Automatic Piecewise Linear Regression

Mathias von Ottenbreit
22.08.2022

Outline

Introduction

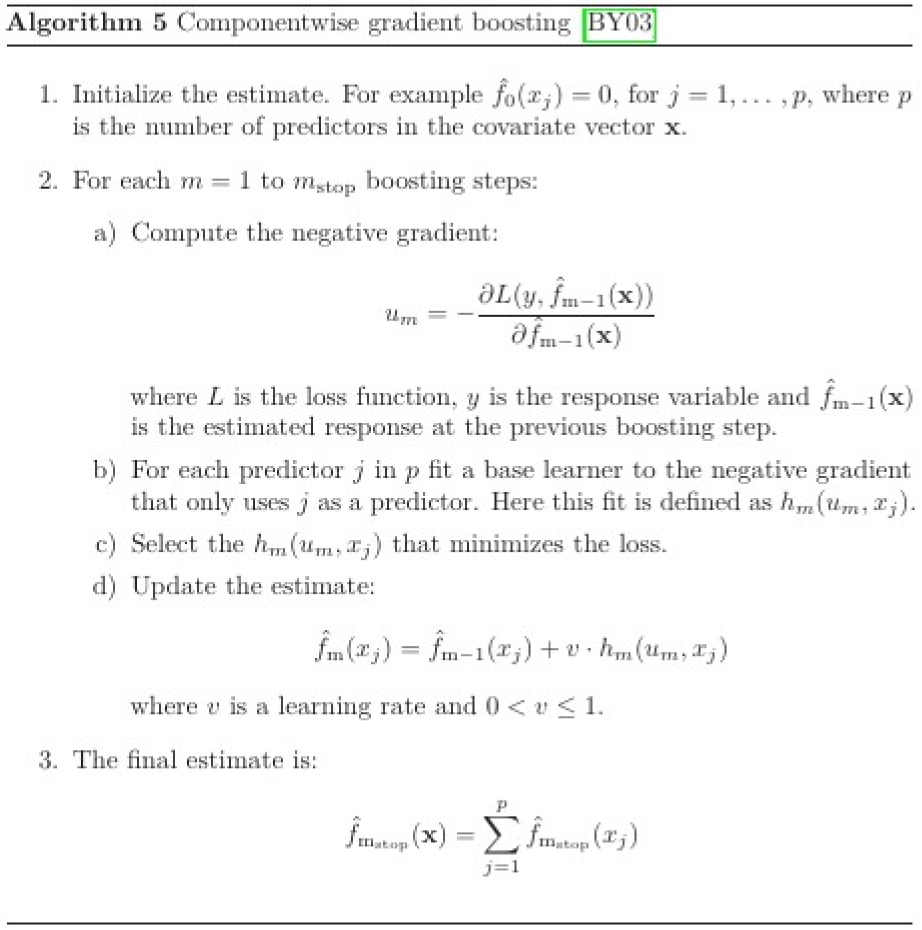
- Focus: regression modelling
- Predictiveness versus interpretability
  - Interpretable regression algorithms are often less predictive
- Ease of use
  - Variable selection, non-linear relationships and interactions
- Goal: Develop a regression algorithm that is easy to use and reduces the trade-off between predictiveness and interpretability
- *Please note: The literature reference for this presentation is my master thesis and the bibliography mentioned there*

Automatic Piecewise Linear Regression (APLR)

- Automatically handles variable selection, non-linear relationships and interactions
- Fits piecewise linear basis functions to the data by using componentwise boosting
- Componentwise boosting handles variable selection
- The piecewise linear basis functions handle non-linear relationships and interactions

General componentwise boosting algorithm

- When linear base learners are used then regression coefficients are estimated
- Variable selection: regression coefficients for irrelevant predictors are not updated
- Important hyperparameters:
  - The number of boosting steps, *mstop*
  - The learning rate, *v*
  - They are dependent. A lower *v* often implies a higher optimal *mstop*
  - Typically *mstop* is tuned and *v is held constant at a low value of <=0.1*



