

Sketch of solutions

STK-IN4300/9300 2022

Problem 1 Bias-variance trade-off

1a L_2 -Boosting

In the first two points we see that the bias of the L_2 -boosting operator decays exponentially fast for increasing m , while the variance has an exponentially slow increase. This represents an advantageous trade-off, which is not common to all procedures seen in the course. Since the mean squared error (MSE) is the sum of the squared bias and variance terms, we see that for $m \rightarrow \infty$ the MSE tends to σ^2 , the noise level, implying that it is not convenient to go on until theoretical convergence (it would only lead to overfitting). Instead, we want to ensure an early stop to avoid overfitting and obtain a lower MSE. The third point provides us with a condition under which boosting the smoother effectively improves over using the smoothing operator alone. The left-hand side can be interpreted as a signal-to-noise ratio, and we need it to be "large enough" to ensure it is larger than the right-hand side; the right-hand side, instead, needs to be "small enough": for this to happen, we want the term $1/(1 - \lambda_k)^2$ to be as small as possible, which means that λ_k should be as small as possible (we are also under the condition $0 < \lambda_k \leq 1$). Since λ_k for $k = 1, \dots, N$ are the eigenvalues of the smoothing operator, this means that the smoothing operator needs to be weak, and/or shrunk, in order for boosting to improve over using the smoother alone. These results are demonstrated not in general, but for boosting with the L_2 -loss and symmetric linear operators as base learners.

1b k-Nearest Neighbours

In kNN, the number of neighbours k is the tuning parameter that controls the complexity of the model. We know that, in general, increasing the model complexity entails a decrease in the bias and an increase in the variance (and viceversa when decreasing the model complexity). We observe this trade-off in the case of kNN: a higher k means a less complex model, hence a larger bias (since we include in the average also points that are far from the point where we are estimating) but a smaller variance (underfit) - the extreme example is considering all points as neighbours, where the estimate is then just an average over the whole sample; a lower k means a more complex model, resulting in a smaller bias but larger variance - the extreme case is $k = 1$, which has a training error equal to 0 but is characterized by a very large variance (overfit) as we are "averaging" using one data point only.

1c Smoothing splines

A spline that is trimming down the hills and filling in the valleys is underfitting the data in high curvature regions. This means that there is too much bias, and the complexity of the model should be increased. On the other hand, increasing the spline complexity will result in a higher variance due to the bias-variance trade-off: this is why it is important to ensure that the spline does not follow

the data too closely, i.e. avoid overfitting which leads to poor generalization ability.

Problem 2 Ensemble methods

2a Bagging

Bagging is an ensemble method based on bootstrap samples. The procedure is as follows:

- Given the original training set Z , a number B of bootstrap samples $Z_1^*, \dots, Z_B^*$ is generated;
- On each of the bootstrap samples, a tree is fitted;
- The results are aggregated, either via probability or via consensus.

In a classification problem with classes $\{1, \dots, K\}$, aggregating via *consensus* means choosing the class that received the highest number of votes (i.e. the class that was most frequently selected by the trees):

$$\hat{G}(x) = \operatorname{argmax}_k q_k(x)$$

where $k \in \{1, \dots, K\}$, and q_k is the proportion of trees voting for class k . Aggregating by *probability*, instead, means computing the probability of each class as an average over the B bootstrap samples, and then choosing the class with the highest resulting probability:

$$\hat{G}(x) = \operatorname{argmax}_k \left(\frac{1}{B} \sum_{b=1}^B p_k^{[b]}(x) \right)$$

where $p_k^{[b]}$ is the probability assigned to class k by the b -th tree.

The strength of the method is that averaging over bootstrap samples leads to a reduction of the variance component of the expected prediction error, in particular when using high-variance classifiers such as trees. Under squared-error loss, averaging reduces the variance while leaving the bias unchanged:

$$\begin{aligned} E_{\mathcal{P}}[(Y - \hat{f}^*(x))^2] &= E_{\mathcal{P}}[(Y - f_{bag}(x) + f_{bag}(x) - \hat{f}^*(x))^2] \\ &= E_{\mathcal{P}}[(Y - f_{bag}(x))^2] + E_{\mathcal{P}}[(f_{bag}(x) - \hat{f}^*(x))^2] \\ &\geq E_{\mathcal{P}}[(Y - f_{bag}(x))^2]. \end{aligned}$$

This does not apply in case of classification with 0-1 loss, due to non-additivity of bias and variance. In this case, bagging makes better a good classifier, and worst a bad one.

