Exploring AutoML frameworks

Introduction to AutoML

AutoML, short for Automated Machine Learning, refers to the process of automating the tasks involved in applying machine learning (ML) to real-world problems. Traditional machine learning tasks often require substantial expertise in data science and machine learning techniques. AutoML aims to make machine learning more accessible to non-experts by automating various steps of the machine learning pipeline, including:

1. **Data Preprocessing:** This involves tasks such as feature engineering, handling missing values, encoding categorical variables, and scaling features.

2. **Model Selection:** AutoML tools can automatically select appropriate machine learning algorithms or even neural network architectures based on the nature of the problem, the data, and the performance metrics.

3. **Hyperparameter Tuning:** Every machine learning model has certain parameters (hyperparameters) that need to be set before training. Hyperparameter tuning involves finding the optimal values for these parameters to maximize the model's performance. AutoML tools can automate this process.

4. **Ensemble Learning:** AutoML systems often employ ensemble learning techniques to combine predictions from multiple models, which can improve performance over using a single model.

5. **Model Evaluation and Deployment:** Once models are trained, AutoML systems can evaluate their performance using various metrics and deploy them into production environments.

Benefits of AutoML:

Accessibility: Makes machine learning accessible to those with limited expertise.
Efficiency: Saves time and resources by automating time-consuming tasks.
Improved performance: Can often find better performing models than manual approaches.
Democratization: Opens up machine learning to a wider range of applications and problems.

Examples of AutoML frameworks:

Some of the popular AutoML frameworks are Google Cloud AutoML, FLAML, Auto-sklearn, TPOT, H2O AutoML, AutoGluon, AutoKeras etc.

Common use cases of AutoML:

AutoML is being utilized across various industries and domains for a wide range of applications. Some common use cases of AutoML include:

- Predictive Analytics
- Image Classification and Object Detection
- Natural Language Processing (NLP)
- Recommendation Systems
- Anomaly Detection
- Time Series Forecasting
- Healthcare Applications
- Financial Analysis
- Customer Segmentation and Marketing
- Supply Chain Optimization

The applications of AutoML continue to expand as the technology evolves and becomes more accessible to organizations across different sectors.

Selecting AutoML frameworks

I've chosen two AutoML frameworks for exploring Automated Machine Learning. They are

1. TPOT (Tree-based Pipeline Optimization Tool)
2. H2O AutoML

TPOT is an open-source Python library for automated machine learning. It uses genetic programming to optimize machine learning pipelines, including feature selection, feature engineering, and model selection. TPOT searches a wide range of possible pipelines and hyperparameters to find the best performing model for a given dataset.

Features of TPOT:

- Automated Pipeline Optimization
- Scalability
- Flexibility
- Ease of use

H2O.ai offers an open-source platform that provides scalable and distributed implementations of various machine learning algorithms, including AutoML capabilities. It automates the process of training and tuning a large number of machine learning models, including algorithms for classification, regression, and clustering. H2O AutoML handles feature engineering, model selection, and hyperparameter tuning, aiming to find the best performing model for a given dataset.

Features of H2O:

- Comprehensive solution
- Scalability
- Performance
- Integration

About the Dataset

The dataset chosen for exploring the AutoML frameworks is a public dataset containing about the parameters responsible for Diabetes and the Outcome gives whether the person has diabetes.

The Features of the dataset are
1. Pregnancies
2. Glucose
3. BloodPressure
4. SkinThickness
5. Insulin
6. BMI
7. DiabetesPedigreeFunction
8. Age

The target of the dataset is
Outcome

Setting up Environment

Importing necessary libraries:

```
import pandas as pd
```
This library is used for analyzing and manipulating a dataset.
```
from sklearn.model_selection import train_test_split
```
This library is used for Splitting the dataset into train and test sets

Loading the dataset:

```
data = pd.read_csv("/content/drive/MyDrive/diabetes.csv")
```

Data Preprocessing:

Checking null values
```
data.isna().sum()
```
Handling null values
```
data.dropna()
```

Splitting features and target:

```
X = data.iloc[:,:-1]
y = data['Outcome']
```

Splitting the data into train and test sets:

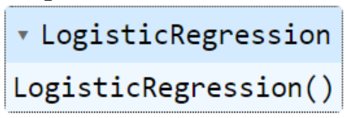
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)
```

Traditional ML (Logistic Regression)

Training the Logistic Regression model

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

Output:



```
▾ LogisticRegression
LogisticRegression()
```

Evaluating the model using Performance metrics

```
y_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall Score:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

