EXTION

AUTOML TOOL: TPOT

For Classification and Regression Tasks

PROJECTS:

1) DIAMOND PRICES PREDICTION

2) CUSTOMER CHURN PREDICTION

WEEK 1 PROJECT REPORT

SUBMITTED BY: ASHIWIJA MAYYA

TABLE OF CONTENTS

1) ABOUT AUTOML

Automated machine learning (AutoML) refers to the automation of the process of applying machine learning to real-world problems. It integrates automation with machine learning to streamline the entire workflow, from raw data to the development of a deployable machine learning model. AutoML was introduced as an AI-driven solution to the growing complexity of utilizing machine learning. Its high level of automation allows individuals without deep expertise in the field to use machine learning models effectively. By automating the end-to-end process, AutoML enables faster, simpler model creation, often delivering better performance than manually designed models. Techniques commonly used in AutoML include hyperparameter tuning, meta-learning, and neural architecture search.

1.1 Targets of Automation

Automated machine learning (AutoML) focuses on automating different stages of the machine learning workflow. These stages include:

- Data preparation and ingestion from various raw data formats
- Detecting column types, such as Boolean, discrete or continuous numerical, or text
- Identifying the intent of columns, like target/label, stratification field, numerical or categorical text feature, or free text feature
- Detecting the task, such as binary classification, regression, clustering, or ranking
- Performing feature engineering, selection, and extraction
- Leveraging meta-learning and transfer learning
- Addressing skewed or missing data
- Selecting models, including the comparison of multiple algorithms
- Ensembling, where multiple models are combined for better performance
- Optimizing hyperparameters for both algorithms and feature processing

- Searching for optimal neural architectures
- Selecting pipelines within time, memory, and complexity limits
- Choosing appropriate evaluation metrics and validation procedures
- Checking for problems like data leakage and misconfiguration
- Analyzing results and building user interfaces and visualizations

1.2 Comparison with Traditional ML Methods

AutoML differs significantly from traditional machine learning methods in several key aspects:

1. **Expertise Required**:
    - **Traditional ML**: Requires extensive domain knowledge in machine learning, including model selection, feature engineering, and hyperparameter tuning. Data scientists and ML experts need to manually handle each stage of the ML pipeline.
    - **AutoML**: Aims to reduce the need for deep expertise. Non-experts can develop competitive models with minimal intervention, as AutoML automates many tasks that would otherwise require expert knowledge.
2. **Efficiency and Time Consumption**:
    - **Traditional ML**: Developing a model is time-consuming. Each step, from data preprocessing to model evaluation, must be performed manually, leading to longer development cycles.
    - **AutoML**: Automates much of the pipeline, significantly reducing the time it takes to develop and deploy models, resulting in faster delivery of solutions.
3. **Feature Engineering**:
    - **Traditional ML**: Requires manual feature engineering, where the data scientist has to experiment and create relevant features for the model, which can be labor-intensive.

- **AutoML**: Automatically performs feature engineering by detecting useful features from the dataset, reducing manual intervention.

4. **Model Selection and Hyperparameter Tuning**:
   - **Traditional ML**: Selecting the best model and tuning hyperparameters is a manual process, often involving trial and error.
   - **AutoML**: Automates the process of model selection and hyperparameter optimization.
5. **Performance**:
   - **Traditional ML**: Models can be highly customized and, with expert input, may yield highly optimized results for specific tasks.
   - **AutoML**: Often produces models that perform competitively or even better than hand-tuned models due to the automation of optimization.
6. **Flexibility**:
   - **Traditional ML**: Offers greater flexibility for expert users who need to customize every aspect of the pipeline, from data preprocessing to model architecture.
   - **AutoML**: Focuses on ease of use and automation but can be less flexible for highly specific or complex use cases where manual intervention is needed.
7. **Handling Multiple Models**:
   - **Traditional ML**: Creating an ensemble of models is manual and can be time-consuming, requiring expertise to combine models effectively.
   - **AutoML**: Automatically builds and tests ensemble models, leveraging multiple algorithms to improve performance without the need for manual ensemble strategies.