The limitation of bagging is that the pairwise correlation between bootstrap trees limits the advantages of reducing the variance by averaging. This is why the trees should be less correlated as possible (ideally, independent), and using random forest brings improvement over bagging.

2b Random Forest

- The first term on the right-hand side is the variance of the random forest estimator:

$$\text{Var}_Z[E_{\Theta|Z}[T(x; \Theta(Z))]] = \text{Var}_Z \hat{f}_{rf}(x) = \rho(x) \sigma^2(x)$$

where $\rho(x)$ is the sampling correlation between any pair of trees used in the averaging, and $\sigma^2(x)$ is the sampling variance of any single randomly drawn tree. Clearly, this term increases for increasing correlation, which in turns increases for m increasing. The second term is the the "within- Z " variance, which comes as a result of the randomization, and increases as m decreases. The coexistence of these two terms is what makes the choice of m non-trivial, in fact the best practice would be to select m via cross-validation.

- The left panel of the figure shows the concept expressed in the previous answer: we see that the "within- Z " variance decreases as m increases. Nonetheless, the total variance is increasing: this is because the sampling variance of the random forest ensemble increases for increasing m , and dominates over the "within- Z " variance. The right panel show the random forest bias-variance trade-off: as m increases, the bias is reduced but the variance increases, leading to an increased mean squared error. It is interesting to note that the squared bias of the ensemble is larger than that of a single fully grown tree (horizontal line in the right plot), but the variance is reduced, especially for small values of m .

2c Boosting

Since

$$\hat{f}^{[m]}(x_i) = \sum_{k=1}^m \hat{\beta}^{[k]} \hat{g}^{[k]}(x_i),$$

at the m -th step the exponential loss function is

$$L(y, f(x)) = \sum_{i=1}^N \exp[-y_i \sum_{k=1}^m \beta^{[k]} g^{[k]}(x_i)]$$

and $\hat{\beta}$ and $\hat{g}$ can be found as

$$\begin{aligned} (\hat{\beta}, \hat{g}) &= \underset{(\beta, g)}{\text{argmin}} \left\{ \sum_{i=1}^N \exp \left[-y_i \sum_{k=1}^m \beta^{[k]} g^{[k]}(x_i) \right] \right\} \\ &= \underset{(\beta, g)}{\text{argmin}} \left\{ \sum_{i=1}^N \exp \left[-y_i \left(\sum_{k=1}^{m-1} \beta^{[k]} g^{[k]}(x_i) + \beta g(x_i) \right) \right] \right\} \\ &= \underset{(\beta, g)}{\text{argmin}} \left\{ \sum_{i=1}^N w_i^{[m-1]} \exp[-y_i \beta g(x_i)] \right\} \end{aligned}$$

where $w_i^{[m-1]} = \exp[-y_i \hat{f}^{[m-1]}]$ does not depend on β and g in the m -th step.

• Minimization with respect to g

$$\begin{aligned}
\hat{g}^{[m]} &= \operatorname{argmin}_g \left\{ \sum_{i=1}^N w_i^{[m-1]} \exp[-y_i \beta g(x_i)] \right\} \\
&= \operatorname{argmin}_g \left\{ \sum_{i:g(x_i)=y_i} w_i^{[m-1]} \exp[-\beta \cdot 1] + \sum_{i:g(x_i) \neq y_i} w_i^{[m-1]} \exp[-\beta \cdot (-1)] \right\} \\
&= \operatorname{argmin}_g \left\{ \sum_{i:g_i=y_i} w_i^{[m-1]} e^{-\beta} + \sum_{i:g_i \neq y_i} w_i^{[m-1]} e^{-\beta} - \sum_{i:g_i \neq y_i} w_i^{[m-1]} e^{-\beta} + \sum_{i:g_i \neq y_i} w_i^{[m-1]} e^{\beta} \right\} \\
&= \operatorname{argmin}_g \left\{ \sum_{i=1}^N w_i^{[m-1]} e^{-\beta} + (e^{\beta} - e^{-\beta}) \sum_{i:g_i \neq y_i} w_i^{[m-1]} \right\} \\
&= \operatorname{argmin}_g \left\{ \sum_{i=1}^N w_i^{[m-1]} e^{-\beta} + (e^{\beta} - e^{-\beta}) \sum_{i=1}^N \mathbb{1}(y_i \neq \hat{g}(x_i)) w_i^{[m-1]} \right\} \\
&= \operatorname{argmin}_g \left\{ \underbrace{\sum_{i=1}^N \mathbb{1}(y_i \neq \hat{g}(x_i)) w_i^{[m-1]}}_{\text{weighted misclassification error}} \right\}
\end{aligned}$$