Output:

```
Accuracy: 0.7402597402597403
Precision: 0.7291666666666666
Recall Score: 0.5645161290322581
F1 Score: 0.6363636363636364
```

Evaluation using AutoML (TPOT)

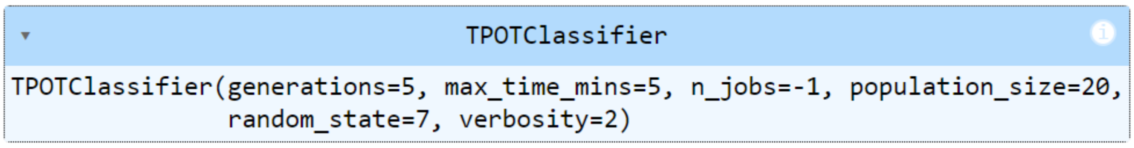
Installing TPOT:

```
!pip install tpot
```

Initializing the TPOT classifier:

```
from tpot import TPOTClassifier
tpot = TPOTClassifier(generations=5, population_size=20, verbosity=2, random_state=7, n_jobs=-1,
                        max_time_mins=5)
tpot.fit(X_train, y_train)
```

Output:

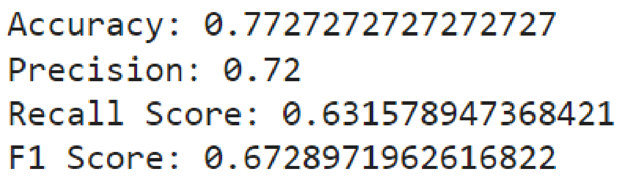


```
TPOTClassifier
TPOTClassifier(generations=5, max_time_mins=5, n_jobs=-1, population_size=20,
               random_state=7, verbosity=2)
```

Evaluating the model:

```
y_pred = tpot.predict(X_test)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall Score:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

Output:



```
Accuracy: 0.7727272727272727
Precision: 0.72
Recall Score: 0.631578947368421
F1 Score: 0.6728971962616822
```

Evaluation using AutoML (H2O)

Installing H2O AutoML framework:

```
!pip install h2o
```

Importing necessary libraries:

```
import h2o
from h2o.automl import H2OAutoML
```

Starting the H2O cluster:

```
h2o.init()
```

Loading the dataset using library of H2O:

```
file_path = "/content/drive/MyDrive/diabetes.csv"
names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age', 'Outcome']
data = h2o.import_file(file_path, col_names=names)
```

Output:

```
Parse progress: |████████████████████████████████████████████████████████████| (done) 100%
```

Converting Outcome to categorical:

```
data['Outcome'] = data['Outcome'].asfactor()
```

Splitting features and target:

```
X = data.drop('Outcome', axis=1)
y = data['Outcome']
```

Splitting data into train and test sets:

```
train, test = data.split_frame(ratios=[0.8])
```

Training AutoML:

```
aml = H2OAutoML(max_runtime_secs=120)
aml.train(y='Outcome', training_frame=train)
```

Output:

```
AutoML progress: |████████████████████████████████████████████████████████████| (done) 100%
```

Getting the best model:

```
best_model = aml.leader
```

Making predictions on test data:

```
predictions = best_model.predict(test)
```

Output:

```
glm prediction progress: |████████████████████████████████████████████████████████████████| (done) 100%
```

Performance metrics:

```
performance = best_model.model_performance(test_data=test)
print(performance)
```

Output:

```
ModelMetricsBinomialGLM: glm
** Reported on test data. **

MSE: 0.1714563551703283
RMSE: 0.4140728863018301
LogLoss: 0.525107937682899
AUC: 0.7822979659153381
AUCPR: 0.6365987342719599
Gini: 0.5645959318306761
Null degrees of freedom: 157
Residual degrees of freedom: 149
Null deviance: 199.5082434637356
Residual deviance: 165.9341083077961
AIC: 183.9341083077961
```

Converting H2O frame to pandas dataframe:

```
y_true = test['Outcome'].as_data_frame()
y_pred = predictions.as_data_frame()['predict']
```

Evaluating the model:

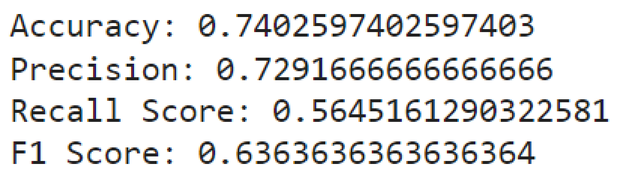
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print("Accuracy:", accuracy_score(y_true, y_pred))
print("Precision:", precision_score(y_true, y_pred))
print("Recall Score:", recall_score(y_true, y_pred))
print("F1 Score:", f1_score(y_true, y_pred))
```

