
rf.fit(X_train,y_train)
```

```
In [ ]:
```

```
# Cross Validation
kfold = model_selection.KFold(n_splits=10)
scoring = 'accuracy'

acc_rf = cross_val_score(estimator = rf, X = X_train, y = y_train, cv = kfold,scoring=scoring)
acc_rf.mean()
```

## Model Evaluation

```
In [ ]:
```

```
y_predict_r = rf.predict(X_test)
roc=roc_auc_score(y_test, y_predict_r)
acc = accuracy_score(y_test, y_predict_r)
prec = precision_score(y_test, y_predict_r)
rec = recall_score(y_test, y_predict_r)
f1 = f1_score(y_test, y_predict_r)

model_results = pd.DataFrame([['Random Forest- Post FS',acc, acc_rf.mean(),prec,rec, f1,roc]],
                             columns = ['Model', 'Accuracy','Cross Val Accuracy', 'Precision', 'Recall',
'F1 Score','ROC'])
results = results.append(model_results, ignore_index = True)
results
```

```
In [ ]:
```

```
cm_rf = confusion_matrix(y_test, y_predict_r)
plt.title('Confusion matrix of the Random Forest classifier')
sns.heatmap(cm_rf,annot=True,fmt="d")
plt.show()
```

```
In [ ]:
```

```
TP = cm_rf[1,1] # true positive
TN = cm_rf[0,0] # true negatives
FP = cm_rf[0,1] # false positives
FN = cm_rf[1,0] # false negatives
# Let us calculate specificity
TN / float(TN+FP)
```

In [ ]:

```
feat_importances = pd.Series(rf.feature_importances_, index=X_train.columns)
feat_importances.nlargest(20).plot(kind='barh')
```

In [ ]:

```
# Plotting ROC
from sklearn import metrics
import matplotlib.pyplot as plt

plt.figure()

# Add the models to the list that you want to view on the ROC plot
models = [
    {
        'label': 'Logistic Regression',
        'model': LogisticRegression(random_state = 0, penalty = 'l2'),
    },

    {
        'label': 'Random Forest',
        'model': RandomForestClassifier(n_estimators = 100, criterion='gini', random_state =
47),
    },
]

# Below for loop iterates through your models list
for m in models:
    model = m['model'] # select the model
    model.fit(X_train, y_train) # train the model
    y_pred=model.predict(X_test) # predict the test data
# Compute False positive rate, and True positive rate
fpr, tpr, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_test)[: ,1])
# Calculate Area under the curve to display on the plot
auc = metrics.roc_auc_score(y_test,model.predict(X_test))
# Now, plot the computed values
plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (m['label'], auc))
# Custom settings for the plot
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

In [ ]:

```
import pickle
filename = 'random_forest.pkl'
pickle.dump(rf, open(filename, 'wb'))
```

In [ ]:

```
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
result
```

In [ ]:

```
X_train.head()
```

In [ ]:

```
y_train
```

In [ ]:

```
X_train.columns
```

```
In [ ]:
```

```
with open(filename, 'rb') as file:  
    pickle_model = pickle.load(file)
```

```
In [ ]:
```

```
result=pickle_model.predict([[1,0,44,1,0,1,0,1,1,1]])  
print(result)
```

```
In [ ]:
```