HOMEWORK 4: LOGISTIC REGRESSION

Introduction to Machine Learning
http://www.cs.cmu.edu/~mgormley/courses/10601/
OUT: Friday, February 18th
DUE: Sunday, February 27th
TAs: Sana, Hayden, Prasoon, Tori, Chu

Summary In this assignment, you will build a sentiment polarity analyzer, which will be capable of analyzing the overall sentiment polarity (positive or negative) . In the Written component, you will warm up by deriving stochastic gradient descent updates for logistic regression. Then in the Programming component, you will implement a logistic regression model as the core of your natural language processing system.

START HERE: Instructions

- **Collaboration Policy**: Please read the collaboration policy here: http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html
- **Late Submission Policy:** See the late submission policy here: http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code. Please follow instructions at the end of this PDF to correctly submit all your code to Gradescope.
  - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, please use the provided template. Submissions can be handwritten onto the template, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Each derivation/proof should be completed in the boxes provided. You are responsible for ensuring that your submission contains exactly the same number of pages and the same alignment as our PDF template. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader.
  - **Programming:** You will submit your code for programming questions on the homework to Gradescope (https://gradescope.com). After uploading your code, our grading scripts will autograde your assignment by running your program on a virtual machine (VM). When you are developing, check that the version number of the programming language environment (e.g. Python 3.9.6) and versions of permitted libraries (e.g. `numpy` 1.21.2 and `scipy` 1.7.1) match those used on Gradescope. You have 10 free Gradescope programming submissions. After 10 submissions, you will begin to lose points from your total programming score. We recommend debugging your implementation on your local machine (or the Linux servers) and making sure your code is running correctly first before submitting your code to Gradescope.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

Instructions for Specific Problem Types

For "Select One" questions, please fill in the appropriate bubble completely:

Select One: Who taught this course?

- ● Matt Gormley
- ○ Marie Curie
- ○ Noam Chomsky

If you need to change your answer, you may cross out the previous answer and bubble in the new answer:

Select One: Who taught this course?

- ● Matt Gormley
- ○ Marie Curie
- ⊗ Noam Chomsky

For "Select all that apply" questions, please fill in all appropriate squares completely:

Select all that apply: Which are scientists?

- ■ Stephen Hawking
- ■ Albert Einstein
- ■ Isaac Newton
- □ I don't know

Again, if you need to change your answer, you may cross out the previous answer(s) and bubble in the new answer(s):

Select all that apply: Which are scientists?

- ■ Stephen Hawking
- ■ Albert Einstein
- ■ Isaac Newton
- ☒ I don't know

For questions where you must fill in a blank, please make sure your final answer is fully included in the given space. You may cross out answers or parts of answers, but the final answer must still be within the given space.

Fill in the blank: What is the course number?

10-601	10-~~6~~301

Written Questions (51 points)

1 LaTeX Bonus Point (1 points)

1. (1 point) **Select one:** Did you use LaTeX for the entire written portion of this homework?

   ● Yes

   ○ No

2 Linear Regression (5 points)

1. We would like to fit a linear regression model to the dataset

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, y^{(1)}\right), \left(\mathbf{x}^{(2)}, y^{(2)}\right), \cdots, \left(\mathbf{x}^{(N)}, y^{(N)}\right) \right\}$$

with $\mathbf{x}^{(i)} \in \mathbb{R}^M$ by minimizing the ordinary least square (OLS) objective function:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} \left(y^{(i)} - \sum_{j=1}^{M} w_j x_j^{(i)} \right)^2 .$$

(a) (2 points) **Select one:** Specifically, we solve for each coefficient w_k ($1 \leq k \leq M$) by deriving an expression of w_k from the critical point $\frac{\partial J(\mathbf{w})}{\partial w_k} = 0$. What is the expression for each w_k in terms of the dataset $(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(N)}, y^{(N)})$ and $w_1, \cdots, w_{k-1}, w_{k+1}, \cdots, w_M$?

