Matrix Methods for Regularised Regression

Dom Owens

12/11/2019

In *regularised regression*, we solve the problem

$$\min_{\mathbf{w}} ||\mathbf{y} - \mathbf{w}^{\mathbf{T}}\mathbf{X}|| + \lambda R(\mathbf{w})$$

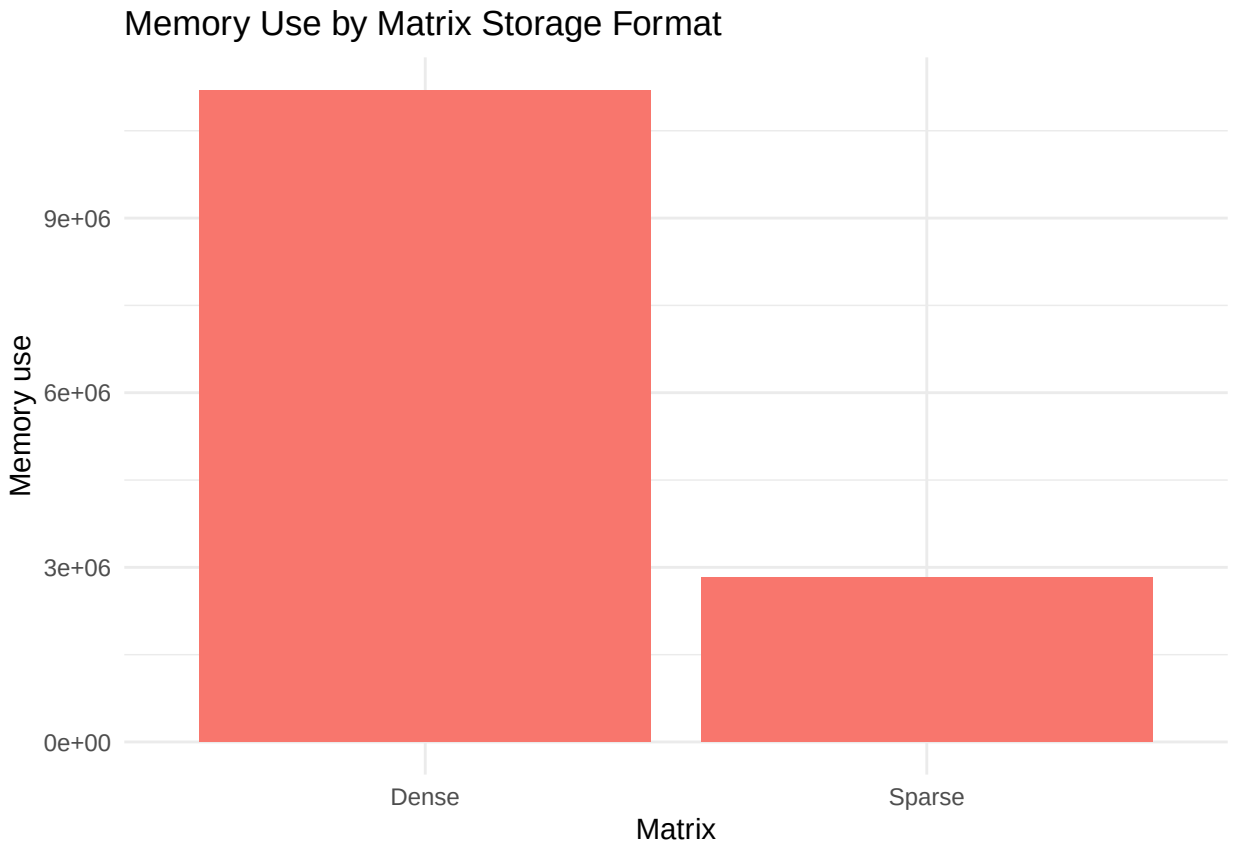
where $R(\mathbf{w})$ is the regularisation function. Common choices include the **l1-** and **l2-norms** $R(\mathbf{w}) = ||\mathbf{w}||_1$ and $R(\mathbf{w}) = ||\mathbf{w}||^2$, which induce the *LASSO* and *Ridge* regressions respectively. Both of these regression methods perform well for high-dimensional data, and can be used to demonstrate associated matrix methods.