Output:

```
Accuracy: 0.7151898734177216
Precision: 0.546875
Recall Score: 0.6862745098039216
F1 Score: 0.6086956521739131
```

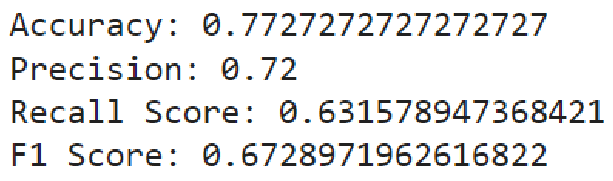
Comparison between Manual Approach and AutoML

Manual Approach (Logistic Regression):



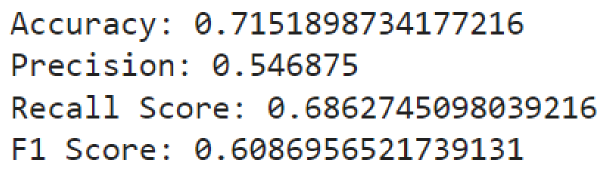
```
Accuracy: 0.7402597402597403
Precision: 0.7291666666666666
Recall Score: 0.5645161290322581
F1 Score: 0.6363636363636364
```

TPOT:



```
Accuracy: 0.7727272727272727
Precision: 0.72
Recall Score: 0.631578947368421
F1 Score: 0.6728971962616822
```

H2O:



```
Accuracy: 0.7151898734177216
Precision: 0.546875
Recall Score: 0.6862745098039216
F1 Score: 0.6086956521739131
```

Comparative Analysis

Percentage difference in Accuracy:	4.39%	&	3.39%
Percentage difference in Precision:	1.25%	&	25.9%
Percentage difference in Recall Score:	11.89%	&	21.57%
Percentage difference in F1 Score:	5.74%	&	4.35%

Based on the ease of implementation, Manual Approach is comparatively easier as it uses basic python libraries for its implementation and doesn't require any additional installation and configuration.On the other hand, both TPOT and H2O require installation of additional libraries for the respective framework.

Based on the training time, the Traditional ML method is efficiently faster and has completed training the model within 5 seconds. On the contrary, both the AutoML frameworks take a time of approximately 2 minutes to train the model.

Manual ML method and AutoML frameworks have nearly the same results for Performance metrics like Accuracy, Precision, F1 Score. AutoML frameworks show improvement in Recall Score as compared to Manual method.

Conclusion

In the nutshell, while logistic regression is a simple and interpretable model, AutoML frameworks offer potentially higher performance by automating the entire process of model selection, hyperparameter tuning, and feature engineering. Manual method is time-consuming and requires manual experimentation to find optimal settings. On the other hand, AutoML frameworks make it accessible to users with limited machine learning expertise.

However, the decision between manual methods and AutoML depends on factors such as specific requirements of the problem, complexity of the problem, available resources, and the importance of interpretability.

Deployment using Gradio

Installing gradio:

```
!pip install gradio
```

Importing necessary libraries:

```
import gradio as gr
```

Training the Logistic Regression model:

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Function for input and output:

```
def greet(Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI,
DiabetesPedigreeFunction, Age):
   inputs = np.array([float(Pregnancies), float(Glucose), float(BloodPressure), float(SkinThickness),
float(Insulin), float(BMI), float(DiabetesPedigreeFunction), float(Age)])
   inputs = inputs.reshape(1, -1)
   prediction = model.predict(inputs)
   return str(prediction[0])
```

Creating Gradio interface:

```
demo = gr.Interface(fn=greet,
            inputs=[gr.Textbox(label="Pregnancies"), gr.Textbox(label="Glucose"),
                  gr.Textbox(label="BloodPressure"), gr.Textbox(label="SkinThickness"),
                  gr.Textbox(label="Insulin"), gr.Textbox(label="BMI"),
                  gr.Textbox(label="DiabetesPedigreeFunction"), gr.Textbox(label="Age")],
            outputs=gr.Label(label="Predicted Outcome (0 or 1)"),
            title="Prediction using Logistic Regression",
            description="Enter the required set of values, and the model predicts whether you have
                  Diabetes.\ ('0' represents that you don't have Diabetes. '1' represents that the chances
                  of you having Diabetes is high.)"
            )
```