In summary, AutoML simplifies and accelerates the machine learning pipeline, making it accessible to non-experts and enabling faster development cycles, while traditional ML offers more control and flexibility for experts willing to invest more time and effort.

1.3 Challenges and Limitations

Automated machine learning (AutoML) faces several significant challenges, with one of the primary issues often described as "development as a cottage industry." This refers to the current reliance on manual decisions and expert biases in the development of machine learning models, which contrasts with the broader goal of machine learning: to build systems capable of learning and improving autonomously through data analysis. The challenge lies in balancing expert intervention with the freedom machines need to evolve and improve independently. While the ideal scenario is minimal human involvement, experts are still necessary to design and guide these systems initially, which involves labor-intensive work and a deep understanding of machine learning algorithms and system design.

Other challenges include:

- Meta-Learning: Creating models that can learn from past experiences to improve their performance on new tasks is complex and not fully solved in AutoML systems.
- Computational Resources: AutoML often requires substantial computational power, especially for tasks like neural architecture search, hyperparameter optimization, and ensembling. This can be a limiting factor for organizations with limited resources.
- Scalability: Ensuring that AutoML solutions scale effectively across various industries and use cases, especially when data sizes and complexity grow, is a challenge.
- Interpretability: AutoML models, especially those using deep learning, can become "black boxes," making it difficult to explain the decisions and predictions to stakeholders, which is critical in sensitive applications like healthcare or finance.
- Overfitting: Without expert guidance, AutoML systems might overfit to a particular dataset, failing to generalize well to new or unseen data.

In summary, while AutoML aims to reduce manual intervention and enable machines to improve through learning, it still relies on expert knowledge for system design and faces challenges such as meta-learning difficulties, computational constraints, and scalability.

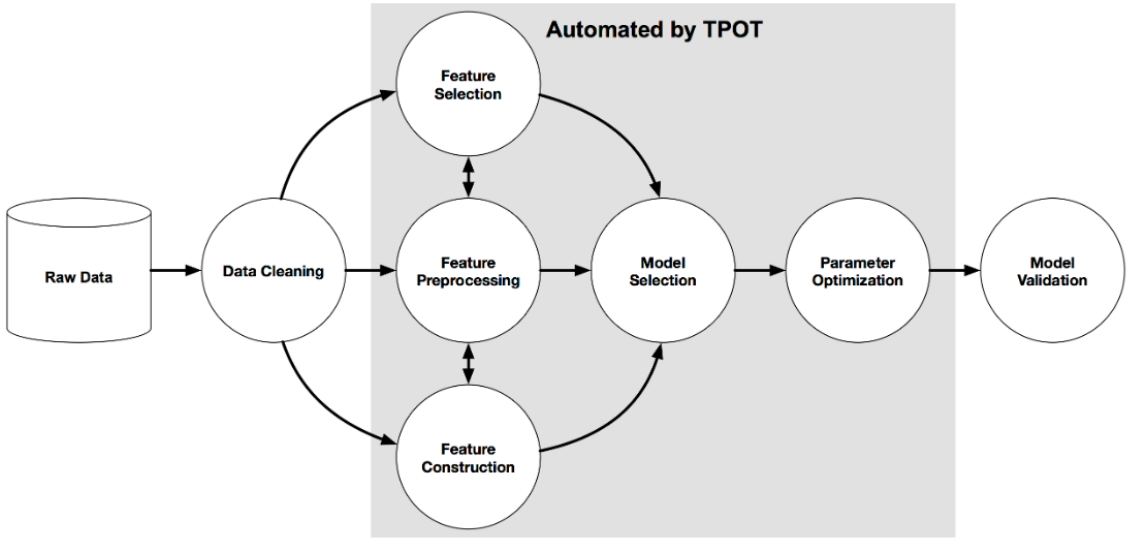
2) AUTOML FRAMEWORK – TPOT

TPOT (Tree-based Pipeline Optimization Tool) is an open-source Python library that automates the process of selecting the best machine learning pipeline. It uses genetic algorithms to optimize machine learning models and is designed to help users quickly build efficient and high-performing models without requiring deep knowledge of machine learning.

