beta_gradient_boosting

Brayden Tang

06/01/2020

Beta Gradient Boosting

Suppose we have a response variable y that can take on any real value in the range (0, 1). Examples of such variables are proportions or rate/frequency distributions in which the exposures or counts are not known.

If we choose to arbitrarily fit such a response with an objective function such as least squares (equivalent to assuming the conditional distribution of Y | X is normally distributed) then there is no guarantee that the predicted response is positive and bounded in [0, 1]. A common workaround for this problem is to transform the response variable to the interval $[-\infty, \infty]$ using the logit link function, letting $Y_{transformed} = ln(\frac{Y}{1-Y})$ and then minimizing least squares on this transformed variable. However, there are shortcomings to this approach. For prediction purposes, the main problem is that setting $E[ln(\frac{Y}{1-Y})|X] = f(x)$ is not the same thing as setting $ln(\frac{E[Y|X]}{1-E[Y|X]}) = f(x)$. The first methodlogy in particular will almost always overestimate $E[Y|X]$ after transformation back to the unit interval due to consequences of Jensen's inquality.

Unfortunately, there is currently no objective function in any of the popular gradient boosting packages that directly deal with the scenario above, despite beta regression being around conceptually for quite some time now.

Maximum Likelihood Estimation

Gradient boosting frameworks like LightGBM and XGBoost both allow for a wide range of loss functions to be chosen for different response variables. Examples include the gamma, Tweedie, Poisson, and L2/Gaussian losses amongst many others. We focus here on these three particular loss functions because they can be derived by maximum likelihood in a similar fashion to that of generalized linear models.

XGBoost and LightGBM both determine their splits (and optimal weights to each terminal node) through closed form expressions using the gradient and Hessian of a loss function L. Thus, we can simply derive the likelihood function for a particular observation y_i and we are done.

Let $y_i \sim$ Gamma with parameters α, θ_i. We are letting α be the same for all observations, for reasons to be discussed below. Then, the density function of y_i is defined as:

$$f(y_i) = \frac{y_i^{\alpha-1} e^{\frac{-y_i}{\theta_i}}}{\Gamma(\alpha) \times \theta_i^{\alpha}}.$$

Now, $E[y_i] = \alpha\theta_i$. If we let $\mu_i = \alpha\theta_i$, then rewrite $f(y_i)$ as:

$$f(y_i) = \frac{y_i^{\alpha-1} e^{\frac{-y_i \alpha}{\mu_i}}}{\Gamma(\alpha) \times (\frac{\mu_i}{\alpha})^{\alpha}}.$$

Typically, we also replace the shape parameter α with the dispersion parameter that satisfies $\phi = \frac{V(\mu)}{Var[Y|X]} \rightarrow \phi = \alpha$ (for this parameterization since the variance function, $V(\mu) = \mu^2$ in this case). However, our predictions will of course just be μ and therefore we have no need to estimate $Var[Y|X]$ (for pure prediction purposes), so we can let α be any arbitrary constant. Let $\alpha = 1$ for convenience. Then:

$$f(y_i) = \frac{e^{\frac{-y_i}{\mu}}}{\mu}.$$

As an aside, this is just an exponential distribution since a Gamma with $\alpha = 1$ is an exponential.

Notice how this expression does not yet relate to the model output given by the model $f(X_i)$. If we use the log link function which is common for the gamma since it respects the support of the gamma distribution,

$$\mu = e^{f(X_i)},$$

then

$$ln(f(y_i)) = \frac{-y_i}{\mu} - ln(\mu) \rightarrow \frac{y_i}{e^{f(X_i)}} - f(X_i)$$

Hence,

$$-\nabla ln(f(y_i)) = 1 - \frac{y_i}{e^{f(X_i)}}$$

and

$$-\nabla^2 ln(f(y_i)) = \frac{y_i}{e^{f(X_i)}}.$$

These expressions are exactly what the package LightGBM uses when we specify `objective = "gamma"`. All of the other distributions can be derived in a similar fashion.

Derivation for Beta Distribution

First, we write the beta density in terms of the parameters μ and ϕ in a similar fashion to what we did above.

Let

$$f(y_i) = \frac{\Gamma(p+q)}{\Gamma(p)\Gamma(q)} y_i^{p-1}(1-y_i)^{q-1},$$

where $0 < y_i < 1$.

Since $\mu = \frac{p}{p+q}$ and $\phi = p + q$,

then:

$$f(y_i) = \frac{\Gamma(\phi)}{\Gamma(\mu_i\phi)\Gamma((1-\mu_i)\phi)} y_i^{\mu_i\phi-1}(1-y_i)^{(1-\mu_i)\phi-1},$$

As before, we will let the dispersion parameter be some arbitrary constant since it is not needed if we just care about $\mu = E[Y|X]$. For convenience, set $\phi = 1$. Then we have

$$f(y_i) = \frac{y_i^{\mu_i-1}(1-y_i)^{-\mu_i}}{\Gamma(\mu_i)\Gamma(1-\mu_i)}$$

.