● $w_k = \frac{\sum_{i=1}^N x_k^{(i)}(y^{(i)} - \sum_{j=1, j \neq k}^M w_j x_j^{(i)})}{\sum_{i=1}^N (x_k^{(i)})^2}$

○ $w_k = \frac{\sum_{i=1}^N x_k^{(i)}(y^{(i)} - \sum_{j=1, j \neq k}^M w_j x_j^{(i)})}{\sum_{i=1}^N (y^{(i)})^2}$

○ $w_k = \frac{\sum_{i=1}^N x_k^{(i)}(y^{(i)} - \sum_{j=1, j \neq k}^M w_j x_j^{(i)})}{\sum_{i=1}^N (x_k^{(i)} y^{(i)})^2}$

○ $w_k = \sum_{i=1}^N x_k^{(i)}(y^{(i)} - \sum_{j=1}^M w_j x_j^{(i)})$

(b) (1 point) **Select one:** How many coefficients (w_k) do you need to estimate? When solving for these coefficients, how many equations do you have?

● M coefficients, M equations

○ M coefficients, N equations

○ N coefficients, M equations

○ N coefficients, N equations

2. (2 points) Consider a dataset D such that we fit a line $y = w_1 x + b_1$. Let $\bar{x}$ and $\bar{y}$ be the mean of the x and y coordinates, respectively. After mean centering the dataset to create $D_{new} = \big((x^{(1)} - \bar{x}, y^{(1)} - \bar{y}), \ldots, (x^{(n)} - \bar{x}, y^{(n)} - \bar{y})\big)$, let the solution to linear regression on D_{new} be $y = w_2 x + b_2$. Explain how w_2 compares to w_1 and justify.

Your Answer

w_2 should be equal to w_1. The D_{new} is just a shifted version of the dataset D. And such shift should only influence the interception term b.

3 Logistic Regression: Warm-Up (7 points)

1. (2 points) **Select all that apply:** Which of the following are true about logistic regression?

   ■ Our formulation of binary logistic regression will work with both continuous and binary features.

   ■ Binary Logistic Regression will form a linear decision boundary in our feature space.

   □ The function $\sigma(x) = \frac{1}{1+e^{-x}}$ is convex.

   ■ The negative log-likelihood function for logistic regression $-\frac{1}{N}\sum_{i=1}^{N}\log(\sigma(\mathbf{x}^{(i)}))$ is not convex so gradient descent may get stuck in a sub-optimal local minimum.

   □ None of the above.

2. (1 point) **Select one:** The negative log-likelihood $J(\boldsymbol{\theta})$ for binary logistic regression can be expressed as

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left[-y^{(i)}\left(\boldsymbol{\theta}^T\mathbf{x}^{(i)}\right) + \log\left(1 + \exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})\right)\right]$$

   where $\mathbf{x}^{(i)} \in \mathbb{R}^{M+1}$ is the column vector of the feature values of the i-th data point, $y^{(i)} \in \{0,1\}$ is the i-th class label, $\boldsymbol{\theta} \in \mathbb{R}^{M+1}$ is the weight vector. When we want to perform logistic ridge regression (i.e. with ℓ_2 regularization), we modify our objective function to be

$$f(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda\frac{1}{2}\sum_{j=0}^{M}\theta_j^2$$

   where λ is the regularization weight, θ_j is the jth element in the weight vector $\boldsymbol{\theta}$. Suppose we are updating θ_k with learning rate α, which of the following is the correct expression for the update?

   ○ $\theta_k \leftarrow \theta_k + \alpha\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k}$ where $\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k} = \frac{1}{N}\sum_{i=1}^{N}\left[x_k^{(i)}\left(y^{(i)} - \frac{\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}{1+\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}\right)\right] + \lambda\theta_k$

   ○ $\theta_k \leftarrow \theta_k + \alpha\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k}$ where $\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k} = \frac{1}{N}\sum_{i=1}^{N}\left[x_k^{(i)}\left(-y^{(i)} + \frac{\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}{1+\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}\right)\right] - \lambda\theta_k$

   ● $\theta_k \leftarrow \theta_k - \alpha\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k}$ where $\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k} = \frac{1}{N}\sum_{i=1}^{N}\left[x_k^{(i)}\left(-y^{(i)} + \frac{\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}{1+\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}\right)\right] + \lambda\theta_k$

