Machine Learning Assignment 2

Group 1
Jonas Kompauer, 11776872
Lukasz Sobocinski, 12123563
Florian Lackner, 11704916

Dataset - Medical Cost Personal

- Small Dataset: 1338 rows
- 3 numeric features and 3 nominal features
- No missing values
- Preprocessing:
  - Z-score standardisation for numerical features
  - One-hot encoding “region” feature
  - “Sex” and “smoker” both contain only 2 categories -> transform into boolean features with 0 and 1

Dataset - Paris Housing Prices

- Big Dataset: 10000 rows
- 14 numeric features and 2 nominal features
- No missing values
- Preprocessing:
  - Z-score standardisation for numerical features
  - One-hot encoding for “cityCode” and “made”

Dataset - Financial Indicators of US Stock

- We only used the “2018_Financial_Data.csv”
- Small Dataset: 4392 rows
- 222 numeric features and 1 nominal features
- Missing values as NaN
- Preprocessing:
  - One-hot encoding for “sector”
  - Impute missing values with mean value of column
  - Z-score standardisation for numerical features

Regression Algorithms

- Random Forest Regressor
  - 6 hyperparameters to optimize
    - “max_depth”, “min_sample_split”, etc.
- LinearSVR
  - 5 hyperparameters to optimize
    - “epsilon”, “loss”, etc.
- SGDRegressor
  - 8 hyperparameters to optimize
    - “loss”, “alpha”, etc.

Full list of Hyperparameters can be found in the code

Implementation

- We use a Simulated Annealing variant
- Each iteration all 3 Regression algorithms generate new parameters from their Neighbourhood
- The Neighbourhood are values which are close to the parameter
- The best solution is updated if the new solution is better, but there is a small probability that we take the new solution if its worse
- This probability gets lower after more iterations
- The scoring is based on the mean squared error (MSE)

Implementation - Neighbourhood

- Neighbourhood is designed to get similar Values for each parameter in every iteration
- Parameter Ranges are lists of values
- We construct our neighbourhood for a parameter by taking X values before and after the current parameter in the list
- X was tested with values 1, 2 and 3; The best results where with X=2
- The neighborhood of “3” in [1,2,3,4,5,6] would then be [1,2,3,4,5]

Implementation - Pseudocode

- The next Slide shows the pseudocode for the basics of our algorithm
- Note that a Solution always consists of each of the three Regression algorithm combined with a specific parameter selection
- The score of a solution is the MSE of the algorithm with the lowest MSE

Implementation - Pseudocode

```
T = T_init
x = initialSolutions()
best = x
P = 0.97
while Time Limit not reached:
        while no Changes happened in the last 10 iterations
                newSolutions = x.getNeighbors()
                if newSolutions < x:
                        best = newSolutions
                        x = newSolutions
                else:
                        if (P < e**(-abs(newSolutions - x) / T)):
                                x = newSolutions
        T = updateTemperature(T)
```

Implementation - Lessons Learned

- Finding a good neighbourhood structure was not easy
- Tweaking of simulated annealing parameters took some time
  - could also be meta learned
- We had problems that simulated annealing did not try every ML algorithm so we came up with the solution to choose hyperparameters for every ML algorithm in every iteration, to get a better variation.
- Finding good parameter ranges took some time

State of the art approaches

After implementing the algorithm, we wanted to test it. To do it, we used two state of the art AutoML approaches: Auto-Sklearn and Tpot. Comparing our algorithm with them allowed us to see how it performs in general and on what datasets it has the best results.

Running these algorithms is a task on its own. They have some parameters that need to be set up. Also, they provide results in different formats, so we had to parse them for each algorithm separately to visualize and analyse the outcome.

State of the Art Approach - Auto Sklearn

- It doesn’t support custom ranges
- Can include a data/feature preprocessor, but was not used by us as we preprocess our data before.
- Performance measured with MSE.
- It needs a lot of memory, significantly more than two other algorithms.
- It has meta-learning implemented which helps the algorithm to arrive at good results faster.
- The holdout method with train size = 0.67 is used here as the resampling method. We decided not to change it for Cross-Validation to observe what is the difference between these two ideas.
- It evaluates if models that it creates are better than Dummy Regressor to assess if they are useful at all.
- It cannot be installed on Windows in a simple way.

