AutoML Comparsion

Machine Learning Exercise 2

Paul Erpenstein
Mihai Truta
Jasmin Wolff

Content

1. Datasets
    a. Beijing PM2.5
    b. Fifa 18
    c. Crab Age
2. Regression Algorithms
    a. Multilayer Perceptron
    b. Elastic Net
    c. Ridge Regression
3. Own AutoML Function
4. State of the Art Algorithms
    a. AutoMl by sklearn
    b. TPOT
5. Experiment Design
6. Result/Lessons Learned

Data Sets

Beijing PM2.5

Data Set connects hourly values of particulate matter (pm2.5) to weather information of Beijing from Jan 1st, 2010 to Dec 31st, 2014.

Facts

- Number of Variables: 8
  Date, Hour, PM2.5, Weather Data, e.g. Temperature, Wind Speed, Wind Direction
- Number of observations: 43824
- Data Types: categorical(1), numeric(11)
- Missing Values: yes, 2067 values
  Marked as â€œNAâ€.

➔ Goal: Predict **PM2.5** with weather data

Preprocessing

- Dropping a column containing IDs (“No”)
- 1 to N Hot-Coding for categorical data combined wind direction (“cbw”)
  4 new columns
- Missing values in only in output variable PM2.5
  Imputation would affect model training and evaluation metrics, so all 2067 rows got deleted

http://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data

Fifa 2018

Characteristics of all available Fifa Players of Fifa 18 including physical attributes (e.g.sprint speed, accuracy, aggression, etc.), position strength (e.g. RM, RS, RW, RWB, etc.), and personal information (e.g. value, wage, club, age, etc.).

Facts

- Number of Variables: 73
- Number of observations: 17981
- Data Types: categorical(9), numeric(64)
- Missing Values: yes, 53002 values
  Marked as NaN.

➔ Goal: Predict player **value** using physical attributes including age

Preprocessing

- Investigate missing values:
  27 rows are affected. Action: plot correlation matrix, discover that all position strength attributes are missing at once and delete all 2029 rows
- Reduce number of columns:
  Name, Photo, Nationality, Flag, Club, Club Logo are deleted.
- One hot encoded preferred positions
- Transform numeric data to maximum values:
  players have standard strength and maximum strength which is displayed as 70+9

https://www.kaggle.com/edith2021/fifa-18-player-prediction

Crab Age

Data set contains physical attributes of crabs from a crab farm business. It was collected to use Regression to optimize the timing of harvesting and reduce costs of farming by decreasing supply times.

Facts

- Number of Variables: 9
  Length, Diameter,Height, Weight, Shucked Weight, Viscera Weight, Shell Weight, Age
- Number of observations: 3893
- Data Types: categorical(1), numeric(8)
- Missing Values: no

➔ Goal: Predict crab **age** with body information

Preprocessing

- Perform 1 to N Hot-Coding for column Sex

https://www.kaggle.com/sidhus/crab-age-prediction

Regression Algorithms

Criterias
Why did we choose the ML solutions we did?

- Ridge Regression: basic and stable regression algo with few hyperparameter
- ElasticNet: more advanced than Ridge which could probably lead to better results
- Multilayer Perceptron: complex, many tuning parameters and great success in exercise Classification

We chose the solutions in order to get **differing amounts of tuning parameters**.

Ridge was great to develop out the Auto-ML, while MLP was good to test it.

Ridge

Hyperparameters: alpha (regularization strength),solver(auto, sag, lbfgs, etc.) or max_iter(maximum of iterations). Few more available.
Complexity: rather simple

Only parameter alpha was used to establish AutoML function due to ease of implementation and default of solver is auto which uses the best fitting computational routine as well as none maximum of iterations

Great candidate to develop simulated annealing because complexity is kept low

Elastic Net

Hyperparameters: alpha (multiplier of penalty terms, 0 similar to OLS), l1_ratio (1= L1 penalty, 0= L2 penalty, between= combination), max_iter(max of iterations), positive (forces coefficients to be positive), and selection (which coefficient is updated every iteration= random or looping sequence). Few more available.

Complexity: rather complex

Alpha and l1_ratio was used for building AutoML function because default for other parameters are reasonably set. Very similar to Ridge but different enough to make for an interesting comparison.

