STK2100- Machine Learning and Statistical Methods for Prediction and Classification

Spring 2026

Boosting and Additive Trees

Covers Sections 10.1-10.3 and 10.9

Ingrid Hobæk Haff
Department of Mathematics
University of Oslo

Boosting methods

- We have seen that we can construct more flexible models for regression and classification through (generalised) additive models using basis functions (e.g. splines), neural networks and classification and regression trees (CART)
- CART models have many advantages, namely that
  - they are computationally efficient
  - they are invariant under strictly monotone transformations of individual covariates (no pre-processing necessary)
  - they automatically perform covariate selection, which is particularly relevant when there are many irrelevant covariates (that have no effect on the response)
- On the other hand, they seldom give predition performance comparable to the best performing of the other mentioned models

Boosting methods

- Boosting is a way to construct regression and classification models, that gives models with many of the same advantages as CART, but also often have very good prediction performance, see for instance https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2433761
- Boosting fits a model for $f(\mathbf{x})$ of the form

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m b(\mathbf{x}; \gamma_m),$$

- Here, the bs are a set of weak base learners with poor prediction performance on their own
- These base learners are obtained by sequentially applying a weak regression or classification algorithm to modified versions of the data, in order to gradually improve (boost) the model

Forward stagewise additive modelling

- The model parameters are β_m, γ_m, $m = 1, \ldots, M$
- The γ_ms are the parameters of the base learners $b(\cdot)$ and the β_ms are the weights that should be put on the different base learners
- Given data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ and a loss function $L(y, f(x))$, the aim is to estimate the parameters that minimise

$$\sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(\mathbf{x}_i; \gamma_m)\right)$$

Forward stagewise additive modelling

Boosting is a form of forward stagewise additive modelling, where each pair β_m, γ_m is fitted in turn for $m = 1, \ldots, M$ as

$$\begin{aligned}(\beta_m, \gamma_m) &= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L\left(y_i, \sum_{l=1}^{m-1} \beta_l b(\mathbf{x}_i; \gamma_l) + \beta b(\mathbf{x}_i; \gamma)\right) \\ &= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma)\right),\end{aligned}$$

keeping β_l, γ_l, $l = 1, \ldots, m-1$, that have already been fitted, fixed

---

Forward stagewise additive modelling

Initialise $f_0(\mathbf{x}) = 0$
for $m = 1, \ldots, M$ **do**
  Let $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma)\right)$
  Set $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}_i) + \beta_m b(\mathbf{x}_i; \gamma_m)$
end for

Loss functions

- For regression, the typical loss function is the squared error loss, which gives

$$\begin{aligned} L\left(y_i, f_{m-1}(\mathbf{x}_i) + \beta_m b(\mathbf{x}_i; \gamma_m)\right) &= \left(y_i - f_{m-1}(\mathbf{x}_i) - \beta_m b(\mathbf{x}_i; \gamma_m)\right)^2 \\ &= \left(r_{mi} - \beta_m b(\mathbf{x}_i; \gamma_m)\right)^2 \end{aligned}$$

  with $r_{mi} = y_i - f_{m-1}(\mathbf{x}_i)$ the residuals from the current model

- The largest residuals r_{mi} will influence the loss function the most
- Hence, the emphasis in stage m is to improve the fit to the data points $(y_i, \mathbf{x}_i)$ that have not yet been well captured by $f_{m-1}(\mathbf{x}_i)$, in order to gradually improve the model

Loss functions

- For binary classification, the response is encoded as $Y \in \{-1, 1\}$, and the typical loss function is the exponential loss

$$
\begin{aligned}
L\left(y_i, f_{m-1}(\mathbf{x}_i) + \beta_m b(\mathbf{x}_i; \gamma_m)\right) &= \exp\left(-y_i(f_{m-1}(\mathbf{x}_i) + \beta_m b(\mathbf{x}_i; \gamma_m))\right) \\
&= \exp\left(-y_i f_{m-1}(\mathbf{x}_i)\right) \cdot \exp\left(-y_i \beta_m b(\mathbf{x}_i; \gamma_m)\right) \\
&= w_{mi} \cdot \exp\left(-y_i \beta_m b(\mathbf{x}_i; \gamma_m)\right),
\end{aligned}
$$