State of the Art Approach - Tpot

- Performance measured with negative MSE. Negative MSE is used because Tpot logic is always trying to maximize the performance metric.
- Cross-validation with 5 folds was used there, so it is the same as one we used in Simulated Annealing.
- Almost everything can be configured in the algorithm, which is not the case in auto-sklearn, where ranges can’t be changed easily.
- In some articles that we have learnt, it is stated that it can be pretty slow

Experimental setup

1. Data preparation

We preprocessed datasets and disabled searching for the best preprocessing techniques in Tpot and Auto-Sklearn. We decided to do it to make the comparison of the algorithms easier.

2. Data split

We decided to split our data into two parts:

- The first split was used by the algorithms to get the optimal hyperparameters (so it contained train+validation).
- The second split (20% of the data) was used to evaluate the final model (so acted as the test data). We decided that we need it because the first set was already used for hyperparameter tuning.

3. Running the experiment

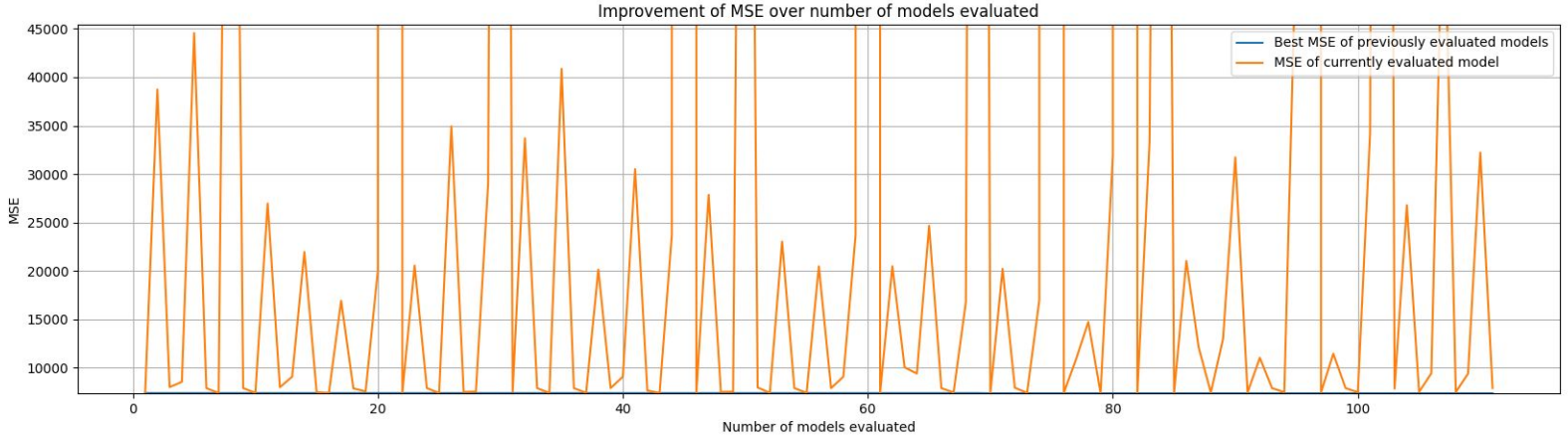
- We ran all three different approaches for each dataset for 1 hour on the same machine.
- We saved the best algorithm for each dataset with its best parameters.
- As the result, we output the MSE and the R^2 of the best algorithm/parameters combination.
- We also saved how does best MSE found by the algorithm changes over time.

