Homework 3

Due: Thursday, February 13, 2025 at 12:00pm **(Noon)**

Written Assignment

Problem 1: Gradient Descent

(13 points)

In this problem, we'll examine a gradient descent type algorithm which could be used to find the minimum of a function $f : \mathbb{R} \to \mathbb{R}$ over a closed interval $[-b, b]$ (given some $b > 0$). Importantly, we must assume that f is convex over $[-b, b]$. You may also assume that f is twice continuously differentiable; in such a setting, f being convex is equivalent to $f''(x) \geq 0$. Finally, we will use α to denote the learning rate parameter, which is some positive number used to scale the optimizer's steps as desired.

The steps of gradient descent are as follows:

1. Begin at $x_0 = 0$
2. At each step, set $x_{t+1} = x_t - \alpha f'(x_t)$.
3. If $x_{t+1} < -b$, set $x_{t+1} = -b$. If $x_{t+1} > b$, set $x_{t+1} = b$. Otherwise, continue.
4. Repeat (2) and (3) until ϵ-convergence.

We say that an optimization algorithm (such as gradient descent) ϵ-*converges* if, at some point, x_t stays within ϵ of the true minimum. Formally, we have ϵ-convergence at time t if

$$|x_{t'} - x_{\min}| \leq \epsilon, \qquad \text{where } x_{\min} = \arg\min_{x \in [-b,b]} f(x)$$

for all $t' \geq t$.

a. For $\alpha = 0.1$, $b = 1$, and $\epsilon = 0.001$, find a convex function f so that running gradient descent does not ϵ-converge. Specifically, make it so that $x_0 = 0$, $x_1 = b$, $x_2 = -b$, $x_3 = b$, $x_4 = -b$, etc.

Solution:
It suffices to have a function in which $f'(1) = 10$, $f'(0) = -5$, and $f'(-1) = -10$.
Consider $f(x) = 100(x - 1/2)^2$.

$f'(x) = 200(x - 1/2)$
$f'(0) = -100$
$f'(1) = 100$
$f'(-1) = -300$

The first step will shoot to $x_1 = 0 - .1(-100) = 10$, which is clipped back to 1.
The second step will shoot to $x_2 = 1 - .1(100) = -9$, which is clipped back to -1.
The third step will shoot to $x_3 = -1 - .1(-300) = 29$, which is clipped back to 1.

b. For $\alpha = 0.1$, $b = 1$, and $\epsilon = 0.001$, find a convex function f so that gradient descent does ϵ-converge, but only after at least 10,000 steps.

Solution:
Consider $f(x) = 0.0001x$. Within the range $[-b, b] = [-1, 1]$, this function is minimized at $x_{min} = -1$. Under gradient descent, each corresponding x_{t+1} would be $0.1 * 0.0001 = 0.00001$ smaller than x_t. Since x_0 starts at 0, after 100,000 steps, $x_{10,000} = 0 - 100,000 * 0.00001 = -1 = x_{min}$.

There are many possible solutions, as long as they converge after at least 10,000 steps.

c. Construct a different optimization algorithm that has the property that it will always ϵ-converge (for any convex f) within $\log_2(2b/\epsilon)$ steps.

Solution:
Consider a binary search, where we start at $-b$ and end at b. Here is the pseudocode:

Algorithm 1 Find-Max($A[1 \dots n]$)

```
start ← −b, end ← b
while start < end do
    x_t ← (start+end)/2
    if f'(x_t) < 0 then
        start ← x_t
    else
        end ← x_t
    end if
    t ← t + 1
end while
return x_t
```

At each iteration, we split the function in half and search for the minimum. To do this, we calculate $f'(x_t)$, which tells us which direction the function is decreasing. If the $f'(x_t) < 0$, the function is decreasing, and thus the minimum would likely be on the right half. Otherwise, we look on the left.