   ○ $\theta_k \leftarrow \theta_k - \alpha\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k}$ where $\frac{\partial f(\boldsymbol{\theta})}{\partial\theta_k} = \frac{1}{N}\sum_{i=1}^{N}\left[x_k^{(i)}\left(-y^{(i)} - \frac{\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}{1+\exp(\boldsymbol{\theta}^T\mathbf{x}^{(i)})}\right)\right] + \lambda\theta_k$

3. (2 points) Data is separable in one dimension if there exists a threshold t such that all values less than t have one class label and all values greater than or equal to t have the other class label. If you train an unregularized logistic regression model for infinite iterations on training data that is separable in at least one dimension, the corresponding weight(s) can go to infinity in magnitude. What is an explanation for this phenomenon?
Hint: Think about what happens to the probabilities if we train an unregularized logistic regression model, and the role of the weights when calculating such probabilities.

Your Answer

The infinite iterations means there will be no training error, which is more likely to lead to overfitting. And since the data is unregularized, there will be noise. The weight in such model will be very large in magnitude.

4. (2 points) **Select all that apply:** How does regularization (such as ℓ_1 and ℓ_2) help correct the problem in the previous question?

□ ℓ_1 regularization prevents weights from going to infinity by penalizing the count of non-zero weights.

■ ℓ_1 regularization prevents weights from going to infinity by reducing some of the weights to 0, effectively removing some of the features.

■ ℓ_2 regularization prevents weights from going to infinity by reducing the value of some of the weights to *close* to 0 (reducing the effect of a feature but not necessarily removing it).

□ None of the above.

4 Logistic Regression: Small Dataset (5 points)

The following questions should be completed before you start the programming component of this assignment.

The following dataset consists of 4 training examples, where $x_k^{(i)}$ denotes the k-th dimension of the i-th training example $\mathbf{x}^{(i)}$, and $y^{(i)}$ is the corresponding label ($k \in \{1, 2, 3\}$ and $i \in \{1, 2, 3, 4\}$).

i	x_1	x_2	x_3	y
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0

A binary logistic regression model is trained on this dataset, and the parameter vector $\boldsymbol{\theta}$ after training is

$$\boldsymbol{\theta} = \begin{bmatrix}1.5 & 2 & 1\end{bmatrix}^T.$$

Note: There is **no intercept term** used in this problem.

Use the data above to answer the following questions. For all numerical answers, please use one number rounded to the fourth decimal place; e.g., 0.1234. Showing your work in these questions is optional, but it is recommended to help us understand where any misconceptions may occur.

1. (2 points) Calculate $J(\boldsymbol{\theta})$, $\frac{1}{N}$ times the negative log-likelihood over the given data after iteration n. (Note here we are using natural log, i.e., the base is e).

$J(\boldsymbol{\theta})$

0.7975

Work

$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}(y^{(i)}log(\sigma(\theta^T x^{(i)})) + (1 - y^{(i)})log(1 - \sigma(\theta^T x^{(i)})))$

$\sigma(\theta^T x^{(i)}) = \frac{e^{(\theta^T x^{(i)})}}{1+e^{(\theta^T x^{(i)})}}$

$J(\theta) = -\frac{1}{4}[log(1 - \sigma(1)) + log(\sigma(2)) + log(\sigma(3)) + log(1 - \sigma(1.5))] = 0.7975$

2. (2 points) Calculate the gradients $\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j}$ with respect to θ_j for all $j \in \{1, 2, 3\}$.