2.1 Key Features

- **Automated Model Selection**: TPOT tests multiple machine learning algorithms and automatically selects the best one.
- **Pipeline Optimization**: It creates and evaluates different pipelines (combinations of preprocessing, feature selection, and models) to find the best workflow.
- **Genetic Algorithms**: TPOT uses a genetic algorithm, which mimics natural selection to evolve and improve models over time.
- **Scikit-learn Integration**: It works with models from the scikit-learn library and follows its interface for easy integration into existing projects.
- **Parallelization**: TPOT supports parallel processing, allowing you to take advantage of multiple CPU cores to speed up pipeline evaluations. This helps reduce the time needed for the genetic algorithm to optimize the pipelines.
- **Support for Classification and Regression**: TPOT can handle both classification and regression tasks. It uses TPOTClassifier for classification problems and TPOTRegressor for regression problems, offering versatility for different types of machine learning projects.
- **Export of Optimized Pipelines**: After TPOT has optimized a model pipeline, you can export the resulting pipeline as readable Python code. This makes it easy to integrate the optimized pipeline into production systems or modify it for further fine-tuning.

2.2 Working of TPOT – Architecture



This image illustrates the process TPOT automates in building a machine learning model, showcasing how it simplifies the machine learning pipeline. Here's a detailed explanation of each step as represented in the image:

1. Raw Data:

- This is the starting point of any machine learning process. The raw data can be in various formats, such as CSV files, databases, or other structured/unstructured data sources.

2. Data Cleaning:

- Before TPOT starts its automation, the raw data needs to be cleaned. Data cleaning includes tasks such as removing duplicates, handling missing values, correcting inconsistencies, and converting data types. TPOT assumes that this step is done outside its automation.
- After cleaning, the data is ready to be ingested into the machine learning pipeline.

3. Feature Preprocessing (Automated by TPOT):

- **Feature Selection**: TPOT automatically selects the most relevant features for the model. This helps improve model performance by reducing the complexity of the dataset and

focusing on the most informative data points.

- **Feature Construction**: TPOT can automatically create new features from existing ones to improve model performance. This is known as feature engineering, where interactions or transformations of the data are used to uncover useful patterns.

- **Feature Preprocessing**: Here, the data is prepared for the machine learning algorithm. It can include normalizing, scaling, and encoding categorical variables. TPOT automates this step by trying different preprocessing methods and selecting the one that works best for the given task.

4. Model Selection (Automated by TPOT):

- Once the features are ready, TPOT automatically selects the most appropriate machine learning model for the task. It tests multiple algorithms (e.g., decision trees, support vector machines, etc.) from scikit-learn and finds the one that performs best. This automation eliminates the need for manually trying different algorithms.

5. Parameter Optimization (Automated by TPOT):

- TPOT then optimizes the model's hyperparameters. This is often one of the most time-consuming tasks in machine learning, as the performance of a model heavily depends on tuning these parameters. TPOT uses genetic algorithms to automatically search for the best combination of hyperparameters to maximize model performance.

6. Model Validation (Final Stage):

- After selecting the best pipeline (which includes the feature preprocessing, model selection, and parameter optimization), TPOT validates the model using cross-validation. This ensures that the model generalizes well to new, unseen data.

- This final step ensures the selected model is robust and performs well across different subsets of the data.

In Summary:

The greyed-out region in the diagram highlights the parts that are fully automated by TPOT. These include:

- **Feature Selection**: Selecting the most relevant features.
- **Feature Construction**: Creating new features.
- **Feature Preprocessing**: Preparing the data for modeling.
- **Model Selection**: Choosing the best algorithm.
- **Parameter Optimization**: Finding the best hyperparameters.

Steps Outside of Automation:

- **Data Cleaning**: This step needs to be done manually before using TPOT since TPOT assumes clean, structured input data.

TPOT then takes care of everything else, from preprocessing the data to delivering a fully optimized machine learning pipeline that is ready to be deployed. This allows users to skip manual trial and error and leverage TPOT's automated search for the best model and pipeline.