The initial search range is of size $2b$. After splitting the range in half for t iterations, the minimum has been pinned into a range of $\frac{2b}{2^t} = \epsilon$. Solving for t, we get $t = log_2(\frac{2b}{\epsilon})$.

d. **(Extra credit)**
Unfortunately, even if x_t is within ϵ of $x_{\min}$, $f(x_t)$ can be arbitrarily greater than $f(x_{\min})$. However, consider the case where the derivative of f is always between $-r$ and r ($\forall x \in [-b, b]$, $f'(x) \in [-r, r]$). In this case, we can make a guarantee about the difference between $f(x_t)$ and $f(x_{\min})$.

Given that $|x_t - x_{\min}| \leq \epsilon$ and that $-r \leq f'(x) \leq r$, find a bound on $|f(x_t) - f(x_{\min})|$ in terms of ϵ and r.

Solution:
We start with our two given statements, then construct a bound for the integral of $f'(x)$.

$$\begin{aligned} &|x_t - x_{min}| \leq \epsilon \\ &-r \leq f'(x) \leq r \implies -rx \leq f(x) + c \leq rx \end{aligned}$$

Next, we apply the bounds to $f(x_t)$ and $f(x_{min})$.

$$-rx_t \leq f(x_t) + c \leq rx_t \tag{1}$$
$$-rx_{min} \leq f(x_{min}) + c \leq rx_{min} \tag{2}$$

From this, we perform (1) − (2) to get:

$$-r(x_t - x_{min}) \leq f(x_t) - f(x_{min}) \leq r(x_t - x_{min})$$
$$|f(x_t) - f(x_{min})| \leq |r(x_t - x_{min})|$$

We know that r is nonnegative (from the statement $-r \leq r$ above), so we can pull it out from the absolute value. This leaves us with:

$$|f(x_t) - f(x_{min})| \leq r|x_t - x_{min}|$$
$$|f(x_t) - f(x_{min})| \leq r\epsilon$$

Problem 2: Logistic Regression

(7 points)

Suppose we collect data on a set of bakeries in Providence. For each bakery, we've measured the average sweetness level of their best selling dessert (on a scale of 1 to 100), x_1, and the average number of layers in their signature cakes, x_2. Each bakery also has a label indicating whether or not it received a "Gold Spoon" award from a prestigious food magazine. From this data, we decide to fit a logistic regression and determine that $\mathbf{w} = (0.05, 1, -6)$, where the last component is the bias.

a. What is the probability that a bakery with a dessert sweetness level of 20 and an average of 3.8 cake layers gets a "Gold Spoon"?

b. How many average cake layers would the bakery in part (a) need in order to have a 50% chance of getting a "Gold Spoon"?

c. Suppose we want to extend our logistic regression model to predict all possible award tiers (Gold Spoon, Silver Spoon, Bronze Spoon, etc.). Why should we use the softmax function in this case, but not previously?

Solution:

1. Let $\mathbf{x} = (20, 3.8, 1)$. Then $\langle \mathbf{w}, \mathbf{x} \rangle = -1.2$, so $h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{1.2}} \approx \boxed{23.15\%}$.

2. Solving $0.5 = \frac{1}{1+e^{-\langle \mathbf{w}, \mathbf{x} \rangle}}$ yields that $e^{-\langle \mathbf{w}, \mathbf{x} \rangle} = 1$. Thus, $\langle \mathbf{w}, \mathbf{x} \rangle = 0$ and $\mathbf{x} = \langle 20, 5, 1 \rangle$ or $\boxed{5}$ layers.

3. Students should address the fact that multiclass logistic regression involves a vector-valued predictor, whereas binary logistic regression is a single linear predictor. Thus, the softmax function is not necessary for binary regression as the value of the predictor alone determines the class. However, with multiclass regression, the values associated with each class must be normalized to ensure we have a valid probability distribution. Only then can we interpret the values of the multiclass predictor as probabilities and select the most likely class.

Programming Assignment

Introduction

In this assignment, you will be using a modified version of the UCI Census Income data set to predict the education levels of individuals based on certain attributes collected from the 1994 census database. You can read more about the dataset here: https://archive.ics.uci.edu/ml/datasets/Census+Income.