We consider a sparse dataset on blog posts.

```
data_raw <- read.csv("blogData_train.csv", header = FALSE)  #read in data
index <- sample(1:dim(data_raw)[1], 5000)
data <- data_raw[index,]#construct sample
#head(data[1:2,])

X <- data[,-281] #create design matrix
y <-  data[,281] # assign response: hits/day
```

We compare the memory useage of storing `X` as a dense or sparse matrix.

```
library(Matrix)
X_d <- matrix(unlist(X), nrow = dim(X)[1], ncol = dim(X)[2]) #as dense matrix
X_s <- Matrix(unlist(X), nrow = dim(X)[1], ncol = dim(X)[2], sparse = TRUE) #as sparse matrix
d <- as.numeric(object.size(X_d))
s <- as.numeric(object.size(X_s))
library(ggplot2)
ggplot(data = data.frame(c("X_d","X_s"), c(d,s)), aes(x= c("Dense","Sparse"), y = c(d,s), fill="red") )
```



How sparse is X?

```
rankMatrix(X_s) # show rank
```

```
## Warning in rankMatrix(X_s): rankMatrix(<large sparse Matrix>, method = 'tolNorm2') coerces to dense m
##  Probably should rather use method = 'qr' !?
```

```
## [1] 234
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 1.110223e-12
```

```
dim(X_s)[2] - rankMatrix(X_s)[1] # find nullity
```

```
## Warning in rankMatrix(X_s): rankMatrix(<large sparse Matrix>, method = 'tolNorm2') coerces to dense m
##  Probably should rather use method = 'qr' !?
```

```
## [1] 46
```

Ridge regression with sparse matrices

For the ridge regression

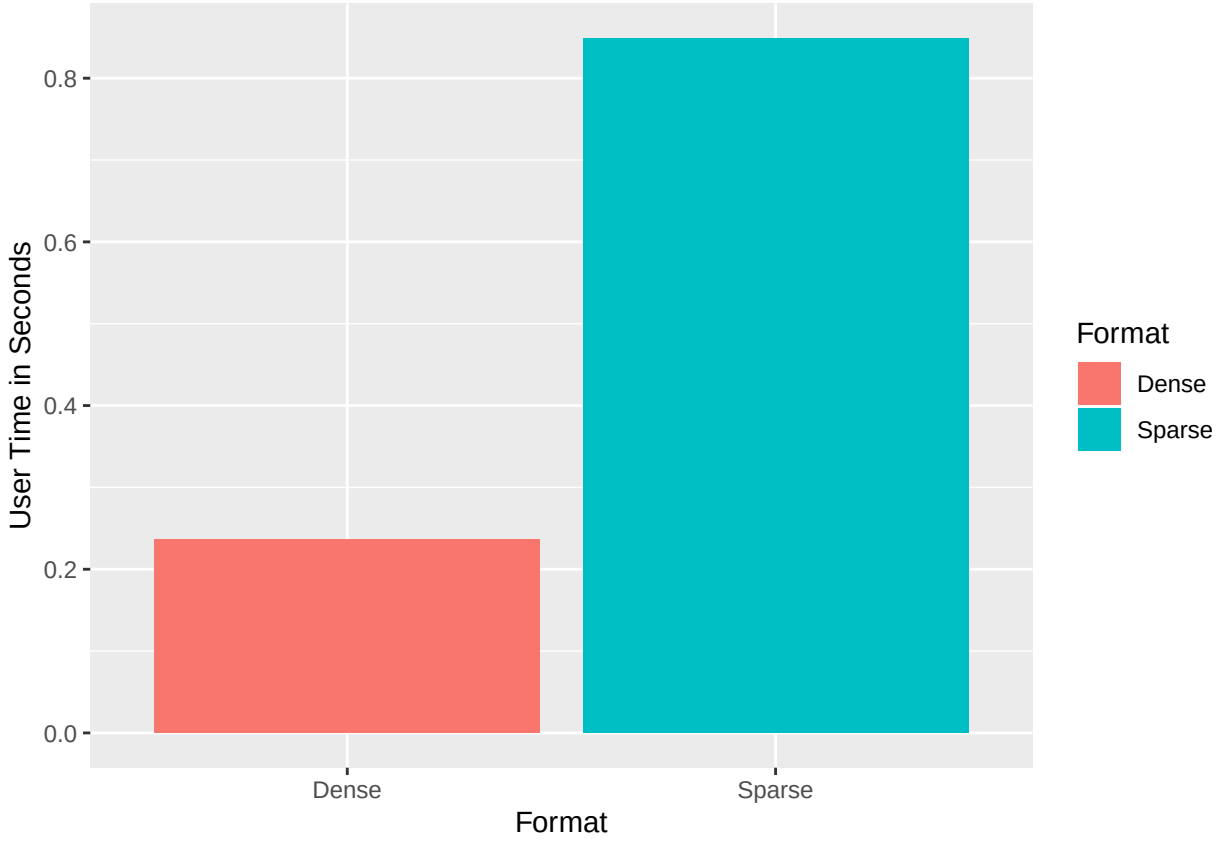
$$min_{\mathbf{w}}||\mathbf{y} - \mathbf{w^T X}|| + \lambda||\mathbf{w}||^2$$

