Exercise 2 - AutoML

Machine learning 2021

Group 22 - Bolcek, Isilksan

13.12.2021

Agenda

- Task description
- Dataset descriptions
- Machine Learning models description
- Used AutoML algorithms
- Settings
- Results
- Comparison with TPOT
- Issues
- Conclusions

Task description

- The goal is this exercise is to implement Auto ML algorithm on regression task.
- Regression if performed on 3 different datasets using 3 different ML models (each on each)
- Auto ML algorithm is used to select the best hyper-parameters of each model on each dataset
- The best ML algorithm with the best performance is selected and applied for test dataset
- Task is solved using Python. ML models are made using scikit-learn library

Dataset descriptions

To perform regression we use 3 different datasets:

1. **A Fine Windy Day**

   Prediction the power generated by Wind turbine (in KW/h)

   https://www.kaggle.com/synergystud/a-fine-windy-day-hackerearth-ml-challenge

2. **Boston House Prices**

   Prediction for the median value of owner-occupied homes

   https://www.kaggle.com/fedesoriano/the-boston-houseprice-data/tasks?taskId=4672

3. **Daily Demand Forecasting Orders**

   Prediction of total of orders for daily treatment

   https://archive.ics.uci.edu/ml/datasets/Daily+Demand+Forecasting+Orders

A Fine Windy Day

Piece of the whole dataset:

	tracking_id	datetime	wind_speed(m/s)	atmospheric_temperature(°C)	shaft_temperature(°C)	blades_angle(°)	gearbox_temperature(°C)	engine_temperature(°C)	motor_torque(N-m)	generator_temperature(°C)	...	windmill_body_temperature(°C)	wind_direction(°)	resistance(ohm)
0	WM_33725	2019-08-04 14:33:20	94.820023	-99.000000	41.723019	-0.903423	82.410573	42.523015	2563.124522	76.665560	...	NaN	239.836388	2730.310605
1	WM_698	2018-11-05 10:13:20	241.832734	27.764785	-99.000000	-99.000000	44.104919	46.258870	2372.384119	78.129803	...	NaN	337.944723	1780.207200
2	WM_39146	2019-09-14 14:03:20	95.484724	NaN	41.855473	12.652763	42.322098	42.878552	1657.169646	67.654469	...	45.033197	227.850294	1666.049900
3	WM_6757	2018-12-25 15:33:20	238.819424	-99.000000	45.443914	15.115323	44.759643	47.282101	2888.134079	95.389974	...	44.827154	492.081520	1964.502895
4	WM_21521	2019-05-04 03:13:20	10.722890	NaN	41.981183	1.715696	-17.616459	43.469852	781.695419	37.423065	...	-99.000000	259.274601	1177.516152
5	WM_17873	2019-03-22 21:03:20	93.769973	30.326226	17.970619	-99.000000	43.816430	40.815795	2119.351653	72.345126	...	101.378184	NaN	1715.244121
6	WM_19873	2019-04-17 18:33:20	16.026249	-99.000000	44.072819	-0.196845	41.680583	43.384904	778.109985	40.284018	...	43.008746	528.003985	1222.931270
7	WM_30330	2019-07-08 21:03:20	48.737826	12.716815	43.217778	-99.000000	-48.405089	44.125843	980.988531	43.691867	...	-99.000000	NaN	1177.637341
8	WM_26069	2019-06-07 17:53:20	47.081729	-99.000000	-33.607048	-99.000000	43.055427	45.253628	957.580151	41.609787	...	43.216062	281.368625	-99.000000
9	WM_28915	2019-06-28 16:13:20	283.789329	18.887932	41.691469	52.337026	-62.724362	41.881256	1042.086135	65.280225	...	-99.000000	352.268521	1662.076277

Shape of the dataset: `(28200, 22)`

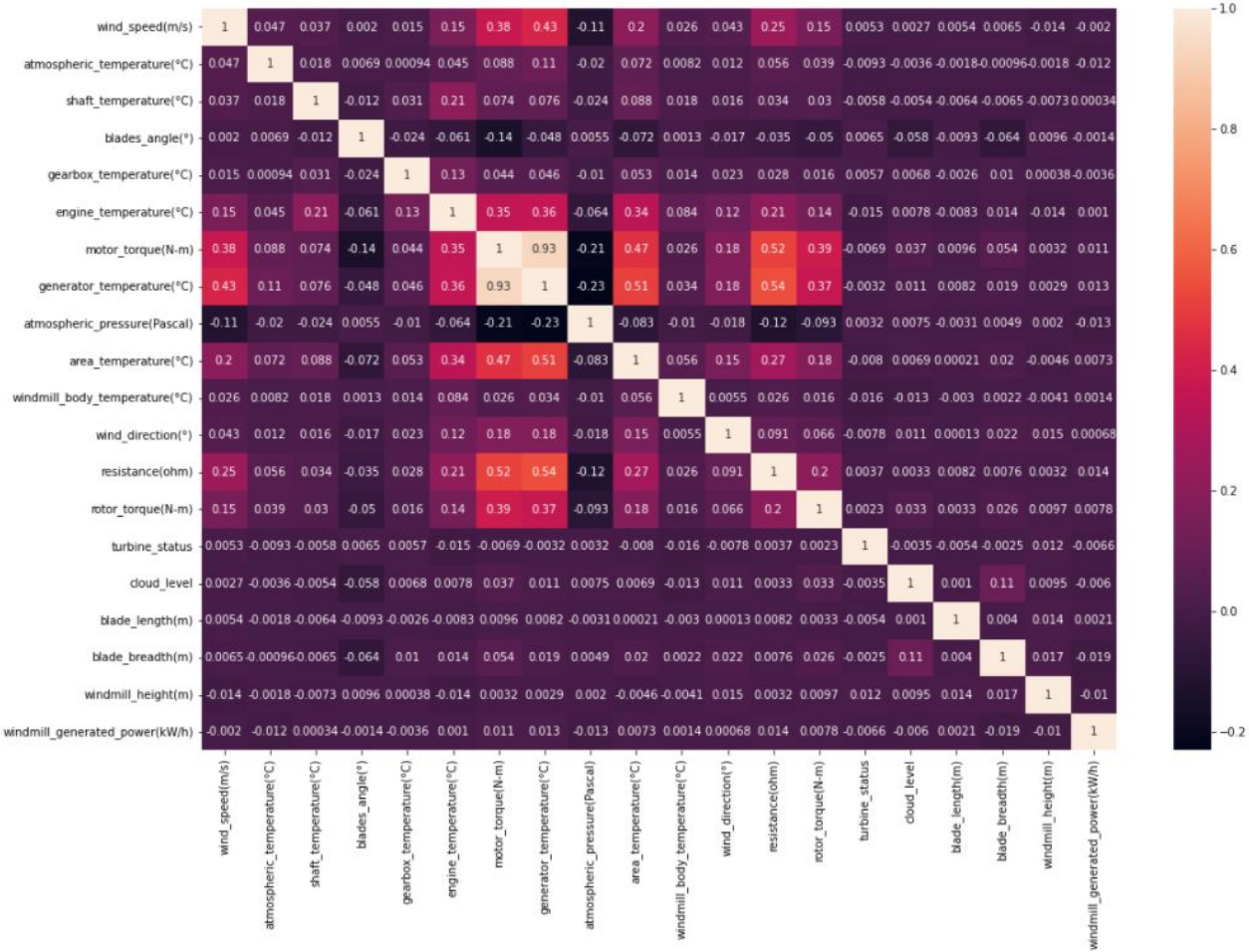
Information about the train dataset features:

#	Column	Non-Null Count	Dtype
0	tracking_id	19740 non-null	object
1	datetime	19740 non-null	object
2	wind_speed(m/s)	19564 non-null	float64
3	atmospheric_temperature(°C)	17354 non-null	float64
4	shaft_temperature(°C)	19738 non-null	float64
5	blades_angle(°)	19598 non-null	float64
6	gearbox_temperature(°C)	19739 non-null	float64
7	engine_temperature(°C)	19733 non-null	float64
8	motor_torque(N-m)	19726 non-null	float64
9	generator_temperature(°C)	19733 non-null	float64
10	atmospheric_pressure(Pascal)	17824 non-null	float64
11	area_temperature(°C)	19740 non-null	float64
12	windmill_body_temperature(°C)	18098 non-null	float64
13	wind_direction(°)	16146 non-null	float64
14	resistance(ohm)	19739 non-null	float64
15	rotor_torque(N-m)	19351 non-null	float64
16	turbine_status	18535 non-null	object
17	cloud_level	19543 non-null	object
18	blade_length(m)	16136 non-null	float64
19	blade_breadth(m)	19740 non-null	float64
20	windmill_height(m)	19355 non-null	float64
21	windmill_generated_power(kW/h)	19588 non-null	float64

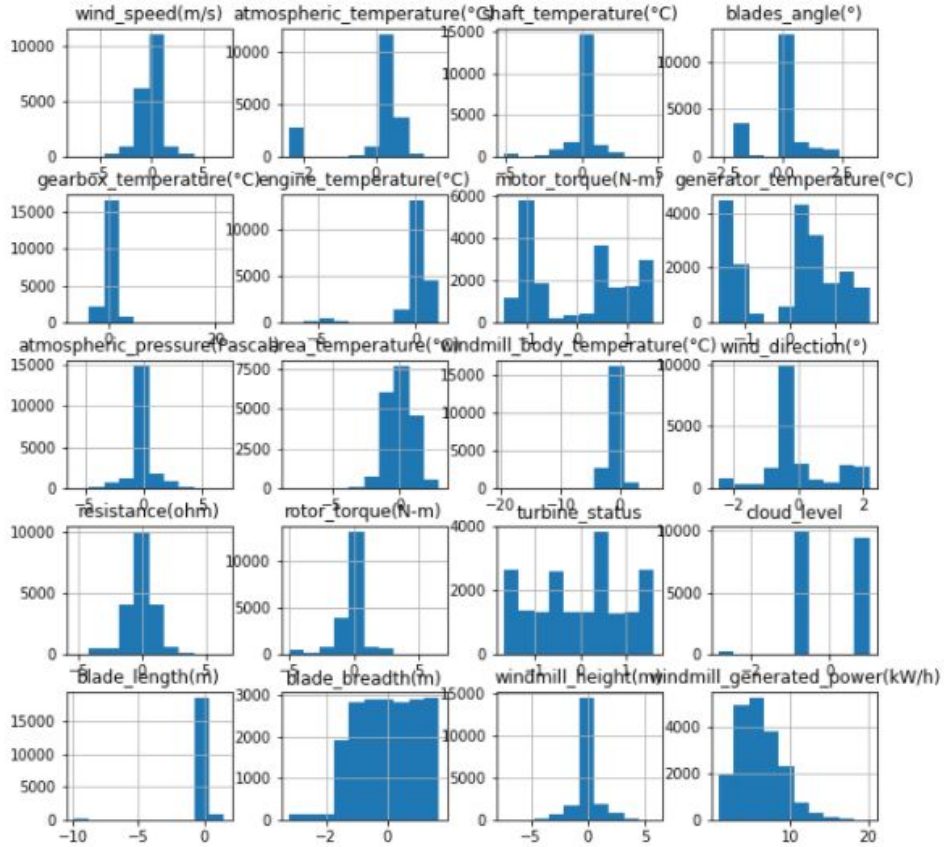
Information about the train dataset features after necessary pre-processing:

#	Column	Non-Null Count	Dtype
0	wind_speed(m/s)	19588 non-null	float64
1	atmospheric_temperature(°C)	19588 non-null	float64
2	shaft_temperature(°C)	19588 non-null	float64
3	blades_angle(°)	19588 non-null	float64
4	gearbox_temperature(°C)	19588 non-null	float64
5	engine_temperature(°C)	19588 non-null	float64
6	motor_torque(N-m)	19588 non-null	float64
7	generator_temperature(°C)	19588 non-null	float64
8	atmospheric_pressure(Pascal)	19588 non-null	float64
9	area_temperature(°C)	19588 non-null	float64
10	windmill_body_temperature(°C)	19588 non-null	float64
11	wind_direction(°)	19588 non-null	float64
12	resistance(ohm)	19588 non-null	float64
13	rotor_torque(N-m)	19588 non-null	float64
14	turbine_status	19588 non-null	float64
15	cloud_level	19588 non-null	float64
16	blade_length(m)	19588 non-null	float64
17	blade_breadth(m)	19588 non-null	float64
18	windmill_height(m)	19588 non-null	float64
19	windmill_generated_power(kW/h)	19588 non-null	float64