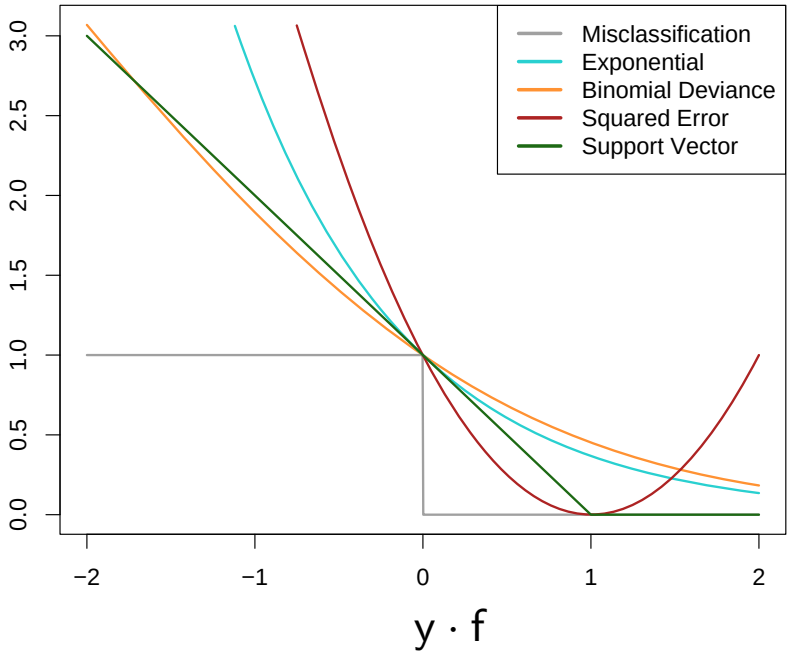
  where $w_{mi} = \exp\left(-y_i f_{m-1}(\mathbf{x}_i)\right)$ can be seen as a weight on data point $(y_i, \mathbf{x}_i)$
- The weights w_{mi} will be largest for the data points for which y_i and $f_{m-1}(\mathbf{x}_i)$ have opposite sign, i.e the ones that have been misclassified
- Hence, most of the effort is put into improving the fit on the misclassified data points, once more gradually improving the model in each stage
- It can be shown that the population minimiser of the exponential loss is the same as when using the deviance (negative log-likelihood) as the loss function

Loss functions

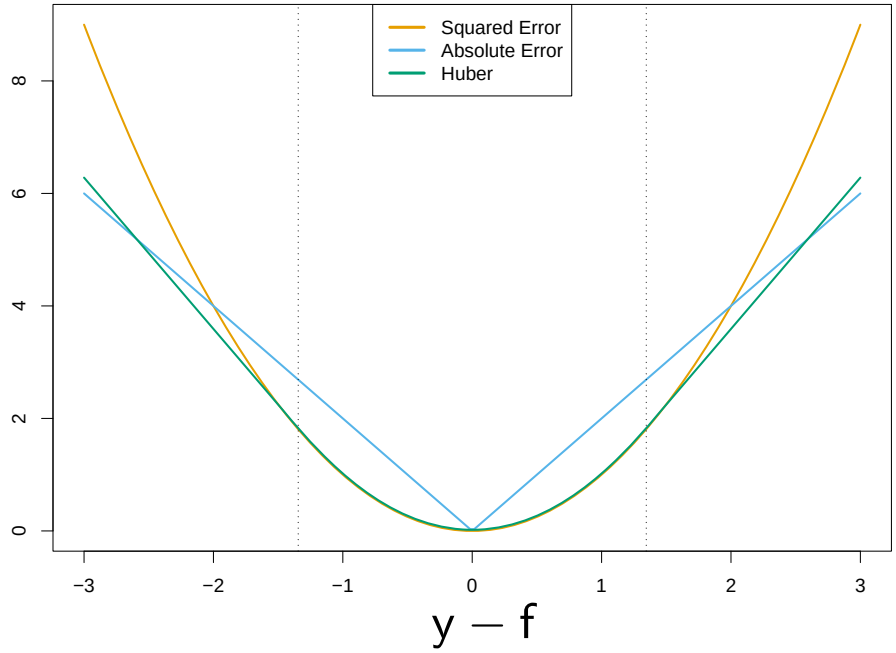
- There are many alternative loss functions, both for regression and classification
- Some of these are more robust to outliers
- For regression, some alternatives to the squared error loss are:
  - The absolute loss: $L(y, \hat{y}) = |y - \hat{y}|$
  - The Huber loss: $L(y, \hat{y}) = \begin{cases} (y - \hat{y})^2, & |y - \hat{y}| \leq \delta \\ 2\delta|y - \hat{y}| - \delta^2, & \text{otherwise} \end{cases}$
- For classification, some alternatives to the exponential loss are:
  - The misclassification loss: $L(y, \hat{y}) = I(y \neq \hat{y})$
  - The binomial deviance: $L(y, \hat{y}) = \log(1 + \exp(-2y\hat{y}))$
  - The squared error loss: $L(y, \hat{y}) = (y - \hat{y})^2$
  - The support vector loss: $L(y, \hat{y}) = (1 - y\hat{y})_+$

Comparison of loss functions for regression and classification

Classification



Regression



AdaBoost algorithm

- One of the most popular boosting algorithms is AdaBoost.M1, which is for binary classification
- This algorithm assumes that the weak learners $b(\cdot)$ are classifiers that return either -1 or 1
- It can be shown that AdaBoost.M1 is equivalent to forward stagewise additive modelling, using the exponential loss

