Real Data Analysis by SGD via Random Scaling

2023-12-12

To assess the predictive performance of Stochastic Gradient Descent (SGD) with random scaling, I will apply the random scaling SGD to the “Mushroom” dataset. This dataset comprises 61,069 observations, each with 2 classes and 20 features. The class labels are are “edible” and “poisonous”. The dataset is sourced from https://archive.ics.uci.edu/dataset/848/secondary+mushroom+dataset. (https://archive.ics.uci.edu/dataset/848/secondary+mushroom+dataset.)

Algorithm

The random scaling SGD algorithm for logistic regression was referenced from the code available on https://github.com/SGDinference-Lab/AAAI-22 (https://github.com/SGDinference-Lab/AAAI-22).

“Algorithm1” in the slide is done by the function below. This function updates A_t(`a.old -> a.new`), b_t(`b.old -> b.new`), and $\hat{V}_t$. `c.new` denotes $\sum_{s=1}^{t} s^2$ in the “Algorithm”. β_t is not updated by the function below, but is updated by `beta_estimator` function.

```
random.scaling.update = function(t.obs, bar.bt_t, a.old, b.old, c.old){

  a.new = a.old + t.obs^2 * bar.bt_t %*% t(bar.bt_t)
  b.new = b.old + t.obs^2 * bar.bt_t
  c.new = c.old + t.obs^2

  V_t = ( a.new - b.new %*% t(bar.bt_t) - bar.bt_t %*% t(b.new) + c.new * bar.bt_t %*% t(bar.bt_t) ) / (t.obs^2)

  return(list(a.new=a.new, b.new=b.new, c.new=c.new, V_t=V_t))
}
```

To perform SGD, gradient of the objective function is necessary. Below is the gradient for logistic model.

```
grad_comp = function(x_old, a_new, b_new, d){
  a_new = matrix(a_new, d, 1)
  x_old = matrix(x_old, d, 1)
  grad = -c((2*b_new-1)*a_new) / c(1+exp( (2*b_new-1) * (t(a_new)%*%x_old)) )
  return (grad)
}
```

This function obtains the estimated β from the dataset. `data_x` denotes the design matrix and it has a form of $n \times d$ matrix where n is the number of observations and d is the dimension of regressors. `data_y` denotes the response varaible matrix with form of $n \times 1$ vector. `alpha` and `lr` are the hyperparameters of the learning rate. Note that from “Assumption 1” (iv), step size γ_t has form of $\gamma_0 t^{-a}$ for some $1/2 < a < 1$. `alpha` and `lr` are the same as a and γ_0, respectively.

```
beta_estimator = function(data_x, data_y, alpha, lr)
{
  #number of observations
  n = dim(data_x)[1]

  # Dimension of regressors
  d = dim(data_x)[2]

  # Initialize the output variables
  X = matrix(0, nrow = d, ncol = 1, byrow = TRUE)
  Xbar_old = matrix(0, nrow = d, ncol = 1, byrow = TRUE)

  # parameters for Random Scaling method updates
  a.old = matrix(0, nrow = d, ncol = d)
  b.old = matrix(0, nrow = d, ncol = 1)
  c.old = 0

  for (obs in 1:n){
    lrnew = lr*obs**(-alpha)
    grad = grad_comp(X, t(data_x[obs, ]), data_y[obs,], d)
    X = X - lrnew*grad
    Xbar_old = (Xbar_old*(obs - 1) + X)/obs

    rs = random.scaling.update(obs, Xbar_old, a.old, b.old, c.old)
    a.old = rs$a.new
    b.old = rs$b.new
    c.old = rs$c.new
    S_hat = diag(rs$V_t)
  }

  return(Xbar_old) #returns the coefficient estimator
}
```

Data Preprocessing and Exploratory Data Analysis(EDA)

Since this analysis is not purposed for developing the best classifier for edible/poisonous mushrooms, only a minimum necessary data preprocessing and EDA have been conducted. Explanations for each feature is available on the data repository linked above.

```
data <- read.csv("secondary_data.csv", sep = ";") #read data
data <- data %>%
  mutate_if(is.numeric, scale) #standaradize numerical variables

data[data == ""] <- NA #"" value is unknowned, so assign NA

(Na_count <- apply(data, 2, function(x) sum(is.na(x))))
```

```
##                class         cap.diameter           cap.shape
##                    0                    0                   0
##          cap.surface            cap.color does.bruise.or.bleed
##                14120                    0                   0
##      gill.attachment         gill.spacing           gill.color
##                 9884                25063                   0
##          stem.height           stem.width            stem.root
##                    0                    0               51538
##         stem.surface           stem.color            veil.type
##                38124                    0               57892
##           veil.color             has.ring            ring.type
##                53656                    0                2471
##    spore.print.color              habitat              season
##                54715                    0                   0
```