$\partial J(\boldsymbol{\theta})/\partial\theta_1$	$\partial J(\boldsymbol{\theta})/\partial\theta_2$	$\partial J(\boldsymbol{\theta})/\partial\theta_3$
0.2044	-0.0417	0.1709

Work

$\bigtriangledown J(\theta) = [\frac{\partial}{\partial\theta_1}J(\theta), \frac{\partial}{\partial\theta_2}J(\theta), \frac{\partial}{\partial\theta_3}J(\theta)]^T$

$\frac{\partial}{\partial\theta_j}J^{(i)}(\theta) = (\sigma(\theta^T x^{(i)}) - y^{(i)})x_j^{(i)}$

$\bigtriangledown J_1(\theta) = [0, 0, 0.7311]^T$

$\bigtriangledown J_2(\theta) = [0, -0.1192, 0]^T$

$\bigtriangledown J_3(\theta) = [0, 0.0474, 0.0474]^T$

$\bigtriangledown J_4(\theta) = [0.8176, 0, 0]^T$

$\bigtriangledown J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \bigtriangledown J_{(i)}(\theta) = [0.2044, -0.0417, 0.1709]^T$

3. (1 point) Update the parameters following the parameter update step $\theta_j \leftarrow \theta_j - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j}$ and write the updated (numerical) value of the vector $\boldsymbol{\theta}$. Use learning rate $\alpha = 1$.

θ_1	θ_2	θ_3
1.2956	2.0417	0.8291

Work

$$\theta = \begin{bmatrix} 1.5 \\ 2 \\ 1 \end{bmatrix} \bigtriangledown J(\theta) = \begin{bmatrix} 0.2044 \\ -0.0417 \\ 0.1709 \end{bmatrix}$$

$$\theta_{new} = \theta - \bigtriangledown J(\theta) = \begin{bmatrix} 1.2956 \\ 2.0417 \\ 0.8291 \end{bmatrix}$$

5 Logistic Regression: Adversarial Attack (7 points)

An image can be represented numerically as a vector of values for each pixel. Image classification tasks then use this vector of pixel values as features to predict an image label.

An automobile company is trying to gather data by asking participants to submit grayscale images of cars. Each pixel has an intensity value in the continuous range $[0, 1]$, zero being the darkest. The company then runs a logistic regression model to predict if the photo actually contains a car. After training the model on a training dataset, the company achieves a mediocre test error. The company wants to improve the model and offers monetary compensation to people who can submit photos that contain a car and make the model predict "false" (i.e., a false negative), as well as photos that do not contain a car and make the model predict "true" (i.e., a false positive). Furthermore, the company releases the parameters of their learned logistic regression model. Let's investigate how to use these parameters to understand the model's weaknesses.

1. (2 points) Given the company's model parameters $\boldsymbol{\theta}$ (i.e., the logistic regression coefficients), gradient ascent can be used to find the vector of pixel values that maximizes the "car" prediction. What is the gradient update rule to do so? Write (1) the objective function, (2) the gradient of the objective function with respect to the feature *values* and (3) the update rule. Use $\mathbf{x}$ as the input vector. *Hint*: You are updating $\mathbf{x}$ to produce an input that confuses the model.

Your Answer

(1) The objective function should be:
Maximize $P(y = 1|\mathbf{x}, \theta) = \sigma(\theta^T\mathbf{x})$

(2) The gradient of the objective function:
$\frac{\partial}{\partial\theta_j}J(\theta) = \sum_{i=1}^{N}(\sigma(\theta^T\mathbf{x}^{(i)})(1 - \sigma(\theta^T\mathbf{x}^{(i)}))\theta_i)$

(3) Update rule:
$\mathbf{x}_{new} = \mathbf{x} + \alpha\frac{\partial}{\partial\theta_j}J(\theta)$

2. (1 point) Modify the procedure in the previous question to find the image that minimizes the "car" prediction.

Your Answer

(1) The objective function should be:
Minimize $P(y = 0|\mathbf{x}, \theta) = 1 - \sigma(\theta^T\mathbf{x})$

(2) The gradient of the objective function:
$\frac{\partial}{\partial\theta_j}J(\theta) = -\sum_{i=1}^{N}(\sigma(\theta^T\mathbf{x}^{(i)})(1 - \sigma(\theta^T\mathbf{x}^{(i)}))\theta_i)$

(3) Update rule:
$\mathbf{x}_{new} = \mathbf{x} + \alpha\frac{\partial}{\partial\theta_j}J(\theta)$

3. (2 points) To generate an image, we require the feature values to be in the range $[0, 1]$. Propose a different procedure that optimizes the "car" prediction subject to this constraint and does not require a gradient calculation. What is the runtime of this procedure?