Results agenda

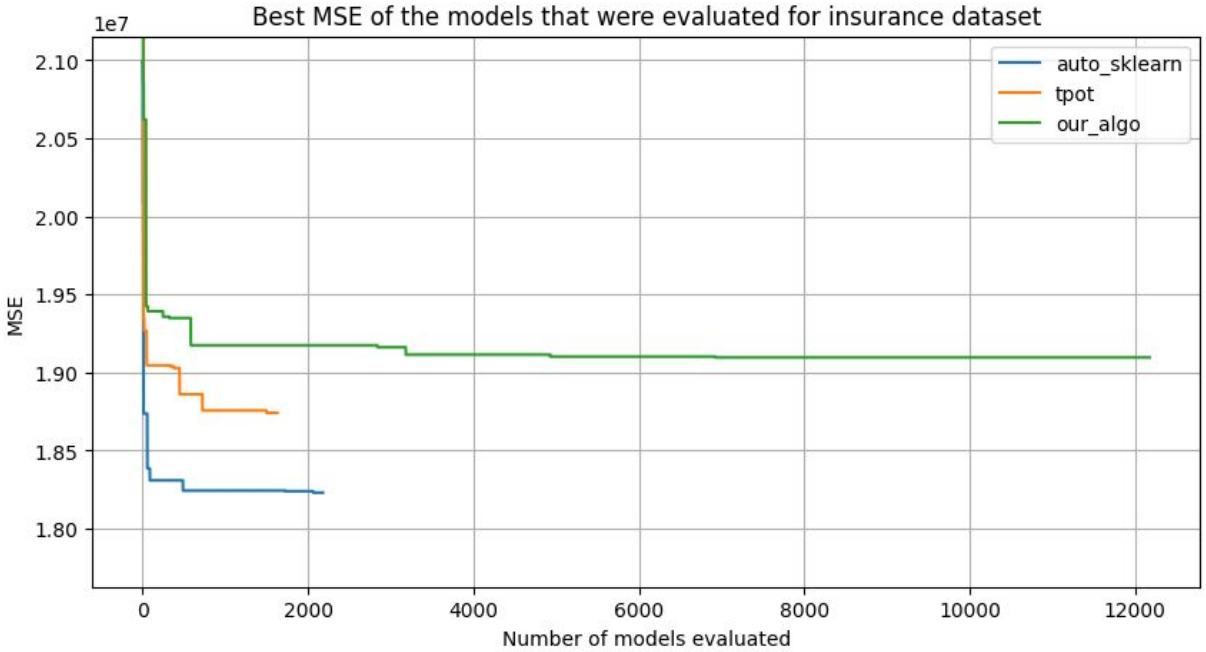
The next step was to analyse the results. To get more insights into them, we decided to focus on the following outcomes:

1. Plots of how does the model's MSE calculated by the algorithm change over time for each algorithm and dataset.
2. Comparison of train and test data MSE for each model.
3. Comparison of train and test data R^2 for each model.

