CSE 250B | Homework 3 — Coordinate descent

Gopal Rander | A53210491

1. A short, high-level description of coordinate descent method

To apply coordinate descent method on Wine Dataset, we define the loss function as the negative log likelihood of the multiclass logistic regression with L2 normalized weights. These steps below mention the details on data pre-processing, the notation and formulas used in the process.

Step 1: Standardizing the features. We use feature scaling for every feature.

$$\vec{X_{strd}} = \frac{\vec{X} - \vec{\mu}}{\sigma}$$

Step 2: Log Likelihood and multiclass regression using structured output spaces.

For multiclass logistic regression, the loss function we are using is negative log likelihood.

$$Log\ likelihood = \sum_t log\ P(Y_t|\vec{X_t})$$

$$P(Y = i|\vec{X}) = \frac{e^{\vec{W_i}.\vec{X}}}{\sum_j e^{\vec{W_j}.\vec{X}}}$$

For coordinate-wise update, instead of using $\vec{W_i}$ for each class, we stack all the $\vec{W_i}$ in single $\vec{W}$ using structured output spaces technique. X is d dimensions and Y can take k different values.

$$\vec{W} = < \vec{W_1}\vec{W_2}\vec{W_3} \dots \vec{W_k} > \qquad \vec{W_i} \in \mathbb{R}^d, \qquad \vec{W} \in \mathbb{R}^{k*d}$$

We define a mapping $\Phi(X, y)$ on $(\mathcal{X}\ x\ \mathcal{Y}) \rightarrow\ \mathbb{R}^{d*k}$

as $\Phi(X, y) = < 0\ 0\ 0 \dots 0\ X_1\ X_2\ X_3 \dots X_d\ 0\ 0 \dots 0 > (\vec{X} at\ Y^{th}\ position, 0\ otherwise)$

Now, the Log likelihood can be written as:

$$Log\ likelihood = \sum_t \log\left[e^{\vec{W}.\ \Phi(\vec{X_t}, y_t)} \Big/ \sum_{y'} e^{\vec{W}.\ \Phi(\vec{X_t}, y')}\right]$$

Step 3: Loss function definition:

$$\mathcal{L}(\vec{W}) = -Log\ likelihood + ||\vec{W}||^2$$

$$\mathcal{L}(\vec{W}) = -\sum_t \log\left[e^{\vec{W}.\ \Phi(\vec{X_t}, y_t)} \Big/ \sum_{y'} e^{\vec{W}.\ \Phi(\vec{X_t}, y')}\right] + \left(\frac{1}{2}\right)||\vec{W}||^2$$

We need to minimize the loss, so we approach the problem by taking the first derivative along each coordinate of $\vec{W}$ separately.

Step 4: Derivative Calculation. The derivative of the loss function with respect to. W_α:

$$\frac{d\mathcal{L}}{dW_\alpha^{\rightarrow}} = -\sum_t \left[\Phi(X_t^{\rightarrow}, y_t)_\alpha - \sum_{y'} \left[\left(e^{W^{\rightarrow} \cdot \Phi(X_t^{\rightarrow}, y')} \Big/ \sum_{y''} e^{W^{\rightarrow} \cdot \Phi(X_t^{\rightarrow}, y'')} \right) * \Phi(X_t^{\rightarrow}, y')_\alpha \right] \right] + W_\alpha^{\rightarrow}$$

Step 5: Coordinate selection and update rule:

We start with $W^{\rightarrow} = 0^{\rightarrow}$.

At every iteration, we calculate the derivative of the Loss function with respect to each coordinate of $W^{\rightarrow}$. We select the coordinate along which the derivate is maximum.

$$Selection\ Rule: \ \omega = \underset{\alpha}{\operatorname{argmax}} \frac{d\mathcal{L}}{dW_\alpha^{\rightarrow}}$$

We update the ω^{th} coordinate of $W^{\rightarrow}$ by subtracting the derivative along ω^{th} direction.

$$Update\ Rule: \ W_\omega^{\rightarrow} \leftarrow W_\omega^{\rightarrow} - \eta \left(\frac{d\mathcal{L}}{dW_\omega^{\rightarrow}} \right)$$

η is a predefined learning rate.