Your Answer

Suppose x has n dimensions. Each position can take two values, i.e. 0 or 1. A full combination of those n dimensions will generate 2^n different vectors. And the running time will be $O(2^n)$.

4. (2 points) **Select all that apply:** Now let's consider whether logistic regression is well-suited for this task. Suppose the exact same white car in a dark background was used to generate the training set. The training photos were captured with the side view of the car centered in the photo at a distance of between 30-50 meters from the camera. Which (if any) of the below descriptions of a **test image** would the model predict as "car"?

■ A new photo with the same car centered and 60 meters away from the camera.

□ A new photo with the same car in the upper right corner of the image.

■ Identical to one of the training photos, but the car replaced with an equal size white cardboard cutout of the car.

□ Identical to one of the training photos, but the background changed to white.

□ None of the above.

6 Vectorization and Pseudocode (10 points)

The following questions should be completed before you start the programming component of this assignment. Assume the `dtypes` of all `ndarrays` are `np.float64`. Vectors are 1D `ndarrays`.

1. (2 points) **Select all that apply:** Consider a matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ and vector $\mathbf{v} \in \mathbb{R}^{M}$. We can create a new vector $\mathbf{u} \in \mathbb{R}^{N}$ whose i-th element is the dot product between $\mathbf{v}$ and the i-th row of $\mathbf{X}$ using NumPy as follows:

```
# X and v are numpy ndarrays
# X.shape == (N, M), v.shape == (M,)
u = np.zeros(X.shape[0])
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        u[i] += X[i, j] * v[j]
```

Which of the following produces the same result?

■ `u = np.dot(X, v)`

□ `u = np.dot(v, X)`

■ `u = np.matmul(X, v)`

□ `u = np.matmul(v, X)`

□ `u = X * v`

□ `u = v * X`

■ `u = X @ v`

□ `u = v @ X`

□ None of the above.

2. Consider a matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ and vector $\mathbf{w} \in \mathbb{R}^{N}$. Let $\mathbf{\Omega} = \sum_{i=0}^{N-1} w_i \left(\mathbf{x}_i - \overline{\mathbf{x}_i}\right) \left(\mathbf{x}_i - \overline{\mathbf{x}_i}\right)^T$ where $\mathbf{x}_i \in \mathbb{R}^M$ is the *column* vector denoting the i-th *row* of $\mathbf{x}$, $\overline{\mathbf{x}_i} \in \mathbb{R}$ is the mean of $\mathbf{x}_i$, and $w_i \in \mathbb{R}$ is the i-th element of $\mathbf{w}$ ($i \in \{0, 1, \cdots, N-1\}$). For the following questions, use `X` and `w` for $\mathbf{X}$ and $\mathbf{w}_i$, respectively. `X.shape == (N, M)`, `w.shape == (N,)`. **You must use NumPy and vectorize your code for full credit.** Do *not* use functions which are essentially wrappers for Python loops and provide little performance gain, such as `np.vectorize`.

   (a) (2 points) Write *one* line of valid Python code that constructs a matrix whose i-th row is $\left(\mathbf{x}_i - \overline{\mathbf{x}_i}\right)^T$.

   Your Answer (CASE SENSITIVE)

```
M = (X - X.mean(axis=1, keepdims=True)).T
```

   (b) (2 points) Assume the result from (a) is stored in `M`. Write *one* line of valid Python code that computes $\mathbf{\Omega}$ from `M`.

   Your Answer (CASE SENSITIVE)

```
omga = w * (np.dot(M.T, M))
```

3. Now we will compare two different optimization methods using pseudocode. Consider a model with parameter $\boldsymbol{\theta} \in \mathbb{R}^M$ being trained with a design matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ and labels $\mathbf{y} \in \mathbb{R}^M$. Say we update $\boldsymbol{\theta}$ using the objective function $J(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N} J^{(i)}(\boldsymbol{\theta}|\mathbf{x}^{(i)}, y^{(i)}) \in \mathbb{R}$. Recall that an epoch refers to one complete cycle through the dataset.