MSE of currently evaluated model

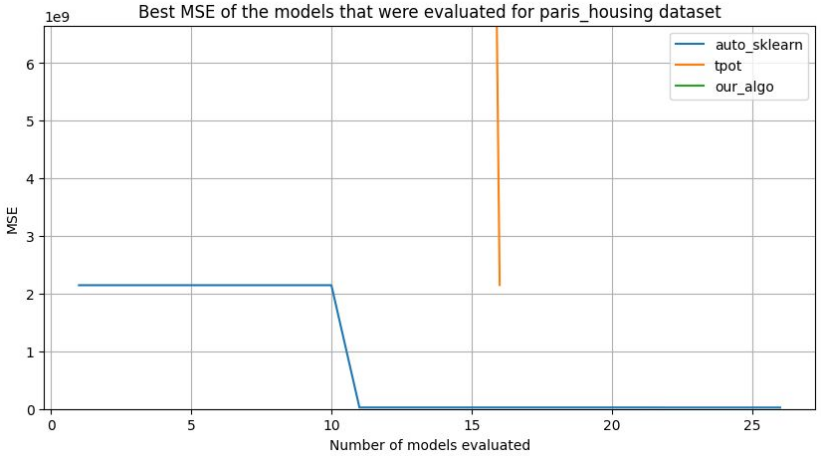


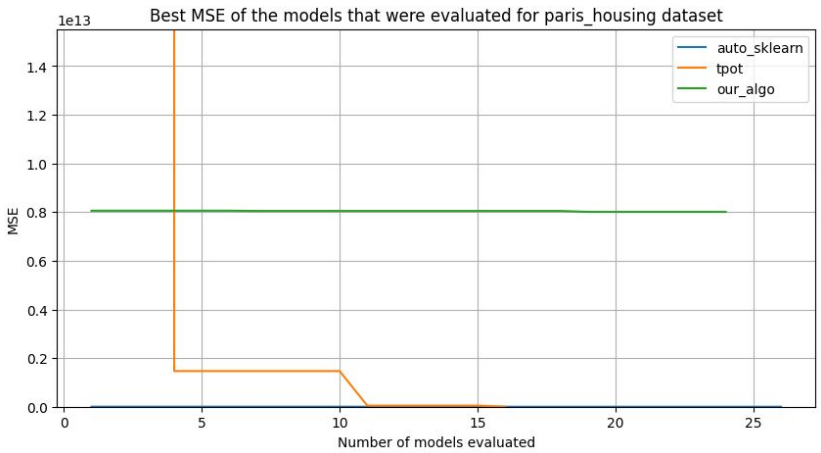
First of all, we plotted how does MSE look for each model that was evaluated by the algorithm. Unfortunately, from these graphs, we couldn't really get a good insight into how does the MSE of the best model found by the algorithm changes. However, we could observe that algorithms are analysing many different models, some of which are really bad.



For the insurance dataset, we could observe that algorithms have a similar “learning curve”. The shape of it is very similar in each case. They started from some bad model, improved quickly in the beginning and then after some time, they arrived at some solution that can’t be beaten easily. However, curves start from different points. It might be caused by different resampling methods used by the algorithm and different splits made inside them. We will see it in the next part of the results: in the end, the error of each algorithm on the whole training+validation set was very similar in each case.

We can also see that our algorithm evaluated much more models than others, which can be a good sign.

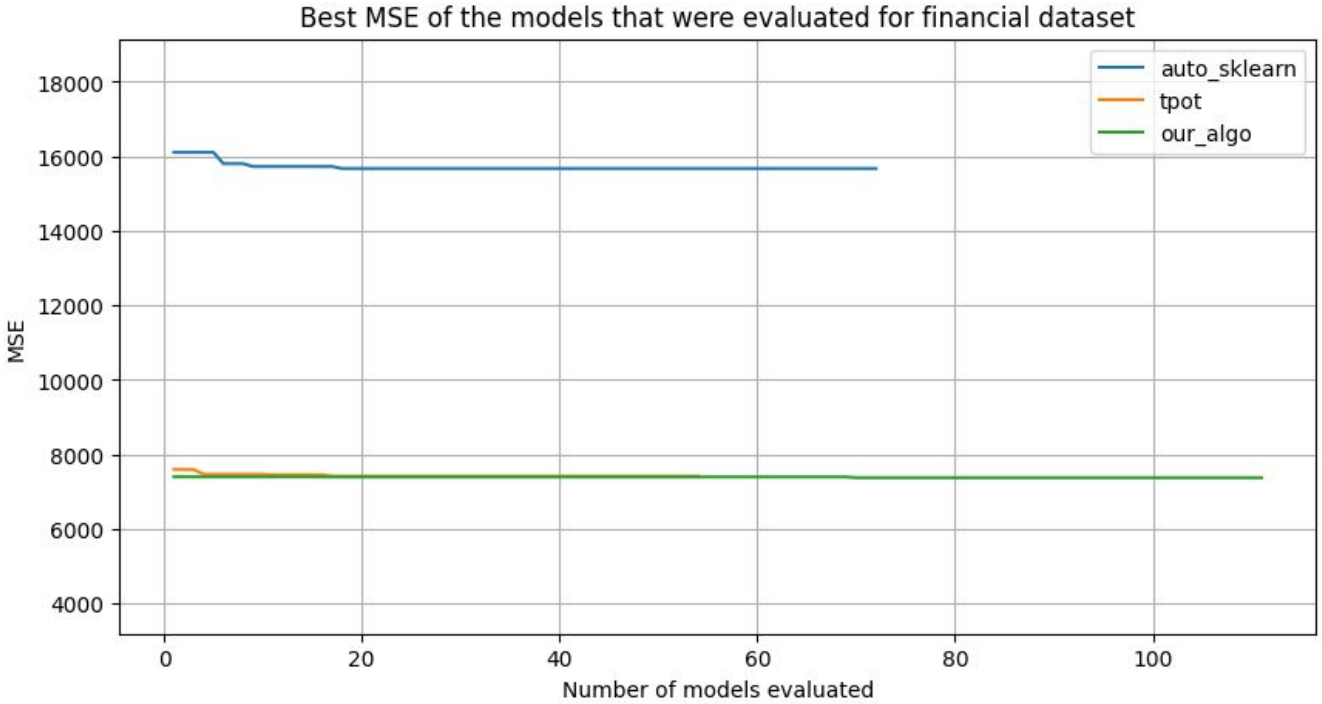




For the Paris Housing dataset, we decided to include two plots with different y-axis limits because MSE for each algorithm is much different.