Algorithm 5 Componentwise gradient boosting [BY03]

```
1. Initialize the estimate. For example $\hat{f}_0(x_j) = 0$, for $j = 1, \dots, p$, where $p$
   is the number of predictors in the covariate vector $\mathbf{x}$.
2. For each $m = 1$ to $m_{\text{stop}}$ boosting steps:
   a) Compute the negative gradient:
          $$u_m = -\frac{\partial L(y, \hat{f}_{m-1}(\mathbf{x}))}{\partial \hat{f}_{m-1}(\mathbf{x})}$$
      where $L$ is the loss function, $y$ is the response variable and $\hat{f}_{m-1}(\mathbf{x})$
      is the estimated response at the previous boosting step.
   b) For each predictor $j$ in $p$ fit a base learner to the negative gradient
      that only uses $j$ as a predictor. Here this fit is defined as $h_m(u_m, x_j)$.
   c) Select the $h_m(u_m, x_j)$ that minimizes the loss.
   d) Update the estimate:
          $$\hat{f}_m(x_j) = \hat{f}_{m-1}(x_j) + v \cdot h_m(u_m, x_j)$$
      where $v$ is a learning rate and $0 < v \leq 1$.
3. The final estimate is:
          $$\hat{f}_{m_{\text{stop}}}(\mathbf{x}) = \sum_{j=1}^{p} \hat{f}_{m_{\text{stop}}}(x_j)$$
```

Piecewise linear basis functions in APLR without interactions

- Definition

$\max(x - t, 0)$

$\min(x - t, 0)$

- Captures non-linearity through local effects, because the basis functions may be zero for large ranges of x (the predictor)
- Similar to the basis functions in MARS, but in APLR only one basis function is fitted in each step to work in componentwise boosting and to be better suited in high dimensions

Piecewise linear basis functions in APLR with interactions (1)

- Definition

$$\max(x - t, 0) \cdot \mathbb{1}(i \neq 0)$$

$$\min(x - t, 0) \cdot \mathbb{1}(i \neq 0)$$

  - The value of the basis function is zero when the value of *i* is zero
  - The value of the basis function only depends on x and t when the value of *i* is non-zero, making interpretation easier
  - *i* is another APLR basis function of a predictor x* with or without interactions
  - Interaction level is one more than the interaction level of *i* => can capture nested interactions
- Models interactions through local effects
- Similar to what happens in regression trees

Piecewise linear basis functions in APLR with interactions (2)

- Avoids some disadvantages that pertain to handling of interactions in MARS:
  - In MARS, an interaction between x1 and x2 is modelled as x1*x2
  - Higher ordered interaction terms modelled in this way can cause numerical problems
  - The importance of the sign of the underlying predictors may not be captured
    - x1*x2 can be the same in multiple scenarios
    - If x1=1 and x2=-1, then x1*x2 = -1
    - If x1=-1 and x2=1 then x1*x2 = -1
    - The response variable may differ in those two segments but x1*x2 does not capture this
  - Difficult to isolate the effect of a single predictor as the value of the interaction term is multiplicatively dependent on multiple predictors

APLR fitting procedure: Data splitting

- APLR splits the training data into a training and validation set
- *mstop* is automatically tuned to minimize validation loss
- This speeds up the fitting procedure significantly compared to tuning by doing a grid search
- Potential drawback: Lower data utilization compared to cross validation

APLR fitting procedure: Boosting

- In each boosting step APLR selects the one of the below options that reduce the training error the most:
  - Add a new term from a set of eligible APLR basis functions (*E*)
  - Update the regression coefficient for a term already in the model
  - Add a new interaction term
  - Update the intercept term
  - Terminate the procedure if none of the above reduce the training error
- The final estimate is the model from the boosting step having the lowest validation error

APLR fitting procedure: Fitting a basis function

- In each boosting step APLR basis functions are fitted to the negative gradient
- Searching for the optimal value of *t* in an APLR basis function can be computationally intensive
- Instead, APLR searches for an approximately optimal *t* using a discretization technique similar to what is implemented in Xgboost (algorithm for boosting of trees)
- The data is discretized into bins (by default up to 300 bins) and the approximately optimal *t* is estimated on the discretized data
- This speeds up the fitting procedure significantly for larger datasets

