Vignette for Fairness-constrained Model Averaging Algorithm

0. Preparation

At the beginning, we first import all external R packages required by the FMA implementation.

```
library(CVXR)
```

```
##
## Attaching package: 'CVXR'
```

```
## The following object is masked from 'package:stats':
##
##     power
```

```
library(reticulate)
```

Next, we import all functions used in the FMA algorithm.

```
path <- path.expand("~/Documents/FairModelAveraging")
setwd(path)

source_code_dir <- "FairAssisted"
file_paths <- list.files(source_code_dir, full.names = T)
for(f_path in file_paths){source(f_path)}

source_code_dir <- "FairConstrained/R Code"
file_paths <- list.files(source_code_dir, full.names = T)
for(f_path in file_paths){source(f_path)}
source_python("FairConstrained/Python Code/optimeopp.py")
```

1. Data Pre-processing

For this vignette example, we use the Communities and Crime dataset from the **fairml** package, which is the same dataset used in the real data evaluation section of the paper. Now we load the data and get the data pre-processed.

```
library(fairml)
data("communities.and.crime")
full_data <- communities.and.crime
# drop non-predictive variables
full_data$state <- NULL
full_data$county <- NULL
full_data$fold <- NULL
# remove 1 incomplete record
full_data <- full_data[complete.cases(full_data),]

# sensitive covariate
median_prop <- median(full_data$racepctblack)
S <- as.factor(ifelse(full_data$racepctblack > median_prop, "High", "Low"))
# response
```

```r
cutoff_rate <- quantile(full_data$ViolentCrimesPerPop, 0.7)
Y <- ifelse(full_data$ViolentCrimesPerPop > cutoff_rate, 1, -1)
# non-protected covariates
full_data$racepctblack <- NULL
full_data$ViolentCrimesPerPop <- NULL
X <- as.matrix(full_data)
# group index set
G <- c(1:ncol(X))

# train test split
set.seed(42)
train_idx <- sample(1:nrow(X), size = 1500, replace = F)
S_train <- S[train_idx]
Y_train <- Y[train_idx]
X_train <- X[train_idx,]
S_test <- S[-train_idx]
Y_test <- Y[-train_idx]
X_test <- X[-train_idx,]
```

2. Implementing FMA

2.1 Statistical Parity

Now, in this section, we fit the proposed FMA algorithm to the training data. We focus on the unfairness measure of statistical parity. First, we set the hyperparameters for the fairness-assisted stepwise method.

```r
fairassist_paras <- list()
# the discount parameter values considered for generating a variety of variable selection paths
fairassist_paras$nu_forward_vec <- seq(0, 1, by = 0.2)
fairassist_paras$nu_backward_vec <- c(20, 1/fairassist_paras$nu_forward_vec[2:6])
# regression function type
fairassist_paras$ftype <- "Logistic"
# by default, the stepwise method determines predictive performance based on objective function
fairassist_paras$altype <- "obj"
# stopping criteria
fairassist_paras$eps_obj <- 1e-4
fairassist_paras$eps_gdt <- 1e-4 # this is not used by default
fairassist_paras$maxit <- (max(G)+10)
# the maximum model sparsity set for each path when identifying candidate models
# from a variable selection path
fairassist_paras$num_model_consider <- 30
# backward elimination is allowed by default
fairassist_paras$flag <- 1
# an optional l2 penalty term can be included in the objective function for the stepwise method,
# by default, this term is omitted
fairassist_paras$lambda <- 0.0
# fairness measurements-related hyperparameters, version="continuous" indicates that we compute
# the average predicted probability of the positive class for a group directly, rather than
# discretizing the prediction into a binary outcome using a cutoff probability. Therefore,
# in this case, cutoff="0.0" (i.e., the cutoff probability is not used)
fairassist_paras$fairness <- "dpar"
fairassist_paras$version <- "continuous"
fairassist_paras$cutoff <- 0.0
# other hyperparameters
```

```
fairassist_paras$QUIET <- 1
fairassist_paras$G <- G
```

After setting the hyperparameters, we can fit the FMA algorithm. Let's assume the threshold value on the covariance-based fairness constraint is 0.1.

```
fair_fit <- FMA(X_train = X_train, Y_train = Y_train, S_train = S_train,
                fairassist_paras = fairassist_paras, tol_level = 0.1,
                tol_type = "covariance", FNR_level = NULL, FPR_level = NULL,
                sum_to_one = TRUE, non_negative = TRUE,
                aic_penalty = TRUE, dccp_paras = NULL)
final_beta <- fair_fit$final_beta
```

Now, **final_beta** provides the fitted model coefficients from FMA. With it, we can check the covariance-based unfairness measure of the fitted model on the training dataset.

```
train_lp <- as.vector(cbind(rep(1,nrow(X_train)), X_train) %*% final_beta)
abs(stats::cov(ifelse(S_train == "High", 0, 1), train_lp))
```

```
## [1] 0.1
```

It turns out that the absolute value of the covariance $\text{Cov}(S, X^T\beta)$ on the training dataset is 0.1, which aligns with the pre-defined threshold value of 0.1 and indicates that the **final_beta** is a fair feasible model.

Next, we evaluate the performance of this fitted model on the test dataset.

