Ozan MÖHÜRCÜ

Data Analyst | Data Scientist

LinkedIn GitHub

What is AutoML

- AutoML (Automated Machine Learning) is a technology that enables the automatic creation, training, and optimization of machine learning models without human intervention.

- It automates tasks such as data preprocessing, model selection, and hyperparameter tuning, allowing even non-expert users to build effective models. AutoML is especially useful for saving time and reducing the need for deep machine learning expertise.

Libraries Import

In [1]:
```
%%capture
!pip install flaml

import pandas as pd
import numpy as np
from flaml import AutoML
import warnings
warnings.filterwarnings('ignore')
```

Data Loading

In [2]:
```
train = pd.read_csv('/kaggle/input/playground-series-s5e5/train.csv', index
test = pd.read_csv('/kaggle/input/playground-series-s5e5/test.csv', index_c
sub = pd.read_csv('/kaggle/input/playground-series-s5e5/sample_submission.c
```

Feature Engineering

- Feature Engineering is the process of creating meaningful input features from raw data to improve the performance of machine learning models.
- It involves transforming, selecting, or generating new features.
- Good feature engineering can significantly enhance model accuracy and efficiency.
- Even simple models can perform well with well-crafted features.

In [3]:
```
def feature_engineering(df):
    df = df.copy()
    df['BMI'] = df['Weight'] / ((df['Height'] / 100) ** 2)
    df['Body_Temp_Duration'] = df['Body_Temp'] * df['Duration']
    df['Weight_Heart_Rate'] = df['Weight'] * df['Heart_Rate']
    df = pd.get_dummies(df, columns=['Sex'], drop_first=True)
    return df


train_fe = feature_engineering(train)
test_fe = feature_engineering(test)
```

Understanding the Code: Data Preparation

1. **X_train = train_fe.drop(columns='Calories')** creates the feature matrix by removing the target column 'Calories' from the dataset.
2. **y_train = np.log1p(train_fe['Calories'])** transforms the target variable by applying the natural logarithm plus one, which helps in stabilizing variance and handling skewness.
3. This process prepares the data for training machine learning models by separating input features (X_train) and the transformed target variable (y_train).

In [4]:
```
X_train = train_fe.drop(columns='Calories')
y_train = np.log1p(train_fe['Calories'])
```

Using AutoML for Regression

1. **aml = AutoML()** creates a new AutoML instance to automate the machine learning workflow.
2. The **fit()** method trains the model using `X_train` as input features and `y_train` as the target variable.

3. The **task='regression'** parameter specifies that the model is solving a regression problem.
4. **metric='rmse'** sets Root Mean Squared Error as the evaluation metric to measure prediction accuracy.
5. **time_budget=3600** limits the training process to one hour to manage computational resources.
6. **eval_method='cv'** and **n_splits=5** enable 5-fold cross-validation for more robust model evaluation.
7. **estimator_list=['xgboost', 'lgbm', 'catboost']** restricts the search to these three popular gradient boosting algorithms.
8. **ensemble=True** allows combining multiple models to improve overall prediction performance.
9. **verbose=3** provides detailed output during training, useful for monitoring progress.

In [5]:

```
aml = AutoML()
aml.fit(
    X_train,
    y_train,
    task='regression',
    metric='rmse',
    time_budget=3600, # 1 Hour
    eval_method='cv',
    n_splits=5,
    estimator_list=['xgboost', 'lgbm', 'catboost'],
    ensemble=True,
    verbose=1
)
```

AutoML Results Summary

1. **aml.best_estimator** displays the name of the best-performing model found during the AutoML process.
2. **aml.best_config** and **aml.best_loss** provide the optimal hyperparameters and the lowest validation loss achieved, respectively.

