BOOSTING ACCURACY AND ROBUSTNESS IN DIABETES PREDICTION : LEVERAGING ENSEMBLE METHODS AND DEEP LEARNING ARCHITECTURES

TEAM MEMBER

720421104053 : VISMAYA B

CMS COLLEGE OF ENGINEERING AND TECHNOLOGY

AI_Phase-2 Document Submission

Project : AI Based Diabetes Prediction System

INTRODUCTION :

- Diabetes is a medical disorder that impacts how well our body uses food as fuel.
- Most food we eat daily is converted to sugar, commonly known as glucose, and then discharged into the bloodstream.
- Our pancreas releases insulin when the blood sugar levels rise.
- Diabetes can cause blood sugar levels to rise if it is not continuously and carefully managed, which raises the chance of severe side effects like heart attack and stroke.
- We, therefore, choose to forecast using machine learning.
- In this project we will explore innovative techniques such as ensemble methods and deep learning architectures to improve the prediction system's accuracy and robustness.
- Briefly introduce the importance of diabetes prediction.
- Emphasize the need for advanced ensemble methods like random forest classifier and deep learning architectures to enhance accuracy.

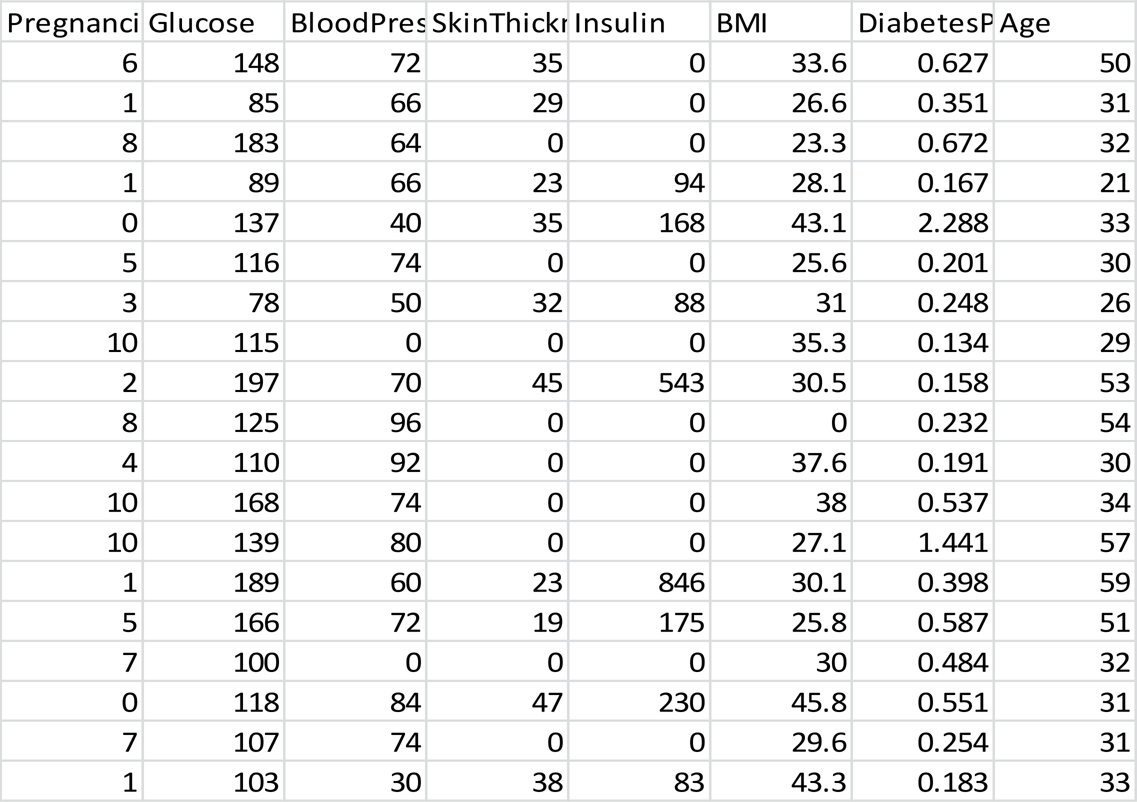
CONTENT FOR PROJECT PHASE-2:

Exploring innovative techniques such as ensemble methods and deep learning architectures to improve the AI based diabetes prediction system's accuracy and robustness.

DATA SOURCE

A good data source for AI based diabetes prediction using random forest should be accurate , complete , covering all the areas of interest , accessible.