APLR fitting procedure: Eligibility of terms

- In APLR, a term is an APLR basis function (with or without interactions) of at least one predictor in the training data
- Evaluating all potential terms in each boosting step can be computationally intensive
- Two hyperparameters limit this search space:
    - *max_eligible_terms* (default value of 5) sets a limit on the number of terms already in the model that can be interaction partners for terms from *E*. *E* is the set of eligible terms in a boosting step
    - Also, in the next boosting step, up to *max_eligible_terms* terms with the lowest loss will remain in *E.* The other terms drop out from *E* for the next *ineligible_boosting_steps_added* (default value of 10) boosting steps
    - Example: With the default hyperparameter values mentioned above, a term that becomes ineligible at the end of the first boosting step will be ineligible in boosting steps 2 to 11 inclusive, but will re-enter *E* in boosting step 12

APLR fitting procedure: Interactions

- In each boosting step APLR first selects a candidate from *E* for entry to the model
- Then APLR tries to find interaction terms. Up to *max_eligible_terms* terms already in the model are considered as interaction partners for terms from *E*
- Interaction terms that reduce the loss more than the candidate from *E* are added to *E* as long as the total number of interactions ever added to *E* does not exceed *max_interactions* (hyperparameter) and given that the interaction level does not surpass *max_interaction_level*
- The interaction term that reduces the loss the most is preferred over the candidate from *E* if it reduces the loss more than the candidate from *E*

APLR fitting procedure: Important hyperparameters

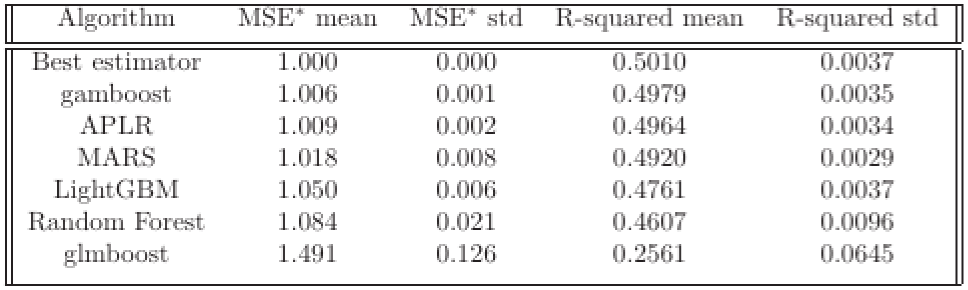
- *M* (default is 1000) is the maximum boosting steps to try. Should be large enough to minimize the validation error but not so large that unnecessary computation costs are incurred
- *v* (default is 0.1) is the learning rate. Should ideally be <= 0.1
- *max_interaction_level* (default is 100) specifies the maximum allowed interaction depth. Should be tuned by for example doing a grid search
- *max_interactions* (default is 0) specifies the maximum number of interactions that can be considered. The higher value the higher computational load. Can be tuned or set to a value that is computationally affordable
- *min_observations_in_split* (default is 20) specifies the minimum number of training observations where a term must not have a value of zero due to the max, min or indicator functions (number of effective observations). The higher value the lower variance (the term relies on more observations) but higher bias (less fine grained splits). Should be tuned by for example doing a grid search

Testing methodology

- APLR was contrasted to LightGBM (tree-boosting), Random Forest, MARS, gamboost and glmboost on simulated and real datasets
- The algorithms were ranked on test MSE
- Four scenarios were simulated:
  - Uncorrelated predictors and an additive true model
  - Correlated predictors and an additive true model
  - Uncorrelated predictors and a non-additive true model
  - Correlated predictors and a non-additive true model
- Real datasets:
  - Auto MPG (small dataset with 8 predictors, low interaction depth)
  - YearPredictionMSD (large dataset with 90 predictors, medium interaction depth)
  - Individual household electric power consumption (large dataset with 6 predictors, high interaction depth)