---

AdaBoost.M1 algorithm

Initialise $w_i = 1/N$, $i = 1, \ldots, N$
for $m = 1, \ldots, M$ **do**
  Fit the classifier $b(\mathbf{x}_i; \gamma_m)$ to the data weighted with weights w_i
  Compute $\text{err}_m = \sum_{i=1}^N w_i I(y_i \neq b(\mathbf{x}_i; \gamma_m)) / \sum_{i=1}^N w_i$
  Compute $\beta_m = \frac{1}{2} \log((1 - \text{err}_m)/\text{err}_m)$
  Set $w_i \to w_i \cdot \exp\left(2\beta_m I(y_i \neq b(\mathbf{x}_i; \gamma_m))\right)$, $i = 1, \ldots, N$
end for
Return the classifier $\hat{y} = \text{sign}\left(\widehat{f}(\mathbf{x})\right) = \text{sign}\left(\sum_{m=1}^M \beta_m b(\mathbf{x}; \gamma_m)\right)$

Boosting trees

- The by far most common weak learner used in boosting algorithms is binary split trees, such as the regression and classification trees from Section 9.2
- The trees are typically smaller than the ones from Section 9.2, and of a pre-fixed size with J end-nodes
- Let $R_{m1}, \ldots, R_{mJ}$ be the regions the covariate space has been split into in stage m
- As each region has its own constant, one sets $\beta_m = 1$, $m = 1, \ldots, M$

Boosting trees

- Then, we have

$$b(\mathbf{x}; \gamma_m) = \sum_{j=1}^{J} \gamma_{mj} I(\mathbf{x} \in R_{mj})$$

and

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{mj} I(\mathbf{x} \in R_{mj})$$

- Given the regions, the γ_{mj}s are estimated as

$$\hat{\gamma}_{mj} = \arg\min_{\gamma_{mj}} \sum_{\{i:\mathbf{x}_i \in R_{mj}\}} L(y_i, f_{m-1}(\mathbf{x}_i) + \gamma_{mj})$$

Boosting trees

- In the regression setting with squared error loss, $b(\mathbf{x}; \gamma_m)$ is the (small) regression tree that minimises the current loss $\sum_{i=1}^{N}(y_i - f_{m-1}(\mathbf{x}_i))^2 = \sum_{i=1}^{N} r_{mi}^2$, i.e. best predicts the current residuals $r_{mi} = y_i - f_{m-1}(\mathbf{x}_i)$
- The γ_{mj}s are estimated by

$$\gamma_{mj} = \frac{1}{N_{mj}} \sum_{\{i:\mathbf{x}_i \in R_{mj}\}} y_i,$$

  with $N_{mj} = \sum_{i=1}^{N} I(\mathbf{x} \in R_{mj})$, and the splits of the tree are found with the same procedure as in Section 9.2 for growing large regression trees
- The main differences from regression trees are that
  - one grows much smaller trees
  - one fits a large number M of trees instead of just one
  - the trees are fitted to different data (residuals) in every stage m

Boosting trees

- In the classification setting with exponential loss, $b(\mathbf{x}; \gamma_m)$ is the (small) classification tree that minimises the weighted misclassification error $\sum_{i=1}^{N} w_{mi} I(y_i \neq \hat{y}_i)$, with weights $w_{mi} = \exp\left(-y_i f_{m-1}(\mathbf{x}_i)\right)$
- The γ_{mj}s are estimated by

$$\gamma_{mj} = \frac{1}{2} \log \frac{\sum_{i=1}^{N} w_{mi} I(y_i = 1)}{\sum_{i=1}^{N} w_{mi} I(y_i = -1)}$$

  with $N_{mj} = \sum_{i=1}^{N} I(\mathbf{x} \in R_{mj})$
- The splits of the tree are found with the same procedure as in Section 9.2 for growing large classification trees, but using the exponential loss as the splitting criterion

Boosting trees

- The size J of the trees used in boosting methods is typically rather small, usually ranging from $J = 2$ (stumps) to $J = 6$
- With size J, the maximum order of interaction effects is $J - 1$, meaning that $J = 2$, only gives main effects, $J = 3$ gives two-way interactions, and so on
- Hence, J should reflect the maximum interaction order one wants in the model
- Minimisation of the loss function L is often done with gradient descent, resulting in gradient boosting
- The most popular implementation of such a procedure is XGBoost (Extreme Gradient Boosting), which is for instance available in the `xgboost` in R

Boosting trees

- One stragegy to avoid overfitting is to introduce shrinkage, i.e. modify $f_m(\mathbf{x})$ to

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu \sum_{j=1}^{J} \gamma_{mj} I(\mathbf{x} \in R_{mj})$$

  with learning rate $0 < \nu < 1$
- Then, both ν and M have to be chosen in order to maximise prediction performance on new data
- They are not independent, as the optimal M will tend to be larger for smaller ν
- A good strategy is to set ν to a small value, more specifically $\nu < 0.1$, and then choose M by cross-validation

Example: E-mail spam data

See R code