   (a) (2 points) Complete the pseudocode for gradient descent.

```
def dJ(theta, X, y, i):
    (omitted)  # Returns $\partial J^{(i)}(\boldsymbol{\theta}|\mathbf{x}^{(i)}, y^{(i)})/\partial\boldsymbol{\theta}$
    # You may call this function in your pseudocode.

def GD(theta, X, y, learning_rate):
    for epoch in range(num_epoch):
        ___Complete this section with the update rule___
    return theta  # return the updated theta
```

Your Answer (CASE SENSITIVE, 7 lines max)

```
for i in range(len(y)):
    summ += dJ(theta, X, y, i)
theta_new = theta - learning_rate * summ / len(y)
```

   (b) (2 points) Complete the pseudocode for stochastic gradient descent that samples *without* replacement.

```
def dJ(theta, i):
    (omitted)  # Returns $\partial J^{(i)}(\boldsymbol{\theta}|\mathbf{x}^{(i)}, y^{(i)})/\partial\boldsymbol{\theta}$
    # You may call this function in your pseudocode.

def SGD(theta, X, y, learning_rate):
    for epoch in range(num_epoch):
        indices = shuffle(range(len(X)))
        for i in indices:
            ___Complete this section with the update rule___
    return theta  # return the updated theta
```

Your Answer (CASE SENSITIVE, 7 lines max)

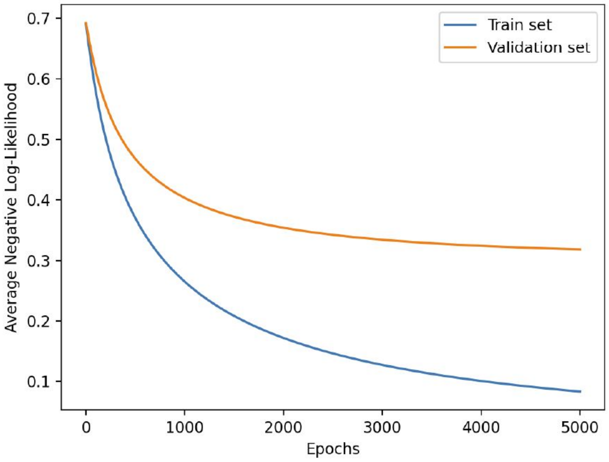
```
theta_new = theta - learning_rate * dJ(theta, i) / len(y)
```

7 Programming Empirical Questions (16 points)

The following questions should be completed as you work through the programming component of this assignment. **Please ensure that all plots are computer-generated**.

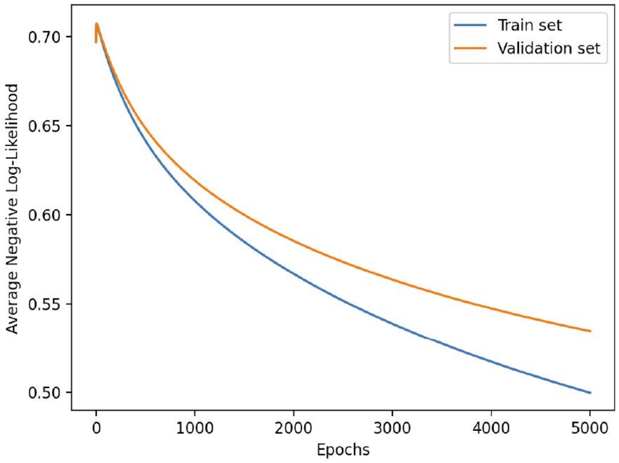
1. (2 points) For *Model 1*, using the data in the `largedata` folder in the handout, make a plot that shows the *average* negative log-likelihood for the training and validation data sets after each of 5,000 epochs. The y-axis should show the negative log-likelihood and the x-axis should show the number of epochs. (Note that running the code for 5,000 epochs might take longer than one minute. This is okay since we won't run your code for more than 500 epochs during auto-grading.)

Your Answer



2. (2 points) For *Model 2*, make a plot as in the previous question.

Your Answer



3. (2 points) Write a few sentences explaining the output of the above experiments. In particular, do the training and validation log-likelihood curves look the same, or different? Why?