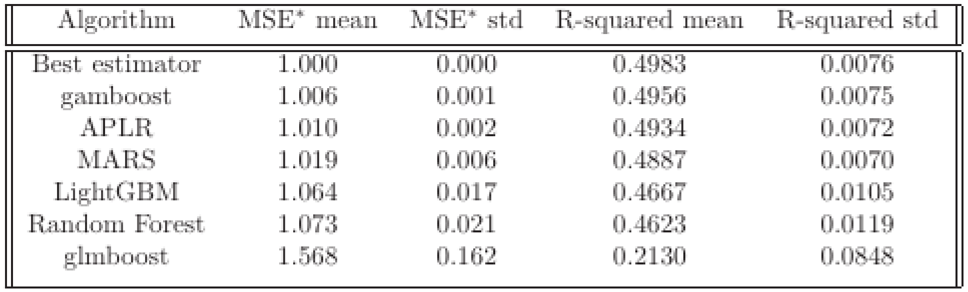
Test results (1)

- APLR had a predictiveness similar to other parametric algorithms in the simulated scenarios where the true model was additive. In these scenarios the tree-based algorithms predicted worse than APLR but still reasonably well



Algorithm	MSE* mean	MSE* std	R-squared mean	R-squared std
Best estimator	1.000	0.000	0.5010	0.0037
gamboost	1.006	0.001	0.4979	0.0035
APLR	1.009	0.002	0.4964	0.0034
MARS	1.018	0.008	0.4920	0.0029
LightGBM	1.050	0.006	0.4761	0.0037
Random Forest	1.084	0.021	0.4607	0.0096
glmboost	1.491	0.126	0.2561	0.0645

Table 4.1: Test results for the additive scenario with uncorrelated predictors.



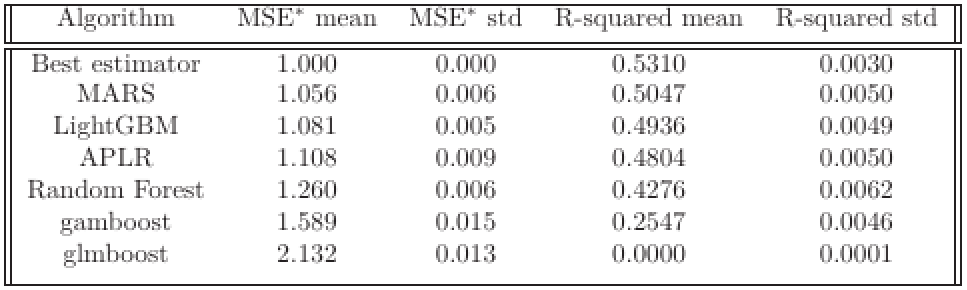
Algorithm	MSE* mean	MSE* std	R-squared mean	R-squared std
Best estimator	1.000	0.000	0.4983	0.0076
gamboost	1.006	0.001	0.4956	0.0075
APLR	1.010	0.002	0.4934	0.0072
MARS	1.019	0.006	0.4887	0.0070
LightGBM	1.064	0.017	0.4667	0.0105
Random Forest	1.073	0.021	0.4623	0.0119
glmboost	1.568	0.162	0.2130	0.0848

Table 4.2: Test results for the additive scenario with correlated predictors.

MSE* is the MSE relative to MSE for the best estimator

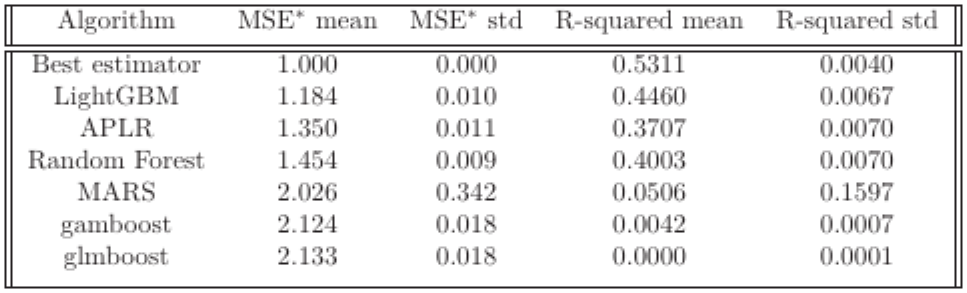
Test results (2)