The result of our algorithm was very bad, it couldn't improve after the first optimal hyperparameters. Also, Tpot struggled in the beginning; however, then it has managed to find much better parameters. Auto-sklearn performed very well, maybe because of its good meta-learning capabilities.

Maybe the bad performance of algorithms was caused by the big number of features in the algorithm.



For the Financial Indicators dataset, we can see that the first configuration found by algorithms was in every case almost the best one. The results don't improve over time that much as in the case of the Insurance dataset. Also, Tpot and our algorithm seem to have quite a similar approach. It is satisfying as it might mean that we are close to the state of the art approach.

Results - MSE

	Medical Cost Personal		Paris Housing Prices		Financial Indicators	
	Training Data	**Test Data**	**Training Data**	**Test Data**	**Training Data**	**Test Data**
Auto Sklearn	16 374 348	25 278 442	14 814 281	16 001 654 158	6 151	5 970
Tpot	14 794 747	25 930 551	892 226 344	16 785 709 006	5 287	15 903
Our Algorithm	14 906 028	25 381 165	8 162 471 254 448	8 039 413 156 428	7 909	4 653

Results - MSE

- The Table on the slide before shows for each dataset and approach the best MSE on the training set and how it performed on the testing set.
- For almost ever approach and dataset the MSE on the test set is worse, especially for the auto sklearn and Tpot for the “paris housing” dataset
- We see that our approach worked quite well for “Medical Cost Personal” and “Financial Indicators” dataset, in the second case better than state of the art approaches
- Our approach has really bad results on “paris housing” both on train and test set.
- We see overfitting for auto sklearn for the “Paris Housing Prices” Dataset, but also tpot overfits at the “Financial Indicators” dataset.
- For our algorithm on the “Financial indicators” we got a quite “bad” result on the train set but a good result on the test set, which could indicate that our algorithm does not overfit that much than other algorithms.

Results - R^2

	Medical Cost Personal		Paris Housing Prices		Financial Indicators	
	Training Data	**Test Data**	**Training Data**	**Test Data**	**Training Data**	**Test Data**
Auto Sklearn	0.8899	0.8160	0.9999	0.9980	0.1790	-0.4381
Tpot	0.9005	0.8153	0.9998	0.9979	0.2943	-2.8308
Our Algorithm	0.8997	0.8113	0.0171	0.0150	-0.0555	-0.1210