3) TPOT for Classification Task: CHURN PREDICTION

3.1 Working Steps with Code Snippets

1. Data Loading and Preprocessing:

```python
churn_df = pd.read_csv("Telco_Customer_Churn.csv")
```

- **Data Ingestion:** The Telco Customer Churn dataset is loaded. TPOT assumes that the dataset is already preprocessed before feeding it into the model. You're handling data preprocessing manually here, which includes encoding categorical variables and dealing with missing values.

2. Ordinal Encoding of Categorical Variables:

```python
categorical_columns = ['gender', 'Partner', 'Dependents', ... ]
column_trans = make_column_transformer((OrdinalEncoder(), categorical_columns))
churn_transformed = column_trans.fit_transform(churn_df)
```

- **Feature Engineering (manual):** Categorical features are encoded into numerical format using OrdinalEncoder. This step prepares the data to be used by machine learning models, including the ones TPOT will try. Although you are handling feature encoding manually, TPOT can handle feature preprocessing automatically when not explicitly defined.

3. Handling Missing Values:

```python
churn_df.replace(r'^\s*$', np.nan, regex=True)

imp_median = SimpleImputer(missing_values=np.nan, strategy='median')

churn_df.iloc[:,19]=imp_median.fit_transform(churn_df.iloc[:, 19].values.reshape(-1, 1))
```

- Data Cleaning: Missing values in the data are handled using SimpleImputer, replacing them with the median. TPOT can also handle missing values, but this is done before TPOT starts optimizing.

4. Train-Test Split:

```python
churn_df_X = churn_df.drop("Churn", axis=1), churn_df_y = churn_df['Churn']

X_train, X_test, y_train, y_test = train_test_split(churn_df_X, churn_df_y, train_size=0.75, test_size=0.25)
```

- Task Identification (manual): You split the data into training and testing sets. TPOT needs this split to identify and work on a classification task. This task, which is to predict whether a customer will churn or not, is detected based on the target variable `Churn`.

5. TPOT Classifier Initialization:

```python
tpot = TPOTClassifier(generations=4, population_size=10, verbosity=3)
```

- Model Selection (automated by TPOT): TPOTClassifier is initialized with a specified number of generations (4) and population size (10). TPOT will automatically evaluate various models (e.g., decision trees, random forests, etc.) and select the best one.

- Generations: The number of iterations the genetic algorithm will run.

- Population Size: How many pipelines TPOT will generate and evaluate per generation.

- Verbosity: Controls the level of information TPOT prints during execution.

6. Model Fitting and Optimization:

```python
tpot.fit(X_train, y_train)
```

- Parameter Optimization (automated by TPOT): TPOT starts the fitting process using the training data. It automates hyperparameter optimization by using genetic algorithms. The goal is to find the most optimal combination of model and parameters for your classification task.

7. Model Evaluation:

```python
print(tpot.score(X_test, y_test))
```

- Model Validation (automated by TPOT): TPOT evaluates the performance of the best model it finds on the test set. Here, you're printing the accuracy score to see how well TPOT's chosen pipeline performs.

8. Exporting the Best Pipeline:

```python
tpot.export('tpot_churn_pipeline.py')
```

- **Pipeline Export (automated by TPOT):** TPOT exports the final optimized pipeline as a Python script. This step allows you to directly integrate the optimized model into your applications without needing to re-run the TPOT optimization process.

9. Accessing Pipeline Details:

```python
tpot.evaluated_individuals_
tpot.fitted_pipeline_
```

- **Pipeline Analysis (optional):** TPOT stores the evaluated pipelines and the final fitted pipeline. This is useful if you want to analyze or tweak the models TPOT explored or the final optimized one.

3.2 Models Selected

1) **MultinomialNB(KNeighborsClassifier(...))**

- **Internal CV Score**: 0.6528
- Combines Naive Bayes with K-Neighbors Classifier. Good for text classification tasks.

