Phase-4: Development Part II

Title: AI Based Diabetes Prediction System

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

Software used: google colab

Introduction:

Diabetes is a health condition that affects how your body turns food into energy. Most of the food you eat is broken down into sugar (also called glucose) and released into your bloodstream. When your blood sugar goes up, it signals your pancreas to release insulin.

Previous step:

- Installing Libraries
- Importing data
- Displaying data
- Data Preprocessing
- Missing value analysis

Next:

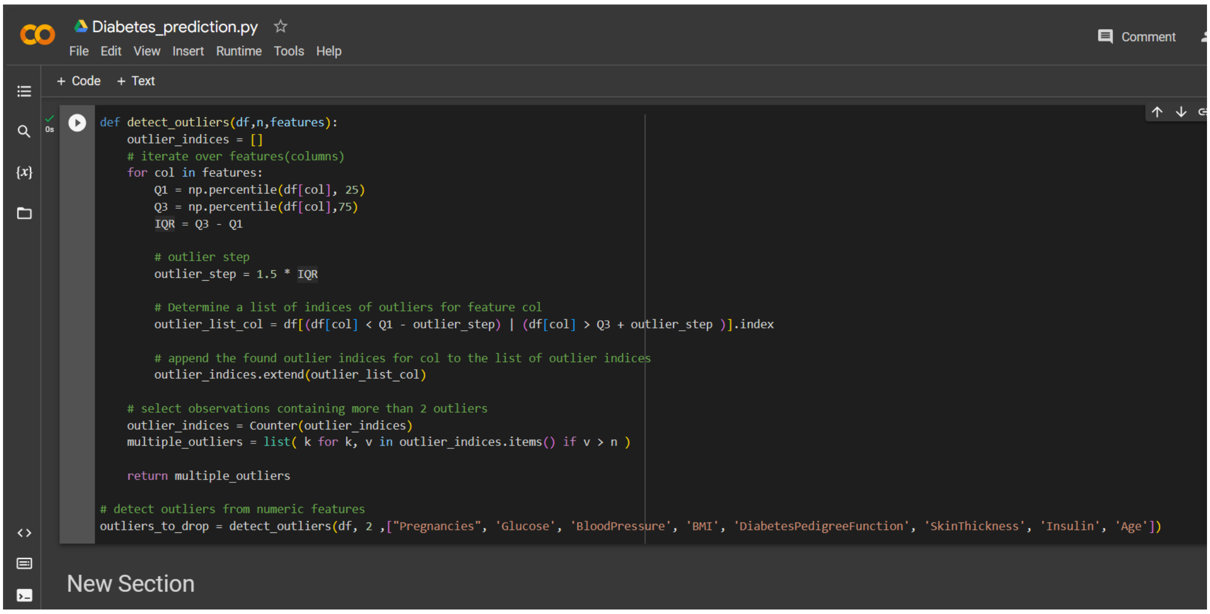
Fearure Engineering:

Till now, i explored the dataset, did missing value corrections and data visualization. Next, i have started feature engineering. Feature engineering is useful to improve the performance of machine learning algorithms and is often considered as applied machine learning.

Selecting the important features and reducing the size of the feature set makes computation in machine learning and data analytic algorithms more feasible.

Outlier Detection:

In this part i removed all the records outlined in dataset. Outliers impacts Model accuracy. I used *Tukey Method* used for outlier detection.



Here,I find outliers from all the features such as Pregnancies,Glucose,BloodPressure,BMI,DiabetesPedigreeFunction, SkinThickness, Insulin, and Age.

```
df.drop(df.loc[outliers_to_drop].index, inplace=True)
print(df)
```

I have successfully removed all outliers from dataset now. The next step is to split the dataset in train and test and proceed the modeling.

Modeling:

In this sections, i tried different models and compare the accuracy for each. Then, i performed Hyperparameter Tuning on Models that has high accuracy.

Before i split the dataset i need to transform the data into quantile using `sklearn.preprocessing` .

```
# Data Transformation
q  = QuantileTransformer()
X = q.fit_transform(df)
transformedDF = q.transform(X)
transformedDF = pd.DataFrame(X)
transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
# Show top 5 rows
transformedDF.head()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_data.py:2627: UserWarning: n_quantiles (1000) is greater than the total number of samples
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but QuantileTransformer was fitted with fe
  warnings.warn(
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.747718	0.810300	0.516949	0.801825	0.000000	0.591265	0.750978	0.889831	1.0
1	0.232725	0.097784	0.336375	0.644720	0.000000	0.227510	0.475880	0.558670	0.0
2	0.863755	0.956975	0.279009	0.000000	0.000000	0.091917	0.782269	0.585398	1.0
3	0.232725	0.131030	0.336375	0.505867	0.662973	0.298566	0.106258	0.000000	0.0
4	0.000000	0.721643	0.050847	0.801825	0.834420	0.926988	0.997392	0.606258	1.0

Data Splitting:

Next, i split data in test and train dataset. Train dataset will be used in Model training and evaluation and test dataset will be used in prediction. Before i predict the test data, i performed cross validation for various models.