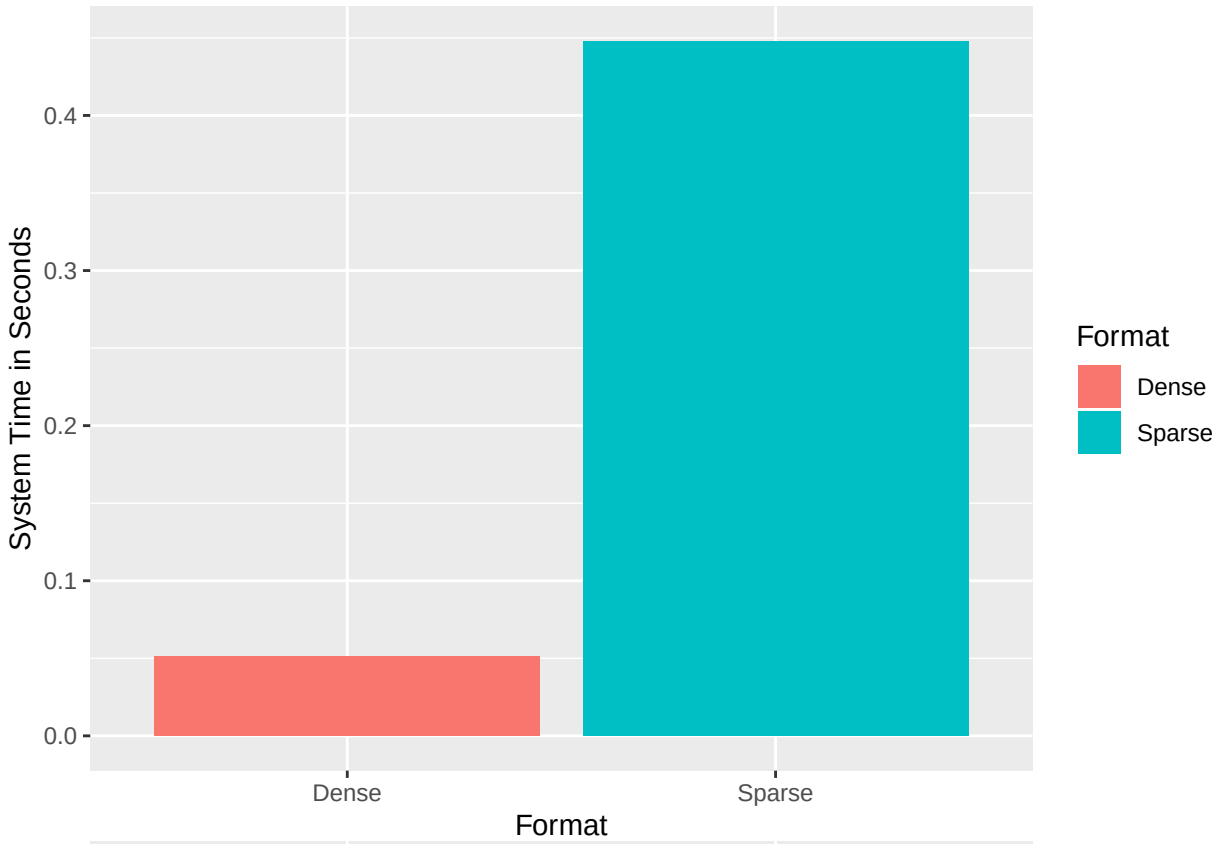
We have the closed-form solution

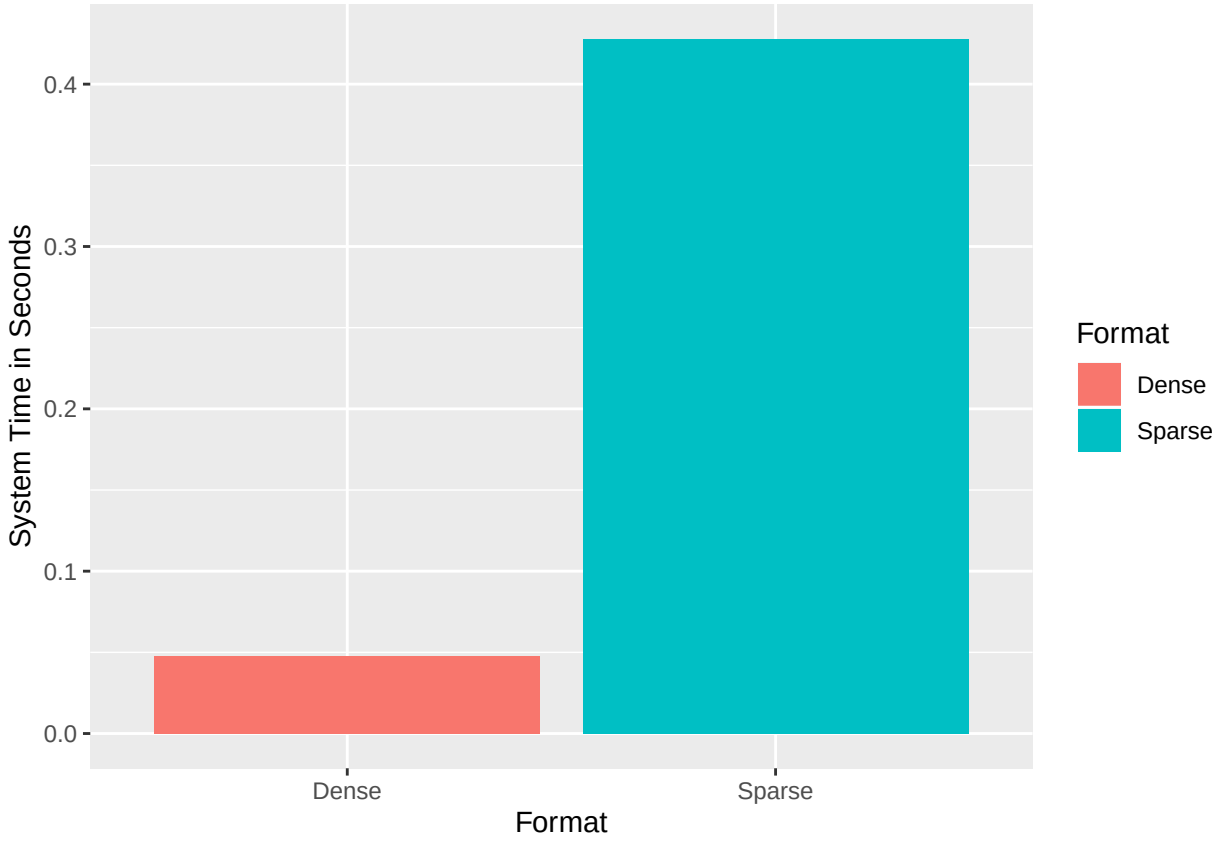
$$\hat{\mathbf{w}}^T = (XX^T + \lambda I)^{-1} X\mathbf{y}^T$$

We compare performance for computing this estimate with dense and sparse matrices, respectively

```
X_d1 <- t(cbind(1, X_d)) # attach column of intercept, then transpose
X_s1 <- t(cbind(1, X_s))
y_1 <- t(y)

ridge <- function(X, y, lambda){ #specify ridge regression
  w <- solve(X %*% t(X) + diag(lambda, dim(X)[1])) %*% X %*% t(y)
  w
}


dense.times <- system.time(w_d <- ridge(X_d1, y_1, lambda = 5))
sparse.times <- system.time(w_s <- ridge(X_s1, y_1, lambda = 5))
```







Interestingly, the dense calculations outperform the sparse calculations for this dataset. This would likely

change for a more sparse set of data.

We see these are stored differently

```
object.size(w_d)
```

```
## 2448 bytes
```

```
head(w_d)
```

```
##             [,1]
## [1,]  3.03846042
## [2,] -1.30808199
## [3,]  1.34953881
## [4,]  0.02915344
## [5,] -0.04100087
## [6,]  0.11569233
```

```
object.size(w_s)
```

```
## 3360 bytes
```

```
head(w_s)
```

