Bagged **L**ogistic Regr**ess**ion (BLESS) Tutorial

Kyle Gardiner, Xuekui Zhang, Li Xing

2024-02-08

Introduction

Bagged **L**ogistic Regr**ess**ion (BLESS) in an ensemble algorithm that is inspired by the bagging procedures used by other machine learning algorithms like Random Forest. The algorithm works by bootstrapping data to fit logistic regression as its base learner over multiple iterations. From these iterations, logistic p-values are aggregated and subject to False Discovery Rate (FDR) multiple testing adjustment. Using the adjusted p-values, we identify any features that surpass a threshold of 0.05 to provide a ranked list of features related to the outcome of interest.

Example Data

Example data has been provided (`data.rda`) and can be found in the GitHub repository **here**. The dimensions of this data are 197 x 1011. The data frame contains 197 observations with 1000 features (`X.1-X.1000`), 10 covariates (`Cov.1-Cov.10`), and the binary outcome variable (`Y`)

```
load("data.rda")
BLESS_data[1:5, 1:5]
```

```
##   X.1 X.2 X.3 X.4 X.5
## 1   2   2   1   2   1
## 2   1   1   2   2   1
## 3   1   1   1   1   2
## 4   2   2   0   1   1
## 5   2   2   2   1   1
```

```
BLESS_data[1:5, c(1001:1004, 1011)]
```

```
##   Cov.1 Cov.2      Cov.3      Cov.4 Y
## 1 78.62     0 -0.0262782 -0.0160650 0
## 2 71.51     0 -0.0249967 -0.0195297 0
## 3 77.01     1 -0.0247467 -0.0130260 1
## 4 69.00     0 -0.0260312 -0.0121795 0
## 5 67.66     1 -0.0264302 -0.0201746 0
```

Model Fitting

Data Setup

First, we need to setup an empty dataframe to store the feature names and p-values from each iteration of the logistic regression process.

```
feature.df<-data.frame(Feature = character(0), pvalues=numeric(0))
```

BLESS Algorithm

The data mentioned above is used as an example to showcase the procedure of applying BLESS. For BLESS, we create a loop that will subsample the data (both observations and predictors), build a logistic regression model using the selected data, store the output of the logistic model (feature name and p-values), apply FDR multiple testing adjustments, and identify any features that surpass a threshold of 0.05.

```
for(ii in 1:1000){ # 1000 is the number of iterations performed.
                   # This number can be changed depending on the data being analyzed

  # Setting seed each iteration for reproducibility
  set.seed(ii)

  # Subsampling Observations and Predictors
  ## Randomly subsampling 90% of observations
  obs.id<-sample(nrow(BLESS_data),round(nrow(BLESS_data)*0.9), replace=FALSE)

  ## Randomly subsampling 30 predictors
  ### 30 predictors is roughly square root the numbers of predictors
  #### In this case, column 1011 is the outcome variable
  sample.SNPs<-sample(ncol(BLESS_data[,-1011]),30, replace=FALSE)

  ## Combining selected observations and predictors with the outcome Y
  ### Again, column 1001 is the outcome variable
  sub.data<-BLESS_data[obs.id,c(sample.SNPs,1011)]


  # Fitting a logistic model with each subset data
  myfit<-glm(Y~., data=sub.data, family="binomial")


   # Extracting the feature names and p-values for this logistic regression model
  selected.feature<-row.names(coef(summary(myfit))[-1,])
  selected.pvalue<-coef(summary(myfit))[-1,4]


  # Storing the extracted features and p-values in a temporary dataframe
  temp.feature.df<-data.frame(Feature=selected.feature, pvalue=selected.pvalue)

  # Binding all the outputs from the logistic regression models over all iterations
  feature.df<- rbind(feature.df, temp.feature.df)
}
```

Now we have a large dataframe containing the results (feature name and p-values) from each iteration. From here, we need to group the data by the feature name. We will achieve this by using the `split()` function from the `dpylr` package.

```
grouped <- split(feature.df, feature.df$Feature)
```

Once we have grouped the results by feature, we apply FDR testing adjustment and set a threshold of 0.05 to identify features associated with the outcome. This code works by dynamically adjusting the p-values by accounting for the fluctuating number of comparisons for each feature.

First, create an empty dataframe to store the adjusted p-values

```
adj.feature<-data.frame(Feature=character(0), adj.pvalue=numeric(0))
```

Now create a loop to apply FDR adjustment to each list/feature of the `grouped` object and extract the feature's name and minimum adjusted p-value within the group for any corrected p-values < 0.05

```
for(ss in 1:length(grouped)){ # looping for how many features are in the `grouped` object

   # applying FDR testing correction to the grouped feature p-values
  adj.p.value<-p.adjust(grouped[[ss]]$pvalue, method="fdr")


  grouped[[ss]]$adj.pvalue<-adj.p.value # creating a new column for the adjusted p-values


  # Checking if any of the newly created adjusted p-values <0.05
  ## If one of the adjusted p.values<0.05, then the feature is added to result
  if(any(grouped[[ss]]$adj.pvalue<0.05)){

    # If the feature passes the threshold, extract the minimum adjust p-value for that feature
    smallest.id<-which(grouped[[ss]]$adj.pvalue==min(grouped[[ss]]$adj.pvalue))[1]

    # Storing the feature name and smallest adjusted p-value in a temporary dataframe
    temp<-grouped[[ss]][smallest.id,c(1,3)]
    row.names(temp)<-NULL


    # Storing the selected feature and adjusted p-value across all features in the `grouped` object
    adj.feature<-rbind(adj.feature, temp)
  }
}
```

Ranking

Finally, we can take the selected features and adjusted p-values and sort them from in ascending order to provide a ranked list based on the adjusted p-value.

```
# Ranking the output based on FDR adjusted p-values
ranked.adj.feature<-adj.feature[order(adj.feature$adj.pvalue, decreasing=FALSE),]
row.names(ranked.adj.feature)<-NULL

head(ranked.adj.feature) # Displaying the top identifed features
```

```
##   Feature adj.pvalue
## 1   Cov.4          0
## 2   X.109          0
## 3    X.11          0
## 4   X.116          0
## 5   X.125          0
## 6    X.14          0
```