```python
features = df.drop(["Outcome"], axis=1)
labels = df["Outcome"]
x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.30, random_state=7)
```

Above code splits dataset into train (70%) and test (30%) dataset.

Cross Validate Models:

```python
def evaluate_model(models):
    """
    Takes a list of models and returns chart of cross validation scores using mean accuracy
    """

    # Cross validate model with Kfold stratified cross val
    kfold = StratifiedKFold(n_splits = 10)

    result = []
    for model in models :
        result.append(cross_val_score(estimator = model, X = x_train, y = y_train, scoring = "accuracy", cv = kfold, n_jobs=4))
```

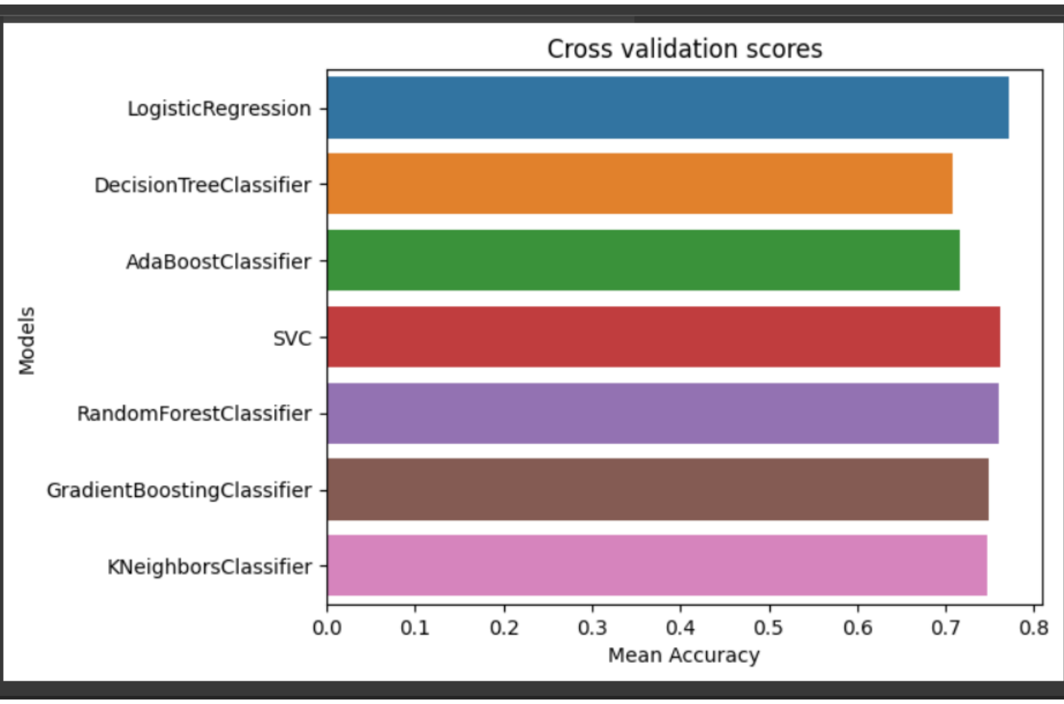
```python
    result_df = pd.DataFrame({
        "CrossValMeans":cv_means,
        "CrossValerrors": cv_std,
        "Models":[
            "LogisticRegression",
            "DecisionTreeClassifier",
            "AdaBoostClassifier",
            "SVC",
            "RandomForestClassifier",
            "GradientBoostingClassifier",
            "KNeighborsClassifier"
        ]
    })

    # Generate chart
    bar = sns.barplot(x = "CrossValMeans", y = "Models", data = result_df, orient = "h")
    bar.set_xlabel("Mean Accuracy")
    bar.set_title("Cross validation scores")
    return result_df
```

Method `evaluate_model` takes a list of models and returns chart of cross validation scores using mean accuracy.