Results - R^2

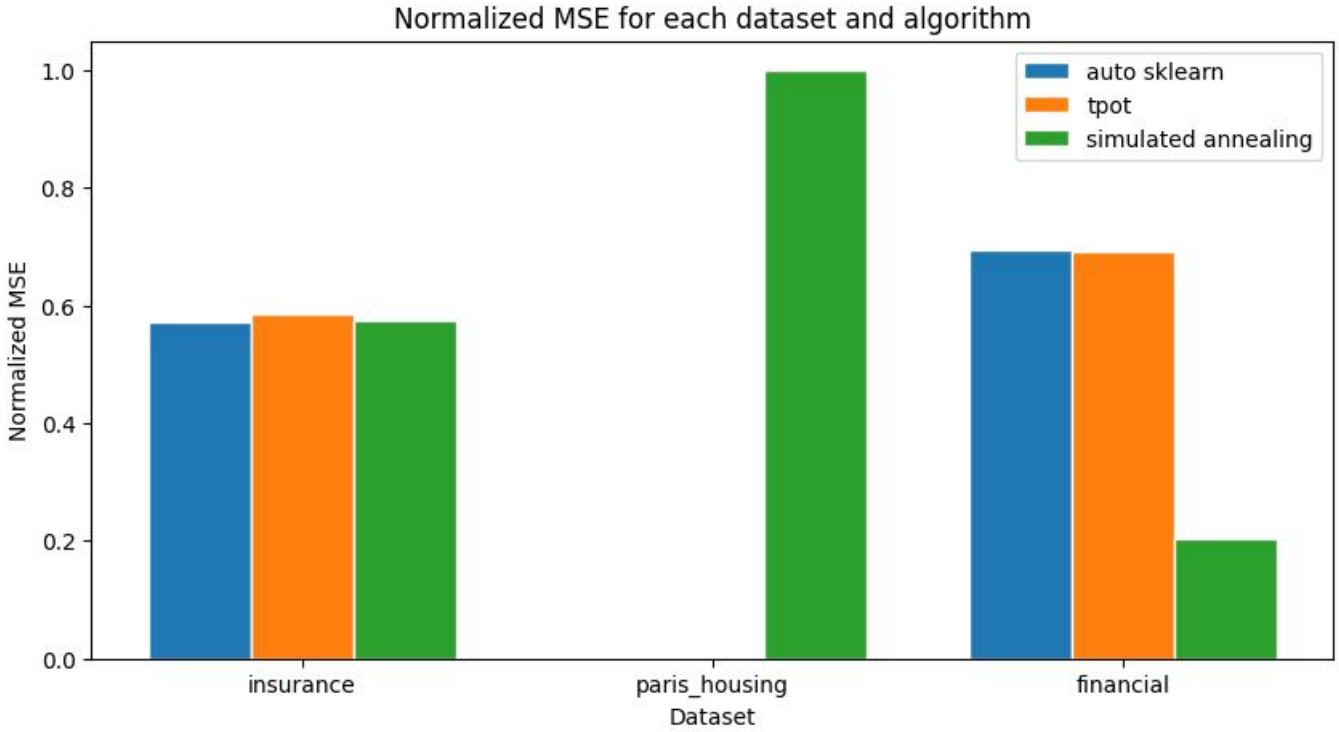
- The Table on the slide before shows again for each dataset and approach the returned best R^2 value on the training set and how it performed on the testing set.
- With R^2 we can now compare the results of different datasets and see that easiest dataset is the “paris housing” dataset, at least for the auto sklearn and tpot, as their R^2 values are close to 1
- On the “medical cost personal” dataset every approach got also decent results
- The “financial indicator” dataset seems to be the hardest, probably because it has over 200 features

Results - ML Algorithms

	Medical Cost Personal	Paris Housing Prices	Financial Indicators
Auto Sklearn	Random Forest	Random Forest	Random Forest
Tpot	SGDRegressor	SGDRegressor	SGDRegressor
Our Algorithm	Random Forest	Random Forest	SGDRegressor

Results - ML Algorithms

- The Table on the slide before shows the chosen machine learning algorithm for each dataset/approach combination.
- We see that auto sklearn always chose Random Forest and tpot always chose SGDRegressor
- Our approach chose 2 time Random Forest and for “Financial indicator” he chose SGDRegressor
- The LinearSVR was never chosen, probably because the SGDRegressor can optimize the same cost function as LinearSVR by adjusting the penalty and loss parameters and has additional features like various loss functions and less memory usage.



MSEs needed to be normalized as they are from very different ranges for each dataset. Also, please note that for Paris Housing dataset bars for Auto-sklearn and Tpot are so small that they don't appear in the plot (because MSE for our algorithm is so big). For the exact values, please consult the tables we have provided.

Summary

- Overall good results for our AutoML approach compared to state-of-the-art approaches, except for “Paris Housing” where our approach did not work at all. We could not find out why it performed so poorly.
- Meta learning for simulated annealing maybe can improve results for Paris.
- LinearSVR was outclassed by other algorithms.
- The “Financial Indicators” was the hardest to get good results on.
- It would be interesting to include different preprocessing methods in the algorithms as probably it could improve the results somehow.

Lessons learned

- Good performing hardware required
- Simulated annealing also needed tweaking of parameters
- The performance of the AutoML approach depends on the dataset
- It is good to try multiple AutoMl approaches because they can return different results.

Thank you for your attention!