Assignment3

Cyril 20109216

20th of April 2020

Question a

We are interested in the following:

$$\begin{aligned} &\arg\min_{\boldsymbol{f}(\boldsymbol{x})} && E_{\boldsymbol{Y}|\boldsymbol{x}} \exp\left(-\frac{1}{K}(Y_1 f_1 + \cdots + Y_K f_K)\right) \\ &\text{subject to} && f_1 + \cdots + f_K = 0. \end{aligned}$$

We can derive the following population minimizer and derive class probabilities as such:

The Lagrange of this constrained optimization problem can be written as:

$$\exp\left(-\frac{f_1(\boldsymbol{x})}{K-1}\right)\text{Prob}(c=1|\boldsymbol{x}) + \cdots + \exp\left(-\frac{f_K(\boldsymbol{x})}{K-1}\right)\text{Prob}(c=K|\boldsymbol{x}) - \lambda\left(f_1(\boldsymbol{x}) + \cdots + f_K(\boldsymbol{x})\right),$$

where λ is the Lagrange multiplier. Taking derivatives with respect to f_k and λ, we reach

$$\begin{aligned} -\frac{1}{K-1}\exp\left(-\frac{f_1(\boldsymbol{x})}{K-1}\right)\text{Prob}(c=1|\boldsymbol{x}) - \lambda &= 0, \\ &\vdots \quad \vdots \\ -\frac{1}{K-1}\exp\left(-\frac{f_K(\boldsymbol{x})}{K-1}\right)\text{Prob}(c=K|\boldsymbol{x}) - \lambda &= 0, \\ f_1(\boldsymbol{x}) + \cdots + f_K(\boldsymbol{x}) &= 0. \end{aligned}$$

Solving this set of equations, we obtain the population minimizer

$$f_k^*(\boldsymbol{x}) = (K-1)\left(\log \text{Prob}(c=k|\boldsymbol{x}) - \frac{1}{K}\sum_{k'=1}^{K}\log\text{Prob}(c=k'|\boldsymbol{x})\right), \quad k=1,\ldots,K. \tag{4}$$

Thus,

$$\arg\max_k f_k^*(\boldsymbol{x}) = \arg\max_k \text{Prob}(c=k|\boldsymbol{x}),$$

which is the Bayes optimal classification rule in terms of minimizing the misclassification error. This justifies the use of this multi-class exponential loss function. Equation (4) also provides a way to recover the class probability $\text{Prob}(c=k|\boldsymbol{x})$ once $f_k^*(\boldsymbol{x})$'s are estimated, i.e.

$$\text{Prob}(c=k|\boldsymbol{x}) = \frac{e^{\frac{1}{K-1}f_k^*(\boldsymbol{x})}}{e^{\frac{1}{K-1}f_1^*(\boldsymbol{x})} + \cdots + e^{\frac{1}{K-1}f_K^*(\boldsymbol{x})}}, \quad k=1,\ldots,K.$$

Question b

Adaboost algorithm is the following:

Algorithm 1 *AdaBoost (Freund & Schapire 1997)*

1. *Initialize the observation weights* $w_i = 1/n,\ i = 1, 2, \dots, n$.
2. *For* $m = 1$ *to* M*:*
   (a) *Fit a classifier* $T^{(m)}(\boldsymbol{x})$ *to the training data using weights* w_i.
   (b) *Compute*
   $$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$
   (c) *Compute*
   $$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}}.$$
   (d) *Set*
   $$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),\ i = 1, 2, \dots, n.$$
   (e) *Re-normalize* w_i.
3. *Output*
   $$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$$

Whereas the multiclass boosting algorithm is as follows:

Algorithm 2 *SAMME*

1. *Initialize the observation weights* $w_i = 1/n,\ i = 1, 2, \dots, n$.
2. *For* $m = 1$ *to* M*:*
   (a) *Fit a classifier* $T^{(m)}(\boldsymbol{x})$ *to the training data using weights* w_i.
   (b) *Compute*
   $$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$
   (c) *Compute*
   $$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(K - 1). \tag{1}$$
   (d) *Set*
   $$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),\ i = 1, \dots, n.$$
   (e) *Re-normalize* w_i.
3. *Output*
   $$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$$

Hence, we can conclude that the special case of the multiclass algorithm reduce to Adaboost as in Step (c), the term log(K-1) where K=2 as in Adaboost vanishes.