For Coordinate descent, we choose the coordinate in W, along which we have the maximum gradient. Update rule for the selected coordinate is somewhat like that of Logistic regression update. But here, we do it coordinate-wise i.e. only one coordinate is updated in one step.

For this method to work, loss function **should be continuous and differentiable**.

2. Convergence

We know that the L2 regularized loss function (negative of log likelihood) is a concave function. So, this method will converge to a unique minimum for Loss function given that the learning rate is moderate or low.

For large values of learning rates, the update rule might cause a coordinate to diverge along its direction and overall Loss might oscillate.

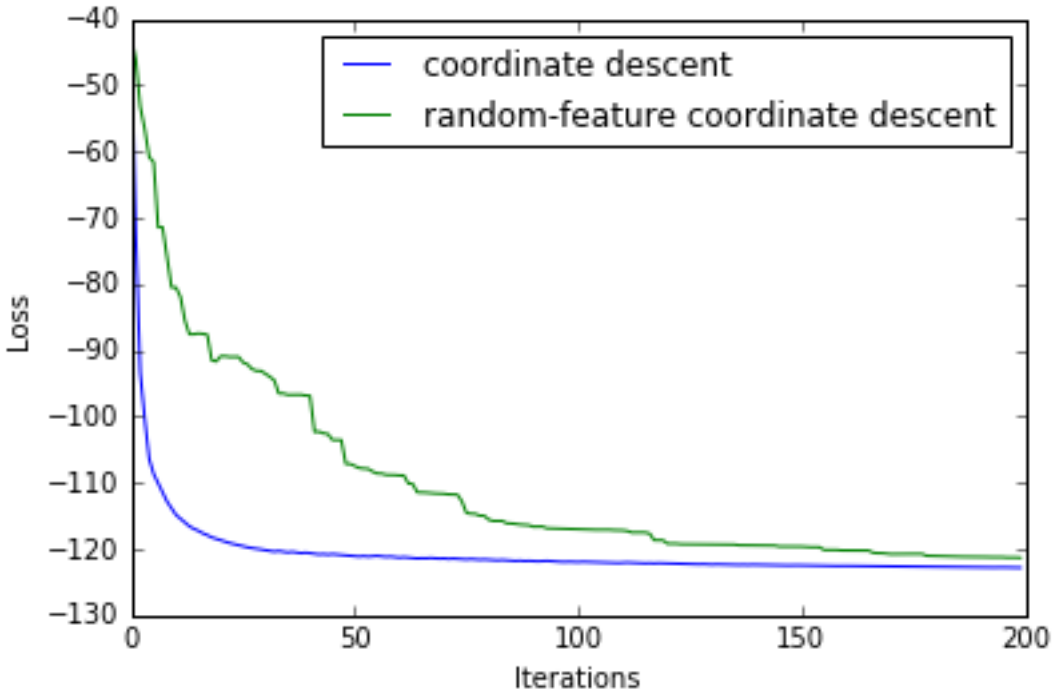