• Minimization with respect to β

$$\begin{aligned}
\hat{\beta}^{[m]} &= \operatorname{argmin}_{\beta} \left\{ \sum_{i=1}^N w_i^{[m-1]} \exp[-y_i \beta g(x_i)] \right\} \\
&= \operatorname{argmin}_{\beta} \left\{ \underbrace{\sum_{i:g_i=y_i} w_i^{[m-1]} e^{-\beta} + \sum_{i:g_i \neq y_i} w_i^{[m-1]} e^{\beta}}_{\mathcal{L}} \right\}
\end{aligned}$$

We compute the partial derivative of $\mathcal{L}$ wrt β and set it to 0:

$$\frac{\partial \mathcal{L}}{\partial \beta} = - \sum_{i:g_i=y_i} w_i^{[m-1]} e^{-\beta} + \sum_{i:g_i \neq y_i} w_i^{[m-1]} e^{\beta} \stackrel{!}{=} 0$$

Multiplying and dividing by e^{β} , we obtain:

$$\begin{aligned}
& - \sum_{i:g_i=y_i} w_i^{[m-1]} + e^{2\hat{\beta}} \sum_{i:g_i \neq y_i} w_i^{[m-1]} \stackrel{!}{=} 0 \\
\iff & e^{2\hat{\beta}} \sum_{i:g_i \neq y_i} w_i^{[m-1]} = \sum_{i:g_i=y_i} w_i^{[m-1]} \\
\iff & e^{2\hat{\beta}} \sum_{i:g_i \neq y_i} w_i^{[m-1]} = \sum_{i=1}^N w_i^{[m-1]} - \sum_{i:g_i \neq y_i} w_i^{[m-1]}
\end{aligned}$$

$$\begin{aligned}
\Longleftrightarrow e^{2\hat{\beta}} &= \frac{\sum_{i=1}^N w_i^{[m-1]} - \sum_{i:g_i \neq y_i} w_i^{[m-1]}}{\sum_{i:g_i \neq y_i} w_i^{[m-1]}} = \frac{\frac{\sum_{i=1}^N w_i^{[m-1]}}{\sum_{i=1}^N w_i^{[m-1]}} - \frac{\sum_{i:g_i \neq y_i} w_i^{[m-1]}}{\sum_{i=1}^N w_i^{[m-1]}}}{\frac{\sum_{i:g_i \neq y_i} w_i^{[m-1]}}{\sum_{i=1}^N w_i^{[m-1]}}} \\
&= \frac{1 - \text{err}}{\text{err}} \quad \text{with err} = \frac{\sum_{i:g_i \neq y_i} w_i^{[m-1]}}{\sum_{i=1}^N w_i^{[m-1]}}
\end{aligned}$$

giving

$$\hat{\beta} = \frac{1}{2} \log\left(\frac{1 - \text{err}}{\text{err}}\right)$$

- If we treated the minimization of the loss function as a "standard" minimization problem to be solved by e.g. gradient descent or other numerical methods, we would likely overfit. In fact, our aim is minimizing the loss *on the new, unseen data*, but we only have the training set to work with. For this reason, we use boosting to ensure an early stop in the minimization process: we do not want to reach the actual minimum, but stop a bit before. This amounts to accepting some bias to reduce the variance, resulting in a lower prediction error.

Problem 3 Projection pursuit and neural networks

3a Projection pursuit

Projection pursuit is an additive model in derived features of the inputs,

$$f(X) = \sum_{m=1}^M g_m(\omega_m^T X) = \sum_{m=1}^M g_m(V_m),$$

where X is a p -dimensional input vector, $\omega_1, \dots, \omega_M$ are p -dimensional unit vectors and $V_m = \omega_m^T X$ are the derived features. V_m is the projection of X onto ω_m : the algorithm searches for the best ω_m to fit the data. The function $g_m(\omega_m^T X)$ is called *ridge function* and varies only in the direction of ω_m - hence, it is estimated together with ω_m in the algorithm, usually with splines or local regression. Projection pursuit can be considered a *universal approximator* because, under a suitable choice of g_m and for M large, it is successful in approximating any continuous function in $\mathbb{R}^p$. However, since the model is complex, interpretability is lost (with the exception of the case $M = 1$) and should be used as a pure-prediction method.