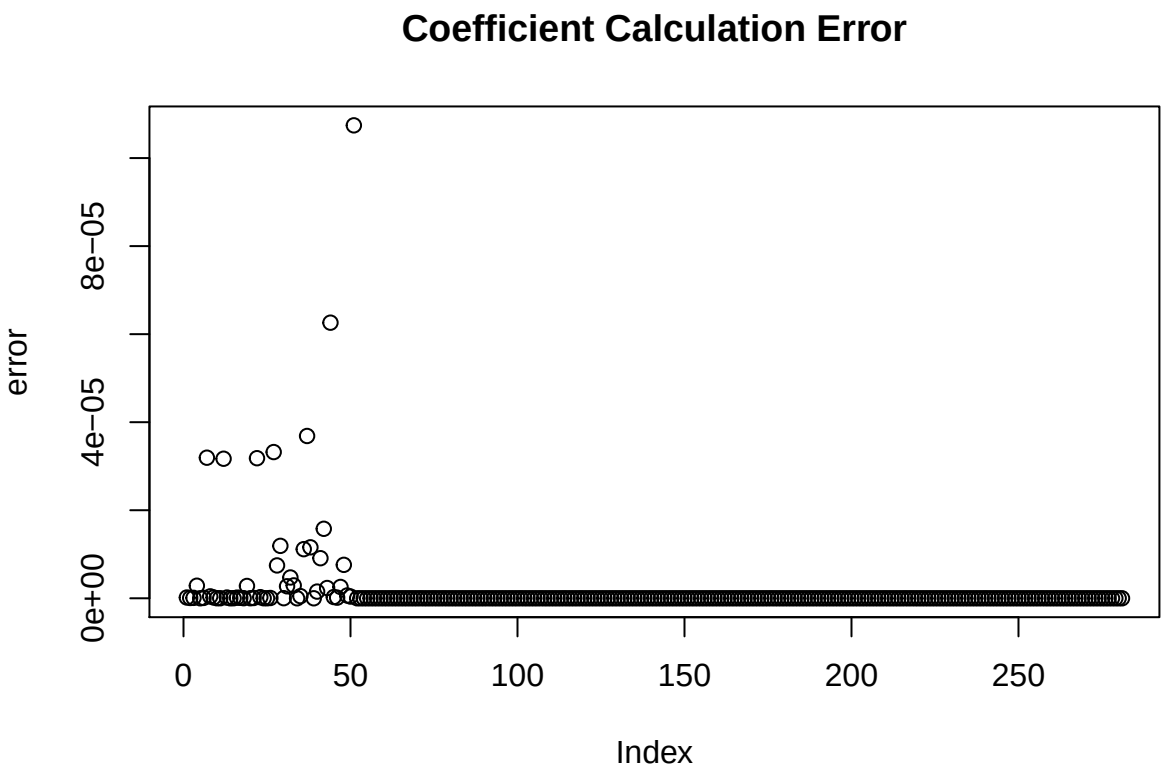
```
## 6 x 1 Matrix of class "dgeMatrix"
##             [,1]
## [1,]  3.03846059
## [2,] -1.30808206
## [3,]  1.34953871
## [4,]  0.02915059
## [5,] -0.04100087
## [6,]  0.11569241
```

And differ by a median magnitude of 2e-11; most of this error comes from the leftmost non-sparse part of the matrix.

```
error <- as.matrix(abs(w_d-w_s))
summary(error)
```

```
##        V1
##  Min.   :0.000e+00
##  1st Qu.:1.000e-11
##  Median :5.000e-11
##  Mean   :1.554e-06
##  3rd Qu.:8.800e-10
##  Max.   :1.074e-04
```

```
plot(error, main = "Coefficient Calculation Error")
```



IRLS Under Sparsity

Credit to https://bwlewis.github.io/GLM/ for non-ridge algorithm code; https://arxiv.org/pdf/1509.09169.pdf for algorithm

A better approach to this might be to split the design matrix into dense and sparse partitions, and conduct separate calculations on each of these. We use the **Iteratively Reweighted Least Squares** optimisation procedure for the L_2 norm regularisation.

We iteratively solve

$$\boldsymbol{\beta} = \arg\min_{\beta}[X^T W X + \lambda I]^{-1} X^T W Z$$

where

$$W_{ii} = \frac{1}{(y_i - X_i \beta^{(t)})}$$

are the diagonal entries of the weight matrix, which we update with each stage.

$$Z = X\beta + W^{-1}(y - X\beta)$$

is the adjusted response.

We first define a function to calculate a matrix product under a sparse/dense partition, preserving the status of each sub-matrix.

```
# Sparse weighted cross product helper function
# Input: Dense Matrix A_dense, sparse Matrix A_sparse, weights W,
# where A = [A_dense, A_sparse] and length W=ncol(A).
# Output: Dense representation of crossprod(A,W*A) + lambda*I
sp_wt_cross <- function(A_dense, A_sparse, W, lambda)
```

```
{
  nd <- ncol(A_dense)
  ns <- ncol(A_sparse)
  n  <- nd + ns
  ATWA <- matrix(0, nrow=n, ncol=n)
  WA_dense  <- W*A_dense
  WA_sparse <- W*A_sparse
  ATWA[1:nd,1:nd] <- as.matrix(crossprod(A_dense,WA_dense))
  ATWA[1:nd,(nd+1):n] <- as.matrix(crossprod(A_dense,WA_sparse))
  ATWA[(nd+1):n,1:nd] <- as.matrix(crossprod(A_sparse,WA_dense))
  ATWA[(nd+1):n,(nd+1):n] <- as.matrix(crossprod(A_sparse,WA_sparse))
  ATWA + diag(lambda, n) #add ridge term
}
```

