4. Multinomial (Softmax) Regression and Gradient Descent

Project due Oct 21, 2020 05:29 IST *Completed*
Daniel suggests that instead of building ten models, we can expand a single logistic regression model into a multinomial regression and solve it with similar gradient descent algorithm.

The main function which you will call to run the code you will implement in this section is `run_softmax_on_MNIST` in main.py (already implemented). In the appendix at the bottom of this page, we describe a number of the methods that are already implemented for you in `softmax.py` that will be useful.

In order for the regression to work, you will need to implement three methods. Below we describe what the functions should do. We have included some test cases in `test.py` to help you verify that the methods you have implemented are behaving sensibly.

You will be working in the file `part1/softmax.py` in this problem

Computing Probabilities for Softmax

5.0/5.0 points (graded)
Write a function `compute_probabilities` that computes, for each data point $x^{(i)}$, the probability that $x^{(i)}$ is labeled as j for $j = 0, 1, \ldots, k-1$.

The softmax function h for a particular vector x requires computing

$$h(x) = \frac{1}{\sum_{j=0}^{k-1} e^{\theta_j \cdot x/\tau}} \begin{bmatrix} e^{\theta_0 \cdot x/\tau} \\ e^{\theta_1 \cdot x/\tau} \\ \vdots \\ e^{\theta_{k-1} \cdot x/\tau} \end{bmatrix},$$

where $\tau > 0$ is the **temperature parameter** . When computing the output probabilities (they should always be in the range $[0, 1]$), the terms $e^{\theta_j \cdot x/\tau}$ may be very large or very small, due to the use of the exponential function. This can cause numerical or overflow errors. To deal with this, we can simply subtract some fixed amount c from each exponent to keep the resulting number from getting too large. Since

$$\begin{aligned} h(x) &= \frac{e^{-c}}{e^{-c} \sum_{j=0}^{k-1} e^{\theta_j \cdot x/\tau}} \begin{bmatrix} e^{\theta_0 \cdot x/\tau} \\ e^{\theta_1 \cdot x/\tau} \\ \vdots \\ e^{\theta_{k-1} \cdot x/\tau} \end{bmatrix} \\ &= \frac{1}{\sum_{j=0}^{k-1} e^{[\theta_j \cdot x/\tau] - c}} \begin{bmatrix} e^{[\theta_0 \cdot x/\tau] - c} \\ e^{[\theta_1 \cdot x/\tau] - c} \\ \vdots \\ e^{[\theta_{k-1} \cdot x/\tau] - c} \end{bmatrix}, \end{aligned}$$

subtracting some fixed amount c from each exponent will not change the final probabilities. A suitable choice for this fixed amount is $c = \max_j \theta_j \cdot x/\tau$.

Reminder: You can implement this function locally first, and run `python test.py` in your project1 directory to validate basic functionality before checking against the online grader here.

Available Functions: You have access to the NumPy python library as `np` ; No need to import anything.

```
def compute_probabilities(X, theta, temp_parameter):
    """
    Computes, for each datapoint X[i], the probability that X[i] is labeled as j
```

```
    Computes, for each datapoint X[i], the probability that X[i] is labeled as j
    for j = 0, 1, ..., k-1

    Args:
        X - (n, d) NumPy array (n datapoints each with d features)
        theta - (k, d) NumPy array, where row j represents the parameters of our model for label j
        temp_parameter - the temperature parameter of softmax function (scalar)
    Returns:
        H - (k, n) NumPy array, where each entry H[j][i] is the probability that X[i] is labeled as j
    """
    #YOUR CODE HERE
    try:
        R = (theta.dot(X.T))/temp_parameter
        c = np.max(R, axis = 0)
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results

See full output

CORRECT

See full output

Submit You have used 1 of 25 attempts

Cost Function

5.0/5.0 points (graded)

Write a function `compute_cost_function` that computes the total cost over every data point.

The cost function $J(\theta)$ is given by: (Use natural log)

$$J(\theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{j=0}^{k-1}[[y^{(i)} == j]]\log\frac{e^{\theta_j \cdot x^{(i)}/\tau}}{\sum_{l=0}^{k-1} e^{\theta_l \cdot x^{(i)}/\tau}}\right] + \frac{\lambda}{2}\sum_{j=0}^{k-1}\sum_{i=0}^{d-1}\theta_{ji}^2$$

Available Functions: You have access to the NumPy python library as `np` and the previous function as `compute_probabilities`

```
def compute_cost_function(X, Y, theta, lambda_factor, temp_parameter):
    """
    Computes the total cost over every datapoint.

