Project 2: Coordinate Descent

Jeff Makings

1 High-level Description

In this project, we seek to implement logistic regression via a coordinate descent learning method. We compared three algorithms to see which converged to the optimal loss the fastest: $coordinate\ search$, $iterative\ coordinate\ descent$, and finally, $random\ feature\ coordinate\ descent$. Each method differs in how a coordinate is chosen, although each method updates the chosen coordinate in the same way. In $coordinate\ search$, the gradient ($\frac{dL}{dw}$) of the loss function $L(w) = -(ylog(wx) + (1-y)log(1-wx))$ is computed for each iteration and the largest absolute value individual $\frac{dL}{dw_i}$ is chosen as the coordinate to update. From here, we set the new value of w_i for iteration $t+1$ via the formula $w_{it+1} = w_{it} - \eta\frac{dL}{dw_{it}}$.
In $iterative\ coordinate\ descent$, we begin from weight w_1 and update this coordinate via the same update rule $w_{1t+1} = w_{1t} - \eta\frac{dL}{dw_{1t}}$.
In the next iteration, we update w_2, then w_3, and so on. For d features, after updating w_d in the d^{th} iteration, we then update w_1 in the next iteration, and the loop continues. To our surprise, this method converged the fastest when compared to the other two methods.
Finally, $random\ feature\ coordinate\ descent$ simply picks a coordinate i at random and updates according to the same update rule $w_{it+1} = w_{it} - \eta\frac{dL}{dw_{it}}$.
Because we are computing the gradient for each iteration, $L(\cdot)$ must be differentiable, as the first derivative is taken in each iteration. To reach a global minima using coordinate descent, it's also important that the Hessian of $L(\cdot)$ be positive semi-definite, which confirms convexity.
This is the reason for using the $log\ loss$ function as opposed to modifying the logistic regression formula $\hat{Y} = \frac{1}{1+e^{-wx}}$ for our cost function. The log loss function we use is convex, and thus coordinate descent can find a global minima, while the logistic function is non-convex and coordinate descent may instead "get stuck" in a local minima, increasing total loss.

2 Convergence

The method converges to optimal loss when the difference in the loss function between iterations is very small, or in the other words, the gradient is approaching 0. This means that each increasing iteration in the training method results in increasingly small gains in accuracy for the model. Visually, this can be seen in a graph when the loss function "flattens out" after a number of iterations. For the method to converge to optimal loss, it is import that the objective function is convex if we are hoping to minimize it, or alternately, concave if we are maximizing the objective. In this case, the loss function $L(w) = -(ylog(wx) + (1-y)log(1-wx))$ can converge to optimal loss due to its convexity.

3 Experimental Methods and Results

3.1 Dataset and Normalization

To begin training our various logistic regression models, we download the UCI $wine$ data set from sklearn.datasets. Then, the data was normalized via the formula

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

where X is each individual feature data point and X_{max} and X_{min} are the maximum and minimum values for each feature class. This was done because the real valued ranges between different feature sets was 3 orders of magnitude in some cases. When trying to run coordinate descent on the raw data, this resulted in the coordinate search method updating only a single weight repeatedly due to the

scaling of this feature.

	Min	Max	Range
alcohol	11.03	14.83	3.80
malic_acid	0.74	5.80	5.06
ash	1.36	3.23	1.87
alcalinity_of_ash	10.60	30.00	19.40
magnesium	70.00	162.00	92.00
total_phenols	1.10	3.88	2.78
flavanoids	0.57	5.08	4.51
nonflavanoid_phenols	0.13	0.66	0.53
proanthocyanins	0.41	3.58	3.17
color_intensity	1.28	8.90	7.62
hue	0.69	1.71	1.02
OD280/OD315	1.59	4.00	2.41
proline	278.00	1680.00	1402.00

Fig. 1: Real-valued Features and their Ranges

The weights for features with high real values such as proline and magnesium were being updated disproportionately in the coordinate search method, even though there's no indication that these were the most predictive features for the classifier. Because of this, coordinate search never converged with real-valued data. Once the data was normalized, this was no longer a problem, as all feature values were between 0 and 1.