In [6]:

```
print("The best model:", aml.best_estimator)
print("Best configuration:", aml.best_config)
print("Best validation loss:", aml.best_loss)
```

```
The best model: catboost
Best configuration: {'early_stopping_rounds': 11, 'learning_rate': 0.005, 'n_
estimators': 8192}
Best validation loss: 0.060354510942818615
```

Data Preparation and Model Training

Pipeline

1. The **feature_engineering()** function creates new meaningful features such as BMI, Body_Temp_Duration, and Weight_Heart_Rate, and applies one-hot encoding to the 'Sex' column.
2. The target variable 'Calories' is log-transformed using `np.log1p` to stabilize variance and reduce skewness.
3. Numerical and categorical columns are identified for preprocessing, where numerical features are standardized and categorical features are one-hot encoded.
4. A pipeline is created that combines preprocessing steps with the LightGBM regression model, configured with specific hyperparameters.
5. A 5-fold cross-validation strategy is implemented using `KFold` to split data into training and validation sets.
6. For each fold, the model is trained and predictions are collected to compute out-of-fold predictions.
7. Finally, the predictions are transformed back from the log scale, clipped to a reasonable range, and evaluated using the RMSLE metric to measure model accuracy.

In [7]:

```python
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_log_error
from sklearn.model_selection import KFold
from lightgbm import LGBMRegressor
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline

print("Train data columns:", train.columns.tolist())


def feature_engineering(df):
    df = df.copy()
    # BMI
    if 'Weight' in df.columns and 'Height' in df.columns:
        df['BMI'] = df['Weight'] / ((df['Height'] / 100) ** 2)
    else:
        print("Error: Body_Temp or Duration column is missing!")
        df['BMI'] = 0
    # Body_Temp_Duration
    if 'Body_Temp' in df.columns and 'Duration' in df.columns:
        df['Body_Temp_Duration'] = df['Body_Temp'] * df['Duration']
    else:
        print("Error: Body_Temp or Duration column is missing!")
        df['Body_Temp_Duration'] = 0
    # Weight_Heart_Rate
    if 'Weight' in df.columns and 'Heart_Rate' in df.columns:
        df['Weight_Heart_Rate'] = df['Weight'] * df['Heart_Rate']
    else:
        print("Error: Weight or Heart_Rate column is missing!")
        df['Weight_Heart_Rate'] = 0
    # Sex için one-hot encoding
    if 'Sex' in df.columns:
        df = pd.get_dummies(df, columns=['Sex'], drop_first=True, dummy_na=
        df = pd.get_dummies(df, columns=['Sex'], drop_first=True, dummy_na
    else:
        print("Error: Sex column is missing!")
        df['Sex_male'] = 0
    return df


train_fe = feature_engineering(train)
print("Columns after feature engineering:", train_fe.columns.tolist())


if 'Calories' in train_fe.columns:
    X_train = train_fe.drop(columns='Calories')
    y_train = np.log1p(train_fe['Calories'])
else:
    raise ValueError("Error: Calories column missing in train_fe!")


numerical_cols = [col for col in ['Age', 'Height', 'Weight', 'Duration', 'H
                                  'BMI', 'Body_Temp_Duration', 'Weight_Hear
categorical_cols = [col for col in ['Sex_male'] if col in X_train.columns]
print("Numeric columns:", numerical_cols)
print("Categorical columns:", categorical_cols)


preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(drop='first', handle_unknown='ignore'), categ
    ])

lgbm_params = {
    'n_estimators': 1125,
    'num_leaves': 110,
    'min_child_samples': 9,
    'learning_rate': 0.0179455702408711,
    'colsample_bytree': 0.5979737441060009,
    'reg_alpha': 0.001975258376030875,
    'reg_lambda': 0.005106256873241264,
    'max_bin': 2**10,  # log_max_bin=10
    'random_state': 42,
    'verbose': -1
}


pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', LGBMRegressor(**lgbm_params))
])


kf = KFold(n_splits=5, shuffle=True, random_state=42)
oof_preds = np.zeros(X_train.shape[0])

for fold, (train_idx, val_idx) in enumerate(kf.split(X_train)):
    print(f"Fold {fold+1}/5")
    X_train_fold, X_val_fold = X_train.iloc[train_idx], X_train.iloc[val_id
    y_train_fold, y_val_fold = y_train.iloc[train_idx], y_train.iloc[val_id
    pipeline.fit(X_train_fold, y_train_fold)
    oof_preds[val_idx] = pipeline.predict(X_val_fold)

# Validasyon RMSLE
y_pred_orig = np.expm1(oof_preds)
y_pred_orig = np.clip(y_pred_orig, a_min=0, a_max=400)
rmsle = np.sqrt(mean_squared_log_error(train['Calories'], y_pred_orig))
```

```
print(f"Validation RMSLE: {rmsle:.6f}")
```

```
Train data columns: ['Sex', 'Age', 'Height', 'Weight', 'Duration', 'Heart_Rat
e', 'Body_Temp', 'Calories']
Columns after feature engineering: ['Age', 'Height', 'Weight', 'Duration', 'H
eart_Rate', 'Body_Temp', 'Calories', 'BMI', 'Body_Temp_Duration', 'Weight_Hea
rt_Rate', 'Sex_male']
Numeric columns: ['Age', 'Height', 'Weight', 'Duration', 'Heart_Rate', 'Body_
Temp', 'BMI', 'Body_Temp_Duration', 'Weight_Heart_Rate']
Categorical columns: ['Sex_male']
Fold 1/5
Fold 2/5
Fold 3/5
Fold 4/5
Fold 5/5
Validation RMSLE: 0.059882
```