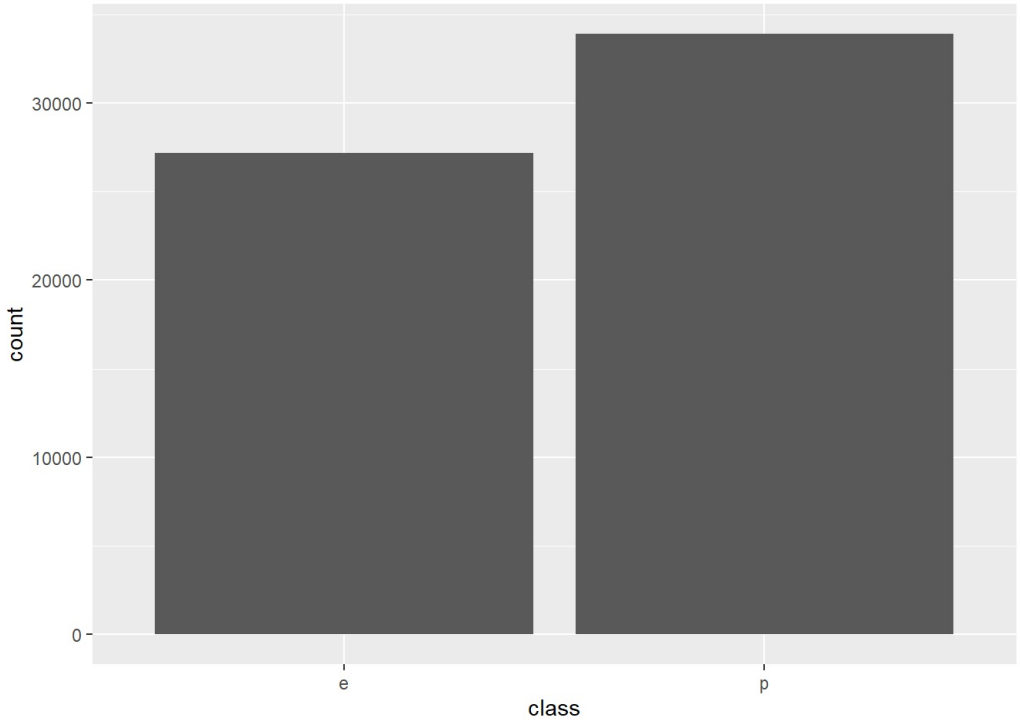
```
#for the sake of convenience, delete columns that have NA values

col_names <- names(Na_count)
col_names <- col_names[Na_count == 0]

data <- data[,col_names]

data <- data %>%
  mutate_if(is.character, function(x) as.factor(x)) #factorize

dim(data)
```

```
## [1] 61069    12
```

We now have 61069 observations with 11 features.

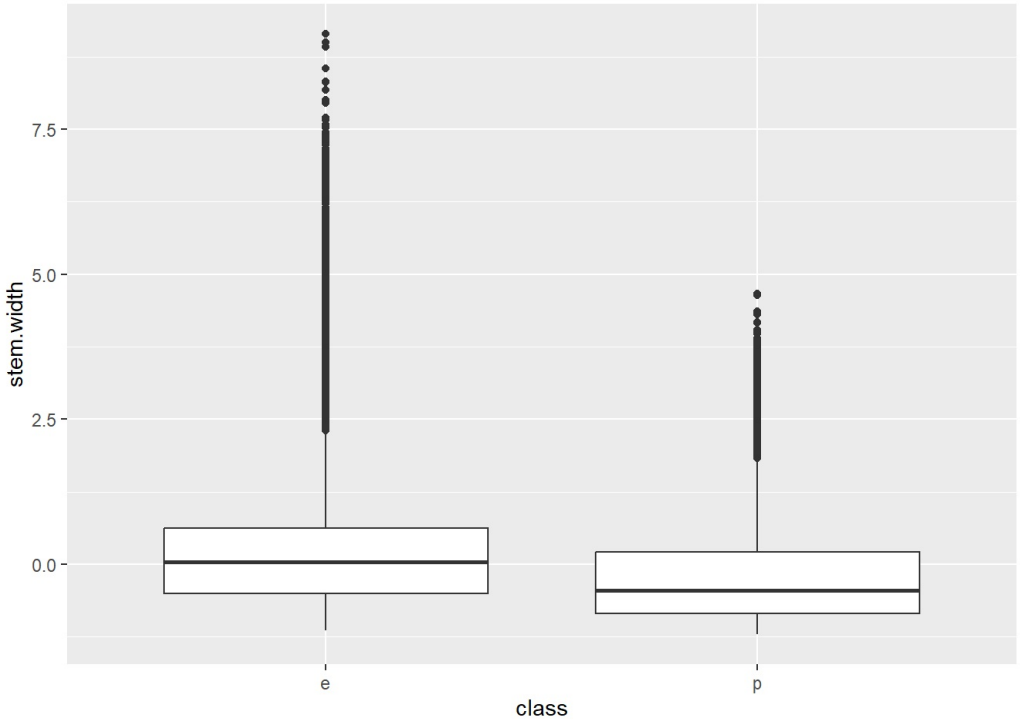
We can see that the class labels are not unbalanced.

```
data %>%
  ggplot(aes(x = class)) +
  geom_bar()
```

