Homework 3

36-708, Spring 2023

Due April 7 at 5PM EST

Please attach all code to your homework. In an RMarkdown document for example, this can be done in one line (see Yihui Xie's website for how to do this).

1 Deriving boosting-like variants for square and logistic losses

- Data: Assume n labeled data points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \{-1, 1\}$.
- Classifier: Denote by $f = \sum_{t=1}^{S} \alpha_t h_t$ the linear combination of base classifiers h_t with weights α_t. Here, S denotes the number of possible decision stumps on $x_1, \ldots, x_n$.
- Loss function: Let the loss function to be used in the boosting derivation be $F = \frac{1}{n} \sum_{i=1}^{n} \phi(-y_i f(x_i))$ for some function ϕ (to be specified). The 0-1 loss is $\phi_{0-1}(-u) = \mathbb{1}(u \leq 0)$ but we will use surrogates of this.
- Algorithm: *Boosting*, which we will derive from the point of view of coordinate descent on the loss function F.

1.1 AdaBoost as coordinate descent

In this problem, we will use $\phi_{\exp}(-u) := \exp(-u)$, the exponential loss. We will rederiving boosting as coordinate descent to recover AdaBoost.

(a) Write the objective function we wish to minimize: $F(\alpha) := \frac{1}{n} \sum_{i=1}^{n} \exp\left(-y_i \sum_{j=1}^{S} \alpha_j h_j(x_i)\right)$. Consider the "coordinate" of steepest descent:

$$\underset{k}{\operatorname{argmin}} \left.\frac{\partial F(\alpha + \eta e_k)}{\partial \eta}\right|_{\eta=0}.$$

Show that

$$\begin{aligned} \left.\frac{\partial F(\alpha + \eta e_k)}{\partial \eta}\right|_{\eta=0} &\propto \sum_{i=1}^{n} -y_i h_k(x_i) w_i \\ &= 2\varepsilon_{t,k} - 1 \end{aligned}$$

where w_i is a weight given by[1]

$$w_i := \frac{\exp\left\{-y_i \sum_{j=1}^{S} \alpha_j h_j(x_i)\right\}}{\sum_{i=1}^{n} \exp\left\{-y_i \sum_{j=1}^{S} \alpha_j h_j(x_i)\right\}}$$

[1]Note that this is the same as the weight discussed in the lecture on AdaBoost, but with an alternatively written normalization constant.

and $\varepsilon_{t,k}$ is the $(w_i)_{i=1}^n$-weighted 0-1 error for the k^{th} learner at time step t, given by

$$\varepsilon_{t,k} := \sum_{i=1}^n w_i \mathbb{1}(y_i \neq h_k(x_i)).$$

Finally, conclude that the argmin_k of the above is the weak learner with smallest weighted 0-1 error.

(b) Show that the step-size η is given by $\frac{1}{2}\log\left(\frac{1-\varepsilon_{t,k}}{\varepsilon_{t,k}}\right)$. That is, show that $\frac{1}{2}\log\left(\frac{1-\varepsilon_{t,k}}{\varepsilon_{t,k}}\right)$ is the solution to

$$\frac{\partial F(\alpha + \eta e_k)}{\partial \eta} = 0.$$

(c) Write the pseudo-code for AdaBoost.

1.2 SquareBoost using the squared loss

We will repeat the previous problem but for the squared loss, $\phi_2(-u) = (1-u)^2\mathbb{1}(u \leq 1)$.

(a) Consider the objective function:

$$F(\alpha) = \frac{1}{n}\sum_{i=1}^n \left(1 - y_i \sum_{j=1}^S h_j(x_i)\alpha_j\right)^2 \mathbb{1}\left(y_i \sum_{j=1}^S h_j(x_i)\alpha_j \leq 1\right).$$

Using similar steps to before, show that the minimizer of the above loss is the weak learner with best 0-1 loss on the weighted data with weights given by

$$w_i := \frac{(1 - y_i f(x_i))\mathbb{1}\left(y_i \sum_{j=1}^S h_j(x_i)\alpha_j \leq 1\right)}{\sum_{i=1}^n (1 - y_i f(x_i))\mathbb{1}\left(y_i \sum_{j=1}^S h_j(x_i)\alpha_j \leq 1\right)}.$$

(b) Unlike AdaBoost, the step size of SquareBoost cannot be computed in closed-form. How would you go about computing it?

(c) Write the pseudo-code for SquareBoost.

1.3 LogisticBoost using the logistic loss

We will repeat the previous problems but for the logistic loss, $\phi_\ell(-u) = \log(1 + e^{-u})$.

(a) Show that the minimizer of the logistic loss is given by the weak learner with smallest 0-1 loss on the weighted data where weights are given by

$$w_i = \frac{\operatorname{logit}^{-1}(-y_i f(x_i))}{\sum_{i=1}^n \operatorname{logit}^{-1}(-y_i f(x_i))}.$$

(b) Show how finding the step-size is equivalent to training a logistic regression model with and offset and no intercept. Explain how you could compute this step size.

(c) Write the pseudo-code for LogisticBoost.

1.4 Comparison of AdaBoost, SquareBoost, and LogisticBoost

(a) How do the three previous boosting algorithms differ?

2 Implementing and evaluating boosting

In this problem, we will implement the previous three boosting algorithms and apply them to the spam dataset.

- Dataset: We will use the spam dataset available here.
- Train and test split: Use a random 3:1 split.

(a) Implement AdaBoost, SquareBoost, and LogisticBoost with decision stumps as the base classifiers. Note that you do not have to minimize the weighted 0-1 loss exactly. *[Hint: In* R*, you can use the package* RPART *to find the best decision stump at each iteration efficiently. In python, you can use the function* SKLEARN.TREE.DECISIONTREECLASSIFER.*]*

(b) Run the boosting algorithms for various values of the number of boosting rounds t (for some reasonable upper limit T).