Great possibility to build up complexity of regression models and to compare results to Ridge

Multilayer Perceptron

Hyperparameter: alpha (regularization term/L2 penalty), hidden_layer_sizes(number of neurons in the ith layer), solver (weight optimization), learning_rate (rate scheduler for weight updates), learning_rate_init (initial learning rate), max_iter (max iterations), tol (tolerance for optimization). Many more specific parameters depending on solver available.
Complexity: very complex

Three tuning parameters used for AutoML: alpha, number of neurons first hidden layer, number of neurons second hidden layer

We chose to use **only two hidden layer** MLPs:

- "number of hidden layers" has a strong relationship with the number of neurons on each layer → We tried to keep the parameters as independent as possible
- Even the smallest possible step size of 1 would lead to a wildly different model, 1 to 2 hidden layer MLPs can behave very differently
- 1 and 2 hidden layers MLPs could be probably used as separate solutions and evaluated against each other

Our AutoML Function

General Facts
Requirements

Idea: Automated Hyperparameter search manually implemented

Implementation:

- in Python
- uses simulated annealing
- covers three different machine learning algorithms

Requirement: Must at least run for one hour

Simulated Annealing
Breakdown

Simulated annealing is a derivative of **stochastic hillclimbing**:

- Imitates the process of metal cooling down → concept of temperature
- Accepts solutions that might not be the best, in order not to get stuck in local optimas rather than finding the global one

Temperature:

- Decreases over the runtime
- Makes it possibly that a “suboptimal” solution is accepted in order to traverse more of the parameter space and find the global optimum

Design decisions
General

Requirement: Runtime at least 1h per data set and each AutoML system

To ensure our AutoML solution runs at least an hour we implemented a parameters called “runtime_seconds”:

- Specifies how long AutoML runs, comparable to max_time_mins of state of the art algorithms
- After time has run out the function terminates and returns the best model at that point

Design decisions
Temperature

How we implemented simulated annealing:

- Temperature is always in interval [1,0]
- Initial temperature is 1 and temperature approaches 0
- A temperature of 0 means that the program terminates
- Temperature is computed in a special way

Actual Computation of temperature:

- Many different possible solutions
  f.e. “fast simulated annealing”: temperature = initial_temperature / (iteration_number + 1)
- Our solution
  temperature = 1 - (time_since_start/runtime_seconds)
  → Temperature linearly goes down during the runtime

Another special trait of our solution:

- Step size of the parameters is influenced by temperature
  → In the beginning: Entire parameter space is accessible
  → In the end: Very fine optimization once the global optimum is very close
- Formula for step regarding parameter i:
  step_size = temperature * (upper_bound_i - lower_bound_i)

Our Auto-ML function
Design decisions

How we implemented simulated annealing:

- Temperature is always in interval [1,0]
- Initial temperature is 1 and temperature approaches 0
- A temperature of 0 means that the program terminates
- Temperature is computed in a special way

Our Auto-ML - Design decisions

Computation of temperature:

- Many different possible solutions

  → f.e. “fast simulated annealing”: temperature = initial_temperature / (iteration_number + 1)

- Our solution: temperature = 1 - (time_since_start/runtime_seconds)

  → Temperature linearly goes down during the runtime

Our Auto-ML - Design decisions

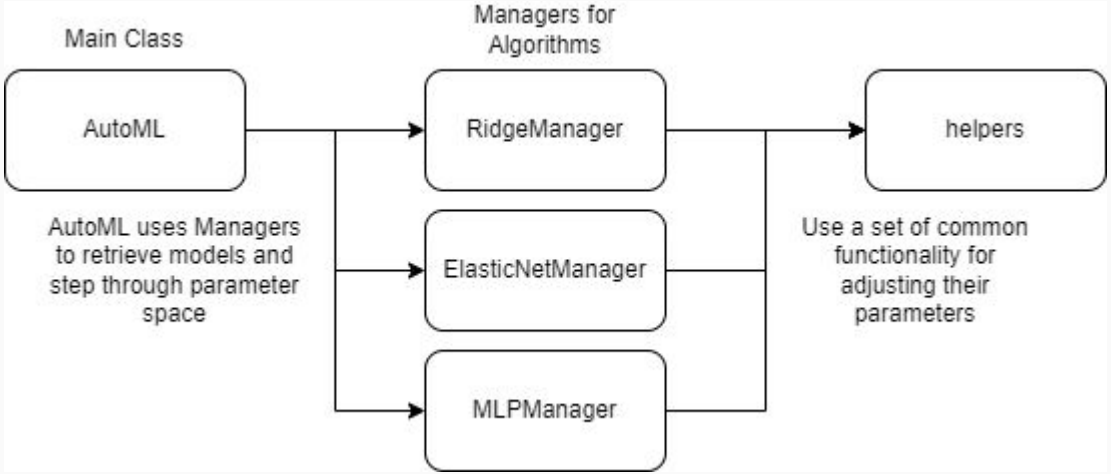
Another special trait of our solution:

- Step size of the parameters is influenced by temperature

  → In the beginning: Entire parameter space is accessible
  → In the end: Very fine optimization once global optimum is hopefully found

- Formula for step regarding parameter i: step_size = temperature * (upper_bound_i - lower_bound_i)

High level architecture
Operating Principle



- Main class handles simulated annealing and performance analysis through cross validation
- Managers handle algorithm specific tasks like parameter stepping and model initialization

→ **“Extensible system” more managers are possible in the future**

Quality Control
Overview

Problem:

- Bugs are a big concern, with systems that run for a long time
  → One bug can waste an hour or more if it happens at the end of execution
- How do we make sure there aren't bugs, that keep the system from doing its job
  → We need a way to observe the function while its working

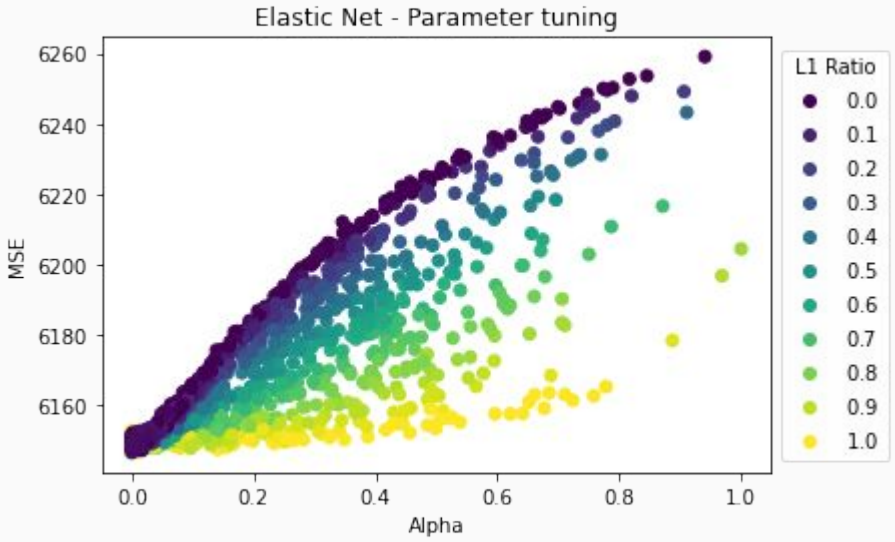
Solution:

- Implemented CSV-"logging"
- During parameter tuning, each Manager writes its own CSV file detailing what happened during the tuning process

→ Visualize data to explore what actually happens and analyse the number of iterations of all used algorithms

→ More details in the README.md (Also contained in submission package)

Quality Control
Elastic Net Parameters

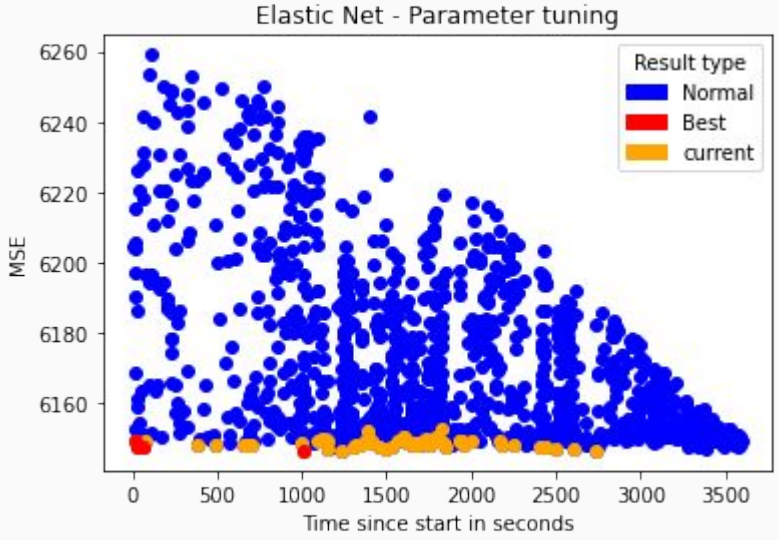


ElasticNet parameters on the Beijing PM2.5 dataset:

- ElasticNet has its best test set results with an alpha of 0
- However if the L1 penalty is used the results are very comparable

This example concerns the beijing data set.

