COMP 330: Mini-Lecture on A5

Chris Jermaine and Kia Teymourian

Rice University

Goal: Perform Logistic Regression

- All begins with the formula from Kia's lecture
- For logistic regression, we have:

$$LLH(r_1, r_2, \ldots, r_d | x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = \sum_i y_i\theta_i - \log(1 + e^{\theta_i})$$

- Where $\theta_i = \sum_j r_j \times x_{i,j}$

Goal: Perform Logistic Regression

- All begins with the formula from Kia's lecture
- For logistic regression, we have:

$$LLH(r_1, r_2, \ldots, r_d | x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = \sum_i y_i \theta_i - \log(1 + e^{\theta_i})$$

- Where $\theta_i = \sum_j r_j \times x_{i,j}$
- Example: have a bunch of (e1score, e2score) pairs
- $(92, 12), (23, 67), (67, 92), (98, 78), (18, 45), (6, 100)$
- result in class: fail, fail, pass, pass, fail, fail

Goal: Perform Logistic Regression

- All begins with the formula from Kia's lecture
- For logistic regression, we have:

$$LLH(r_1, r_2, ..., r_d | x_1, x_2, ..., x_n, y_1, y_2, ..., y_n) = \sum_i y_i \theta_i - \log(1 + e^{\theta_i})$$

- Where $\theta_i = \sum_j r_j \times x_{i,j}$
- Example: have a bunch of (e1score, e2score) pairs
- $(92, 12), (23, 67), (67, 92), (98, 78), (18, 45), (6, 100)$
- result in class: fail, fail, pass, pass, fail, pass
- If coefs are (-1, -1), LLH is -335
- If coefs are (-1, 1), LLH is -78
- If coefs are (1, -1), LLH is -48
- If coefs are (1, 1), LLH is 32 (BEST!)

Loss Function

Learn this model using gradient descent

- Remember, GD tries to MINIMIZE

So our loss function is the NEGATIVE LLH

$$\sum_i -y_i\theta_i + \log(1 + e^{\theta_i})$$

Regularization

But regularize! Why?

- In previous example, if coefs are (1, 1), LLH is 32
- If coefs are (3, 3), LLH is 96
- If coefs are (4, 4), LLH is 127

So our loss regularized loss function is

$$\sum_i -y_i\theta_i + \log(1 + e^{\theta_i}) + \gamma \sum_j r_j^2$$

- With $\gamma = \mathbf{10}$, minimized at $r_1 = r_2 = \mathbf{0.8}$

Deriving Gradient Descent

Just have to take partial derivative wrt each r_k

Use this in the gradient calculation

Just to refresh your memory:

$$\frac{\partial f}{\partial r_{j'}} = -x_{i,j'}y_i + x_{i,j'}\left(\frac{e^{\theta}}{1+e^{\theta}}\right) + 2\gamma r_{j'}$$

Deriving Gradient Descent

Just to refresh your memory:

$$\frac{\partial f}{\partial r_{j'}} = -x_{i,j'}y_i + x_{i,j'}\left(\frac{e^{\theta}}{1+e^{\theta}}\right) + 2\gamma r_{j'}$$

Makes sense!

- Imagine model says 0, answer is 1. Then:

$$\frac{\partial f}{\partial r_{j'}} = -x_{i,j'} + x_{i,j'}(\text{small}) + 2\gamma r_{j'}$$

- So if $x_{i,j'}$ is positive, this gradient update will try to INCREASE $r_{j'}$
- Intuitively correct, since this will INCREASE θ, hence move us towards a 1
- But update tempered by $2\gamma r_{j'}$

Note on Implementation

Depending upon learning rate, will need a lot of iters to converge

So core loop MUST be fast

No joins, sorts, anything other than a MapReduce

Pseudo-code:

```
cur_answer = # some intialization
while no convergence:
    nextgradient = big_ass_list_of_vectors.map(
        # use cur_answer to get grad at current point
    ).reduce(
        # add up all of the one-point gradients
    )
    curanswer -= learning_rate * gradient
```

Questions?