(c) Plot the train and test errors of all three algorithms as a function of t.

(d) Summarize your findings. In particular, comment on the following.

- How fast does the training error decrease?
- Does the training error reach zero? What happens to the testing error if you train longer?

(e) Is the weak learning hypothesis satisfied? If so, explain why. If not, are you able to modify the set of base classifiers so that it is?

3 Kernel calculus and the Gaussian kernel

To hack the so-called kernel trick into a machine learning algorithm, we need to be able come up with (or justify the choices of) valid kernel functions. One approach to do this is to first construct explicit feature maps and then get the kernels using corresponding inner products. An alternate approach is to directly construct kernels that are appropriate for a given application by building them out of simpler kernels using calculus of kernels. In this questions, we play around with some of such calculus rules and use them to show the validity of the popular Gaussian kernel. Consider the following setup.

- $K_1, K_2 : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and $K_3 : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ are any valid kernel functions.
- Constants: c_1, c_2 are nonnegative real constants
- Polynomial: p is a polynomial with nonnegative coefficients.
- Generic function: $\phi : \mathbb{R}^d \to \mathbb{R}^D$ is any function.
- Generic points: $u, v \in \mathbb{R}^d$.

Prove that the following are valid kernels:

(a) $c_1K_1(u, v) + c_2K_2(u, v)$,

(b) $K_1(u, v)K_2(u, v)$,

(c) $p(K_1(u, v))$,

(d) $\exp(K_1(u, v))$, and

(e) $K_3(\phi(u), \phi(v))$.

(f) Using the above, show that

$$K(u, v) = \exp\left(-\frac{\|u - v\|^2}{2\sigma^2}\right)$$

is a valid kernel.

4 Two flavours of kernel regression

In this question we explore two flavors of kernel regression. One is a local fit perspective and the other is a global fit perspective, but both of them lead to smoothers involving kernels.

4.1 Kernel regression

Consider the following local linear regression setup.

- Data: features $x_i \in \mathbb{R}^d$ and responses $y_i \in \mathbb{R}$ for $i = 1, \ldots, n$.
- Regression estimate: at a point $x \in \mathbb{R}^d$ the regression estimate is denoted by $\widehat{f}(x) = \widehat{\beta}^T x$ where

$$\widehat{\beta} = \operatorname*{argmin}_{\beta} \sum_{i=1}^{n} w_i(x) \left(y_i - \beta^T x_i\right)^2, \quad \text{and}$$

$$w_i(x) := \frac{K(x, x_i)}{\sum_{i=1}^{n} K(x, x_i)},$$

  for some kernel K.

(a) Verify that the objective function can be re-written as

$$(y - X\beta)^T \Omega(x)(y - X\beta)$$

where $y = \begin{pmatrix} y_1 & y_2 & \cdots & y_n \end{pmatrix}^T \in \mathbb{R}^n$ and $X = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix}^T \in \mathbb{R}^{n \times d}$, and $\Omega(x) = \operatorname{diag}(w_1(x), \ldots, w_n(x))$.

(b) Show that $\widehat{f}(x)$ is a linear combination of $\{y_i\}_{i=1}^n$.

4.2 Kernelized ridge regression

The nonparametric kernel regression in the question above performs a local fit around the test point. Let us now investigate the use of kernels for regression in another way. We apply the kernel trick in regular regression as follows. Note that this produces a global fit to the data. Consider the following ridge regression setup.

- Data: features $x \in \mathbb{R}^d$ and responses $y_i \in \mathbb{R}$ for $i = 1, \ldots, n$.
- Feature map: we use a feature mapping function ϕ to map the original d-dimensional feature vector x to a new D-dimensional feature vector $\phi(x)$, where $D \gg d$.
- Regression estimate: the regression estimate at a point $x \in \mathbb{R}^d$ is denoted by $\widehat{f}(x) := \widehat{\beta}^T \phi(x)$, where

$$\widehat{\beta} = \operatorname*{argmin}_{\beta} \frac{1}{2}\|y - \Phi\beta\|^2 + \frac{\lambda}{2}\|\beta\|^2,$$

  and $\Phi := \begin{pmatrix} \phi(x_1) & \ldots & \phi(x_n) \end{pmatrix}^T \in \mathbb{R}^{n \times D}$ denotes the kernel design matrix and λ is a regularization parameter.

(a) Show that we can write $\widehat{\beta} = \Phi^T(\Phi\Phi^T + \lambda I_n)^{-1} y$ where I_n is the identity matrix.

(b) Show that we can alternately write $\widehat{\beta} = (\Phi^T\Phi + \lambda I_D)^{-1}\Phi^T y$.

(c) In practice, designing ϕ is nontrivial, and when it is designed, we must compute the inner product $\phi(x)^T\phi(x')$ which may be costly. Instead we may want to use the kernel $K(x, x') = \phi(x)^T\phi(x')$ directly if it is computationally cheaper. Show how to 'kernelize' ridge regression (both in the training and evaluation phases) in the sense that neither $\phi(x)$ nor $\phi(x)^T\phi(x')$ would need to be computed, but $K(x, x')$ would. In particular, show that the predicted value at a new test point $x^\star$ can be computed as

$$y^\star = y^T(M + \lambda I_n)^{-1}c(x^\star)$$

for some matrix $M \in \mathbb{R}^{n\times n}$ and vector $c(x^\star) \in \mathbb{R}^n$. Write down expressions for M and $c(x)$ in terms of the kernel.