Exercise 3 - Automated Machine Learning

Clara Pichler
Hannah Knapp
Sibel Toprakkiran

Group 18

Automated Machine Learning

- automating the complex model development process
- using automated tools and processes
- making machine learning more accessible to individuals and organizations with limited expertise in data science

Can include data preparation, feature selection/extraction, model selection and training, hyperparameter tuning, monitoring and maintenance…

Our implementation will only consider model selection and hyperparameter tuning

Setup

- Python 3.12.5 - Jupiter Notebook files
- Testing the algorithms with four dataframes:
  - Congressional Voting Data (Classification)
  - Iris Data (Classification)
  - Airfoil Noise Data (Regression)
  - Abalone Data(Regression)

- Using Git to collaborate and merge our code
- Multiple meetings on discord to discuss our work

Chosen Models

Classifiers

- Multilayer Perceptron
- K-Nearest Neighbour
- Random Forest
- Support Vector Machine
- AdaBoost

Regressors

- Random Forest
- K-Nearest Neighbour
- Gradient Boosting
- Linear Regression
- Lasso Regression

Chosen Hyperparameters

MLP: `["max_iter", "activation", "solver", "alpha"]`

RF: `["n_estimators", "max_depth", "min_samples_split", "max_features", "criterion"]`

KNN: `["n_neighbors", "weights", "leaf_size"]`

SVC: `["C", "kernel", "gamma"]`

AdaBoost: `["n_estimators", "learning_rate"]`

GradientBoosting: `["n_estimators", "learning_rate", "loss"]`

LinearRegression: `['n_jobs']`

Lasso: `["alpha", "max_iter"]`

Simulated Annealing (Part 1)

Initialization

- Set the initial solution to a "dummy" classifier or regressor
- evaluate the initial solution to initialize best score, best solution
- Init starting temperature, maximum iterations, and minimum training time

Main loop

Result

- After the loop, set the best solution as the final model
- Fit the best model
- Print the best model and its score

Simulated Annealing (Part 2)

Main Loop

- Start the timer and track the elapsed time
- generate a new solution in the neighbourhood and evaluate
- possibility of accepting a state with a better outcome is always 1
- calculate possibility of accepting bad values (escape local maximums)
- reduce temperature using cooling rate (gradually decrease the frequency of accepting bad values)

Generating Neighbourhood

purpose -> generate a new candidate solution

1. modify the current solution
    a. no parameters -> select a new algorithm
    b. append randomly selected parameters to the new solution list
    c. randomly select a parameter index and replace with randomly selected value
2. or select a completely new algorithm with a probability of 10%

Creating a Model

purpose -> instantiate the appropriate machine learning algorithm

- extract the name and hyperparameters from the function parameter
- determine, if classification or regression
- for the chosen model call the corresponding class with correct hyperparameters

Objective Function (evaluation function)

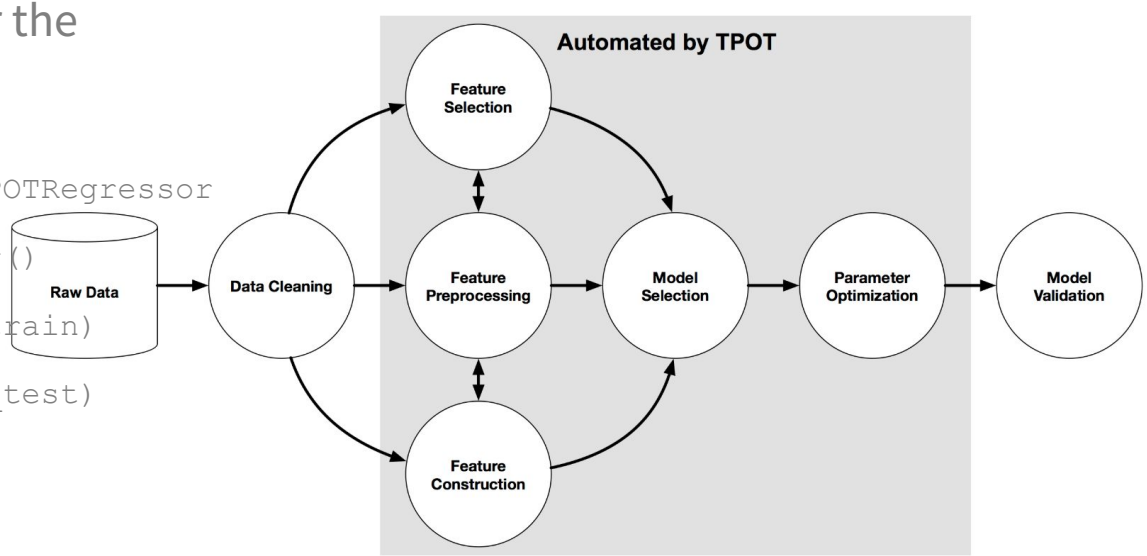
- fitting the model to the training data and making prediction
- using validation set and holdout method so **data leakage can be avoided**
- calculating **accuracy** for classification and **negative MSE** for regression