Relevant textbook sections: 9.2 (pg 123), 9.3 (pg 126), 14.3 (pg 191)

Stencil Code & Data

You can find the stencil code and dataset for this assignment on Github classroom at this link. For more details, please see the download/submission guide.

We have provided the following stencil code:

- `main.py` is the entry point of program which will read in the datasets, run the models and print the results.
- `models.py` contains the `LogisticRegression` model which you will be implementing.

You should only need to modify code marked by `#TODO` in `models.py` to complete the project. If you edit anything else for other purposes, please make sure all of your additions are commented out in the final handin.
To run the program, run `python main.py` in a terminal. Make sure you activate the virtual environment first when working over ssh or on a department machine:

```
source /course/cs1420/cs142_env/bin/activate
```

The Assignment

In `models.py`, there are a few functions you will implement. They are:

- `LogisticRegression`:
  - **train()** uses stochastic gradient descent to train the weights of the model.
  - **loss()** calculates the log loss of some dataset divided by the number of examples.
  - **predict()** predicts the labels of data points using the trained weights. For each data point, you should apply the softmax function to it and return the label with the highest assigned probability.
  - **accuracy()** computes the percentage of the correctly predicted labels over a dataset.

Note: You are not allowed to use any packages that have already implemented these models (e.g. scikit-learn). We have also included some code in `main.py` for you to test out the different random seeds and calculate the average accuracy of your model across those random seeds.

Logistic Regression

Logistic Regression, despite its name, is used in classification problems. It learns sigmoid functions of the inputs

$$h_{\mathbf{w}}(\mathbf{x})_j = \phi_{sig}(\langle \mathbf{w}_j, \mathbf{x} \rangle)$$

where $h_{\mathbf{w}}(\mathbf{x})_j$ is the probability that sample $\mathbf{x}$ is a member of class j.

In multi-class classification, we need to apply the `softmax` function to normalize the probabilities of each class. The loss function of a Logistic Regression classifier over k classes on a *single* example (x, y) is the **log-loss**, sometimes called **cross-entropy loss**:

$$\ell(h_{\mathbf{w}}, (\mathbf{x}, y)) = -\sum_{j=1}^{k} \left\{ \begin{array}{ll} \log(h_{\mathbf{w}}(\mathbf{x})_j), & y = j \\ 0, & \text{otherwise} \end{array} \right\}$$

Therefore, the ERM hypothesis of $\mathbf{w}$ on a dataset of m samples has weights

$$\mathbf{w} = \text{argmin}_{\mathbf{w}}(-\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} \left\{ \begin{array}{ll} \log(h_{\mathbf{w}}(\mathbf{x}_i)_j), & y_i = j \\ 0, & \text{otherwise} \end{array} \right\})$$

To learn the ERM hypothesis, we need to perform gradient descent. The partial derivative of the loss function on a single data point $(\mathbf{x}, y)$ with respect to an individual weight w_{st} is

$$\frac{\partial l_S(h_{\mathbf{w}})}{\partial w_{st}} = \left\{ \begin{array}{ll} h_{\mathbf{w}}(\mathbf{x})_s - 1, & y = s \\ h_{\mathbf{w}}(\mathbf{x})_s, & \text{otherwise} \end{array} \right\} x_t$$

With respect to a single row in the weights matrix, $\mathbf{w}_s$, the partial derivative of the loss is

$$\frac{\partial l_S(h_{\mathbf{w}})}{\partial \mathbf{w}_s} = \left\{ \begin{array}{ll} h_{\mathbf{w}}(\mathbf{x})_s - 1, & y = s \\ h_{\mathbf{w}}(\mathbf{x})_s, & \text{otherwise} \end{array} \right\} \mathbf{x}$$

You will need to descend this gradient to update the weights of your Logistic Regression model.

Stochastic Gradient Descent

You will be using Stochastic Gradient Descent (SGD) to train your `LogisticRegression` model. Below, we have provided pseudocode for SGD on a sample S.