Quality Control
Elastic Net Runtime

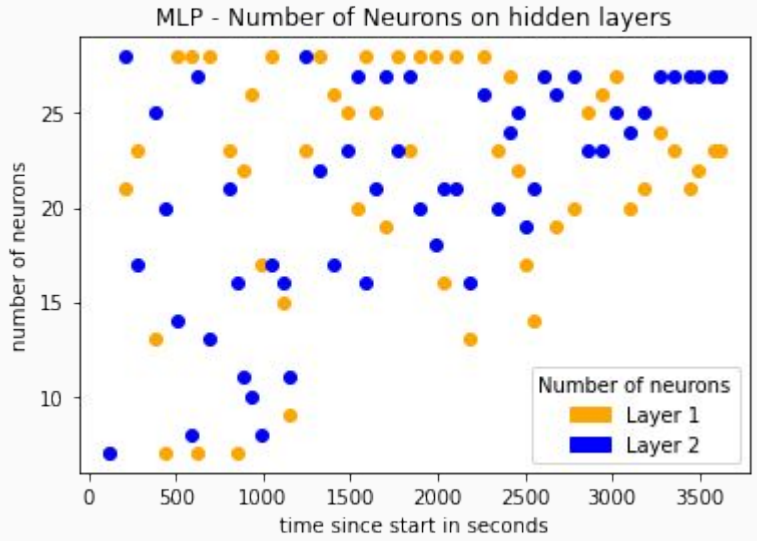


Best result was found at about a third of the maximum runtime (Beijing PM2.5):

- ElasticNet does not seem to need such a long time frame for it to optimize
- Maybe a shorter timeframe would be more appropriate

.

Quality Control
Multilayer Perceptron Runtime



MLP number of neurons for the Beijing PM2.5 Data Set:

- conversion at the end of runtime
- Entire parameters space seems to be covered → However not very densely
- **much less** results than for ElasticNet!

Quality Control
Comparison of used regression algorithms

Machine learning algo.	Multilayer Perceptron	ElasticNet	Ridge
Number of iterations	52	1263	7206

Numbers only concern the beijing data set.

Numbers of iteration **vary greatly**
→ **Starving occurs**: Jobs are running in parallel. Ridge and ElasticNet work so quickly that they take away processing power from MLP which leads to less number of iterations.
→ Number of iterations for Elastic and Ridge are most-likely **over-excessive**

Solution: Limit the number of iterations and use fast simulated annealing
This was sadly not possible, since we ran out of time.

State of the Art AutoML

Overview of State of the Art

AutoML refers to automate **some or all steps** of building ML models. Multiple algorithms are considered and hyperparameter tuning is performed to find top-performing model. In the following, 5 of the **most popular** are summarized.

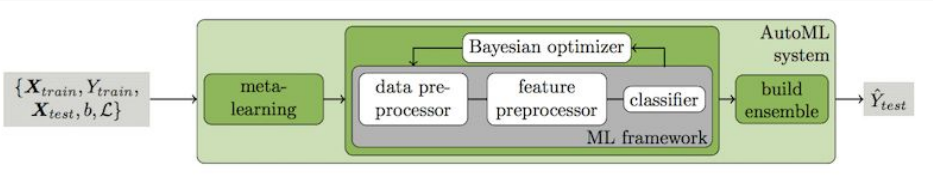
01	**Auto-SKLearn**	• chosen for Exercise • open source founded by researchers • supports Python
02	**TPOT**	• chosen for Exercise • open source founded by researchers • supports Python
03	**H2O AutoML**	• open source by H20.ai • supports R and Python • includes functions as automatic visualization and interpretation
04	**Google Cloud AutoML**	• paid service by Google • No Code environment • includes functions for explanation ,e.g feature importance values.
05	**Auto-Keras**	• open source by DATA LAB • supports Python • follows SciKit-Learn API design • includes functions for hyperparameter tuning

Auto-sklearn

Auto-sklearn is build in a open source library developed by Matthias Feurer, et al. and described in their 2015 paper titled "Efficient and Robust Automated Machine Learning." It does combined Algorithm Selection and Hyperparameter optimization.