    Args:
        X - (n, d) NumPy array (n datapoints each with d features)
        Y - (n, ) NumPy array containing the labels (a number from 0-9) for each
            data point
        theta - (k, d) NumPy array, where row j represents the parameters of our
                model for label j
        lambda_factor - the regularization constant (scalar)
        temp_parameter - the temperature parameter of softmax function (scalar)

    Returns
        c - the cost value (scalar)
    """
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results

See full output

CORRECT

See full output

Submit You have used 1 of 25 attempts

Gradient Descent

5.0/5.0 points (graded)

Solution to this problem available before due date: The function `run_gradient_descent_iteration` is necessary for the rest of the project. Hence, once you have either submitted the correct function or finished your attempts for this problem, the solution to this function will be available.

Now, in order to run the gradient descent algorithm to minimize the cost function, we need to take the derivative of $J(\theta)$ with respect to a particular θ_m. Notice that within $J(\theta)$, we have:

$$\frac{e^{\theta_j \cdot x^{(i)}/\tau}}{\sum_{l=0}^{k-1} e^{\theta_l \cdot x^{(i)}/\tau}} = p\left(y^{(i)} = j | x^{(i)}, \theta\right)$$

so we first compute: $\frac{\partial p(y^{(i)}=j|x^{(i)},\theta)}{\partial \theta_m}$,
when $m = j$,

$$\frac{\partial p\left(y^{(i)} = j | x^{(i)}, \theta\right)}{\partial \theta_m} = \frac{x^{(i)}}{\tau} p\left(y^{(i)} = m | x^{(i)}, \theta\right) \left[1 - p\left(y^{(i)} = m | x^{(i)}, \theta\right)\right]$$

when $m \neq j$,

$$\frac{\partial p\left(y^{(i)} = j | x^{(i)}, \theta\right)}{\partial \theta_m} = -\frac{x^{(i)}}{\tau} p\left(y^{(i)} = m | x^{(i)}, \theta\right) p\left(y^{(i)} = j | x^{(i)}, \theta\right)$$

Now we compute

$$\begin{aligned}
\frac{\partial}{\partial \theta_m}\left[\sum_{j=0}^{k-1} [[y^{(i)} == j]] \log \frac{e^{\theta_j \cdot x^{(i)}/\tau}}{\sum_{l=0}^{k-1} e^{\theta_l \cdot x^{(i)}/\tau}}\right] &= \sum_{j=0, j\neq m}^{k-1} \left[[[y^{(i)} == j]] \left[-\frac{x^{(i)}}{\tau} p\left(y^{(i)} = m | x^{(i)}, \theta\right)\right]\right] \\
&\quad + [[y^{(i)} == m]] \frac{x^{(i)}}{\tau} \left[1 - p\left(y^{(i)} = m | x^{(i)}, \theta\right)\right] \\
&= \frac{x^{(i)}}{\tau}\left[[[y^{(i)} == m]] - p\left(y^{(i)} = m | x^{(i)}, \theta\right) \sum_{j=0}^{k-1} [[y^{(i)} == j]]\right] \\
&= \frac{x^{(i)}}{\tau}\left[[[y^{(i)} == m]] - p\left(y^{(i)} = m | x^{(i)}, \theta\right)\right]
\end{aligned}$$

Plug this into the derivatite of $J(\theta)$, we have

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta_m} &= \frac{\partial}{\partial \theta_m}\left[-\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{j=0}^{k-1} [[y^{(i)} == j]] \log p\left(y^{(i)} = j | x^{(i)}, \theta\right)\right] + \frac{\lambda}{2}\sum_{j=0}^{k-1}\sum_{i=0}^{d-1} \theta_{ji}^2\right] \\
&= -\frac{1}{\tau n}\sum_{i=1}^{n}\left[x^{(i)}\left([[y^{(i)} == m]] - p\left(y^{(i)} = m | x^{(i)}, \theta\right)\right)\right] + \lambda\theta_m
\end{aligned}$$

To run gradient descent, we will update θ at each step with $\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$, where α is the learning rate.