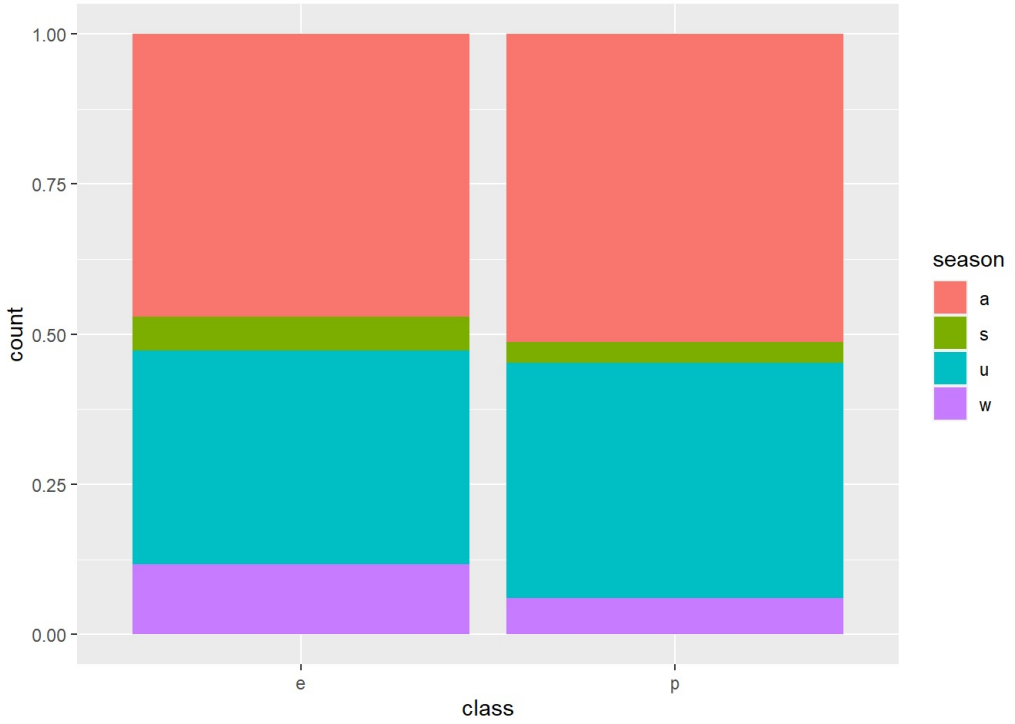


These two features relatively discriminate the two labels well.(not very well)