```
# Modeling step Test differents algorithms
random_state = 30
models = [
    LogisticRegression(random_state = random_state, solver='liblinear'),
    DecisionTreeClassifier(random_state = random_state),
    AdaBoostClassifier(DecisionTreeClassifier(random_state = random_state), random_state = random_state, learning_rate = 0.2),
    SVC(random_state = random_state),
    RandomForestClassifier(random_state = random_state),
    GradientBoostingClassifier(random_state = random_state),
    KNeighborsClassifier(),
]
evaluate_model(models)
```

	CrossValMeans	CrossValerrors	Models
0	0.770964	0.058524	LogisticRegression
1	0.707687	0.065109	DecisionTreeClassifier
2	0.717016	0.061262	AdaBoostClassifier
3	0.761670	0.033724	SVC
4	0.759748	0.055512	RandomForestClassifier
5	0.748532	0.065303	GradientBoostingClassifier
6	0.746506	0.054171	KNeighborsClassifier



As per above observation, I found that Logistic Regression model has more accuracy.next, I will do hyperparameter tuning on model.

Hyperparameter Tuning:

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

```
# Import libraries
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
def analyze_grid_result(grid_result):
    '''
    Analysis of GridCV result and predicting with test dataset
    Show classification report at last
    '''
    # Best parameters and accuracy
    print("Tuned hyperparameters: (best parameters) ", grid_result.best_params_)
    print("Accuracy :", grid_result.best_score_)

    means = grid_result.cv_results_["mean_test_score"]
    stds = grid_result.cv_results_["std_test_score"]
    for mean, std, params in zip(means, stds, grid_result.cv_results_["params"]):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
    print()
    print("Detailed classification report:")
    y_true, y_pred = y_test, grid_result.predict(x_test)
    print(classification_report(y_true, y_pred))
    print()
```

First of all i have imported GridSearchCV and classification_report from sklearn library. Then, i have defined `analyze_grid_result` method which will show prediction result. I called this method for each Model used in SearchCV. In next step, i will perform tuning for each model.

Logistic Regression:

```
# Define models and parameters for LogisticRegression
model = LogisticRegression(solver='liblinear')
solvers = ['newton-cg', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
# Define grid search
grid = dict(solver = solvers, penalty = penalty, C = c_values)
cv = StratifiedKFold(n_splits = 50, random_state = 1, shuffle = True)
grid_search = GridSearchCV(estimator = model, param_grid = grid, cv = cv, scoring = 'accuracy', error_score = 0)
logi_result = grid_search.fit(x_train, y_train)
# Logistic Regression Hyperparameter Result
analyze_grid_result(logi_result)
```

Output:

```
Tuned hyperparameters: (best parameters)  {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
Accuracy : 0.774909090909091
0.773 (+/-0.241) for {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.773 (+/-0.241) for {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.241) for {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.775 (+/-0.226) for {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.240) for {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.773 (+/-0.224) for {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.773 (+/-0.242) for {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.720 (+/-0.225) for {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.764 (+/-0.245) for {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.687 (+/-0.256) for {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}

Detailed classification report:
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       147
           1       0.74      0.58      0.65        84

    accuracy                           0.77       231
   macro avg       0.77      0.73      0.74       231
weighted avg       0.77      0.77      0.77       231
```

As per my observation, in LogisticRegression it returned best score 0.78 with `*{'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}*` parameters.

Prediction:

Till now, i worked on Feature Engineering, Cross Validation of Models, and Hyperparameter Tuning and find the best working Model for my dataset. Next, I did prediction from my test dataset and storing the result in CSV.

```
# Test predictions
y_pred = logi_result.predict(x_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       147
           1       0.74      0.58      0.65        84

    accuracy                           0.77       231
   macro avg       0.77      0.73      0.74       231
weighted avg       0.77      0.77      0.77       231
```

Finally append new feature column in test dataset called Prediction and print the dataset.

Final output:

```
x_test['pred'] = y_pred
print(x_test)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
353            1       90             62             12       43  27.2
236            7      181             84             21      192  35.9
323           13      152             90             33       29  26.8
98             6       93             50             30       64  28.7
701            6      125             78             31        0  27.6
..           ...      ...            ...            ...      ...   ...
188            8      109             76             39      114  27.9
351            4      137             84              0        0  31.2
120            0      162             76             56      100  53.2
108            3       83             58             31       18  34.3
616            6      117             96              0        0  28.7

     DiabetesPedigreeFunction  Age  pred
353                     0.580   24     0
236                     0.586   51     1
323                     0.731   43     1
98                      0.356   23     0
701                     0.565   49     0
..                        ...  ...   ...
188                     0.640   31     0
351                     0.252   30     0
120                     0.759   25     1
108                     0.336   25     0
616                     0.157   30     0

[231 rows x 9 columns]
```

Conclusion:

1. Diabetes is one of the ricks during Pregnancy. It has to be treat to avoid complications.

2. BMI index can help to avoid complications of diabetes a way before

3. Diabetes start showing in age of 35 – 40 and increase with person age.