Heatmap of the train data



Distribution of unique values in train attributes



Boston House Prices

First 10 rows of the dataset:

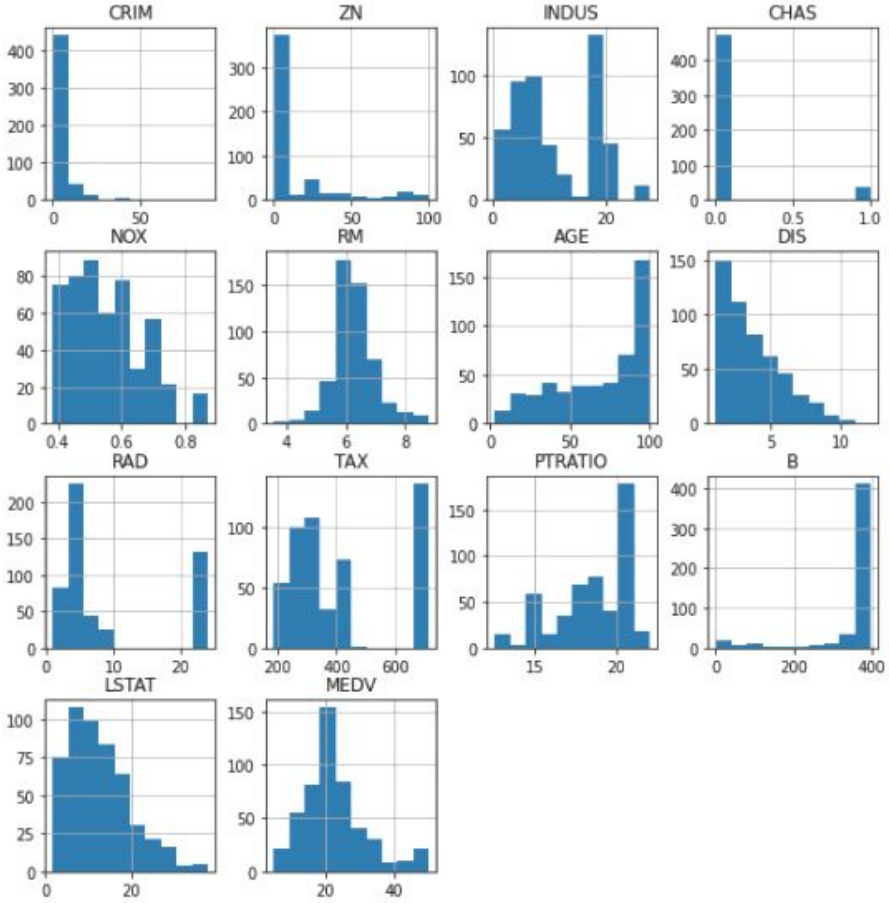
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311.0	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311.0	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311.0	15.2	386.71	17.10	18.9

Information about the features:

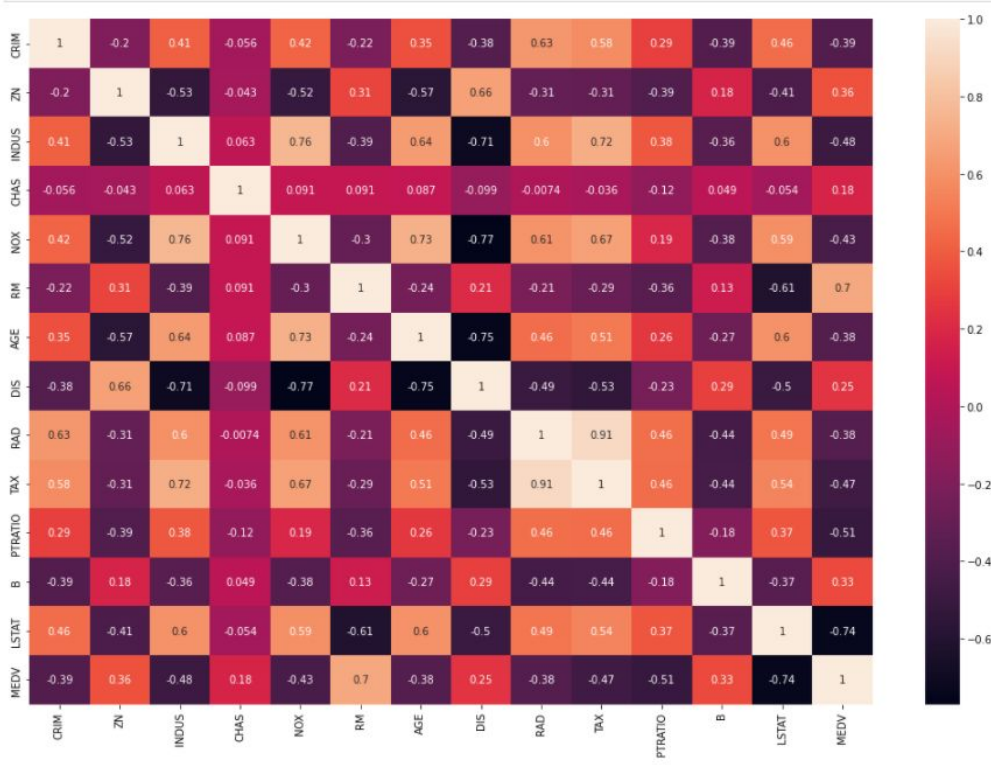
```
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
```