And then define a function performing the IRLS procedure, making use of the sparse/dense partition.

```
# Example IRLS implementation that can take advantage of sparse model matrices.
# Here we assume that the model matrix A is already permuted and partitioned
# into A = [A_dense, A_sparse] dense and sparse columns.  The response vector b
# is also assumed to be permuted conformably with the matrix partitioning.

irls_sparse <-
function(A_dense, A_sparse, b, lambda =1, family=gaussian, maxit=25, tol=1e-08)
{
  nd <- ncol(A_dense)
  ns <- ncol(A_sparse)
  n  <- nd + ns
  x <- rep(0, n)
  for(j in 1:maxit)
  {
    eta   <- as.vector(A_dense %*% x[1:nd] + A_sparse %*% x[(nd+1):n])
    g     <- family()$linkinv(eta)
    gprime <- family()$mu.eta(eta)
    z     <- eta + (b - g) / gprime
    W     <- as.vector(gprime^2 / family()$variance(g))
    xold  <- x
    ATWA  <- sp_wt_cross(A_dense,A_sparse,W,lambda)
    wz    <- W*z
    ATWz  <- c(as.vector(crossprod(A_dense, wz)), as.vector(crossprod(A_sparse, wz)))

    C   <- chol(ATWA, pivot=TRUE)
   # if(attr(C,"rank") < ncol(ATWA)) stop("Rank-deficiency detected.")
    p   <- attr(C, "pivot")
    s   <- forwardsolve(t(C), ATWz[p])
    x   <- backsolve(C,s)[p]

    if(sqrt(crossprod(x-xold)) < tol) break
  }
  list(coefficients=x,iterations=j)
}
```

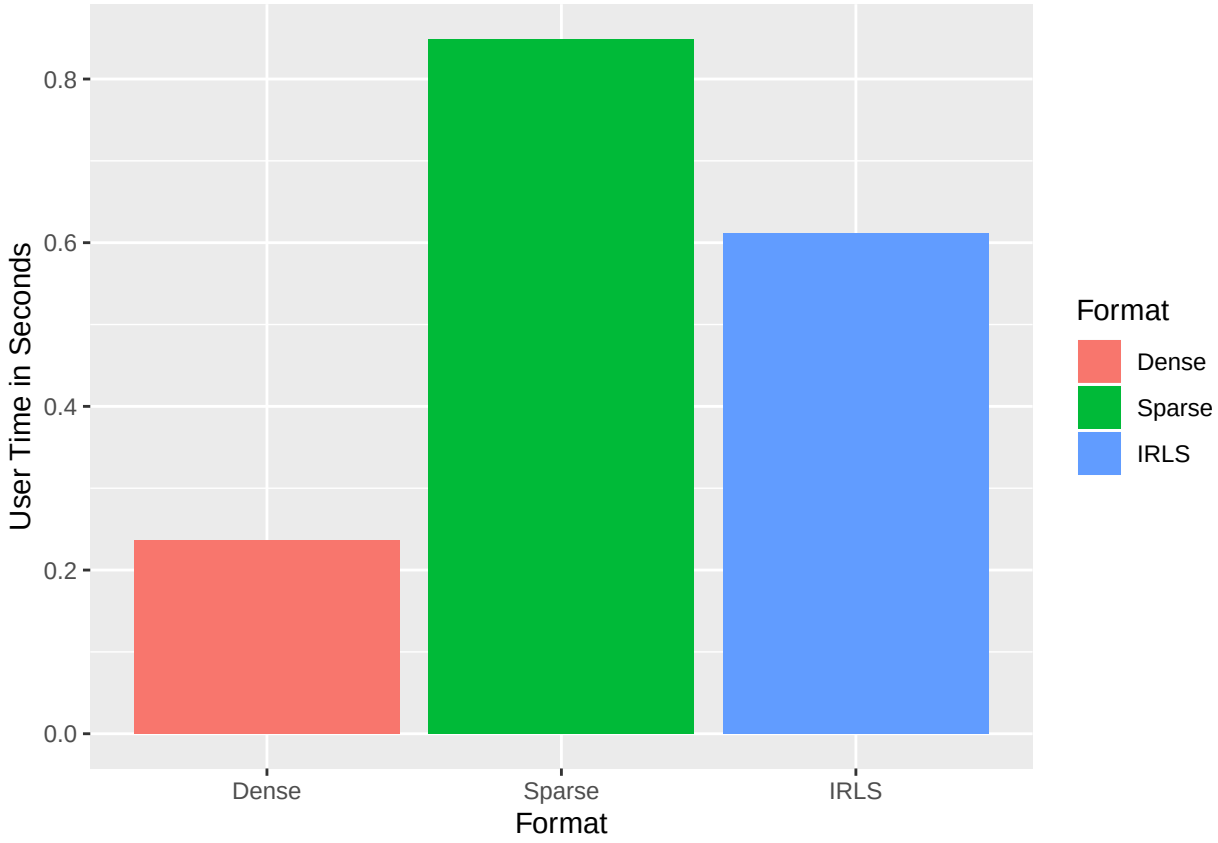
We partition `X` into dense and sparse parts, and record run times.

```
X_dense <- as.matrix(X[,1:50]) #partition X into dense and sparse sub-matrices
X_sparse <- as.matrix(X[, 51:280])
```