3.2 Unregularized Sklearn Logistic Regression

To get a baseline for other models, an unregularized logistic regression model from Sklearn was trained using the whole dataset (130 samples) of normalized data. To use Sklearn's coordinate descent method, "solver" was set to 'liblinear'. Regularization was removed by setting 'C=1000000'. Using Sklearn's Log Loss function, the total loss is computed via the formula:

$$L_{total} = -\sum_{i=1}^{n} ylog(p) + (1-y)log(1-p)$$

where p=Pr(y=1). For Sklearn's classifier L_{total} = **0.0044**. The mean loss is determined by dividing this by the number of samples. We will be using **mean** loss when referring to training loss L^* in the paper:

$$L_{mean} = \frac{-1}{n}\sum_{i=1}^{n} ylog(p) + (1-y)log(1-p)$$

Sklearn's mean loss:

$$L^* = L_{mean} = 3.394 * 10^{-5}$$

This baseline measurement $L^* =$ **0.000034** will be what we compare our models to.

3.3 Coordinate Descent Algorithms

Much is conserved between the three coordinate descent algorithms. Each algorithm's weights w were initialized as a vector of all 0s.
Then, for each iteration, the predicted probability Pr(Y=1) was computed for each sample via the formula

$$Pr(Y=1) = \hat{y} = \frac{1}{1+e^{-w_t * x}}$$

where w is the weights vector and x is the vector of features for a sample. Then the gradient of the loss function

$$\nabla L(w) = \frac{dL}{dw} = x[y - \hat{y}]$$

is computed to obtain the partial derivatives for each w_i.
At this point, depending on the algorithm being used, a coordinate i is chosen via the following methods:

3.3.1 Coordinate Search (Largest Partial Derivative)

In this method, the coordinate i is chosen as the max absolute value partial derivative in the gradient.

$$i = argmax_{(1..d)}(|\frac{dL}{dw_i}|)$$

then the weight w_i is updated via the formula

$$w_{it+1} = w_{it} - \eta\frac{dL}{dw_{it}}$$

3.3.2 Iterative Coordinate Descent

In the set of coordinates $i \in \{1, 2..., d\}$, i is incremented by 1 in each iteration. At iteration t=1, i =1, at t=2 i=2, etc. At $t = d$, $i = d$, and after this update each weight has been updated exactly once. Then in the next iteration at $t = d + 1$, the loop begins again and the coordinate to update is $i = 1$. Then the weight w_i is updated via the formula

$$w_{it+1} = w_{it} - \eta\frac{dL}{dw_{it}}$$

3.3.3 Random-Feature Coordinate Descent

In this method, the coordinate i to update in each iteration is chosen at random. This is done by choosing a number between 1 and d via the "random" python library.

$$i = random.randint(1, d)$$

Then, as usual, the weight w_i is updated via the formula

$$w_{it+1} = w_{it} - \eta \frac{dL}{dw_{it}}$$

3.3.4 Finishing up the iteration

Finally, for each of the above methods, the predicted probabilities are computed once again with the updated weights,

$$Pr(Y = 1) = \hat{y} = \frac{1}{1 + e^{-w_{(t+1)} * x}}$$

then the loss $L(w)$ is computed

$$L(w) = \frac{-1}{n} \sum_{i=1}^{n} ylog(\hat{y}) + (1 - y)log(1 - \hat{y})$$

and stored in an array for graphing later. This completes a single iteration.

3.4 Results

After testing with various η (learning rate) values and numbers of iterations for each model, we've set $\eta = 4$ and the number of iterations to 1 million to evaluate each coordinate descent method under the same parameters.