Shape of the dataset: `(506, 14)`

Distribution of unique values in attributes



Heatmap of the dataset



Daily Demand Forecasting Orders

First 10 rows of the dataset:

	week of the month	day of the week	non-urgent order	urgent order	order type A	order type B	order type C	fiscal sector orders	orders from traffic controller sector	banking orders 1	banking orders 2	banking orders 3	Target(total orders)
0	1	4	316.307	223.270	61.543	175.586	302.448	0	65556	44914	188411	14793	539.577
1	1	5	128.633	96.042	38.058	56.037	130.580	0	40419	21399	89461	7679	224.675
2	1	6	43.651	84.375	21.826	25.125	82.461	1.386	11992	3452	21305	14947	129.412
3	2	2	171.297	127.667	41.542	113.294	162.284	18.156	49971	33703	69054	18423	317.120
4	2	3	90.532	113.526	37.679	56.618	116.220	6.459	48534	19646	16411	20257	210.517
5	2	4	110.925	96.360	30.792	50.704	125.868	79	52042	8773	47522	24966	207.364
6	2	5	144.124	118.919	43.304	66.371	153.368	0	46573	33597	48269	20973	263.043
7	2	6	119.379	113.870	38.584	85.961	124.413	15.709	35033	26278	56665	18502	248.958
8	3	2	218.856	124.381	33.973	148.274	162.044	1.054	66612	19461	103376	10458	344.291
9	3	3	146.518	101.045	36.399	43.306	168.723	865	58224	7742	82395	11948	248.428

Shape of the dataset:

```
pd_data.shape[0]
```

```
60
```

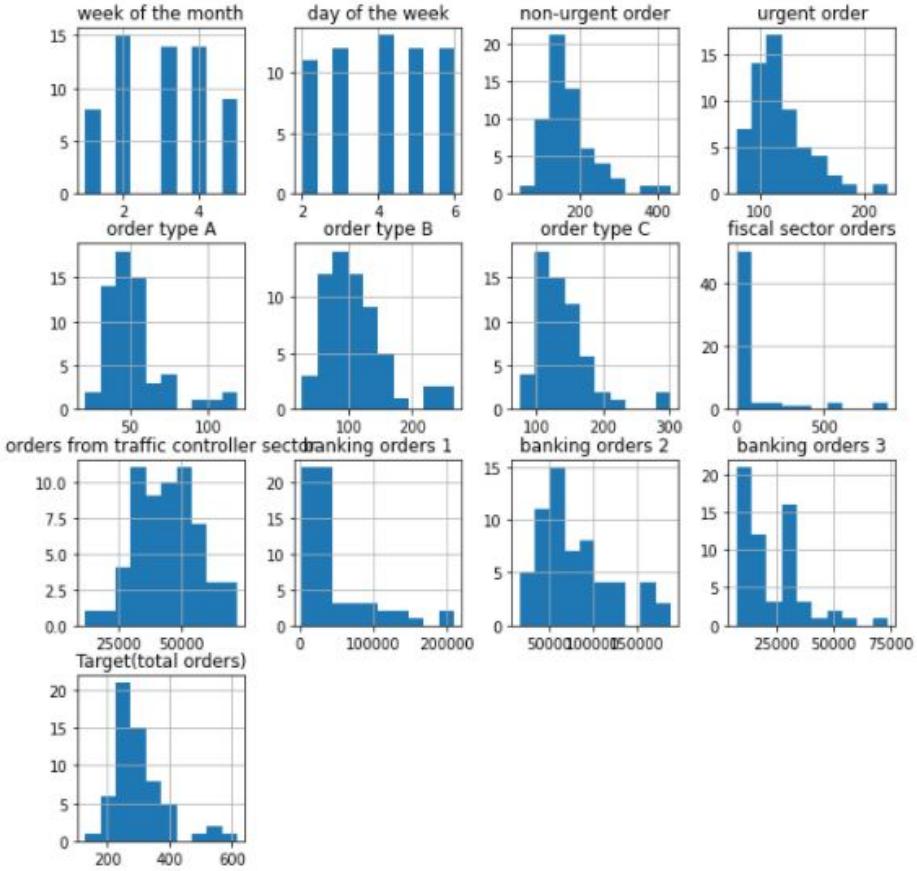
```
pd_data.shape[1]
```

```
13
```

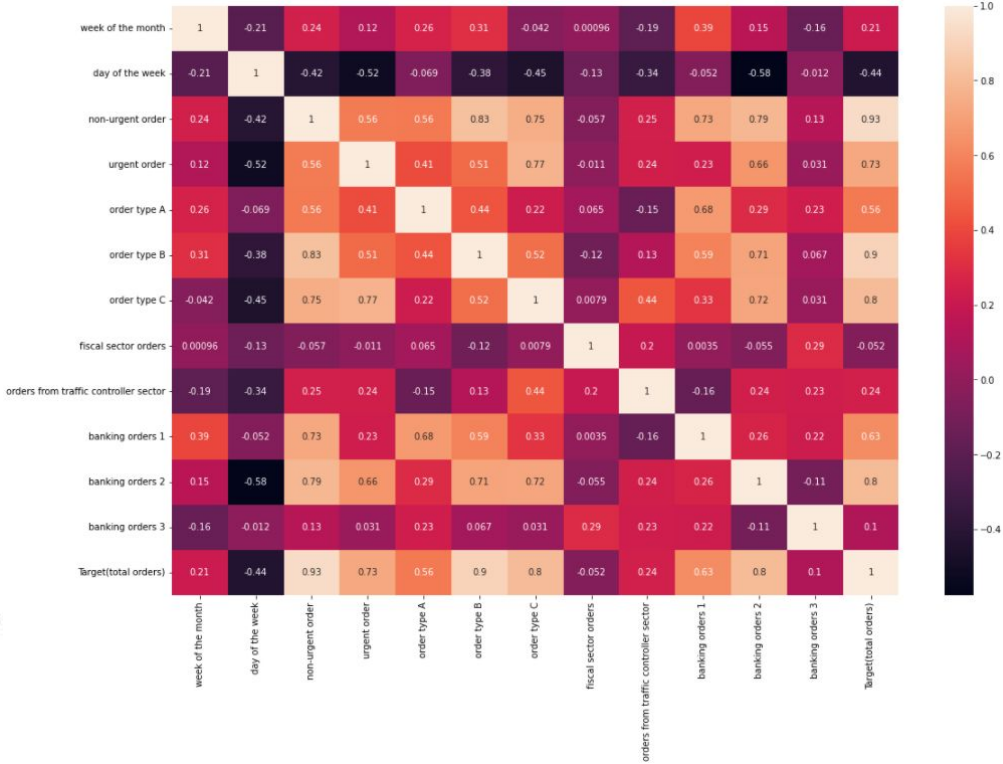
Information about the features:

```
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   week of the month                     60 non-null     int64
 1   day of the week                       60 non-null     int64
 2   non-urgent order                      60 non-null     float64
 3   urgent order                          60 non-null     float64
 4   order type A                          60 non-null     float64
 5   order type B                          60 non-null     float64
 6   order type C                          60 non-null     float64
 7   fiscal sector orders                  60 non-null     float64
 8   orders from traffic controller sector 60 non-null     int64
 9   banking orders 1                      60 non-null     int64
 10  banking orders 2                      60 non-null     int64
 11  banking orders 3                      60 non-null     int64
 12  Target(total orders)                  60 non-null     float64
```