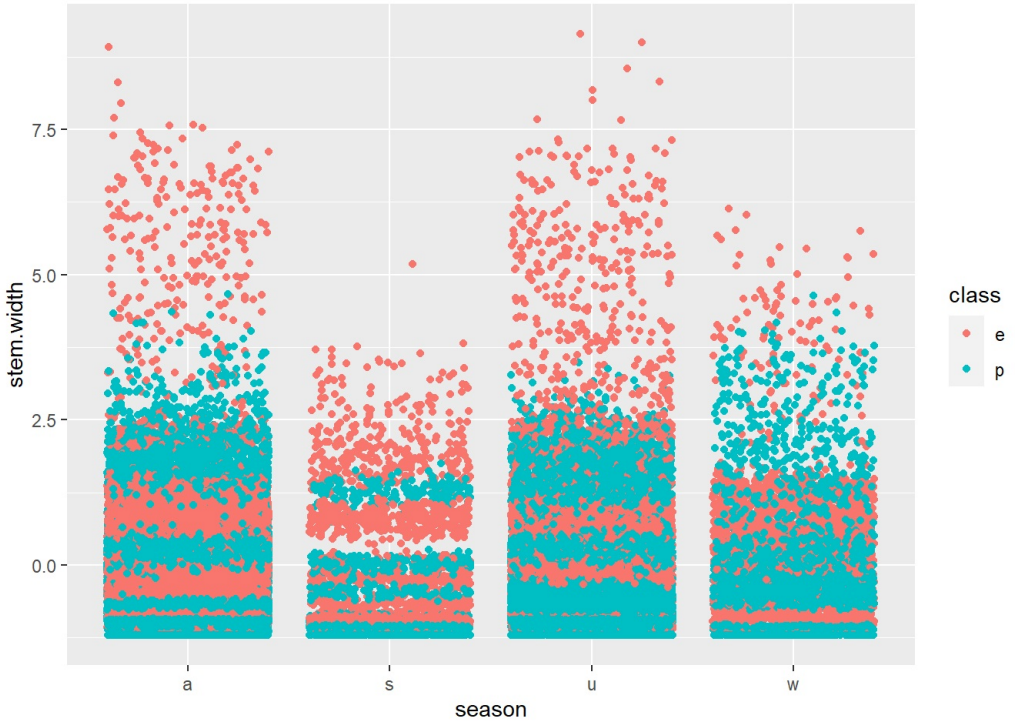
```
data %>%
  ggplot(aes(x = class, y = stem.width)) +
  geom_boxplot()
```




```
data %>%
  ggplot() +
  geom_bar(aes(x = class, fill = season), position = "fill")
```




```
data %>%
  ggplot(aes(x = season, y = stem.width, color = class)) +
  geom_jitter()
```



Prediction

We will split the data into train and test by 9:1 and compare the prediction performance of logistic regression estimated by default in R(maximizing conditional likelihood) and by random scaling SGD. Unlike estimating coefficient by maximizing conditional likelihood, random scaling SGD requires tuning hyperparameters. This will be executed by 5-fold cross-validation on the test set.

First, check the performance of default logistic regression.

```
set.seed(1213)
#perform one-hot encoding
dummy <- dummyVars(~., data = data, fullRank = TRUE, sep = "_")
data <- data.frame(predict(dummy, newdata=data))
head(data)
```

```
##   class_p cap.diameter cap.shape_c cap.shape_f cap.shape_o cap.shape_p
## 1       1     1.619449           0           0           0           0
## 2       1     1.873967           0           0           0           0
## 3       1     1.393421           0           0           0           0
## 4       1     1.412415           0           1           0           0
## 5       1     1.501686           0           0           0           0
## 6       1     1.634644           0           0           0           0
##   cap.shape_s cap.shape_x cap.color_e cap.color_g cap.color_k cap.color_l
## 1           0           1           0           0           0           0
## 2           0           1           0           0           0           0
## 3           0           1           0           0           0           0
## 4           0           0           1           0           0           0
## 5           0           1           0           0           0           0
## 6           0           1           0           0           0           0
##   cap.color_n cap.color_o cap.color_p cap.color_r cap.color_u cap.color_w
## 1           0           1           0           0           0           0
## 2           0           1           0           0           0           0
## 3           0           1           0           0           0           0
## 4           0           0           0           0           0           0
## 5           0           1           0           0           0           0
## 6           0           1           0           0           0           0
##   cap.color_y does.bruise.or.bleed_t gill.color_e gill.color_f gill.color_g
## 1           0                      0            0            0            0
## 2           0                      0            0            0            0
## 3           0                      0            0            0            0
## 4           0                      0            0            0            0
## 5           0                      0            0            0            0
## 6           0                      0            0            0            0
##   gill.color_k gill.color_n gill.color_o gill.color_p gill.color_r gill.color_u
## 1            0            0            0            0            0            0
## 2            0            0            0            0            0            0
## 3            0            0            0            0            0            0
## 4            0            0            0            0            0            0
## 5            0            0            0            0            0            0
## 6            0            0            0            0            0            0
##   gill.color_w gill.color_y stem.height stem.width stem.color_e stem.color_f
## 1            1            0    3.076679  0.4922890            0            0
## 2            1            0    3.385283  0.6018949            0            0
## 3            1            0    3.328904  0.5570561            0            0
## 4            1            0    2.726533  0.3816866            0            0
## 5            1            0    2.952051  0.5032496            0            0
## 6            1            0    3.340773  0.6616799            0            0
##   stem.color_g stem.color_k stem.color_l stem.color_n stem.color_o stem.color_p
## 1            0            0            0            0            0            0
## 2            0            0            0            0            0            0
## 3            0            0            0            0            0            0
## 4            0            0            0            0            0            0
## 5            0            0            0            0            0            0
## 6            0            0            0            0            0            0
##   stem.color_r stem.color_u stem.color_w stem.color_y has.ring_t habitat_g
## 1            0            0            1            0          1         0
## 2            0            0            1            0          1         0
## 3            0            0            1            0          1         0
## 4            0            0            1            0          1         0
## 5            0            0            1            0          1         0
## 6            0            0            1            0          1         0
##   habitat_h habitat_l habitat_m habitat_p habitat_u habitat_w season_s season_u
## 1         0         0         0         0         0         0        0        0
## 2         0         0         0         0         0         0        0        1
## 3         0         0         0         0         0         0        0        0
## 4         0         0         0         0         0         0        0        0
## 5         0         0         0         0         0         0        0        0
## 6         0         0         0         0         0         0        0        1
##   season_w
## 1        1
## 2        0
## 3        1
## 4        1
## 5        1
## 6        0
```

```
n <- dim(data)[1]
test <- sample(n, n%/%10)

logistic_mod <- glm(class_p ~ ., data[-test,], family = "binomial") #model trained in train data

prob <- predict(logistic_mod, newdata = data[test, -1], type = "response") #obtain the probability for test data

pred_class <- rep(0, length(test))
pred_class[prob > 0.5] = 1

true_class <- data[test,1]

table(pred_class, true_class) #confusion matrix
```

```
##           true_class
## pred_class    0    1
##          0 1877  741
##          1  883 2605
```

```
mean(pred_class == true_class) #accuracy
```

```
## [1] 0.7340321
```