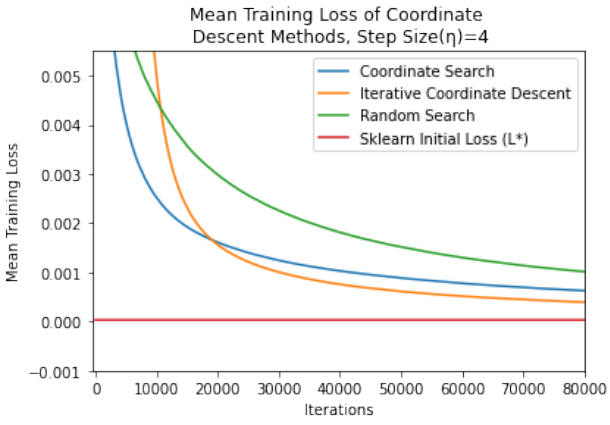

Figure 2: Convergence of 3 Algorithms

As seen above, the loss of all three methods decreases after each iteration. In the first 15,000 iterations, the Coordinate Search method has the lowest $L(w)$. However, just after this point the Iterative Coordinate Descent method has the lowest $L(w)$.

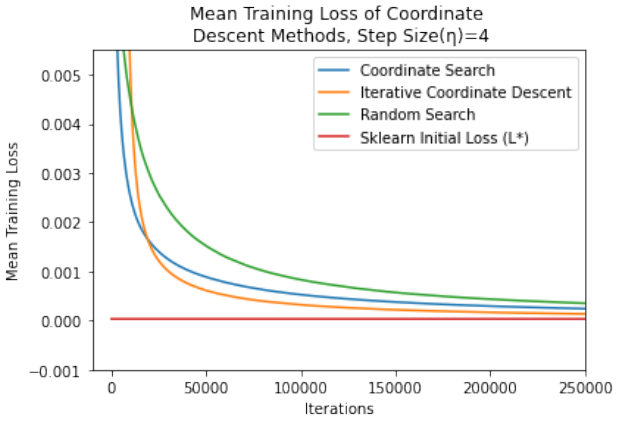

Figure 3: Loss Graph to 250000 Iterations

This demonstrates that all three methods converge towards L^*.

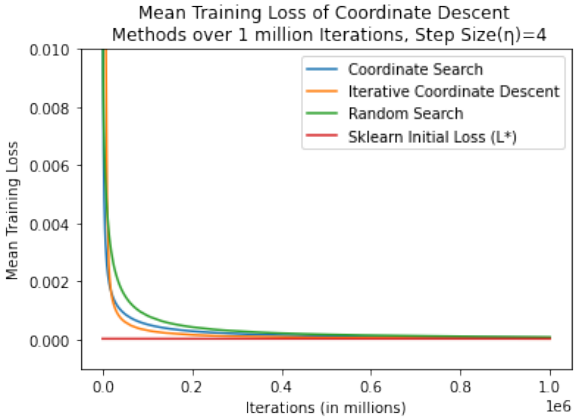

Fig. 4 (above): Loss over 1 Million Iterations

	Coordinate	Iterative	Random	Sklearn Loss
0	4.353555	6.760726	1.419935	0.000034
5000	0.004053	0.027691	0.006115	0.000034
10000	0.002518	0.005003	0.004483	0.000034
15000	0.001924	0.002314	0.003570	0.000034
20000	0.001611	0.001563	0.002986	0.000034
25000	0.001400	0.001216	0.002565	0.000034
100000	0.000527	0.000321	0.000831	0.000034
200000	0.000294	0.000168	0.000436	0.000034
250000	0.000242	0.000136	0.000352	0.000034
300000	0.000205	0.000114	0.000296	0.000034
400000	0.000157	0.000087	0.000224	0.000034
500000	0.000128	0.000070	0.000180	0.000034
750000	0.000087	0.000048	0.000121	0.000034
999999	0.000066	0.000037	0.000091	0.000034

Fig. 5 (above): Loss over 1 Million Iterations (X axis: CD methods Y axis: Iterations)

As illustrated in graph and table above, the Iterative Coordinate Descent method begins with the largest training loss and surpasses the Random

Feature method before 10000 iterations and finally surpasses the Coordinate Search Method before 20000 iterations. After 1 million iterations, the final $L(w)$ of Iterative Descent is **0.000037**, while Coordinate Search reaches **0.000066**, and Random Search reaches **0.000091**. All of these, and especially the Iterative Descent Method, come very close to the Sklearn loss of **0.000034**.