Distribution of unique values in attributes



Heatmap of the dataset



Machine learning models description

We use 3 ML models, where we tune this parameters

1. **Random Forest Regressor (RFR)**

   max_depth, n_estimators, criterion, max_features

2. **Support Vector Regressor (SVR)**

   kernel, C, gamma

3. **KNeighbours Regressor (KNNR)**

   n_neighbours, weights, algorithms, leaf_size, metric

Used AutoML algorithms

1. Simulated annealing - from scratch
   - Randomly initialize single solution, in every iteration it randomly changes one hyper-parameter. It evaluate the performance on the model (objective function). Check if the performance is better than the current best. If not, it will not throw the solution but decide if it will keep the solution for future computation. Decision is based on current temperature and metropolis function. If it will decide to keep the solution, it starts from it the next step (so it will not stuck in some local minima)
2. Bayes optimization - from scratch
   - Randomly generate n-solution and evaluate their performance by objective function. Keep the best. The other are used to tune another model (Surrogate model - fit on hyper-parameters and score). Than randomly initialize new set of solution and predict their performance on Surrogate model. Acquitizion function will select the solution that will probably have the best performance and evaluate it on objective function. This new solution is pomrated with the current best . All solution evaluated on objective function are used to update the surrogate model
3. TPOT - Python library
   - *http://epistasislab.github.io/tpot/*

Simulated annealing

```
Input: Tempretature, n_iterations,
Ouput: BEST_solution

1. Current_solution = Initialize random hyperparameters
2. Current_fit      = Evaluate(Solutions)
3. BEST_solution    = Save this as the best
4. BEST_fit         = Current_fit

4. for (i = 1 to n_iterations)
    4.1 Candidate_to_solution = Randomly change one parameter in current solution
    4.2 Candiate_fit          = Evaluate(Candidate_to_solution)
    4.3 If Candiate_fit > BEST_fit then BEST_fit = candidate_fit and BEST_solution = Candidate_to_solution
    4.4 Difference_in_solu = Current_fit - Candiate_fit
    4.5 Evaluate current temperature
    4.6 Evaluate Metropolis function
    4.6 if (Difference_in_solu > 0 or random.number() < Metropolis_function) than
        Current_solution = Candidate_to_solution
        Current_fit = Candiate_fit

5. return BEST_solution
```

Simulated annealing - Pseudocode

Bayes optimization

```
Input: n_iterations
Output: BEST_solution

1. Solutions = Inicialize n_solutions
2. Evaluation = evaluate(Solutions)
3. surrogate_model.Fit(solutioms,evaluation)
4. BEST_Solution = best(Solutions)
5. BEST_fit      = best(Evaluation)

4. for (i = 1 to n_iterations)
   4.1 random_solu = Generate random solutions
   4.2 probably_best = (acquitizion_function(suggogate_model(random_solu))
   4.3 probably_best_eval = evaluate(probably_best)
   4.4 surrogate_model.Fit(probably_best,probably_best_eval)
   4.5 if (probably_best_eval > BEST_fit), then
       BEST_fit = probably_best_eval
       BEST_Solution = probably_best

5. return BEST_solution
```

Bayes optimization - Pseudocode

Settings - general

- For evaluating model performance we use metric: *negative root mean square error*
- AutoML algorithms evaluates the model performance using cross-validation, after that the Best hyper-parameters are also evaluated on test set (if available)
- Each Auto-ML algorithm is run multiple-times (5x - 20x) to cover the searchspace better (Only the best of the Best results would be presented)
- Each dataset use a bit different search space
- Cost/objective function takes a hyper-parameters (and Training data) as an input and gives n-RMS as an output, that we want to maximalize

Settings - Simulated annealing

For Simulated annealing needs to define some extra parameters and functions

- Temperature = 100 (initial temperature - in every case)

```
def get_temperature(self,temp,n_iter):
    # n_iter - accual number of iteration
    # temp - initial temperature
    t = temp / float(n_iter + 1)
    return t

def get_metropolis(self, diff):
    #diff - difference between candidate score and current score

    metropol = math.exp(-10*diff / self.temretature_list[-1])
    self.metropolis_list.append(metropol)
    return metropol
```

Settings - Bayes optimization

For Bayes optimization needs to define some extra parameters and functions

- n_samples = 40 (len of initial population)

```
def acquisition_Function(random_param,real_X,real_Y,surrogate):
    #get the best parameters so far
    score_pred = surrogate.predict(real_X)
    best = max(real_Y)


    #predict accurcy
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        mu, std = surrogate.predict(random_param, return_std=True)

        mu = mu[:, 0]

        probs = norm.cdf((mu - best) / (std))

    ix = np.argmax(probs)

    return random_param[ix]

pr_model = GaussianProcessRegressor(random_state=0)
```

Acquisition function

Search space - Boston House Prices

```
SVR_parameters_ranges = {'kernel':['linear', 'poly', 'rbf', 'sigmoid'],
                         'C': [0.00001,0.0001,0.001,0.01,0.1,1], 'gamma': ['scale','auto']}

RFR_parameters_ranges = {'max_depth': np.arange(1,250), 'n_estimators': np.arange(1,250),
                    'criterion': ['mae','mse'], 'max_features' : ["sqrt","log2"]}

KNNR_parameters_ranges = {'n_neighbors':np.arange(1,50), 'weights' : ['uniform', 'distance'],
                          'algorithm': ['ball_tree', 'kd_tree', 'brute'],'leaf_size': np.arange(1,50),
                         'metric' : ['euclidean', 'manhattan', 'minkowski']}
```