- In the simulated scenario with uncorrelated predictors and a non-additive true model, MARS had the highest predictiveness, followed by LightGBM, APLR and Random Forest. However, APLR was not far behind MARS
- APLR was significantly more predictive than the other parametric algorithms in the simulated scenario with correlated predictors and a non-additive true model
- In both scenarios, APLR predicted better than Random Forest



Algorithm	MSE* mean	MSE* std	R-squared mean	R-squared std
Best estimator	1.000	0.000	0.5310	0.0030
MARS	1.056	0.006	0.5047	0.0050
LightGBM	1.081	0.005	0.4936	0.0049
APLR	1.108	0.009	0.4804	0.0050
Random Forest	1.260	0.006	0.4276	0.0062
gamboost	1.589	0.015	0.2547	0.0046
glmboost	2.132	0.013	0.0000	0.0001

Table 4.3: Test results for the non-additive scenario with uncorrelated predictors.



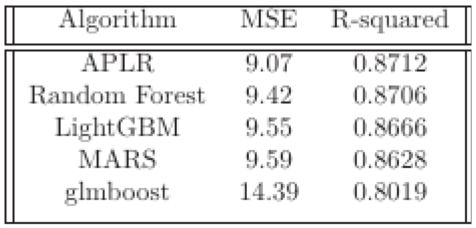
Algorithm	MSE* mean	MSE* std	R-squared mean	R-squared std
Best estimator	1.000	0.000	0.5311	0.0040
LightGBM	1.184	0.010	0.4460	0.0067
APLR	1.350	0.011	0.3707	0.0070
Random Forest	1.454	0.009	0.4003	0.0070
MARS	2.026	0.342	0.0506	0.1597
gamboost	2.124	0.018	0.0042	0.0007
glmboost	2.133	0.018	0.0000	0.0001

Table 4.4: Test results for the non-additive scenario with correlated predictors.

MSE* is the MSE relative to MSE for the best estimator

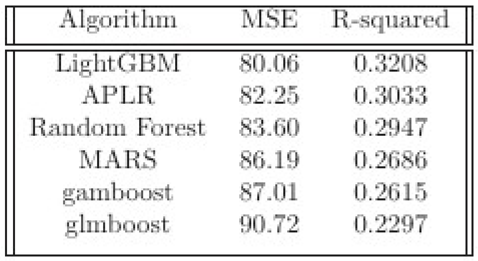
Test results (3)

- On the Auto MPG dataset APLR had a slightly better predictiveness than Random Forest, LightGBM and MARS. Those four were close on predictiveness. Glmboost was worst
- On the YearPredictionMSD dataset LightGBM predicted best, slightly ahead of APLR and Random Forest. MARS and gamboost were somewhat worse than APLR
- On the Individual household electric power consumption dataset LightGBM and Random Forest predicted slightly better than APLR. This was the only dataset where Random Forest beat APLR on predictiveness. Other parametric algorithms predicted somewhat worse



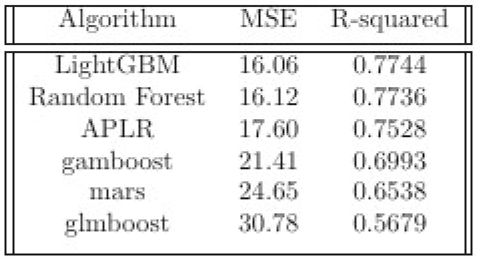
Algorithm	MSE	R-squared
APLR	9.07	0.8712
Random Forest	9.42	0.8706
LightGBM	9.55	0.8666
MARS	9.59	0.8628
glmboost	14.39	0.8019

Table 4.5: Test results for the Auto MPG datase



Algorithm	MSE	R-squared
LightGBM	80.06	0.3208
APLR	82.25	0.3033
Random Forest	83.60	0.2947
MARS	86.19	0.2686
gamboost	87.01	0.2615
glmboost	90.72	0.2297

Table 4.6: Test results for the YearPredictionMSD dataset



Algorithm	MSE	R-squared
LightGBM	16.06	0.7744
Random Forest	16.12	0.7736
APLR	17.60	0.7528
gamboost	21.41	0.6993
mars	24.65	0.6538
glmboost	30.78	0.5679

Table 4.7: Test results for the Individual household electric power consumption dataset.