Dataset Link : https://www.kaggle.com/datasets/mathchi/diabetes-data-set



Pregnanci	Glucose	BloodPres	SkinThickr	Insulin	BMI	DiabetesP	Age
6	148	72	35	0	33.6	0.627	50
1	85	66	29	0	26.6	0.351	31
8	183	64	0	0	23.3	0.672	32
1	89	66	23	94	28.1	0.167	21
0	137	40	35	168	43.1	2.288	33
5	116	74	0	0	25.6	0.201	30
3	78	50	32	88	31	0.248	26
10	115	0	0	0	35.3	0.134	29
2	197	70	45	543	30.5	0.158	53
8	125	96	0	0	0	0.232	54
4	110	92	0	0	37.6	0.191	30
10	168	74	0	0	38	0.537	34
10	139	80	0	0	27.1	1.441	57
1	189	60	23	846	30.1	0.398	59
5	166	72	19	175	25.8	0.587	51
7	100	0	0	0	30	0.484	32
0	118	84	47	230	45.8	0.551	31
7	107	74	0	0	29.6	0.254	31
1	103	30	38	83	43.3	0.183	33

PROGRAM :

```
import numpy as np
 import pandas as pd
import os

for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import missingno as msno
from sklearn import preprocessing
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import *
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")

pd.set_option("display.float_format",lambda x: "%.5f" % x)
pd.set_option("display.max_rows",None)
pd.set_option("display.max_columns",None)
df = pd.read_csv("/kaggle/input/pima-indians-diabetes-database/diabetes.csv")
df.head()
```

```
def check_df(dataframe):
    print("##################### Shape #####################")
    print(dataframe.shape)
    print("##################### Types #####################")
    print(dataframe.dtypes)
    print("##################### Head #####################")
    print(dataframe.head(3))
    print("##################### Tail #####################")
    print(dataframe.tail(3))
    print("##################### NA #####################")
    print(dataframe.isnull().sum())
    print("##################### Quantiles #####################")
    print(dataframe.quantile([0, 0.05, 0.50, 0.95, 0.99, 1]).T)

check_df(df)
##################### Shape #####################
(768, 9)
##################### types #####################

Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
                BMI                           float64
DiabetesPedigreeFunction    float64
Age                         int64
Outcome                     int64
dtype: object
##################### Head #####################
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin      BMI  \
0            6      148             72             35        0 33.60000
1            1       85             66             29        0 26.60000
2            8      183             64              0        0 23.30000

DiabetesPedigreeFunction  Age  Outcome
0                   0.62700   50        1
1                   0.35100   31        0
2                   0.67200   32        1
##################### Tail #####################
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin      BMI  \
765             5      121             72             23      112 26.20000
766             1      126             60              0        0 30.10000
767             1       93             70             31        0 30.40000

     DiabetesPedigreeFunction  Age  Outcome
765                   0.24500   30        0
```

```
766                   0.34900   47        1
767                   0.31500   23        0
#################### NA ####################
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
#################### Quantiles ####################
                          0.00000  0.05000   0.50000   0.95000   0.99000  \
Pregnancies               0.00000  0.00000   3.00000  10.00000  13.00000
Glucose                   0.00000 79.00000 117.00000 181.00000 196.00000
BloodPressure             0.00000 38.70000  72.00000  90.00000 106.00000
SkinThickness             0.00000  0.00000  23.00000  44.00000  51.33000
Insulin                   0.00000  0.00000  30.50000 293.00000 519.90000
BMI                       0.00000 21.80000  32.00000  44.39500  50.75900
DiabetesPedigreeFunction  0.07800  0.14035   0.37250   1.13285   1.69833
Age                      21.00000 21.00000  29.00000  58.00000  67.00000
Outcome                   0.00000  0.00000   0.00000   1.00000   1.00000

                           1.00000
Pregnancies               17.00000
Glucose                  199.00000
BloodPressure            122.00000
SkinThickness             99.00000
Insulin                  846.00000
BMI                       67.10000
DiabetesPedigreeFunction   2.42000
Age                       81.00000
Outcome                    1.00000
```

```
cols = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]
for col in cols:
    df[col].replace(0, np.NaN, inplace=True)
```

```
msno.bar(df);
```

```
msno.heatmap(df);
```

```
for col in df.columns:
    df.loc[(df["Outcome"] == 0) & (df[col].isnull()), col] = df[df["Outcome"]
== 0][col].median()