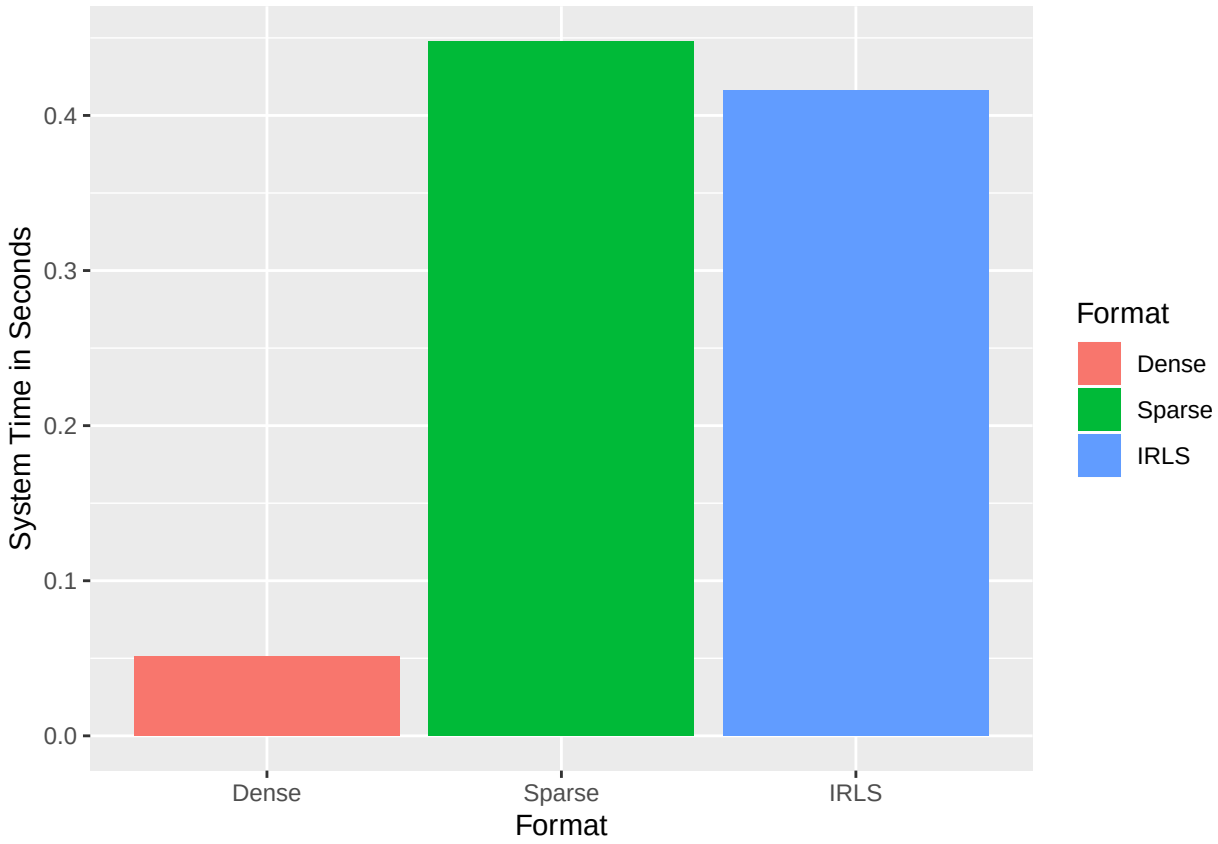
```
irls.times <- system.time(w_irls <- irls_sparse(X_dense, X_sparse, y)) #record run times
```

We inspect the performance and compare to the strictly dense/ strictly sparse methods from before.