Hints: Consistent with the notation in the lecture, $\mathbf{w}$ are initialized as a $k \times d$ matrix, where k is the number of classes and d is the number of features (with the bias term). With n as the number of examples, X is a $n \times d$ matrix, and $\mathbf{y}$ is a vector of length n.

Tuning Parameters

Convergence is achieved when the change in loss between iterations is some small value. Usually, this value will be very close to but not equal to zero, so it is up to you to tune this threshold value to best optimize your model's performance. Typically, this number will be some magnitude of 10^{-x}, where you experiment with x. Note that when calculating the loss for checking convergence, you should be calculating the loss for the entire dataset, not for a single batch (i.e., at the end of every epoch).

You will also be tuning batch size (and one of the report questions addresses the impact of batch size on model performance). In order to reach the accuracy threshold, you will need to tune both parameters. α would typically be tuned during the training process, but we are fixing $\alpha = 0.03$ for this assignment. **Please do not change α in your code**.

You can tune the batch size and convergence threshold in `main.py`.

Algorithm 2 Stochastic Gradient Descent

Require: $b > 0$ ▷ Batch size
Require: $\alpha > 0$ ▷ Learning rate
Require: n_{features} ▷ Number of features
Require: n_{classes} ▷ Number of classes
Require: w initialized ▷ Weights matrix
Require: `CONV_THRESHOLD` > 0 ▷ Convergence threshold

procedure (X, Y)
    `converged` $\leftarrow$ False
    `epoch` $\leftarrow 0$
    $n \leftarrow$ `len`(X)
    $L_0 \leftarrow +\infty$
    while `converged` is False **do**
        `epoch` $\leftarrow$ `epoch`$+1$
        Shuffle (X, Y) indices
        ▷ You may find `np.random.shuffle` useful
        for $i = 0$ **to** $\lceil n/b \rceil - 1$ **do**
            $X' \leftarrow X[ib : (i+1)b]$
            $Y' \leftarrow Y[ib : (i+1)b]$
            ▷ Grabs the current batch of examples and labels together
            $n' \leftarrow$ `len`(X')
            $\nabla L_w \leftarrow 0^{n_{\text{features}} \times 1}$
            for $(x, y) \in (X', Y')$ **do** ▷ You may find `zip` useful
                for $j = 0$ **to** $n_{\text{classes}} - 1$ **do**
                    if $y = j$ **then**
                        $\nabla L_{w_j} \leftarrow \nabla L_{w_j} + (\texttt{softmax}(w \cdot x)_j - 1) \cdot x$
                    else
                        $\nabla L_{w_j} \leftarrow \nabla L_{w_j} + (\texttt{softmax}(w \cdot x)_j) \cdot x$
                    end if
                end for
            end for
            $w = w - \frac{\alpha}{n'} \nabla L_w$
        end for
        if $|\mathcal{L}(X, Y) - L_0| <$`CONV_THRESHOLD` **then**
            `converged` $\leftarrow$ True
        else
            $L_0 \leftarrow \mathcal{L}(X, Y)$
        end if
        ▷ If the change in loss $\mathcal{L}(\cdot) - L_0$ has reached the threshold, we end the loop. Otherwise, we need to store the current value to compare in the next epoch
    end while
end procedure

Project Report

This section outlines some guiding questions that you should answer in your report. Please leave any code that you use in your final handin but make sure that it is **not** run by default when your program is run (i.e., comment it out). You may use any program to create the PDF file, but we highly recommend using LaTeX. We have provided an example report available on our course website to get you started.

Report Questions

1. Make sure that you have implemented a variable batch size using the constructor given for `LogisticRegression`. Try different batch sizes (e.g. 1, 5, 10, 75, etc.) and report the accuracy and number of epochs taken to converge.

   a. What tradeoffs exist between good accuracy and quick convergence?

   Solution: For a lower convergence threshold, SGD will take longer to converge and SGD will better approximate the ideal weights that correctly classify the training set. This leads to higher accuracy initially as you decrease the criteria, but could also result in overfitting. For a higher convergence threshold, SGD will converge faster, but could result in a lower accuracy (since training loss will remain higher). This is why it is important to experiment with different convergence thresholds and batch sizes to find the right balance.

   b. Why do you think the batch size led to the results you received?