=> Maximization Problem

TPOT

- a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming
- exploring thousands of possible pipelines
- provides the Python code for the best pipeline

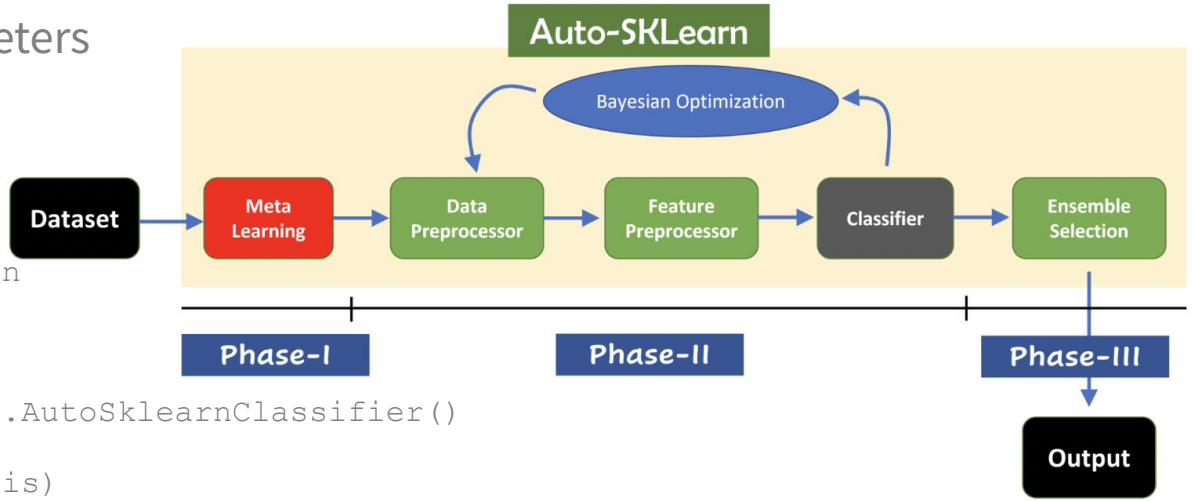
```
from tpot import TPOTClassifier, TPOTRegressor

pipeline_optimizer = TPOTClassifier()

pipeline_optimizer.fit(X_train, y_train)

pipeline_optimizer.score(X_test, y_test)
```



An example machine learning pipeline

auto sklearn

- Auto-sklearn provides out-of-the-box supervised machine learning
- Built around the scikit-learn machine learning library
- auto-sklearn automatically searches for the right learning algorithm and optimizes its hyperparameters




```
import autosklearn.classification

import autosklearn.regression

cls = autosklearn.classification.AutoSklearnClassifier()

cls.fit(X_train_iris, y_train_iris)

predictions = cls.predict(X_test_iris)
```

Differences TPOT/Auto Sklearn and Our Algorithm

- Auto Sklearn applies Bayesian optimization to find the best combinations and allows for automatic ensemble construction
- TPOT uses a pipeline structure, optimization of feature preprocessing, feature selection, and classifiers/regressors + hyperparameters
- Our model can only use few predefined classifiers/regressors and cannot do preprocessing/feature selection as part of the optimization

Training

- Training our AutoML Algorithm for at least an hour with parameter `min_training_time`
- Setting an initial Temperature of 100 and using a cooling rate of 0.99
- End the loop if we get over our minimal training time or when max iteration 100 is reached
- Give back model with the best score -> highest accuracy score (classification), lowest mean squared error (regression)

Evaluation AutoML_18

- **Iris**: `KNClassifier(n_neighbour=9, weights=distance, leaf_size=20)` with a final score of 0.9778

  Accuracy on test set: 0.9778

```
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        18
  versicolor       1.00      0.91      0.95        11
   virginica       0.94      1.00      0.97        16

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45
```

- **Voting**: `MLPClassifier(max_iter=1000, activation=logistic, solver=adam, alpha=0.0001)` with a final score of 1.0