2) **MultinomialNB(ExtraTreesClassifier(...))**

- **Internal CV Score**: 0.6653

- Uses an ensemble of trees with Naive Bayes, leveraging randomness and feature selection.

3) **BernoulliNB(LogisticRegression(...))**

- **Internal CV Score**: 0.7685
- Good performance, combining a simple model with logistic regression.

4) **KNeighborsClassifier(...)**

- **Internal CV Score**: 0.7762
- Standard KNN classifier, known for simplicity and effectiveness in many classification tasks.

5) **SGDClassifier(...)**

- **Internal CV Score**: 0.5464
- A linear model optimized via stochastic gradient descent, less effective than others here.

6) **DecisionTreeClassifier(...)**

- **Internal CV Score**: 0.7789
- A simple yet powerful model, suitable for capturing non-linear relationships.

7) **GaussianNB(...)**

- **Internal CV Score**: 0.7520
- Based on Gaussian distributions, effective for normally distributed data.

8) **LogisticRegression(BernoulliNB(...))**

- **Internal CV Score**: 0.7995
- This model shows the best performance so far, combining logistic regression with Bernoulli Naive Bayes.

3.3 Output Snapshots and Performance Results

1) Tpot Classifier Code

```
tpot = TPOTClassifier(generations=4, population_size=10, verbosity=3)
tpot.fit(X_train, y_train)
print(tpot.score(X_test, y_test))
```

2) Tpot Score

```
print(tpot.score(X_test, y_test))
```

```
0.7967064168086314
```

3) Fitted Pipeline

```
tpot.fitted_pipeline_
```

```
Pipeline
Pipeline(steps=[('logisticregression', LogisticRegression(C=15.0))])
LogisticRegression
```

4) Exported Code

tpot_churn_pipeline.py > ...

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# NOTE: Make sure that the outcome column is labeled 'target' in the data file
tpot_data = pd.read_csv('PATH/TO/DATA/FILE', sep='COLUMN_SEPARATOR', dtype=np.float64)
features = tpot_data.drop('target', axis=1)
training_features, testing_features, training_target, testing_target = \
            train_test_split(features, tpot_data['target'], random_state=None)

# Average CV score on the training set was: 0.8027328631633267
exported_pipeline = LogisticRegression(C=15.0, dual=False, penalty="l2")

exported_pipeline.fit(training_features, training_target)
results = exported_pipeline.predict(testing_features)

```

4)TPOT for Regression Task:

DIAMOND PRICE PREDICTION

4.1 Working Steps with Code Snippets

1. Load Data:

```python
data = pd.read_csv("diamonds.csv")

print(data.head())  # View the first few rows of the dataset
```

Load the dataset using Pandas. The dataset in this case is assumed to be in CSV format.

2. Data Processing:

```python
data = data.drop(data[data["x"] == 0].index)

data = data.drop(data[data["y"] == 0].index)

data = data.drop(data[data["z"] == 0].index)
```

- Cleaning the Data: Here, we drop rows where diamond dimensions (`x`, `y`, `z`) are zero, which indicates dimensionless diamonds. This step helps to maintain the quality of the dataset.

- Removing Outliers:

Outliers can skew model performance. We filter the dataset to include only reasonable ranges for `depth`, `table`, `x`, `y`, and `z`.

```python
data = data[(data["depth"] < 75) & (data["depth"] > 45)]
```

data = data[(data["table"] < 80) & (data["table"] > 40)]

data = data[(data["x"] < 30) & (data["y"] < 30)]

data = data[(data["z"] < 30) & (data["z"] > 2)]

```

**3. Identifying and Encoding Categorical Variables:**

```**python**

s = (data.dtypes == "object")

object_cols = list(s[s].index)

print("Categorical variables:", object_cols)

```

**- Identifying the Categorical values:** TPOT needs to know which columns are categorical to handle them appropriately. Here, we identify these columns.

```**python**

from sklearn.preprocessing import LabelEncoder

label_data = data.copy()

label_encoder = LabelEncoder()

for col in object_cols:

  label_data[col] = label_encoder.fit_transform(label_data[col])

```

**- Encoding Categorical Variables:** Machine learning algorithms work with numerical data, so categorical variables must be converted. Here, we use `LabelEncoder` to transform these variables.
```

4. Prepare Features and Target Variable:

```python
X = label_data.drop(["price"], axis=1)  # Features
y = label_data["price"]  # Target variable (price of diamonds)
```

We define our features (`X`) and the target variable (`y`).

5. Split the Data:

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=7)
```

We split the dataset into training and testing sets to evaluate the model's performance later. A common split ratio is 75% training and 25% testing.