Ensemble Submission Creation

Why and How:
1. We load multiple submission files, each containing predictions from different models or folds.
2. By averaging the predictions across these files, we reduce individual model biases and variance, improving overall prediction robustness.
3. The `id` column is taken from the first submission file assuming all files have the same order and IDs.
4. The averaged predictions are stored in a new DataFrame, which is then saved as a combined submission file.
5. This simple ensemble technique often leads to better performance than using a single model's predictions.

In [8]:

```
df1 = pd.read_csv("/kaggle/input/my-best-sub/submission_1.csv")
df2 = pd.read_csv("/kaggle/input/my-best-sub/submission_2.csv")
df3 = pd.read_csv("/kaggle/input/my-best-sub/submission_3.csv")
df4 = pd.read_csv("/kaggle/input/my-best-sub/submission_4.csv")
df5 = pd.read_csv("/kaggle/input/my-best-sub/submission_5.csv")

ground_truth = pd.read_csv("/kaggle/input/playground-series-s5e5/sample_sub

all_preds = np.stack([df['Calories'] for df in [df1, df2, df3, df4, df5]],
ground_truth['Calories'] = np.median(all_preds, axis=1)
ground_truth.to_csv('submission.csv', index=False)
```

Project Summary & Key Highlights 🚀

In this project, we predicted **Calories burned** using physical and activity data, combining powerful feature engineering with advanced models and ensemble techniques.

Feature Engineering 🛠️

- Created features like *BMI*, *Body_Temp_Duration*, and *Weight_Heart_Rate* to enhance model understanding.
- Applied one-hot encoding to categorical variables for better representation.

AutoML Framework & Models 🤖

We used AutoML to efficiently tune and select models, reducing manual effort and improving performance.

CatBoost 🐱	LightGBM 🌲	XGBoost ⚡
RMSLE: 0.05930	**RMSLE:** 0.05937	**RMSLE:** 0.05925
Train Time: ~45 mins (higher due to complex features and bins)	**Train Time:** ~35 mins (due to max_bin=750 increasing training complexity)	**Train Time:** ~40 mins (more exhaustive training)

Validation Strategy 🔍

- Implemented **5-fold cross-validation** to ensure robust performance estimates.
- Used ensemble averaging to reduce variance and avoid overfitting.

Ensembling 🤝

Combined multiple model predictions by averaging, improving prediction stability and accuracy.

This comprehensive approach ensures a balance of feature richness, model power, and validation rigor — delivering reliable calorie predictions.

What is AutoML ? 🤖✨

AutoML (Automated Machine Learning) automates the process of building, training, and tuning machine learning models, making ML accessible even to non-experts. It streamlines complex steps like data preprocessing, feature engineering, model selection, and hyperparameter tuning.