Homework 3

Due: Thursday, February 17, 2022 at 12:00pm **(Noon)**

Written Assignment

Problem 1: Gradient Descent

(20 points)

Consider using gradient descent to find the minimum of f, where:

- f is a convex function over the closed interval $[-b, b]$, $b > 0$
- f' is the derivative of f
- α is some positive number, which will represent a learning rate parameter

The steps of gradient descent are as follows:

1. Start at $x_0 = 0$
2. At each step, set $x_{t+1} = x_t - \alpha f'(x_t)$
3. If x_{t+1} falls below $-b$, set it to $-b$, and if it goes above b, set it to b.

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

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

for all $t' \ge t$.

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.

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.

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.

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

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

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

where $h_{\mathbf{w}}(x)_j$ 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

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

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:

```
initialize parameters $\mathbf{w}$, learning rate $\alpha$, and batch size $b$
converge = False
while not converge:
    epoch + 1
    shuffle training examples
    for i = 0,1,...,$\lceil n\_examples/b \rceil - 1$: # iterate over batches:
        $X_{batch} = X[i \cdot b : (i+1) \cdot b]$ # select the X in the current batch
        $\mathbf{y}_{batch} = \mathbf{y}[i \cdot b : (i+1) \cdot b]$ # select the labels in the current batch
        initialize $\nabla L_{\mathbf{w}}$ to be a matrix of zeros
        for each pair of training data point $(\mathbf{x}, y) \in (X_{batch}, \mathbf{y}_{batch})$:
            for j = 0,1,..., $n\_classes - 1$:
                # calculate the partial derivative of the loss with respect to
                # a single row in the weights matrix
                if $y = j$: $\nabla L_{\mathbf{w}_j}$ += $(softmax(\langle \mathbf{w}_j, \mathbf{x} \rangle) - 1) \cdot \mathbf{x}$
                else: $\nabla L_{\mathbf{w}_j}$ += $(softmax(\langle \mathbf{w}_j, \mathbf{x} \rangle)) \cdot \mathbf{x}$
        $\mathbf{w} = \mathbf{w} - \frac{\alpha \nabla L_{\mathbf{w}}}{len(X_{batch})}$ # update the weights
    if $|Loss(X, \mathbf{y})_{this\_epoch} - Loss(X, \mathbf{y})_{last\_epoch}| <$ CONV_THRESHOLD:
        converge = True # break the loop if loss converged
```

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`.

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?

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

2. Take a look at the Colab notebook `here` 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: To use our data, right click on `hw03` in the `Shared with me` > `hw03` section, and click `Add to My Drive`. Within the hw03 folder, double click the file (`cleaning_data.ipynb`). 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 dropdown again to open the notebook in Google Colab. To run the notebook, you will either need to open in playground mode, or make a copy.

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.

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?

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

To hand in the programming component of this assignment, first ensure that your code runs on *Python 3* using our course `virtualenv`. You can activate the `virtualenv` on a department machine by running the following command in a Terminal:

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

Once the `virtualenv` is activated, run your program and ensure that there are no errors. We will be using this `virtualenv` to grade all programming assignments in this course so we recommend testing your code on a department machine each time before you hand in. Note that handing in code that does not run may result in a significant loss of credit.

Please handin the written assignment, coding assignment and report via Gradescope. If you have questions on how to set up or use Gradescope, ask on Edstem!

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.