The connection between projection pursuit and neural networks resides in both being sums of non-linearly transformed linear models. Instead of the non-parametric functions g_m of projection pursuit, for neural network we have the *activation function* σ (very often a sigmoid function) which is multiplied with a weight β_m . The formal connection is:

$$\begin{aligned}
g_m(\omega_m^T X) &= \beta_m \sigma(\alpha_{0,m} + \alpha_m^T X) = \\
&= \beta_m \sigma\left(\alpha_{0,m} + \frac{\|\alpha_m\|}{\|\alpha_m\|} \cdot \alpha_m^T X\right) = \beta_m \sigma(\alpha_{0,m} + \|\alpha_m\| \cdot \omega_m^T X),
\end{aligned}$$

where $\omega_m = \frac{\alpha_m}{\|\alpha_m\|}$ is, as before, the m -th unit vector. Equivalent formulations or a shorter notation such as $g_m(V) = \beta_m \sigma(V)$ with a brief explanation of the quantities are also valid solutions.

3b Activation functions

- The activation function is the non-linear transformation applied to the linear model constructed with the inputs. The idea behind it was to mimic what was believed to be the nerve system in a human body. In fact, a nerve was supposed to be "activated" after a certain threshold was reached, and the activation would stop after having received the signal: the same behaviour is reproduced e.g. by a sigmoidal activation function. Note that the activation function does not have to be non-linear, in principle, but a non-linear choice is what makes neural networks powerful on many kinds of different complex problems. Choosing $\sigma = I$, the identity, one ends up with a linear model in the inputs. Common choices of activation function nowadays, on top of the sigmoidal one, are the *arctan* and the *ReLU* (rectified linear) functions.
- The curve for $s = 1/2$ is the blue curve in the figure, while the purple one is obtained for $s = 10$. The parameter s controls the steepness of the activation curve: in fact, a larger s implies a hard activation in form of a step at $v = 0$. This is why this function is also called step or Heaviside function. On the other hand, for small values of s we get an activation curve approximating a linear function. In the limit where s tends to 0, the curve is flattened out and becomes an horizontal line at 0.5; when s tends to ∞ , instead, we get the step function which is 0 for $x \leq 0$ and 1 for $x > 0$. An equally acceptable solution is to define the step function as

Problem 4 Penalized regression

4a Smoothly Clipped Absolute Deviation (SCAD)

- The idea behind the SCAD algorithm is to have a version of lasso where larger coefficients are shrunk less severely. In addition, the solution becomes continuous. Another benefit of the SCAD algorithm is that it is an oracle and therefore $\sqrt{n}$ - consistent estimator.

4b Properties SCAD

- In the above figure, we see a comparison of the size of the theoretical size of a coefficient $\hat{\beta}$ in an ordinary least squares model on the x-axis versus its actual size in the lasso (left) or SCAD model (right). We see that in both cases, $\hat{\beta}$ is set to 0 if it is too small. If it is above a certain threshold,

then $|\hat{\beta}_{\text{lasso}}|$ is always smaller than $|\hat{\beta}_{\text{OLS}}|$ which is the dotted line in the middle of each graph. Similar to lasso, $\hat{\beta}$ needs to be sufficiently large or else it will be set to 0. If it is larger than that, it will be shrunk similarly to lasso. However, at a certain point, the shrinkage is reduced. If $|\hat{\beta}_{\text{SCAD}}|$ is sufficiently large, we recover the OLS estimates since the shrinkage is removed.¹

- The parameter α dictates at which point the coefficients are no longer shrunk as strongly as it would be the case with the typical lasso. This can also be seen in the figure in task 4b. For a certain value for $|y_i|$, the black line is no longer parallel to the dotted line in the center of the graph. Instead, it approaches the dotted line. We can see this in the second equation in 4a) as well. Furthermore, α also dictates, together with λ (see next subtask), when the coefficients are no longer shrunk. Increasing α means that the coefficients $|\beta_i|$ need to be larger in order to not be shrunk. For $\alpha \rightarrow \infty$, SCAD and lasso behave effectively the same way.
- If $|\beta| > \alpha\lambda$, it is no longer shrunk and the OLS estimate is recovered.

4c Oracle properties

There are two properties which an estimator should fulfil (asymptotically) in order to be an oracle estimator. On the one hand, the estimator should identify the right subset model. By that we mean that the true relevant variables are selected and that no meaningless variables are included.

On the other hand, the fitting procedure should converge to the right value for the coefficients. As an example, we have seen in the previous sub-task that lasso does not fulfill this condition. The absolute value of the coefficients is always less than its true value, except for the null case. In contrast, the coefficients of the SCAD procedure do converge asymptotically.

Stating the oracle properties in a mathematically correct manner (using words) is also a valid answer.

¹This is also the reason why SCAD is a $\sqrt{n}$ -consistent estimator (though this is not part of the expected answer).