Coordinate Descent and Coordinate Gradient Descent

It is generally believed that coordinate descent should perform better than first-order methods. We have our minimization problem $\min_{\mathbf{x}} f(\mathbf{x})$ where f is convex.

We can use coordinate descent. This algorithm is also called 'coordinatewise minimization'.

let the initial value be $\mathbf{x}^{(0)} \in \mathbb{R}^n$, and repeat: $x_i^{(k)} = \text{argmin}_{x_i} f(x_1^{(k)}, \ldots, x_{i-1}^{(k)}, x_i, x_{i+1}^{(k-1)}, \ldots, x_n^{(k)})$, for $i = 1, \ldots, n$ and $k = 1, 2, 3, \ldots$. Essentially, we minimize f with respect to one element x_i, plug it back in f, and move to the next index.

Some guidelines for coordinate descent:

- The order of cycle through coordinates is arbitrary, we can use any permutation of $\{1, \ldots, n\}$;
- we can replace everywhere individual coordinates with blocks of coordinates (Just remember that this is possible if you see some such step in the future.)
- The "one-at-a-time" update scheme is critical, and "all-at-once" scheme does not necessarily converge;

Why is it used?

- Very simple and easy to implement.
- Careful implementations can achieve state-of-the-art solution.
- Scalable, e.g., don't need to keep full data in memory

1 Example: linear regression

Given $\mathbf{y} \in \mathbb{R}^n$ and $\mathbf{X}$ is $n \times p$ with columns $X_1, \ldots, X_p$, consider the linear regression problem: $\min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$.

Minimizing over β_i with all $\beta_j, j \neq i$ fixed: $0 = \frac{\partial f(\boldsymbol{\beta})}{\partial \beta_i} = X_i^T(\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) = X_i^T(X_i\beta_i + \mathbf{X}_{-i}\boldsymbol{\beta}_{-i} - \mathbf{y})$ where $\mathbf{X}_{-i}$ and $\boldsymbol{\beta}_{-i}$ are original matrix or vector with *i-th* column or element removed, respectively.

Thus

$$\beta_i = \frac{X_i^T(\mathbf{y} - \mathbf{X}_{-i}\boldsymbol{\beta}_{-i})}{X_i^T X_i}. \tag{1}$$

Like gradient descent, we will update $\boldsymbol{\beta}$ sequentially. At $k+1$-th iteration, the update for $\boldsymbol{\beta}$ be $\boldsymbol{\beta}^{(k+1)}$. Then according to the gradient descent algorithm $\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - L\frac{2}{n}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^{(k)}) = \boldsymbol{\beta}^{(k)} - L\frac{2}{n}\mathbf{X}^T(\mathbf{r}^k)$, where $\mathbf{r}_k = \mathbf{y} - \mathbf{X}\boldsymbol{\beta}^{(k)}$ is the residual error vector.

In case of coordinate descent, $\beta_i^{k+1} = \frac{X_i^T(\mathbf{y} - \mathbf{X}_{-i}\boldsymbol{\beta}_{-i} - \mathbf{X}_i\boldsymbol{\beta}_i^{(k)} + \mathbf{X}_i\boldsymbol{\beta}_i^{(k)})}{X_i^T X_i} = \beta_i^{(k)} + \frac{X_i^T\mathbf{r}^{(k)}}{X_i^T X_i}$. Thus, the two updates look very similar. The only difference is in the step size. For gradient-descent, it is fixed at L/n. However, here, it is coordinate specific, $1/X_i^T X_i$.

Coordinate descent repeats this update for $i = 1, 2, \ldots, p$, Note that the computational cost for one cycle of coordinate descent is $O(np)$ where $O(n)$ to compute $X_i^T(\mathbf{y} - \mathbf{X}_{-i}\boldsymbol{\beta}_{-i})$ for each update in a cycle (it is $O(n)$ because we can precompute $X_i^T X_i \beta_i$), which is the same as gradient descent.

```
x <- rnorm(1000)
z <- rnorm(1000)
w <- rnorm(1000)

y <- rnorm(1000, 2*x+3*z)

#We fit the model y=beta1*x+beta2*z+beta3*w + error
```