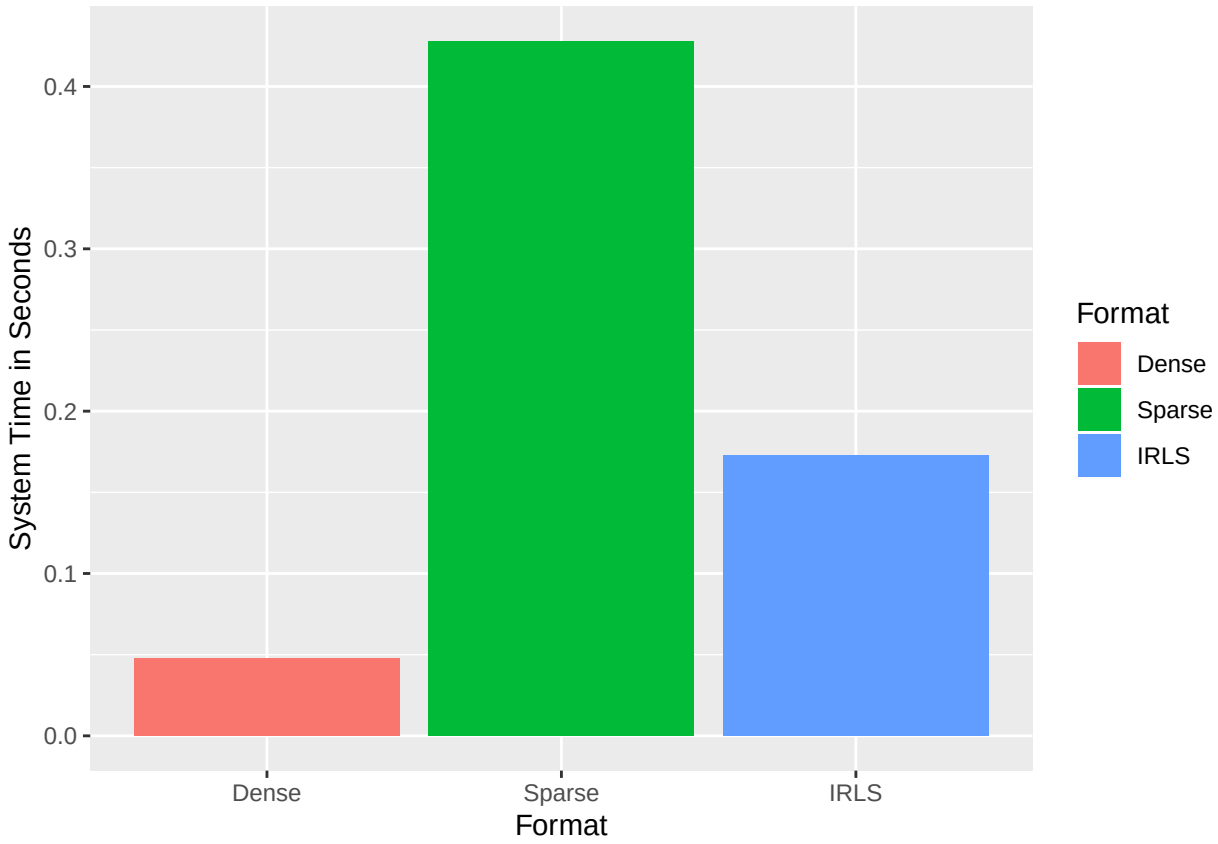
```
performance <- rbind(performance, data.frame(Format = 'IRLS', #append dense performance
                                             UserTime = irls.times[1],
                                             SystemTime = irls.times[2],
                                             ElapsedTime = irls.times[3]))

par(mfrow = c(1,3))
ggplot(performance, aes(x = Format, y = UserTime, fill = Format)) +
  geom_bar(stat = "identity") +
  ylab('User Time in Seconds')
```




```
ggplot(performance, aes(x = Format, y = SystemTime, fill = Format)) +
  geom_bar(stat = "identity") +
  ylab('System Time in Seconds')
```




```
ggplot(performance, aes(x = Format, y = ElapsedTime, fill = Format)) +
  geom_bar(stat = "identity") +
  ylab('System Time in Seconds')
```



We can see that the IRLS hybrid method outperforms the sparse method, but the dense method performs best overall. We should investigate further with increasingly sparse design matrices to see when the IRLS/sparse methods begin to outperform the dense method.