Write a function `run_gradient_descent_iteration` that runs one step of the gradient descent algorithm.

Available Functions: You have access to the NumPy python library as `np`, `compute_probabilities` which you previously implemented and `scipy.sparse` as `sparse`

previously implemented and `scipy.sparse` as `sparse`.

You should use `sparse.coo_matrix` so that your function can handle larger matrices efficiently (and not time out for the online graders). The sparse matrix representation can handle sparse matrices efficiently.

Hint

Show

```
def run_gradient_descent_iteration(X, Y, theta, alpha, lambda_factor, temp_parameter):
    """
    Runs one step of batch gradient descent

    Args:
        X - (n, d) NumPy array (n datapoints each with d features)
        Y - (n, ) NumPy array containing the labels (a number from 0-9) for each
            data point
        theta - (k, d) NumPy array, where row j represents the parameters of our
                model for label j
        alpha - the learning rate (scalar)
        lambda_factor - the regularization constant (scalar)
        temp_parameter - the temperature parameter of softmax function (scalar)

    Returns:
        theta - (k, d) NumPy array that is the final value of parameters theta
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results

See full output

CORRECT

See full output

Submit You have used 1 of 20 attempts

Test Error on Softmax Regression

1.0/1.0 point (graded)

Finally, report the final test error by running the `main.py` file, using the temperature parameter $\tau = 1$. If you have implemented everything correctly, the error on the test set should be around 0.1, which implies the linear softmax regression model is able to recognize MNIST digits with around 90 percent accuracy.

Note: For this project we will be looking at the error rate defined as the fraction of labels that don't match the target labels, also known as the "gold labels" or ground truth. (In other contexts, you might want to consider other performance measures such as precision and recall, which we have not discussed in this class.

Please enter the **test error** of your Softmax algorithm (copy the output from the `main.py` run).

0.10050000000000003 ✔ **Answer:** 0.1005

Submit You have used 1 of 20 attempts

Answers are displayed within the problem

Discussion

Hide Discussion

Topic: Unit 2 Nonlinear Classification, Linear regression, Collaborative Filtering (2 weeks):Project 2: Digit recognition (Part 1) / 4. Multinomial (Softmax) Regression and Gradient Descent

Add a Post

Show all posts by recent activity

Tips for Cost Function Implementation Pinned Staff		29
super confused about the summation part of gradient descent For each theta, we run the steps, Can someone please help me understand as to What is the desired output of the first part(the sum...		4
Gradient of J(theta)		3
Beat sparse.coo_matrix with np.where Title says it all. sparse.coo_matrix is something like 16 times faster than naive. np.where is something like 5 times faster than sparse....		5
SOFTMAX useful link Stanford, I wonder why was not explained in the video lessons... At Standford is well explained, in this course insted was instead not explained during the course and the description, given in the pr...		1
How long is softmax taking for you guys to converge? Initially, it was taking around 8-9 minutes for me and then I vecotorized the code and now it takes around 2-3 minutes. I still feel it's ...		8
Useful tip for Gradient Descent		8
Can't get test error on Softmax regression due to long running time I tried to make sure there's no for loop in my code at all and also use sparse.coo.matrix when did one-hot encoding part, but still got...		1
what does this line of code do ? I do not understand this, hel please M = sparse.coo_matrix(([1]*n, (Y, range(n))), shape=(k,n)).toarray()		4
Why test error higher than for Support Vector Machine Just wondering why we find a test error higher in the conclusion of Part 4 compared to the conclusion of Part 3. Isn't Softmax suppo...		3
I give up I have been trying to catch up with all the assignments of this course but I'm giving up. In my personal opinion, this program is not f...		1
[Staff] On the deadline for the Verified Track Unfortunately, due to being incredibly busy in university, i've managed to miss the verification deadline for this course. Would it be p...		1
Gradient Descent I am getting only zeros in my returned theta, can anyone suggest a solution for this?		2

Previous Next

/