Search space for Boston House Prices dataset

- 1000 iterations of autoML were used to search through this space
- AutoML were run 10x

Search space - A Fine Windy Day

```
SVR_parameters_ranges = {'kernel':['linear', 'poly', 'rbf', 'sigmoid'],
                         'C': [0.00001,0.0001,0.001,0.01,0.1,1], 'gamma': ['scale','auto']}

RFR_parameters_ranges = {'max_depth': np.arange(1,70), 'n_estimators': np.arange(1,70),
                     'criterion': ['mae','mse'], 'max_features' : ["sqrt","log2"]}

KNNR_parameters_ranges = {'n_neighbors':np.arange(1,50), 'weights' : ['uniform', 'distance'],
                          'algorithm': ['ball_tree', 'kd_tree', 'brute'],'leaf_size': np.arange(1,50),
                         'metric' : ['euclidean', 'manhattan', 'minkowski']}
```

Search space for A Fine Windy Day Dataset

- 100 iterations of autoML were used to search through this space
- AutoML were run 5x

Search space - Daily Demand Forecasting Orders

```
SVR_parameters_ranges = {'kernel':['linear', 'poly', 'rbf', 'sigmoid'],
                         'C': [0.00001,0.0001,0.001,0.01,0.1,1], 'gamma': ['scale','auto']}

RFR_parameters_ranges = {'max_depth': np.arange(1,250), 'n_estimators': np.arange(1,250),
                     'criterion': ['mae','mse'], 'max_features' : ["sqrt","log2"]}

KNNR_parameters_ranges = {'n_neighbors':np.arange(1,40), 'weights' : ['uniform', 'distance'],
                          'algorithm': ['ball_tree', 'kd_tree', 'brute'],'leaf_size': np.arange(1,50),
                         'metric' : ['euclidean', 'manhattan', 'minkowski']}
```

Search space for Boston House Prices dataset

- 100 iterations of autoML were used to search through this space
- AutoML were run 25x

Results - Boston House Prices

Simulated annealing | Bayes optimization

	RNFR	SVR	KNNR	BEST/Total
Best score	-4.17967	-5.56103	-4.45608	**-4.17967(RNF)**
Mean time [s]	2610.9798	25.8638	12.9671	833.207
Total time [s]	26109.7983	258.6388	129.6715	**26498.1086 (~7.5h)**
Best score	-4.17967	-5.56103	-4.45608	**-4.17967 (RNF)**
Mean time [s]	3018.3737	97.1544	85.8876	1067
Total time [s]	30183.7374	971.5449	858.8761	**32014.1584 (~7,8h)**

Note: Total time = total time to fill all runs of AutoML, Mean_time = time/single_run

Boston House Prices (Error during AutoML iterations)