3. Experimental results

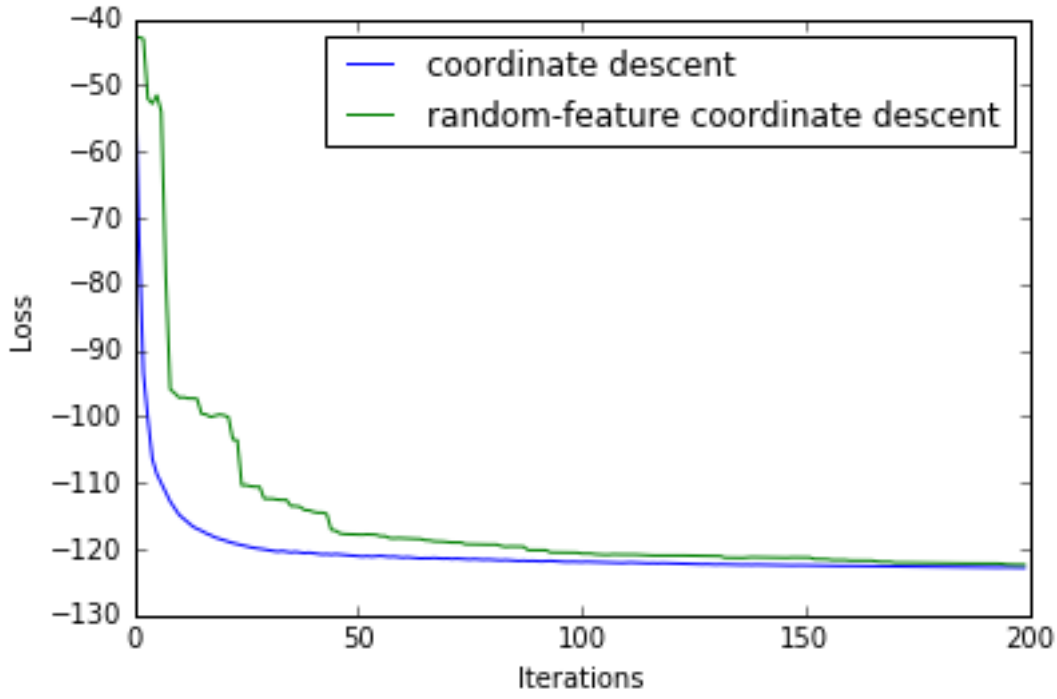
Training Data: 128 Selected randomly from the set of 178 observations of Wine dataset.
Test Data: Rest 50 observations
Number of Iterations: 200
Learning rate: 0.05

Results of running scikit-LogisticRegression multiclass with 'lbfgs' solver and default parameters [1].
Training Error: 0%
Test Error: 2%
Final Loss L^* = - 121.743017402

Results of running Coordinated descent method mentioned above:
Training Error: 0%
Test Error: 4%
Final Loss L = - 122.817135084

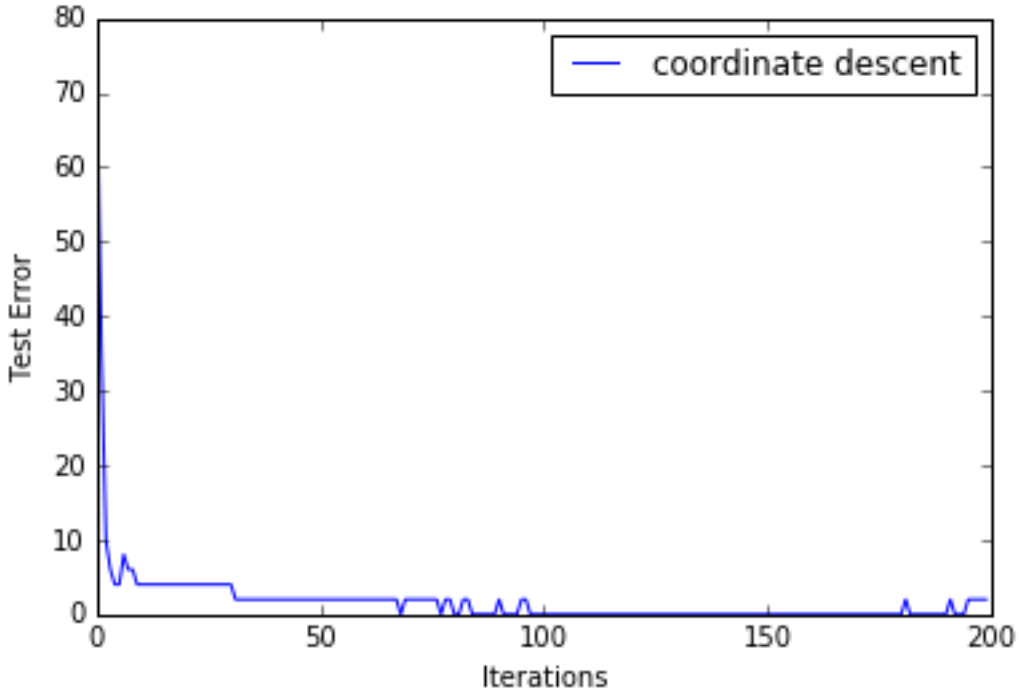
These plots compare the Random coordinate selection and update with the same rule. We observe that convergence with heuristic mentioned in coordinate descent above converges much faster than the random selection.

