Homework 2

Due: Thursday, February 14, 2019 at 12:00pm **(Noon)**

Introduction

Your neighbourhood fine dining establishment has seen more and more empty tables in recent years because customers haven't enjoyed the selections on the wine list. Unfortunately, not one employee working there is a wine connoisseur, so they have asked you to help them choose which wines to stock using Machine Learning.

At the same time, they invited locals in for a wine tasting to learn their tastes. They've collected quite a lot of data, and need you to electronically determine what each person rated a wine out of 10.

In this assignment, you'll implement algorithms that perform linear regression and logistic regression, and use them to predict wine quality and classify handwritten digits, respectively. The book sections relevant to this assignment are 9.2 on page 123, and 9.3 on page 126. You may also find the book section on stochastic gradient descent, 14.3 on page 191, to be helpful, although the mathematical rigour is beyond the scope of this homework assignment. If you are having trouble with the derivations involved for gradient descent, you may find re-watching lecture 4 or reading the slides to be useful.

Stencil Code & Data

You can find the stencil code and datasets for this assignment in the course directory. You can copy the files to your personal directory on a department by machine running the command

```
cp -r /course/cs1420/pub/hw2/* <DEST DIRECTORY>
```

where `<DEST DIRECTORY>` is the directory where you would like to deposit the files. If you are working remotely, you will need to use the `scp` command to copy the files to your computer over `ssh`:

```
scp -r <login>@ssh.cs.brown.edu:/course/cs1420/pub/hw2/* <DEST DIRECTORY>
```

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 `LinearRegression` model and the `LogisticRegression` model which you will be implementing.

You should not need to modify any code in the `main.py`. If you do for debugging or other purposes, please make sure all of your additions are commented out in the final handin. All the functions you need to fill in reside in `models.py`, marked by `TODO`s. You can see a full description of them in the section below. To run the program, run `python main.py` in a terminal with the course environment set up.

Datasets

UCI Wine Quality

For the Linear Regression model, you will be using the UCI Wine Quality Dataset,[1] which contains information about various attributes of a wine and its corresponding quality rating (out of 10). It includes 4898

[1]P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.
Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

examples, which will be split into training and testing datasets using the `sklearn` library. Each example contains 12 attributes, and your model will train on the first eleven attributes (and a 12th bias term) to predict the last value. More information can be found at `https://archive.ics.uci.edu/ml/datasets/Wine+Quality`.

MNIST Dataset

For the Logistic Regression model, you will be using the well-known MNIST Dataset, which includes 60,000 examples for training and 10,000 for testing. All images are size normalized to fit in a 20×20 pixel box and this box is centered in a 28×28 image based on the center of mass of its pixels. This gives a total of 784 features, and we have added a 785th bias term to each example. The full description of this dataset is available at `http://yann.lecun.com/exdb/mnist/`.

The original feature values in the MNIST dataset are integers ranging from 0 to 255 (encoding grey scale values), but we have normalized those features to the range $[0, 1]$. The labels are the same as the original dataset, which are integers in the range $[0, 9]$.

The Assignment

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

- Helper functions:
  - **l2_loss()** calculates the sum squared difference between two arrays.

  In addition, one helper function is provided for you. You should not change it.

  - `softmax()` applies the softmax function to an array and returns the result. What softmax does is normalize input values from the range $[-\infty, \infty]$ into the range $[0, 1]$ so that they add up to 1. You can think of the output of the softmax function as a probability distribution over n classes.
- `LinearRegression`:
  - **train_sgd()** uses stochastic gradient descent to train the weights of the model.
  - **train_solver()** uses matrix inversion to find the optimal set of weights for the given data.
  - **predict()** predicts the values of test data points using the trained weights.

  In addition, three methods are provided for you. You should not change them.

  - `train()` calls either `train_sgd` or `train_solver`, depending on whether the model was instantiated with stochastic gradient descent.
  - `loss()` computes the squared error loss of the predicted labels over a dataset.
  - `average_loss()` computes the average squared error loss per prediction over a dataset.
- `LogisticRegression`:
  - **train()** uses stochastic gradient descent to train the parameters of the model.
  - **predict()** predicts the labels of data points using the trained weights. For each data point, you should apply the sigmoid 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 off-the-shelf packages that have already implemented these models, such as scikit-learn or any linear regression functions. We're asking you to implement them yourself.

Linear Regression

Linear Regression learns linear functions of the inputs:

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

As we are using squared loss, the ERM hypothesis has weights

$$\mathbf{w} = \text{argmin}_{\mathbf{w}} \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2$$

Squared Loss

For this assignment, we will be evaluating and training the Linear Regression model using mean squared loss (or L2 loss). Recall that the L2 loss function is defined as:

$$L_S(h_{\mathbf{w}}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2$$

where y_i is the target value of i^{th} sample and $h_{\mathbf{w}}(\mathbf{x}_i)$ is the predicted value of that sample given the learned model weights. Your model should seek to minimize this loss through either stochastic gradient descent or matrix inversion to learn the ERM hypothesis.