Question c

```
error = function (w, misses)
{
    sum(w * misses)
}

alpha = function (err, K)
{
    if (err > 0.5) {
        1
    }
    else if (err <= 0) {
        20
    }
    else {
        log((1 - err)/err) + log(K - 1)
    }
}

weights = function (w, a, misses)
{
    tmp_w <- w * exp(a * misses)
    tmp_w/(sum(tmp_w))
}
samplePrediction = function (sample, A)
{
    votes.table <- votesTable(sample, A)
    prediction(votes.table)
}
weightedClassVote = function (k, sample, A)
{
    sum(A * (sample == k))
}
votesTable = function (sample, A)
{
    classes <- unique(sample)
    sapply(classes, function(k) weightedClassVote(k, sample,
        A))
}
prediction = function (votes.table)
{
    max.votes <- names(which(votes.table == max(votes.table)))
    ifelse(length(max.votes) == 1, max.votes, sample(max.votes,1))
}
finalPrediction  = function (C, A)
{
    apply(C, 1, function(sample) samplePrediction(sample, A))
}
```

And finally the algorithm:

```
samme = function (formula, data, test, m = 5)
{
```

```
  outcome.label <- toString(formula[[2]])
  y <- data[, outcome.label]
  C <- matrix(nrow = nrow(test), ncol = m)
  n <- nrow(data)
  w <- rep(1/n, n)
  A <- numeric(m)
  K <- nlevels(y)
  for (i in 1:m) {
    t <- data[sample(n, n, replace = T, prob = w), ]
    t[, outcome.label] <- droplevels(t[, outcome.label])
    fit <- ranger(formula, data = t)
    C[, i] <- as.character(predict(fit, test)$predictions)
    h <- predict(fit, data)$predictions
    levels(h) <- levels(y)
    misses <- as.numeric(h != y)
    err <- error(w, misses)
    A[i] <- alpha(err, K)
    w <- weights(w, A[i], misses)
  }
  finalPrediction(C, A)
}
```

Load data

```
data(iris)
colnames(iris)[length(colnames(iris))] = 'y'
train_index <- sample(1:nrow(iris), nrow(iris)*0.75)
train = iris[train_index,]
test = iris[-train_index,]
```

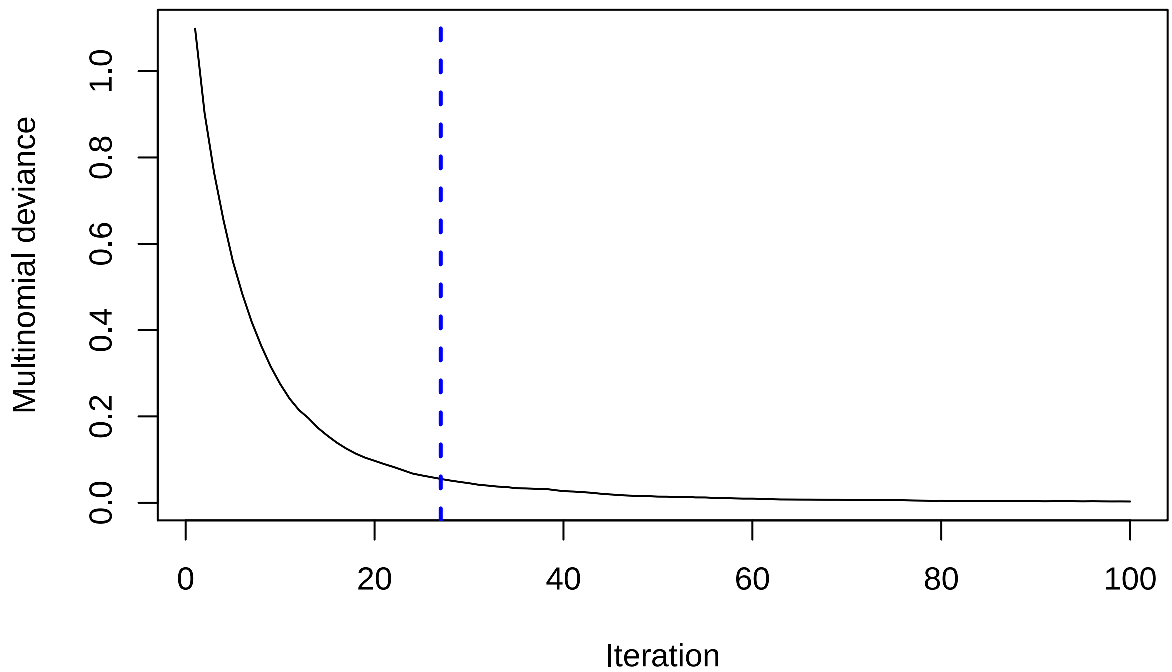
Results

```
preds_same = samme(y ~., train,test)
model_gbm = gbm(y ~.,data= train)
```

```
## Distribution not specified, assuming multinomial ...
```

```
best.iter <- gbm.perf(model_gbm, method = "OOB")
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance is rea
```




```
preds_gbm = predict(model_gbm,newdata=test, n.trees = best.iter,type='link')
preds_gbm2 = matrix(preds_gbm,ncol=nlevels(train$y))
colnames(preds_gbm2) = colnames(preds_gbm[,,1])
preds_gbm2 = apply(preds_gbm2,1,function(x) colnames(preds_gbm2)[which.max(x)])
acc_gbm = accuracy(preds_gbm2,test$y)
acc_same = accuracy(preds_same,test$y)
print(acc_gbm)
```

```
## [1] 0.9210526
```

```
print(acc_same)
```

```
## [1] 0.9210526
```

Hence, we can conclude that both algorithms have nearly similar performance.

Sources https://web.stanford.edu/~hastie/Papers/samme.pdf https://www.cs.ucr.edu/~eamonn/time_series_data/