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
test_lp <- as.vector(cbind(rep(1,nrow(X_test)), X_test) %*% final_beta)
test_prob <- 1 / (1 + exp(-test_lp))
auc_test <- auc(response = factor(Y_test, levels = c(-1,1)), predictor = test_prob,
                direction = "<", quiet = TRUE)
cov_test <- abs(stats::cov(ifelse(S_test == "High", 0, 1), test_lp))
cat("The AUC score on the test set is: ", auc_test, sep = "")
```

```
## The AUC score on the test set is: 0.8830357
```

```
cat("The covariance-based unfairness measure on the test set is: ", cov_test, sep = "")
```

```
## The covariance-based unfairness measure on the test set is: 0.09058265
```

It indicates that the fitted model has an AUC of 0.883 and a covariance-based unfairness measure of 0.09 on the test dataset. The unfairness value 0.09 is very close to the pre-defined threshold of 0.1.

2.2 Equal Odds

In this section, we focus on the unfairness measure of equal odds. We skip the equal opportunity measure in this vignette, as it can be viewed as a special, simplified case of equal odds. Similar to the processes in Section 2.1, we begin by setting the hyperparameters for the fairness-assisted stepwise method.

```
fairassist_paras <- list()
```

```
fairassist_paras$nu_forward_vec <- seq(0, 1, by = 0.2)
fairassist_paras$nu_backward_vec <- c(20, 1/fairassist_paras$nu_forward_vec[2:6])

fairassist_paras$ftype <- "Logistic"
fairassist_paras$altype <- "obj"

fairassist_paras$eps_obj <- 1e-4
fairassist_paras$eps_gdt <- 1e-4 # this is not used by default
fairassist_paras$maxit <- (max(G)+10)

fairassist_paras$num_model_consider <- 30

fairassist_paras$flag <- 1
fairassist_paras$lambda <- 0.0
fairassist_paras$fairness <- "eodd"
fairassist_paras$version <- "continuous"
fairassist_paras$cutoff <- 0.0

fairassist_paras$QUIET <- 1
fairassist_paras$G <- G
```

Next, to solve the model averaging optimization problem for equal odds, we need to use the Disciplined Convex-Concave Programming (DCCP) solver in Python. Here, we set the hyperparameters for the DCCP solver.

```
dccp_paras <- list()
# some hyperparameters for the DCCP solver
dccp_paras$EPS <- 1e-6
dccp_paras$tau <- 0.005
dccp_paras$mu <- 1.2
# whether a unconstrained solution is taken as the initialization
dccp_paras$take_initial_sol <- TRUE
# the solver type to be used
dccp_paras$solver_type <- "ECOS"
```

After setting the hyperparameters, we can fit the FMA algorithm. Assume that the threshold values on the covariance-based fairness constraints related to the FNR (1-TPR, Y = 1) and FPR (Y = –1) are both set to 0.02.

```
fair_fit <- FMA(X_train = X_train, Y_train = Y_train, S_train = S_train,
                fairassist_paras = fairassist_paras, tol_level = NULL,
                tol_type = NULL, FNR_level = 0.02, FPR_level = 0.02,
                sum_to_one = TRUE, non_negative = TRUE,
                aic_penalty = TRUE, dccp_paras = dccp_paras)
```

```
## Problem is DCCP: True
```

```
final_beta <- fair_fit$final_beta
```

With this fitted **final_beta**, again, we first check the covariance-based unfairness measures related to the FNR (Y = 1) and FPR (Y = -1) on the training dataset.

```
train_lp <- as.vector(cbind(rep(1,nrow(X_train)), X_train) %*% final_beta)
train_cov <- covariance_fnr_fpr(fitted = train_lp, response = Y_train,
                                sensitive = as.matrix(ifelse(S_train == "High", 0, 1)))
train_cov$cov_fnr
```

```
## [1] 0.02004608
```

```
train_cov$cov_fpr
```

```
## [1] 0.01010171
```

The covariance-based unfairness measures related to FNR and FPR of the fitted model on the training dataset are found to be either close to or below the threshold of 0.02 (not exactly 0.02 because the non-convex problem is solved using heuristics), indicating that the pre-defined fairness constraint is satisfied and that **final_beta** represents a fair feasible model.

Next, we evaluate the performance of this fitted model on the test dataset.

```
test_lp <- as.vector(cbind(rep(1,nrow(X_test)), X_test) %*% final_beta)
test_prob <- 1 / (1 + exp(-test_lp))
auc_test <- auc(response = factor(Y_test, levels = c(-1,1)), predictor = test_prob,
                direction = "<", quiet = TRUE)
cov_test <- covariance_fnr_fpr(fitted = test_lp, response = Y_test,
                               sensitive = as.matrix(ifelse(S_test == "High", 0, 1)))
cat("The AUC score on the test set is: ", auc_test, sep = "")
```

```
## The AUC score on the test set is: 0.9107143
```

```
cat("The covariance-based unfairness measure related to FNR on the test set is: ",
    cov_test$cov_fnr, sep = "")
```

```
## The covariance-based unfairness measure related to FNR on the test set is: 0.01343335
```

```
cat("The covariance-based unfairness measure related to FPR on the test set is: ",
    cov_test$cov_fpr, sep = "")
```

```
## The covariance-based unfairness measure related to FPR on the test set is: 0.008268575
```

It indicates that the fitted model has an AUC of 0.911, a covariance-based unfairness measure related to FNR of 0.013, and a covariance-based unfairness measure related to FPR of 0.008 on the test dataset. The unfairness values are below and thus satisfy the pre-defined threshold of 0.02.