Your Answer

Both lines have the same decaying tendency. Model 1 converges faster without the increasing tendency at first. One of the reasons should be that model1 has a fixed matrix with value 0 or 1.

4. (2 points) Make a table with your train and test error for the large data set (found in the `largedata` folder in the handout) for each of the two models after running for 5,000 epochs. Please use one number rounded to the fourth decimal place, e.g., 0.1234.

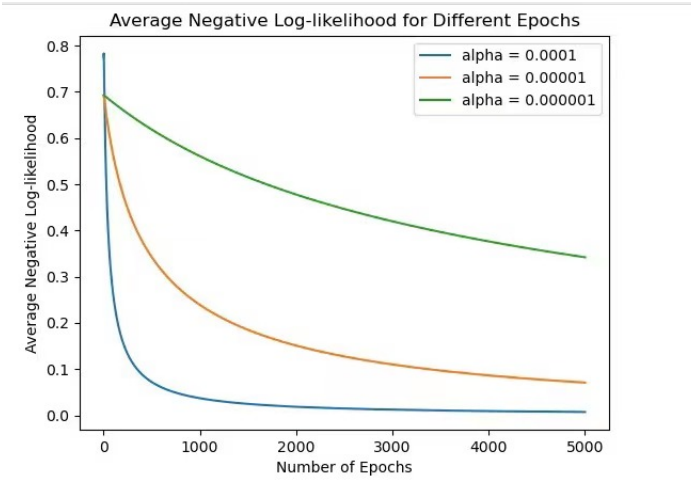
Error Rates

	Train Error	Test Error
Model 1	0.0000	0.1400
Model 2	0.2083	0.2400

Table 1: "Large Data" Results

5. (2 points) For *Model 1*, using the data in the `largedata` folder of the handout, make a plot comparing the training average negative log-likelihood over epochs for three different values for the learning rates, $\alpha \in \{10^{-4}, 10^{-5}, 10^{-6}\}$. The y-axis should show the negative log-likelihood, the x-axis should show the number of epochs (from 1 to 5,000 epochs), and the plot should contain three curves corresponding to the three values of α. Provide a legend that indicates the learning rate α for each curve.

Your Answer



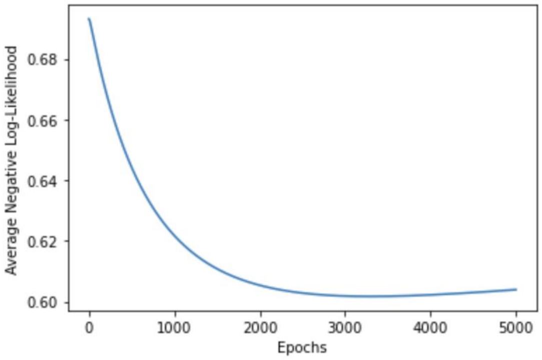
6. (2 points) Compare how quickly each curve in the previous question converges.

Your Answer

the smaller the learning rate is, the faster it converges.

7. (2 points) Now we will compare the effectiveness of bag-of-words and word2vec. Consider *Model 3*, which is *Model 1* with the size of the dictionary reduced to 300 to match *Model 2*'s embedding. We provided you the *validation* average negative log-likelihood over 5,000 epochs in `model3_val_nll.txt`. Using this, make a plot that compares the validation average negative log-likelihood of all three models over 5,000 epochs. The y-axis should show the negative log-likelihood and the x-axis should show the number of epochs.

Your Answer



8. (2 points) Compare and contrast the performance of the three models based on the curves in the previous question. Recall that a better model is one that attains lower negative log-likelihood faster. Explain the relative difference in performance focusing on the dimensions and design of the input data.

Your Answer

Model 3 is the best model, since it converges fastest. Model 2 is the slowest. One of the reasons is that not only model 3 is a binary matrix,compared to model 2, its matrix has fewer dimensions,compared to model 1, which means less uncertainty, and makes its log-likelihood more easier to converge.

8 Collaboration Questions

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found here.

1. Did you receive any help whatsoever from anyone in solving this assignment? If so, include full details.
2. Did you give any help whatsoever to anyone in solving this assignment? If so, include full details.
3. Did you find or come across code that implements any part of this assignment? If so, include full details.