Now, check the performance of random scaling SGD.

```r
set.seed(1213)
train_x <- as.matrix(data[-test,-1])
train_y <- as.matrix(data[-test, 1])

shuffle <- sample(54963, 54963) #to prevent bias due to ordering

train_x <- as.matrix(train_x[shuffle,])
train_y <- as.matrix(train_y[shuffle,])

#perform 5-fold validation grid search and obtain the best hyperparameters

k_fold_result <- createFolds(train_y, k = 5, list=TRUE, returnTrain = FALSE)

param_grid <- expand.grid(alpha = seq(0.5001, 0.9999, length.out = 20),
            lr = seq(5, 12, length.out = 20))

param_grid$acc <- 0

alpha <- param_grid$alpha
lr <- param_grid$lr
for(i in 1:400){
    acc_vec = c() #to obtain 5 accuracies for each fold
    for(f in 1:5){
    beta <- beta_estimator(data_x = train_x[-k_fold_result[[f]],],
                           data_y = as.matrix(train_y[-k_fold_result[[f]],]),
                           alpha = alpha[i],
                           lr = lr[i])

    # Calculate the linear combination of predictors and beta
    lin_comb <- as.matrix(train_x[k_fold_result[[f]], ]) %*% beta
    # Apply the logistic function to obtain predicted probabilities
    prob <- 1 / (1 + exp(-lin_comb))
    true_class = train_y[k_fold_result[[f]], ]
    pred_class <- rep(0, length(true_class))
    pred_class[prob > 0.5] = 1
    acc_vec[f] = mean(pred_class == true_class)
    }
  param_grid[i,3] = mean(acc_vec) #mean accuracy
}

#obtain the best hyperparameters

tmp <- param_grid %>%
  arrange(desc(acc)) %>%
  .[1,]

alpha_optim <- tmp[1,1]; lr_optim <- tmp[1,2]

#obtain the coefficient estimator by using the hyperparameters above

beta <- beta_estimator(data_x = train_x,
                           data_y = as.matrix(train_y),
                           alpha = alpha_optim,
                           lr = lr_optim)

# Calculate the linear combination of predictors and beta
lin_comb <- as.matrix(data[test, -1]) %*% beta

# Apply the logistic function to obtain predicted probabilities
prob <- 1 / (1 + exp(-lin_comb))
pred_class <- rep(0, length(test))
pred_class[prob > 0.5] = 1
true_class <- data[test,1]
table(pred_class, true_class) #confusion matrix
```

```
##           true_class
## pred_class    0    1
##          0 1892  727
##          1  868 2619
```

```r
mean(pred_class == true_class) #accuracy
```

```
## [1] 0.7387815
```

It is almost sure that other classification methods such as LASSO will perform better in this case. Also there is no significant enhancement in accuracy with random scaling SGD. However, it is notable that random scaling SGD has strongness in inference and shows moderate performance in real data prediction. If random scaling SGD is applied to more sophisticated models, it may outperform other models in prediction as well.