Facts:

- uses Scikit-Learn for data transforms and machine learning algorithms
- Bayesian Optimization search procedure for estimating top-performing model pipeline
- not optimal for large data sets
- Following the depiction of their system, taken from their paper:



Important default Parameters for AutoSklearnRegression:

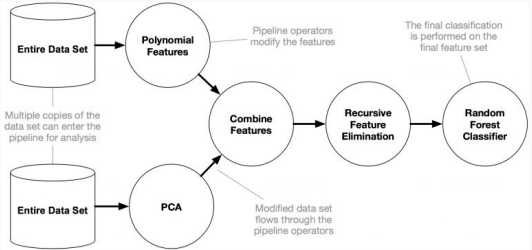
- time_left_for_this_task=3600 given in seconds
- include_estimators = None means all features are used
- exclude_estimators=None means no features are excluded
- metric=None will select best based on task

TPOT (Tree-Based Pipeline Optimization Tool)

(TPOT) was developed by Dr. Randal Olson (as postdoctoral student) and Dr. Jason H. Moore around 2016 and is still extended and supported by their team. Early publications about TPOT won multiple awards for best paper.

Facts:

- uses Scikit-Learn for data transforms and machine learning algorithms
- Statistical search procedures, e.g. genetic programming, combined with flexible expression trees used for estimating top-performing model pipeline
- Following the depiction of their system:



Important default Parameters for tpotRegressor:

- max_time_mins = None; changed to 60 minutes
- generations= 100; means 100 iterations are done; changed to None because max_time_mins has been set
- population_size= 100; number of individuals to retain in genetic programming population every gen.
- scoring = negative MSE to evaluate the quality of a given pipeline
- cv= 5; CV strategy used when evaluating pipelines; changed to own CV with 10 kfold and 5 jobs

Experiment Design

Experiment Steps

Pre-Processing	Model Fit	Prediction and Evaluation
Investigate which preprocessing methods are required for each data set. - Missing Values - Scaling - Imbalanced Data - Transforming Values - 1 to N Hot-Coding	Run three AutoMl Algorithms for each data set at least 1 hour	Use best performing pipeline to predict outcome variable of test set and evaluate result using MSE

Results

Test set performance comparison

Runtime: 3600 seconds / 60 min
Evaluation Metrics: MSE from sklearn mean_squared_error().
Lower value represents predictions are close to actual value.

Name	Performance Beijing PM2.5 // Model	Performance Crabs // Model	Performance FIFA 18 // Model
Our Auto-ML	ca. 5324.55 (MLP)	ca. 4.7 (MLP)	682,885,074,507,187.6 (MLP)
auto-sklearn	ca. 1.8e-11 (Ensemble solution)	ca. 4.8 (Ensemble solution)	ca. 71,987,844,336,480.5 (Ensemble solution)
TPOT	ca. 8.5e-28 (LassoLarsCV)	ca. 4.5 (Ensemble: PCA, StackingEstimator, RandomForestRegressor)	174,617,172,439,499.38 (extraTreesRegressor)

Test set performance analysis

- Our solution highly favors MLP even though it has the least number of iterations for optimizing its parameters
  → Not ideal: Next iteration of program would absolutely need to increase the number of iterations for MLP
- Big performance discrepancy for the Beijing data set
  → Most likely fixable by introducing more machine learning algorithms in the form of managers
- Sadly the output of autosklearn is really unintuitive and convoluted
  → Comparison possible only in terms of performance measures
- Overall really bad performance for FIFA data set

AutoML Comparison
Overall

Documentation: informative and understable documentation available for all

Preprocessing: done manually, TPOT also includes preprocessing steps in its pipelines like PCA or Scaling

Handling: own solution and TPOT easy to handle, auto-sklearn only available for Linux operating systems

Performance: 2 of 3 best results was achieved by TPOT. Own Solution worked comparable on small data sets with less variables, but struggles with larger data sets, esp. with high number of variables

Lessons Learned

Summary

- joblib is great when training for a long time and wanting to quickly retrieve the results again
- auto-sklearn runs on Windows when you install it on Windows-Subsystem for Linux
- TPOT is a great choice for AutoML, since you get a pipeline exported to train the model, that is very easy to read and understand
- Interesting to encounter the phenomenon of process starvation in a practical example (MLPs having so many less executions)

On bucket list: Try out solutions to auto-preprocess data sets!

Help

For more information about the structure of our submission package please look inside the README.md file.