Automatic Piecewise Linear Regression version 7.0.0

Von Ottenbreit Data Science

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 $j = 1, \ldots, p$, where p is the number of predictors in the covariate vector $\mathbf{x}$.
2. For each $m = 1$ to m_{stop} boosting steps:
   a) Compute the negative gradient:
   $$u_m = -\frac{\partial L(y, \hat{f}_{\text{m}-1}(\mathbf{x}))}{\partial \hat{f}_{\text{m}-1}(\mathbf{x})}$$
   where L is the loss function, y is the response variable and $\hat{f}_{\text{m}-1}(\mathbf{x})$ 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:
   $$\hat{f}_{\text{m}}(x_j) = \hat{f}_{\text{m}-1}(x_j) + v \cdot h_m(u_m, x_j)$$
   where v is a learning rate and $0 < v \leq 1$.
3. The final estimate is:
   $$\hat{f}_{\text{m}_{\text{stop}}}(\mathbf{x}) = \sum_{j=1}^{p} \hat{f}_{\text{m}_{\text{stop}}}(x_j)$$

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

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

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

  - *i* is another APLR basis function of a predictor x* with or without interactions
  - 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
  - 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

Useful functionality

- Allows to specify the loss function among several built-in options such as *mse* (default), *poisson*, *gamma*, etc.
- Allows to specify the link function among built in variants: *identity* (default), *logit* and *log*
- Allows to specify the function for calculating the validation set tuning metric among built-in variants such as *default* (same as loss function), *mse*, *negative_gini* etc
- Alternatively allows to pass custom Python functions for each of the above
- The user can specify a list of predictors to be prioritized. Terms based on these will be added or updated in each boosting step as long as training error is reduced. This can be particularly useful to include the full effect of a predictor that is weak but important
- The user can specify a list of predictors with monotonic constraints. This can improve model realism, usually with only a minimal loss increase
- The user can provide constraints on which predictors are allowed in interaction terms
- In addition to solving regression problems APLR can also be used for classification. The object APLRClassifier fits a logit APLR model for each response category. When predicting, each observation is predicted to the class with the highest predicted class probability

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
- The user can choose whether to 1) use a randomly selected proportion of the training data as a validation set, or 2) directly specify which observations should be used as a validation set

APLR fitting procedure: Boosting

- Each boosting step in APLR consists of these sub-steps:
  - Update the intercept term
  - Optionally add or update a term for each prioritized predictor if this reduces the training error. Requires that *prioritized_predictors_indexes* is provided as an argument to the *fit* method
  - Add or update a term that minimizes the training error the most. This is usually an APLR basis function with or without interactions, but can also be a linear effect
  - Prune model terms every *boosting_steps_before_pruning_is_done* boosting step
- 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 a new term without interactions from *E*, where *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 selects the best candidate from *E* for entry to the model. This can be an interaction term or not
- Then APLR tries to find new interaction terms. Up to *max_eligible_terms* terms already in the model are considered as interaction partners for each term from *E* without interactions
- Interaction terms that reduce the loss more than the best 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 new interaction term that reduces the loss the most is preferred over the previously selected best candidate from *E* if it reduces the loss more

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 1) specifies the maximum allowed interaction depth. Should be tuned by for example doing a grid search
- *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

Conclusions based on empirical tests

- 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 (mars, gamboost and glmboost) 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

Sources

- Von Ottenbreit, Automatic Piecewise Linear Regression, 2022