Thus,

$$ln(f(y_i)) = (\mu_i - 1)ln(y_i) - \mu_i ln(1 - y_i) - ln(\Gamma(\mu_i)) - ln(\Gamma(1 - \mu_i)).$$

Using the link function $\mu_i = \frac{e^{f(X_i)}}{1+e^{f(X_i)}}$, we end up with the following expression:

$$ln(f(y_i)) = \left(\frac{-1}{1 + e^{f(X_i)}}\right) ln(y_i) - \left(\frac{e^{f(X_i)}}{1 + e^{f(X_i)}}\right) ln(1 - y_i) - ln\left(\Gamma\left(\frac{e^{f(X_i)}}{1 + e^{f(X_i)}}\right)\right) - ln\left(\Gamma\left(\frac{1}{1 + e^{f(X_i)}}\right)\right).$$

First, we will derive some needed quantities.

$$\frac{d}{df(x_i)} \frac{e^{f(X_i)}}{1 + e^{f(X_i)}} = \frac{e^{f(X_i)}}{(1 + e^{f(X_i)})^2},$$

$$\frac{d}{df(x_i)} \frac{1}{1 + e^{f(X_i)}} = -\frac{e^{f(X_i)}}{(1 + e^{f(X_i)})^2},$$

$$\frac{d^2}{df(x_i)^2} \frac{e^{f(X_i)}}{(1 + e^{f(X_i)})^2} = \frac{e^{f(X_i)}(1 - e^{f(X_i)})}{(1 + e^{f(X_i)})^3}.$$

We also need to take the derivative of $ln(\Gamma(x))$ which is denoted as $\psi(x)$. This is often referred to as the digamma function. For the Hessian, we will also need the second derivative of $ln(\Gamma(x))$, known as the trigamma function. Denote this as $\psi^2(x)$

We now calculate the gradient and Hessian as required.

$$\nabla ln(f(y_i)) = \frac{e^{f(X_i)}}{(1 + e^{f(X_i)})^2} ln(y_i) - \frac{e^{f(X_i)}}{(1 + e^{f(X_i)})^2} ln(1-y_i) - \psi\left(\frac{e^{f(X_i)}}{1 + e^{f(X_i)}}\right) \frac{e^{f(X_i)}}{(1 + e^{f(X_i)})^2} + \psi\left(\frac{1}{1 + e^{f(X_i)}}\right) \frac{e^{f(X_i)}}{(1 + e^{f(X_i)})^2} =$$

$$\frac{e^{f(X_i)}}{(1 + e^{f(X_i)})^2} \left(\psi\left(\frac{1}{1 + e^{f(X_i)}}\right) - \psi\left(\frac{e^{f(X_i)}}{1 + e^{f(X_i)}}\right) + ln\left(\frac{y_i}{1 - y_i}\right)\right).$$

Hence,

$$-\nabla ln(f(y_i)) = -\frac{e^{f(X_i)}}{(1 + e^{f(X_i)})^2} \left(\psi\left(\frac{1}{1 + e^{f(X_i)}}\right) - \psi\left(\frac{e^{f(X_i)}}{1 + e^{f(X_i)}}\right) + ln\left(\frac{y_i}{1 - y_i}\right)\right).$$

For the Hessian,

$$-\nabla^2 ln(f(y_i)) = \frac{e^{2f(X_i)}}{(1 + e^{f(X_i)})^4} \left(\psi^2\left(\frac{1}{1 + e^{f(X_i)}}\right) + \psi^2\left(\frac{e^{f(X_i)}}{1 + e^{f(X_i)}}\right)\right) - \frac{e^{f(X_i)}(1 - e^{f(X_i)})}{(1 + e^{f(X_i)})^3} \left(\psi\left(\frac{1}{1 + e^{f(X_i)}}\right) - \psi\left(\frac{e^{f(X_i)}}{1 + e^{f(X_i)}}\right)\right.$$

Implementation in R/Python

We write functions that take in two arguments: a vector of real values preds = $f(X_i)$ and a LightGBM matrix representing the training data set.

```
library(tidyverse)
library(lightgbm)

beta_loss <- function(preds, dtrain) {

  labels <- getinfo(dtrain, "label")

  grad <- (exp(preds) / ((1 + exp(preds))^2)) * (
    digamma(1 / (1 + exp(preds))) -
    digamma(exp(preds) / ((1 + exp(preds))^2)) +
    log(labels/ (1 - labels))
    )

  hess <- (exp(preds)*(1 - exp(preds)) / (1 + exp(preds))^3) * (
    digamma(1 / (1 + exp(preds))) -
    digamma(exp(preds) / ((1 + exp(preds))^2)) +
    log(labels/ 1 - labels)
    ) -
    (exp(2 * preds) / ((1 + exp(preds))^4)) * (
    trigamma(1 / (1 + exp(preds))) +
    trigamma(exp(preds) / (1 + exp(preds)))
    )

  return(list(grad = -grad, hess = -hess))

}
```

In Python:

```
from scipy.special import polygamma
import numpy as np

def beta_loss(preds, dtrain):
  labels = dtrain.get_label()

  grad = ((np.exp(preds)/((1 + np.exp(preds))^2)) * (
    polygamma(0, 1/(1 + np.exp(preds))) -
    polygamma(0, np.exp(preds)/((1 + np.exp(preds))^2)) +
    np.log(labels/(1-labels))
    ))

  hess = ((np.exp(preds)*(1 - np.exp(preds)) / (1 + np.exp(preds))^3) * (
    polygamma(0, 1 / (1 + np.exp(preds))) -
    polygamma(0, np.exp(preds) / ((1 + np.exp(preds))^2)) +
    np.log(labels/ (1 - labels))
    ) -
    (np.exp(2 * preds) / ((1 + np.exp(preds))^4)) * (
    polygamma(1, 1 / (1 + np.exp(preds))) +
    polygamma(1, np.exp(preds) / (1 + np.exp(preds)))
    )
    )


  return -grad, -hess
```