The partial derivatives of this loss function on the prediction $h_{\mathbf{w}}(\mathbf{x})$ of a single data point $(\mathbf{x}, y)$ with respect to the individual weights $[w_1, w_2, \ldots, w_s, \ldots, w_d]$ is

$$\frac{\partial L}{\partial w_s} = 2 \cdot (h_{\mathbf{w}}(\mathbf{x}) - y) \cdot \mathbf{x}_s$$

So the derivative of the loss with respect to the w vector is

$$\frac{dL}{d\mathbf{w}} = 2 \cdot (h_{\mathbf{w}}(\mathbf{x}) - y) \cdot \mathbf{x}$$

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

Stochastic Gradient Descent

You will be using Stochastic Gradient Descent (SGD) as one way to train your `LinearRegression` model. Below, we have provided example pseudocode for SGD on a sample S. Note that you do not have to follow this exactly, and you can use other techniques such as batching and learning rate scheduling.

```
initialize parameters $\mathbf{w}$ and learning rate $\alpha$
repeat until $\mathbf{w}$ converges:
    shuffle training examples
    for each training example $(\mathbf{x}, y) \in S$:
        $\hat{y} = \langle \mathbf{x}, \mathbf{w} \rangle$
        $\nabla L_{\hat{y}} = 2(\hat{y} - y)$
        $\nabla L_{\mathbf{w}} = \langle \mathbf{x}, \nabla L_{\hat{y}} \rangle$
        $\mathbf{w} := \mathbf{w} - \alpha \nabla L_{\mathbf{w}}$
```

Matrix Inversion

We showed in lecture that you can use matrix inversion to compute the set of weights w that minimizes the squared loss. The equation to find $\mathbf{w}$, for a set of data points X and their labels Y is

$$\mathbf{w} = (X^T X)^{-1} X^T Y$$

Implement this in `LinearRegression`, using the `np.linalg.pinv` function to calculate matrix inverses.

Logistic Regression

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

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

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 d classes on a single example (x, y) is the log-loss, sometimes called cross-entropy loss

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

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

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

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 column 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 $\mathbf{w}$ and learning rate $\alpha$
repeat until $\mathbf{w}$ converges:
    shuffle training examples
    for each training example $(\mathbf{x}, y) \in S$:
        $l = \langle \mathbf{x}, \mathbf{w} \rangle$
        $p = \text{softmax}(l)$
        for $j \in$ the set of classes
            if $y = j$: $\nabla L_{p_j} = p_j - 1$
            else: $\nabla L_{p_j} = p_j$
        $\nabla L_{\mathbf{w}} = \langle \mathbf{x}, \nabla L_p \rangle$
        $\mathbf{w} := \mathbf{w} - \alpha \nabla L_{\mathbf{w}}$
```

Project Report

Each programming assignment in this course will be accompanied by a short report in which you will answer a few guiding questions about the results of your algorithm. To answer these questions, you may need to write new code that generates some output (potentially a value or graph) that you will want to include in your writeup. For example, you may be asked to contrast the results of running the same algorithm on different datasets or to explore the effect of changing a certain hyperparameter in your algorithm. By the end of this course you will not only be able to implement common machine-learning algorithms but also develop intuition as to how the results of a given algorithm should be interpreted.

The next 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. We ask that your final report be a PDF file named `report.pdf`, which must be handed in along with your code. 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.

Guiding Questions

- Discuss in a few sentences your choices of hyperparameters. If you deviated from the implementation described by our pseudocode, what did you do differently and why? What did you choose as an indicator of convergence, if any?
- When training the `LinearRegression` model with SGD, graph the training loss at each iteration (weight update) of the algorithm. To save time, you can generate a data point after every 100 iterations (instead of one for each iteration). Produce 3 graphs, each with a different learning rate. Compare their losses over time and to the loss produced by the Matrix Inversion model.
- Compare the different applications of Linear Regression and Logistic Regression. Could you use Linear Regression on the MNIST dataset for digit identification? Could you use Logistic Regression on the wine dataset to predict wine quality? Discuss in a paragraph.

Grading Breakdown

We expect that the `LinearRegression` model should have a training loss of less than 0.55 on the training dataset when trained using matrix inversion, and less than 0.57 when trained using SGD. Because SGD is stochastic, we will evaluate your implementation on the lowest loss among five trials. `LogisticRegression` should reach a training accuracy of above 87%. We will evaluate your implementation on the highest accuracy among five trials.

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:

Linear Regression	50%
Logistic Regression	40%
Report	10%
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.

To hand in the coding portion of the assignment, run `cs142_handin hw2` from the directory containing all of your source code and your report in a file named `report.pdf`.

Anonymous Grading

You need to be graded anonymously, so do not write your name anywhere on your handin. Instead, you should use the course ID that you generated when filling out the collaboration policy form. If you do not have a course ID, you should email the HTAs as soon as possible.

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.