  Accuracy on test set: 0.9545

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.95      0.96        40
         1.0       0.93      0.96      0.94        26

    accuracy                           0.95        66
   macro avg       0.95      0.96      0.95        66
weighted avg       0.96      0.95      0.95        66
```

Evaluation AutoML_18

- **Airfoil**: `RandomForestRegressor(n_estimators=25, max_depth=15, min_samples_split=2, max_features=None, criterion=absolute_error)` with a final score of -4.8266

  Evaluating on the test data:

  - Test MSE: 5.1866
  - Test RMSE: 2.2774
  - Test MAE: 1.6701
  - Test R^2: 0.8964

- **Abalone**: `RandomForestRegressor(n_estimators=100, max_depth=10, min_samples_split=3, max_features=None, criterion=squared_error)` with a final score of -4.8067

  Evaluating on the test data:

  - Test MSE: 4.9738
  - Test RMSE: 2.2302
  - Test MAE: 1.5996
  - Test R^2: 0.5291

Evaluation TPOT

- **Iris**: `LogisticRegression(input_matrix, C=15.0, dual=False, penalty=l2)` with a final score of 1.0

  Accuracy on test set: 0.9714

- **Voting**: `GradientBoostingClassifier(input_matrix, learning_rate=0.1, max_depth=10, max_features=0.45, min_samples_leaf=6, min_samples_split=14, n_estimators=100, subsample=0.8)` with a final score of 0.9846

  Accuracy on test set: 0.9542

Evaluation TPOT

- **Airfoil**: `GradientBoostingRegressor(PolynomialFeatures(input_matrix, degree=2, include_bias=False, interaction_only=False), alpha=0.95, learning_rate=0.1, loss=huber, max_depth=7, max_features=0.8, min_samples_leaf=2, min_samples_split=6, n_estimators=100, subsample=0.7000000000000001)` with a final score of -5.4071

  MSE on test set: -5.4541

- **Abalone**: `RandomForestRegressor(RidgeCV(PCA(input_matrix, iterated_power=9, svd_solver=randomized)), bootstrap=False, max_features=0.45, min_samples_leaf=12, min_samples_split=9, n_estimators=100)` with a final score of -4.0099

  MSE on test set: -4.6168

Evaluation AutoSklearn

- We had some difficulties to run AutoSklearn
- Did some research and used docker to run it with linux
- got some output we tried to interpret but it didn’t run for all models
- only got output for iris dataset:

Accuracy score 1.0, {9: {'model_id': 9, 'rank': 1, 'cost': 0.0, 'ensemble_weight': 0.02, 'data_preprocessor': <autosklearn.pipeline.components.data_preprocessing.DataPreprocessorChoice object at 0x7f98945a8eb0>, 'balancing': Balancing(random_state=1, strategy='weighting'), 'feature_preprocessor': <autosklearn.pipeline.components.feature_preprocessing.FeaturePreprocessorChoice object at 0x7f98942c3910>, 'classifier': <autosklearn.pipeline.components.classification.ClassifierChoice object at 0x7f98942c3970>, 'sklearn_classifier': LinearDiscriminantAnalysis(shrinkage='auto', solver='lsqr', tol=0.00010000000000000009)

How to run auto-sklearn in docker

download: `docker pull mfeurer/auto-sklearn:master`
(https://automl.github.io/auto-sklearn/master/installation.html)

get docker id: `docker ps`

copy datasets/files to docker: `docker cp AutoSklearn.py <container_id>:/AutoSklearn.py`

`docker cp airfoil_noise_data.csv <container_id>:/airfoil_noise_data.csv`

`docker cp CongressionalVotingID.shuf.lrn.csv <container_id>:CongressionalVotingID.shuf.lrn.csv`

`docker cp abalone.csv <container_id>:/abalone.csv`

Install python in docker: `docker run -it mfeurer/auto-sklearn:master` then `pip install python3`

To run our auto sklearn python script: `docker exec -it <container_id> bash`

`python3 /AutoSklearn.py`

Results

	AutoML_18	TPOT
Iris (accuracy)	97%	97%
Voting (accuracy)	95%	95%
Airfoil (mse)	5.18	5.45
Abalone (mse)	4.97	4.61

Conclusions

- For the smaller data sets our AutoML class works quite well also in comparison to TPOT
- For the complexer data sets we still get good results but not as good as TPOT
- The class could be expanded by adding optimization to find the best preprocessing and feature selection for the data sets
- Comparison with AutoSklearn was difficult because we didn’t get all the results for all data sets