Conclusion

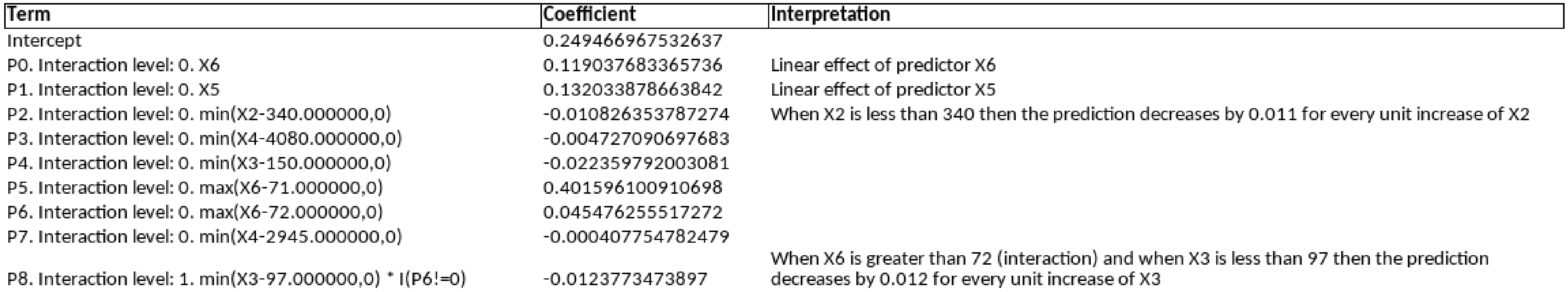
- Test results indicate that APLR may reduce the loss in predictiveness when increasing interpretability, as it seems to predict 1) better than other parametric algorithms when there are interactions in the data and 2) as accurately as other parametric algorithms when the true model is additive
- APLR looks competitive with Random Forest but slightly below LightGBM on predictiveness
- Predictiveness relative to the tree-based algorithms seems best when there are no interactions and deteriorates when the depth of interactions increases

Suggestions for further work

- Do more tests to find out when APLR predicts well relative to other algorithms and when it does not
  - Additional simulated scenarios with different types of interactions
  - Additional real datasets
- Develop a classification algorithm based on APLR
- Develop regression algorithms that use other parametric learners than piecewise linear basis functions (such as smoothing splines), but that automatically handle variable selection, interactions and non-linearity

Interpretation example of model terms

- Below is an example of how one can interpret model terms



Term	Coefficient	Interpretation
Intercept	0.249466967532637	
P0. Interaction level: 0. X6	0.119037683365736	Linear effect of predictor X6
P1. Interaction level: 0. X5	0.132033878663842	Linear effect of predictor X5
P2. Interaction level: 0. min(X2-340.000000,0)	-0.010826353787274	When X2 is less than 340 then the prediction decreases by 0.011 for every unit increase of X2
P3. Interaction level: 0. min(X4-4080.000000,0)	-0.004727090697683	
P4. Interaction level: 0. min(X3-150.000000,0)	-0.022359792003081	
P5. Interaction level: 0. max(X6-71.000000,0)	0.401596100910698	
P6. Interaction level: 0. max(X6-72.000000,0)	0.045476255517272	
P7. Interaction level: 0. min(X4-2945.000000,0)	-0.000407754782479	
P8. Interaction level: 1. min(X3-97.000000,0) * I(P6!=0)	-0.0123773473897	When X6 is greater than 72 (interaction) and when X3 is less than 97 then the prediction decreases by 0.012 for every unit increase of X3

- In APLR, the predictors can be assigned names (instead of X1, X2 and so on) for increased readability

Where to get APLR

- Run *pip install aplr* in a Python environment
- Available for Windows and most Linux distributions, but currently not available for Mac
- Documentation and code examples can be found on https://github.com/ottenbreit-data-science/aplr