    df.loc[(df["Outcome"] == 1) & (df[col].isnull()), col] = df[df["Outcome"]
== 1][col].median()
```

```
for col in df.columns:
    if col != "Outcome":
        sns.catplot("Outcome", col, data = df)
```

```
def outlier_thresholds(dataframe, col_name, th1=0.05, th3=0.95):
    quartile1 = dataframe[col_name].quantile(th1)
    quartile3 = dataframe[col_name].quantile(th3)
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquantile_range
    low_limit = quartile1 - 1.5 * interquantile_range
    return low_limit, up_limit


def check_outlier(dataframe, col_name):
    low_limit, up_limit = outlier_thresholds(dataframe, col_name)
    if dataframe[(dataframe[col_name] > up_limit) | (dataframe[col_name] < low_limit)].any(axis=None):
        return True
    else:
        return False

def replace_with_thresholds(dataframe, col_name, th1=0.05, th3=0.95):
    low_limit, up_limit = outlier_thresholds(dataframe, col_name, th1, th3)
    if low_limit > 0:
        dataframe.loc[(dataframe[col_name] < low_limit), col_name] = low_limit
        dataframe.loc[(dataframe[col_name] > up_limit), col_name] = up_limit
    else:
        dataframe.loc[(dataframe[col_name] > up_limit), col_name] = up_limit
```

```
num_cols = [col for col in df.columns if df[col].dtypes in [int, float]
            and df[col].nunique() > 10]
```

```
for col in df.columns:
    print(check_outlier(df, col))
```

```
for col in df.columns:
    replace_with_thresholds(df, col)
```

```
for col in df.columns:
    print(check_outlier(df, col))
```

```
def label_encoder(dataframe, binary_col):
    labelencoder = preprocessing.LabelEncoder()
    dataframe[binary_col] = labelencoder.fit_transform(dataframe[binary_col])
    return dataframe


def one_hot_encoder(dataframe, categorical_cols, drop_first=False):
    dataframe = pd.get_dummies(dataframe, columns=categorical_cols, drop_first=drop_first)
    return dataframe


def rare_analyser(dataframe, target, rare_perc):
    rare_columns = [col for col in dataframe.columns if dataframe[col].dtypes == 'O'and (dataframe[col].value_counts() / len(dataframe) < rare_perc).any(axis=None)]

    for col in rare_columns:
        print(col, ":", len(dataframe[col].value_counts()))
        print(pd.DataFrame({"COUNT": dataframe[col].value_counts(),
                            "RATIO": dataframe[col].value_counts() / len(dataframe
),
"TARGET_MEAN": dataframe.groupby(col)[target].mean()}), end="\n\n\n")


def rare_encoder(dataframe, rare_perc):
    temp_df = dataframe.copy()

    rare_columns = [col for col in temp_df.columns if temp_df[col].dtypes == 'O'
and (temp_df[col].value_counts() / len(temp_df) < rare_perc).any(axis=None)]


    for var in rare_columns:
        tmp = temp_df[var].value_counts() / len(temp_df)
        rare_labels = tmp[tmp < rare_perc].index
        temp_df[var] = np.where(temp_df[var].isin(rare_labels), 'Rare', temp_df[va
r])


    return temp_df


df['NEW_BMI_CAT'] = pd.cut(x=df['BMI'], bins=[0, 18.4, 25.0, 30.0, 70.0],
                           labels=['weakness', 'normal', 'slightly_fat', 'obese'])
.astype('O')


df['NEW_GLUCOSE_CAT'] = pd.cut(x=df['Glucose'], bins=[0, 139, 200],
                               labels=['Normal', 'Prediabetes']).astype('O')

df['NEW_BLOOD_CAT'] = pd.cut(x=df['BloodPressure'], bins=[0, 79, 90, 123],
                             labels=['Normal', 'Hypertension_S1', 'Hypertension_S2
']).astype('O')

df['NEW_SKINTHICKNESS_CAT'] = df['SkinThickness'].apply(lambda x: 1 if x <= 18.0 e
lse 0)


df['NEW_INSULIN_CAT'] = df['Insulin'].apply(lambda x: 'Normal' if 16.0 <= x <=166
else 'Abnormal')

df.head()


label_cols = [col for col in df.columns if df[col].dtypes == 'O' and df[col].nuniq
ue() <= 2]
for col in label_cols:
    label_encoder(df, col)

ohe_cols = [col for col in df.columns if 10 >= len(df[col].unique()) > 2]
df = one_hot_encoder(df, ohe_cols, drop_first=True)


df.columns = [col.upper() for col in df.columns]


df.head()
```

```
y = df[['OUTCOME']]
X = df.drop('OUTCOME', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_s
tate=42)

rf = RandomForestClassifier().fit(X_train, y_train)
y_pred = rf.predict(X_test)


acc_random_forest = round(rf.score(X_test, y_pred) * 100, 2)
acc_random_forest
```

CONCLUSION AND FUTURE WORK(PHASE-2) :

Project conclusion :

- In the phase-2 conclusion , We will iterate the impact of advanced ensemble methods and deep learning architectures on improving the accuracy and robustness of diabetes prediction system.
- We have used the random forest classifier method for prediction and machine learning models for improving accuracy.
- Future work : We will discuss potential avenues for future work, such as incorporating additional data sources, exploring deep learning models for prediction etc...