3.5 Analysis

These results surprised us, as we hypothesized the Coordinate Search method would have the lowest training loss for any number of iterations. This prediction stemmed from the Coordinate Search method choosing the largest partial derivative and therefore taking larger steps towards the global minima in each iteration.

However, it appears that updating each of the weights sequentially, as in Iterative Descent, results in faster convergence with sufficient iterations. Therefore, if the number of iterations/training runtime is not a constraint for creating a logistic regressor via coordinate descent, the Iterative Coordinate Descent method is superior to both Random-feature Coordinate Descent and the Coordinate Search method. All three methods converge to L^* and would likely be accurate classifiers.

4 Critical Evaluation

Iterative Coordinate Descent appears to be the superior coordinate descent method for this dataset and classifier, achieving a final $L(w)$ very close to L^*. However, there is still room for improvement in the Iterative Coordinate Descent learning method.

4.1 Learning Rates

The most obvious way to improve upon the algorithm would be to test a number of learning rates to see how an increase or decrease in η would change the rate of convergence on the training data, and how much these different learning rates could change the accuracy on a test set as well. For the analysed method, η was fixed at 4 in order to keep the learning rate constant for all tested methods.

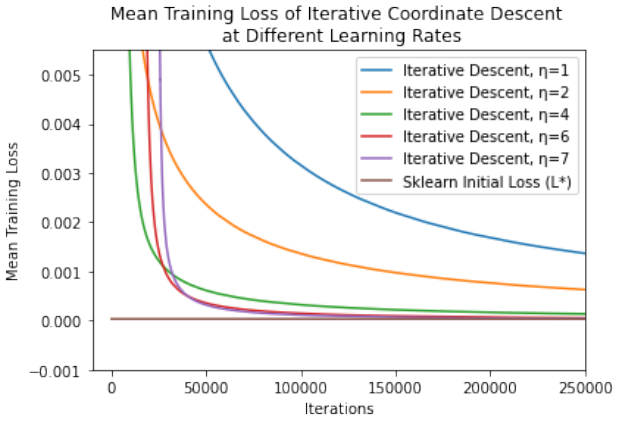

Fig. 6: Testing Different η for Iterative Descent
The above figure shows the various learning rates that could be used to increase or decrease the convergence of the Iterative Coordinate Descent method. Up until η = 7, the increased learning rate results in a faster rate of convergence on the training set. However, at η = 8, the final training loss skyrocked to 0.27, and the coordinate descent method was no longer able to converge.

4.2 Gradient efficiency

Another possible improvement of this algorithm would be to improve the efficiency of the gradient computation. Currently, for each iteration of the training method, the entire gradient is computed, getting partial derivatives with respect to all of the weights. However, we only need the partial derivative $\frac{dL}{dw_i}$ with respect to the coordinate weight w_i that is being updated. Furthermore, in our method, we know exactly which coordinate i is being updated, because it increases by exactly one in each iteration.

To very easily improve on this, we could determine which coordinate i is being updated in the iteration and take $\frac{dL}{dw_i}$ without computing the whole gradient. This would then be used in the update rule $w_{it+1} = w_{it} - \eta\frac{dL}{dw_{it}}$. The decreased amount of computations completed would hopefully result in a total decrease in the runtime of the learning method.

4.3 Further Testing

Finally, a way to improve and confirm the usefulness of the model that has been trained by the coordinate descent algorithm would be to test it on a test set. In this analysis, the models were trained on all 130 samples from the data set. There's no samples left to test the model on unfortunately, so we do not know if the model would be a useful classifier, or if it is horribly overfit to the data that

we have.

4.4 Final Thoughts

In conclusion, this project has shown that coordinate descent is a fairly straightforward learning algorithm that is in some ways simpler to implement than gradient descent, yet requires many updates to begin converging towards optimal loss.
However, when building a machine learning model, one never knows until they try a method to determine if said method is ideal for the dataset and goal. In this project, the ”best” learning method in theory (coordinate search) ended up being less effective than a more trivial approach (iterative descent). So although not as frequently discussed as gradient descent, coordinate descent is a key learning method to know for building machine learning models in the future.