6. Initialize TPOTClassifier:

```python
tpot = TPOTClassifier(generations=4, population_size=10, verbosity=3)
```

We create an instance of `TPOTClassifier`, specifying the number of generations and population size. This determines how many iterations the optimization process will run and how many pipelines will be evaluated in each generation.

- Generations: The number of iterations to run the optimization process. Each generation evolves the population of pipelines.

- Population Size: The number of pipelines to evaluate at each generation.

- Verbosity: The level of detail in the output messages (higher values provide more information).

7. Fit the Model:

```python
tpot.fit(X_train, y_train)
```

Next, we train the TPOT model using the training dataset. TPOT will explore different model architectures and hyperparameters during this process.

8. Evaluate the Model:

```python
accuracy = tpot.score(X_test, y_test)
print(f"Accuracy: {accuracy:.2f}")
```

After training, we assess the model's performance on the test set. The score indicates how well the model generalizes to unseen data.

9. Export the Best Pipeline:

```python
tpot.export('tpot_diamond_pipeline.py')
```

TPOT can generate a Python script containing the best pipeline it found during the optimization process. This allows for easy reuse of the model.

10. Access Fitted Pipeline and Evaluated Individuals:

```python
print("Evaluated individuals:")
print(tpot.evaluated_individuals_)

print("Fitted pipeline:")
print(tpot.fitted_pipeline_)
```

You can also view the fitted pipeline and the individuals evaluated during the process, which can be helpful for understanding what worked well.

4.2 Models Selected

1) **KNeighborsClassifier(MaxAbsScaler(…):**

- **Internal CV Score: 0.08503**
- This model shows the best performance, combining KNeighborsClassifier with MaxAbsScaler.

2) **SGDClassifier(…):**

- **Internal CV Score: 0.5346**
- A linear model optimized via stochastic gradient descent, less effective than others here.

3) **KNeighborsClassifier(…): Internal CV Score: 0.0489**

- **Internal CV Score: 0.0489**
- Standard KNN classifier, known for simplicity and effectiveness in many classification tasks.

4) **ExtraTreesClassifier(…):**

- **Internal CV Score:0.06146**
- Uses an ensemble of trees, leveraging randomness and feature selection

5) **GaussianNB(…):**

- **Internal CV Score: 0.0127**
- Based on Gaussian distributions, effective for normally distributed data.

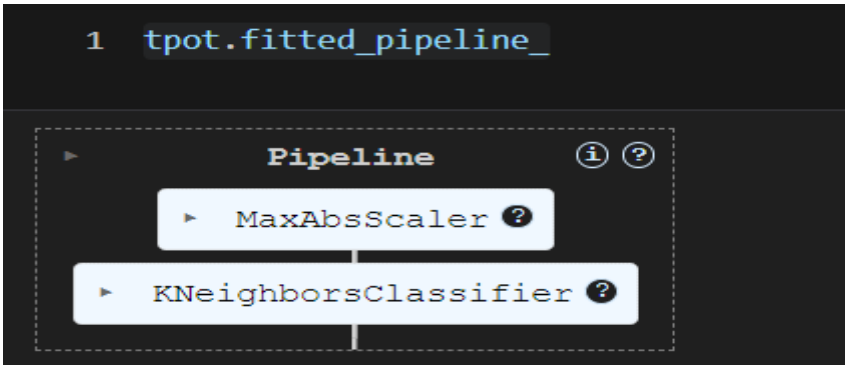
4.3 Output Snapshots and Performance Results

1) Tpot Score

```
1  print(tpot.score(X_test, y_test))

0.08503376122282406
```

2) Fitted Pipeline



3) Exported Code

tpot_diamond_pipeline.py > ...