Apply coordinate descent in the above problem to estimate beta1, beta2 and beta3. Use (1) to get the updates of components of beta.

```
#Initialize
beta <- rep(0, 3)

#Design matrix
X = cbind(x, z, w)

for(itr in 1:10000){
  beta[1] <- t(X[,1])%*%(y-X[,-1]%*%beta[-1])/(t(X[,1])%*%X[,1])
  beta[2] <- t(X[,2])%*%(y-X[,-2]%*%beta[-2])/(t(X[,2])%*%X[,2])
  beta[3] <- t(X[,3])%*%(y-X[,-3]%*%beta[-3])/(t(X[,3])%*%X[,3])
}

#Least square estimate
```

```
betals <- solve(crossprod(X))%*%crossprod(X, y)

#Compare with least square estimate
sum((beta-betals)^2)
```

Here is the quicker code along with a stopping rule.

```
#Quicker code
x <- rnorm(1000)
z <- rnorm(1000)
w <- rnorm(1000)

y <- rnorm(1000, 2*x+3*z)


beta <- rep(0, 3)
#Design matrix

X = cbind(x, z, w)
res <- y-X%*%beta
for(itr in 1:10000){
  beta0itr <- beta0
  beta0 <- beta
  beta[1] <- beta[1] + sum(X[,1]*res)/sum(X[,1]*X[,1])
  res <- res + beta0[1] * X[,1] - beta[1] * X[,1]

  beta0 <- beta
  beta[2] <- beta[2] + sum(X[,2]*res)/sum(X[,2]*X[,2])
  res <- res + beta0[2] * X[,2] - beta[2] * X[,2]

  beta0 <- beta
  beta[3] <- beta[3] + sum(X[,3]*res)/sum(X[,3]*X[,3])
  res <- res + beta0[3] * X[,3] - beta[3] * X[,3]

  #stopping rule
  if(sum(((beta0itr-beta)/beta0itr)^2)<1e-3){
    break;
  }
```

```
}

beta
```

2 Pathwise Coordinate Descent for Lasso

The objective function is given by,

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} (\frac{1}{n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1),$$

where the part $\lambda\|\boldsymbol{\beta}\|_1$ is called the LASSO penalty [2]. Here $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{p} |\beta_j|$ where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$.

If we minimize above expression with respect to β_i, keeping everything else fixed, the minimizer would be $\hat{\beta}_i$?

Start with an initial solution of $\hat{\boldsymbol{\beta}}$. Define, $\beta_i^* = \frac{X_i^T(\mathbf{y} - \mathbf{X}_{-i}\hat{\boldsymbol{\beta}}_{-i})}{X_i^T X_i}$, the coordinate descent update 'like' in the least square case.

$$\hat{\beta}_i = \begin{cases} \beta_i^* - \frac{1}{X_i^T X_i}\frac{n}{2}\lambda & \text{if } \beta_i^* > 0, \frac{1}{X_i^T X_i}\frac{n}{2}\lambda \leq |\beta_i^*| \\ \beta_i^* + \frac{1}{X_i^T X_i}\frac{n}{2}\lambda & \text{if } \beta_i^* < 0, \frac{1}{X_i^T X_i}\frac{n}{2}\lambda \leq |\beta_i^*| \\ 0 & \text{if } \frac{1}{X_i^T X_i}\frac{n}{2}\lambda > |\beta_i^*|, \end{cases}$$

When the predictors are normalized to one i.e., $X_i^T X_i = 1$, the steps will be simplified. This above estimate help us to get the following way to get 'solution-path' of LASSO. Specifically, [1] proposed this the pathwise-coordinate descent method for Lasso problem. The algorithm runs over two loops. We start with the maximum value of $\lambda_{\max}$ for which the solution is zero. Outer Loop (pathwise strategy): • Compute the solution over a sequence $\lambda_{\max} = \lambda_1 > \lambda_2 > \cdots > \lambda_r$ of tuning parameter values • For tuning parameter value λ_k, initialize coordinate descent algorithm at the computed solution for λ_{k-1} for warm start.

3 Coordinate Gradient Descent

For a smooth function f, the iterations $x_i^{(k)} = x_i^{(k-1)} - t_{ki} \bigtriangledown f(x_1^{(k)}, \ldots, x_{i-1}^{(k)}, x_i, x_{i+1}^{(k-1)}, \ldots, x_n^{(k-1)})$, $i = 1, \ldots, n$ for $k = 1, 2, 3 \ldots$ are called coordinate gradient descent.

References

[1] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[2] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B*, 58:267–288, 1996.