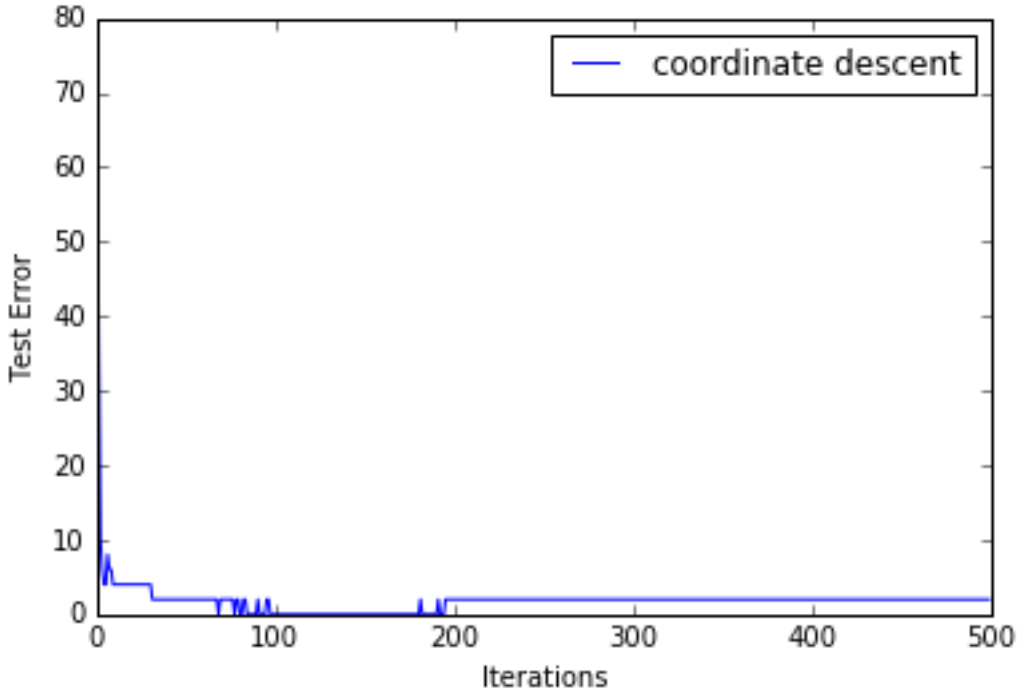




Also, We observed that the Final loss in coordinate descent is very close to the loss L^* observed for multiclass logistic regression. Both of them asympotes for more iterations.

This plot shows the test error rate (%) for the coordinate descent methods for first 200 iterations. The next plot shows the same for 500 iterations.





4. Critical evaluation

Yes, I believe that there is scope for further improvement in the coordinate descent method mentioned above.

First, coordinate descent method described here depends strongly on derivate. If the function is not derivable, we cannot use this method. We can define a better or let's say, derivative free update rule so that this method can be used for all functions. Furthermore, there are various heuristics which we can use to determine the learning rate on the run instead of fixing it at the beginning.

Coordinate Selection: In this method, the coordinate selection is dependent on the derivative with respect to each coordinate. Hence, it does not directly consider minimizing the overall loss during training. So, to select the coordinate, we can check the loss for each coordinate update and pick the one which gives a minimum loss. Although this will be computationally heavy, but in practice, it will converge faster than the derivate base selection method.

We used the first derivative in the update rule with a learning rate. One more update rule we could have tried was newton update, which is free of learning rate and supposed to converge much faster. That would require the function to have continuous second derivative defined.

[1]

Default parameters used in the sklearn.linear_model.LogisticRegression.

```
solver = 'lbfgs',
max_iter = T,
random_state = 0,
multi_class = 'multinomial'
C = 1.0
penalty = 'l2'
```