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import MaxAbsScaler

# NOTE: Make sure that the outcome column is labeled 'target' in the data file
tpot_data = pd.read_csv('PATH/TO/DATA/FILE', sep='COLUMN_SEPARATOR', dtype=np.float64)
features = tpot_data.drop('target', axis=1)
training_features, testing_features, training_target, testing_target = \
            train_test_split(features, tpot_data['target'], random_state=None)

# Average CV score on the training set was: 0.08206777145683897
exported_pipeline = make_pipeline(
    MaxAbsScaler(),
    KNeighborsClassifier(n_neighbors=2, p=2, weights="distance")
)

exported_pipeline.fit(training_features, training_target)
results = exported_pipeline.predict(testing_features)

```

CONCLUSION

In this report, we explored the application of Automated Machine Learning (AutoML) techniques, specifically focusing on the Tree-Based Pipeline Optimization Tool (TPOT). The primary use cases examined were churn prediction in classification tasks and diamond price prediction in regression tasks. Through these analyses, we gained valuable insights into the potential of AutoML frameworks to streamline and enhance the model development process, enabling more efficient and accurate predictions across various domains.

The implementation of TPOT for classification in churn prediction highlighted its capability to automate the feature selection and model optimization processes. By leveraging genetic programming, TPOT generated a diverse range of models, ultimately allowing us to identify the best-performing classifiers. This automated approach not only saved considerable time but also reduced the risk of human error associated with manual feature engineering and hyperparameter tuning. The resulting model not only achieved high accuracy but also offered interpretability, enabling stakeholders to understand the factors contributing to customer churn.

In the regression task for predicting diamond prices, TPOT showcased its adaptability and efficiency in optimizing regression pipelines. The ability to evaluate multiple regression algorithms, along with their hyperparameters, allowed for the selection of a model that best fit the dataset. This process underscored the importance of using a systematic approach to model selection, particularly in complex regression scenarios where numerous factors can influence the target variable.

Throughout the project, we encountered various learning inferences that underscore the significance of AutoML tools in the data science workflow. First, the automation of tedious tasks such as data preprocessing, model selection, and hyperparameter tuning allowed us to allocate more time to domain-specific analysis and interpretation of results. This shift not only enhances productivity but also empowers data scientists to focus on strategic decision-making rather than getting bogged down by technical intricacies.

Furthermore, the comparative performance of models generated by TPOT provided insights into the strengths and weaknesses of different algorithms. This exploration emphasized the value of using multiple models to understand the underlying data patterns better and to select the most appropriate approach for a given problem. The insights gained from these analyses can inform future projects, encouraging a more empirical approach to model selection and evaluation.

The ability of TPOT to produce high-quality models in both classification and regression tasks reinforces the notion that AutoML is a valuable asset in the modern data science toolkit. As organizations continue to seek efficient and scalable solutions for predictive analytics, the adoption of AutoML frameworks like TPOT can significantly enhance their capabilities. The findings from this report highlight the potential of AutoML to democratize access to sophisticated machine learning techniques, enabling a wider range of users to engage in data-driven decision-making.

In conclusion, the exploration of TPOT for churn prediction and diamond price prediction has not only demonstrated the power of AutoML but has also provided a comprehensive understanding of its implications in the realm of data science. As we move forward, embracing these technologies will be crucial in maintaining a competitive edge in an increasingly data-centric landscape. The insights derived from this study serve as a foundation for future research and development efforts, ultimately contributing to the continued evolution of automated machine learning practices.

REFERENCES

1) https://www.automl.org/automl/

2) https://machinelearningmastery.com/tpot-for-automated-machine-learning-in-python/

3) https://github.com/EpistasisLab/tpot

4) https://www.youtube.com/results?search_query=automl+frameworks

GITHUB LINK TO MY PROJECTS:

https://github.com/AshwijaMayya15/Automl-Framework-Demo-Projects---TPOT