Launching the interface:

```
demo.launch(debug=True)
```

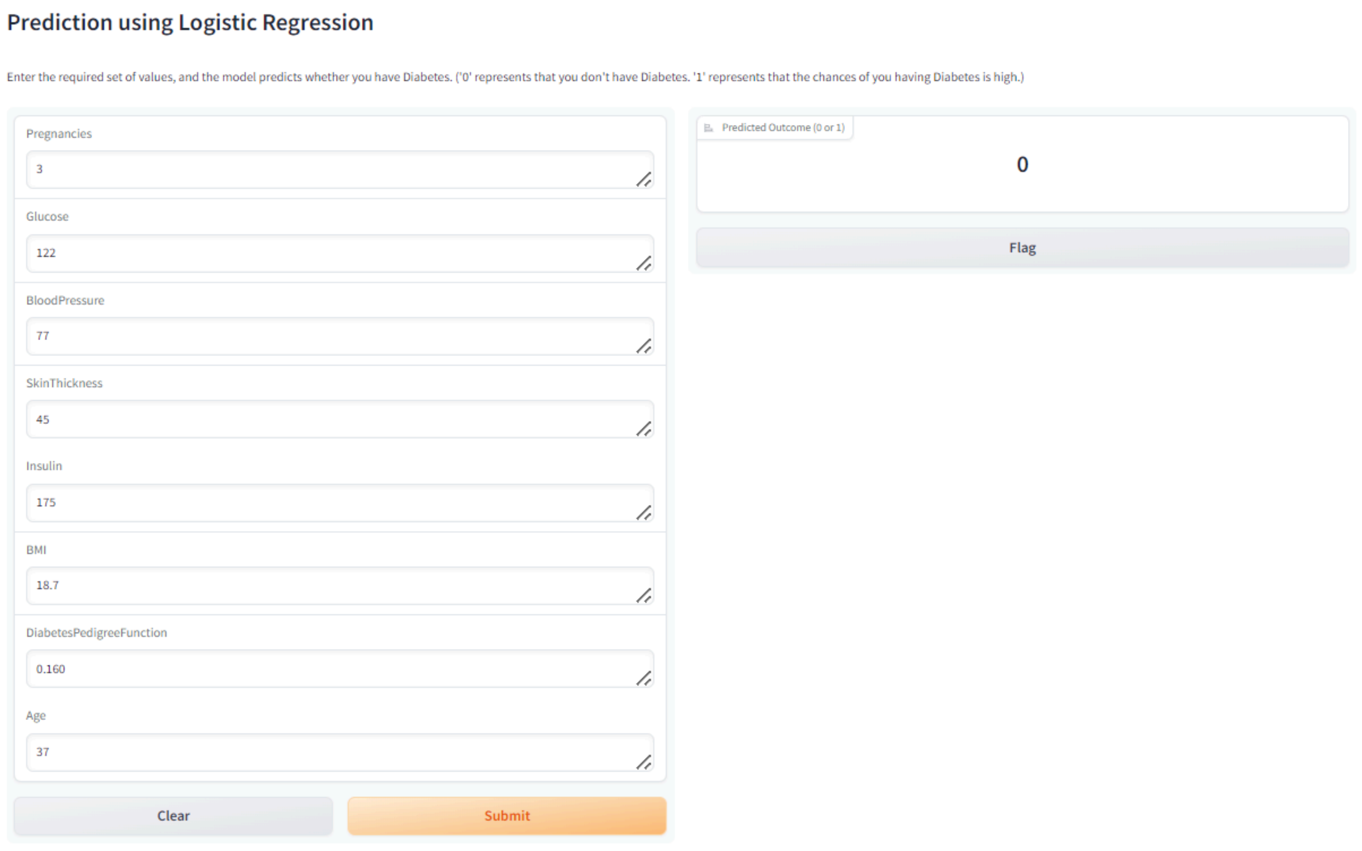
Output:

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to c
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Setting queue=True in a Colab notebook requires sharing enabled. Setting `share=True` (you can turn this off by sett

Colab notebook detected. This cell will run indefinitely so that you can see errors and logs. To turn off, set debug
Running on public URL: https://1f352e6fb0d7606a0a.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Terminal
```

OUTPUT IN GRADIO INTERFACE:

URL: https://1f352e6fb0d7606a0a.gradio.live/

Output:



Prediction using Logistic Regression

Enter the required set of values, and the model predicts whether you have Diabetes. ('0' represents that you don't have Diabetes. '1' represents that the chances of you having Diabetes is high.)

Pregnancies: 3
Glucose: 122
BloodPressure: 77
SkinThickness: 45
Insulin: 175
BMI: 18.7
DiabetesPedigreeFunction: 0.160
Age: 37

Predicted Outcome (0 or 1): 0