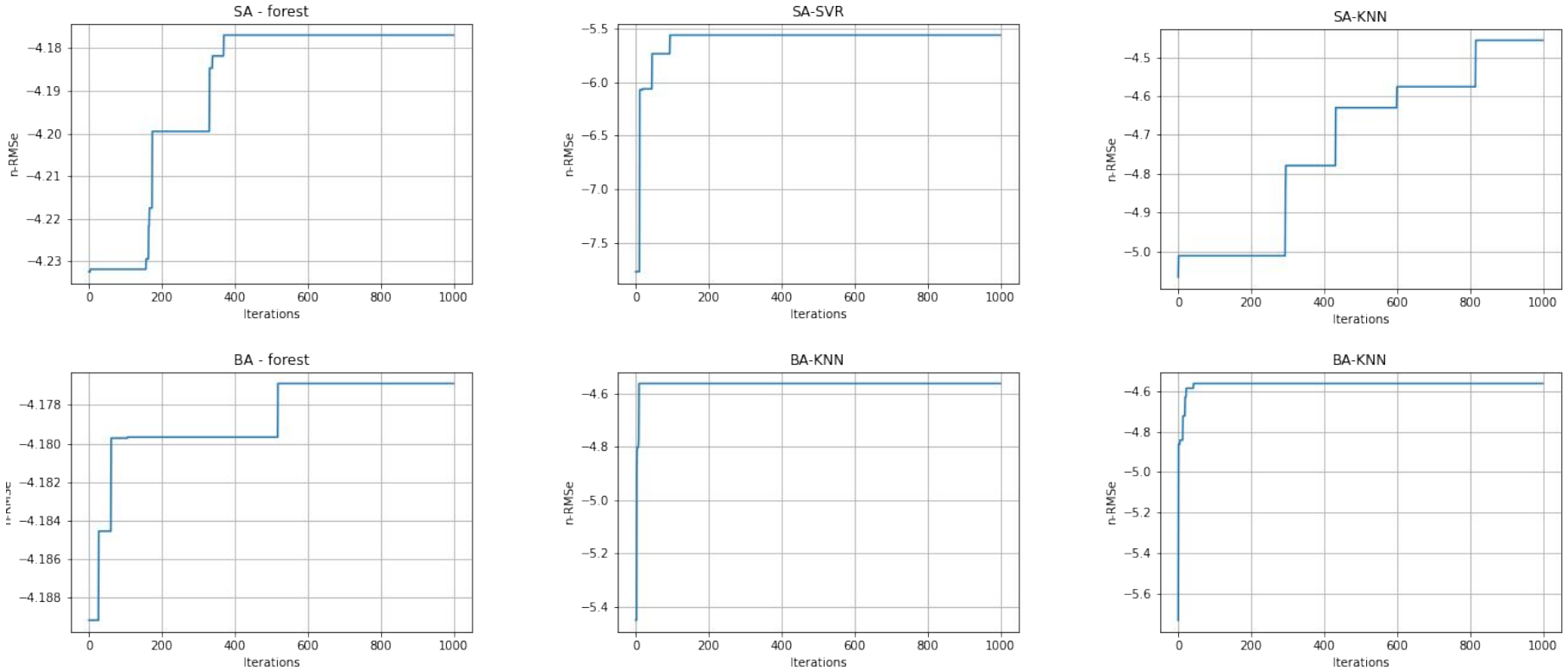


Fig. 5: Search space for Boston House Prices dataset

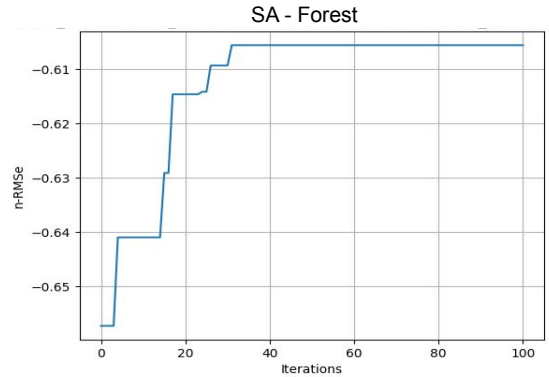
Results - A Fine Windy Day

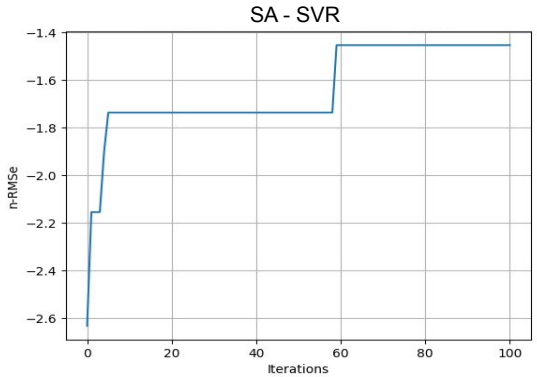
Simulated annealing | Bayes optimization

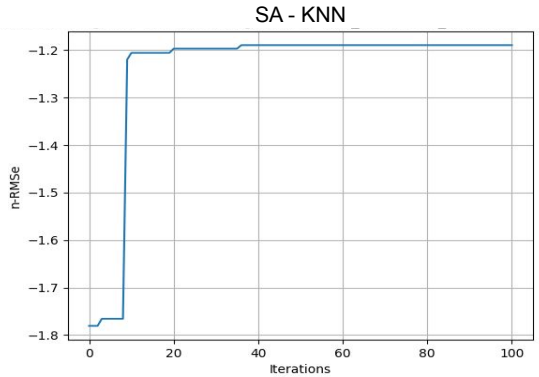
	RNFR	SVR	KNNR	BEST/Total
Best score	-0.6011	-1.4531	-1.1892	-0.6011(RFR)
Mean time [s]	2461.3556	1000.4472	316.89093	2519.1291
Total time [s]	24613.5561	10004.4722	3168.9093	**37786.937 (~10.4h)**
Best score	-0.6011	-1.4531	-1.1892	-0.6011(RFR)
Mean time [s]	2486.7811	1245.0430	363.0574	2729.9210
Total time [s]	24867.811	12450.43041	3630.5741	**40948.81 (~11.3h)**

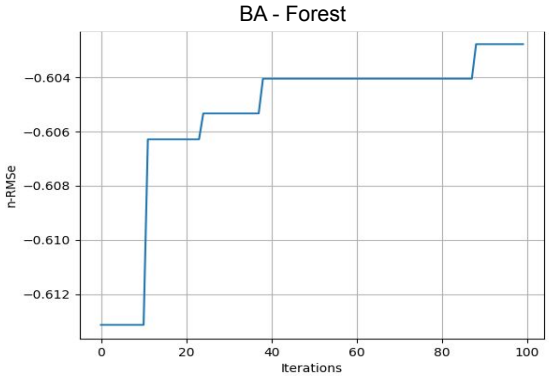
Note: Total time = total time to fill all runs of AutoML, Mean_time = time/single_run

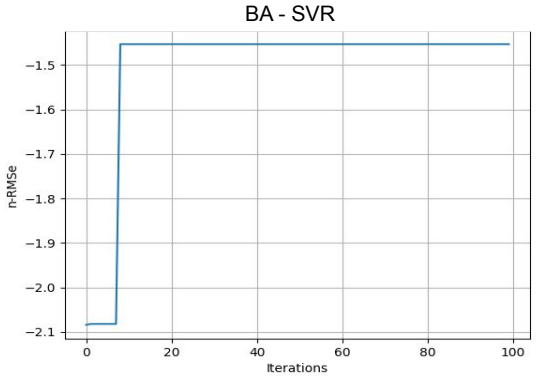
A Fine Windy Day (Error during AutoML iterations)

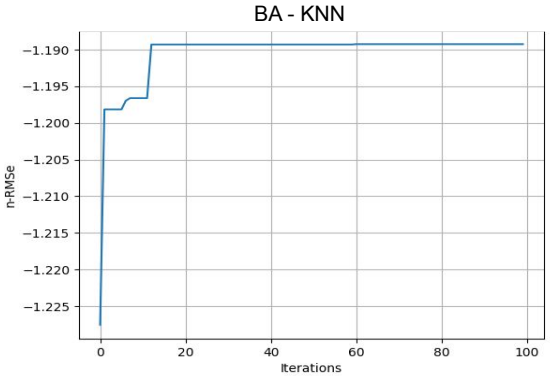












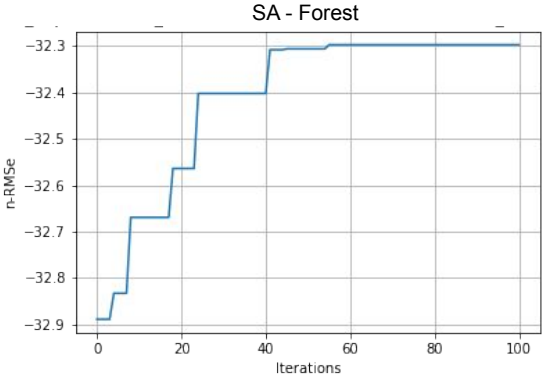
Results - Daily Demand Forecasting Orders

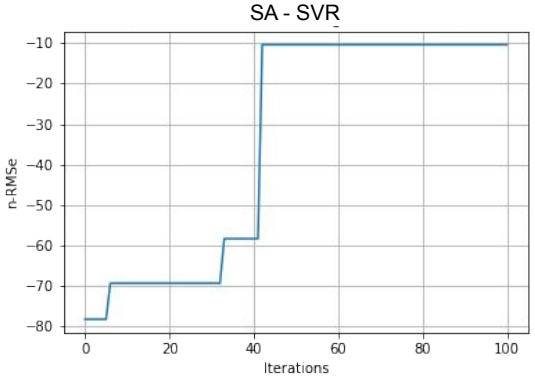
Simulated annealing | Bayes optimization