   Solution: Students should notice that as they increase the batch size, the number of epochs it takes to cause their loss to converge decreases. However, if one uses a batch size that is too large, the number of epochs will increase again.

2. Take a look at the Colab notebook in `this folder` we provided to clean and process the data. Which categories did we one-hot encode and why? How does a one-hot representation compare to an enumerations of the possible values for that feature?

   Using Google Colab: If the notebook looks like a long text file, then click the button at the top center that says "Open with Google Colaboratory." If this is your first time using Google Colab, you may need to click the dropdown on the top center, click `Connect more apps`, and connect Google Colab first. Afterwards, click the button again to open the notebook in Google Colab. To run the notebook, you may either need to open in playground mode or make a copy.

   Solution: We will one-hot encode attributes with multiple categories by turning each category into an attribute with a 1 if that example contains that category and a 0 if it does not. This will allow us to numerically express that an example belongs to a category without implying that its attribute follows some kind of cardinal order (that the order of the numbers matters). This is better than the alternative of assigning the attribute's categories unique numbers since that would imply that the order of the numbers means something about the attribute.

3. Try to run the model with `unnormalized_data.csv` instead of `normalized_data.csv`. Report your findings when running the model on the unnormalized data. In a few short sentences, explain what normalizing the data does and why it affected your model's performance.

   Solution: There is a gap in accuracy when training on normalized vs unnormalized data. This gap is huge because the optimizer cannot tune the weights for all features at the same time very well, since there's one step size (i.e., alpha in the pseudocode, also referred to as learning rate) for all weights. In other words, if feature A ranges from 0 to 1000 and feature B ranges from 0 to 1, then the weights corresponding to feature A should in theory have a larger degree of change than the weights corresponding to feature B because of the sheer range of feature A's values compared to feature B's in order to have a satisfying accuracy.

4. Try the model with `normalized_data_nosens.csv`; in this data file, we have removed sensitive information such as the `race` and `sex` attributes. Report your findings on the accuracy of your model on this dataset (averaging over many random seeds here may be useful). Can we make any conclusion based on these accuracy results about whether there is a correlation between sex/race and education level? Why or why not?

   Solution: We expect the accuracy to stay approximately the same. However, we can't make a claim that there is no correlation between sex/race and education level. We expect answers like: *accuracy* is distinct from *correlation*; or there may be other attributes that serve as proxy variables for race and gender.

Grading Breakdown

We expect your `LogisticRegression` model to reach a test accuracy of 80% or above and run in under one minute. Since we are setting a random seed in the stencil code, you should not have to worry about randomness affecting your model performance.

As always, you will primarily be graded on the correctness of your code and not based on whether it does or does not achieve the accuracy targets.

The grading breakdown for the assignment is as follows:

Written Assignment	20%
Logistic Regression	50%
Report	30%
Total	100%

Handing in

You will hand in both the written assignment and the coding portion on gradescope, separately.

1. Your written assignment should be uploaded to gradescope under "Homework 3".
2. Submit your hw3 github repo containing all your source code and your project report named **report.pdf** on gradescope under "Homework 3 Code". **report.pdf** should live in the root directory of your code folder; the autograder will check for the existence of this file and inform you if it is not found. For questions, please consult the download/submission guide.

If you have questions on how to set up or use Gradescope, ask on Edstem! For this assignment, you should have written answers for Problems 1, 2, and 3.

Anonymous Grading

You need to be graded anonymously, so do not write your name anywhere on your handin.

Obligatory Note on Academic Integrity

Plagiarism—don't do it.

As outlined in the Brown Academic Code, attempting to pass off another's work as your own can result in failing the assignment, failing this course, or even dismissal or expulsion from Brown. More than that, you will be missing out on the goal of your education, which is the cultivation of your own mind, thoughts, and abilities. Please review this course's collaboration policy and, if you have any questions, please contact a member of the course staff.