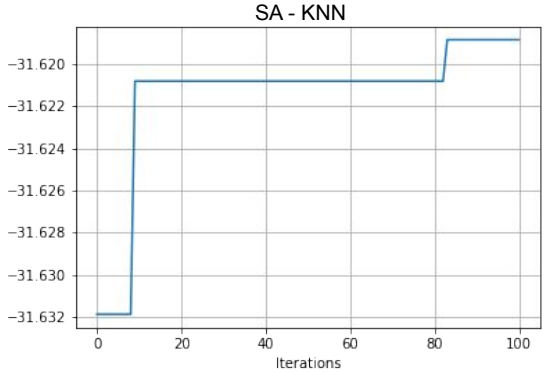
	RNFR	SVR	KNNR	BEST/Total
Best score	-31.604	-10.3616	-37.6302	-10.3616(SVR)
Mean time [s]	325.10862	25.028	24.98932	50.0147
Total time [s]	3251.0862	250.28	249.8932	**3751 (~1.04h)**
Best score	-31.604	-10.3616	-37.6302	-10.3616(SVR)
Mean time [s]	343.2045	26.282	27.3304	162.705
Total time [s]	3432.0456	262.28	273.3042	**4067.62 (~1.12h)**

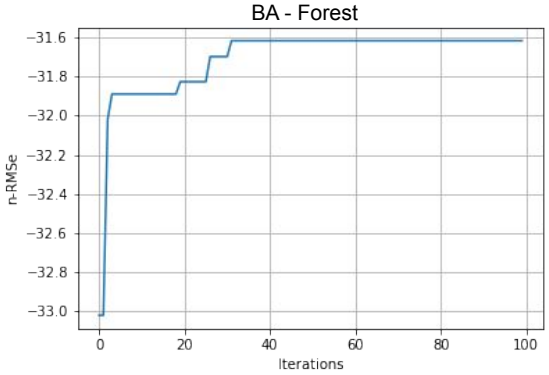
Note: Total time = total time to fill all runs of AutoML, Mean_time = time/single_run

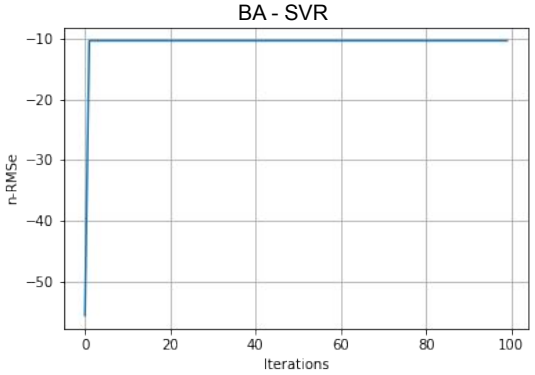
Daily Demand Forecasting Orders (Error during AutoML iterations)

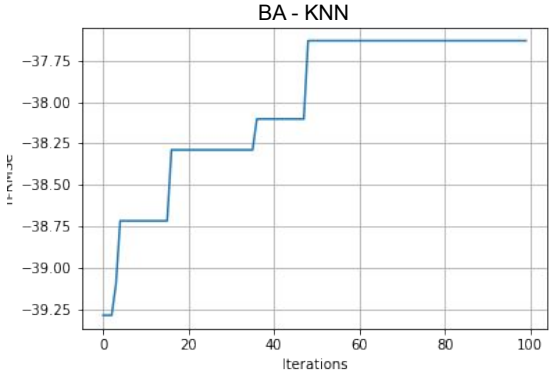












Results - the Best of the best

1. Boston House Prices
   RFR - {'max_depth': 33, 'n_estimators': 245, 'criterion': 'mae', 'max_features': 'log2'}

2. A Fine Windy Day
   RFR - {'max_depth': 33, 'n_estimators': 14, 'criterion': 'mae', 'max_features': 'log2'}

3. Daily Demand Forecasting Orders
   SVC - {'kernel': 'linear', 'C': 1, 'gamma': 'scale' }

Consider TPOT your **Data Science Assistant**. TPOT is a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming.

Comparison with TPOT

	Simulated annealing			Bayes optimization			TPOT		
	CV	Test	time[s]	CV	Test	time[s]	CV	Test	time[s]
Boston	-4.17967	-3.3942	26498.108	-4.17967	-3.3942	32014.158	-3.7876	-3.4353	18318.016
A Fine Windy	-0.6011	-0.6466	37786.937	-0.6011	-0.6466	40948.81	-0.5531	-0.60425	11403.899
Daily Demand	-10.3616	-75.866	3751	-10.3616	-75.8666	4067.62	-0.1260	-134.0477	

Comparison with TPOT - Comments

- TPOT was able to find the best solution much faster that our algorithms (it it because maybe we run it more times than it was necessary

- The performace on solutions found by our algorithms is comparable to performance achieved by TPOT

- From the results achieved on the third dataset could be seen strong over-fitting.

Issues

- The single AutoML did not cover the search space very well, do that's why we run it multiple times (So it took very much computational time)

- In some cases the models performance did not very improved during AutoML iterations (sometimes it did not improved at all) - keep attention of hyper-parameters of auto ML algorithms

- When the Bayes optimization initial population is too big, than it find usually the best solution in initialization and after that this technique is equal to random search (with an extra unnecessary iterations)

Issues 2

- Bayes optimization was paradoxically slower than Simulated annealing (maybe it is because it took long time for initialization)

- Over-fitting in the last dataset (the regression techniques were too brutal and search space too big)

- Computational time was too long

Conclusions

- RFC had the best performance, on the other hand it took much longer time to fit it
- It it necessary to still keep attention on the solved task and try to choose search spaces, that makes sense, otherwise the results will not be sufficient (as in the third dataset)
- The autoML algorithms have also hyper-parameters (n_of_initial population, temperature….). This parameters had to be chossed carefully otherwise the autoML algorithms will not work well (or in different way than expected)