Your Answer

1. No
2. No
3. No

9 Programming (70 points)

Your goal in this assignment is to implement a working Natural Language Processing (NLP) system using binary logistic regression. Your algorithm will determine whether a movie review is positive or negative. You will also explore various approaches to feature engineering for this task.

Note: Before starting the programming, you should work through the written component to get a good understanding of important concepts that are useful for this programming component.

9.1 The Task

Datasets Download the zip file from the course website, which contains the data for this assignment. This data comes from the Movie Review Polarity dataset.[1] In the data files, each line is a single training example that consists of a label (0 for negative reviews and 1 for positive ones) and a set of words. The format of each training example (each line) is `label\tword1 word2 word3 ... wordN\n`, where words are separated from each other with white-space and the label is separated from the words with a tab character.[2]

Examples of the data are as follows:

```
1 david spade has a snide , sarcastic sense of humor that works ...
0 " mission to mars " is one of those annoying movies where , in...
1 anyone who saw alan rickman's finely-realized performances in ...
1 ingredients : man with amnesia who wakes up wanted for murder ...
1 ingredients : lost parrot trying to get home , friends synopsi...
1 note : some may consider portions of the following text to be ...
0 aspiring broadway composer robert ( aaron williams ) secretly ...
0 america's favorite homicidal plaything takes a wicked wife in ...
```

Feature Engineering In lecture, we saw that we can apply logistic regression to real-valued inputs of fixed length (e.g. $\mathbf{x}^{(i)} \in \mathbb{R}^n$). However, each training example (movie review) above has variable length and are not real-valued, so we cannot directly run logistic regression on the dataset above.

To be able to run logistic regression on the dataset, we first need to transform it using some basic feature engineering techniques. In this homework, we will use two common techniques: a bag-of-words (BoW) model and a word embeddings model. These feature engineering models are described in full detail in the next section (9.2).

Programs At a high level, you will write two programs for this homework: `feature.py` and `lr.py`. `feature.py` takes in the raw input data and produces a real-valued vector for each training, validation, and test example. `lr.py` then takes in these vectors and trains a logistic regression model to predict whether each example is a positive or negative review.

9.2 Feature Models

In order to transform a set of words into vectors, we rely on two popular methods of feature engineering: bag-of-words and word embeddings. In this homework, you will have the opportunity to implement both and reason about each method's strengths and weaknesses.

In the subsections below, we use ϕ to denote a feature engineering method (bag-of-words or word embeddings) and $\mathbf{x}^{(i)}$ to denote a training example (a set of English words as seen in 9.1).

[1] For more details, see http://www.cs.cornell.edu/people/pabo/movie-review-data/

[2] The data files are in tab-separated-value (`.tsv`) format. This is identical to a comma-separated-value (`.csv`) format except that instead of separating columns with commas, we separate them with a tab character, `\t`

9.2.1 Model 1: Bag-of-Words

A *bag-of-words* feature vector $\phi_1\left(\mathbf{x}^{(i)}\right) = \mathbf{1_{occur}}(\mathbf{x^{(i)}}, \mathbf{Vocab})$ indicates which words in vocabulary $\mathbf{Vocab}$ occur at least once in the i-th movie review $\mathbf{x}^{(i)}$. Specifically, the bag-of-words method ϕ_1 produces an indicator vector of length $|\mathbf{Vocab}|$, where the j-th entry will be set to 1 if the j-th word in $\mathbf{Vocab}$ occurs at least once in the movie review. The j-th entry will be set to 0 otherwise.

Vocabulary We provide a dictionary file (`dict.txt`) that contains the vocabulary (the set of words we recognize) and the order of the words. This dictionary is constructed from the training data, so it includes all the words from the training data, but some words in the validation and test data may not be present in the dictionary. Each line in the dictionary file is in the following format: `word index\n`. Words and indexes are separated with *whitespace*. Examples of the dictionary content are as follows:

```
films 0
adapted 1
from 2
comic 3
```

As an example, if a movie review $\mathbf{x}^{(i)}$ contained only the word `films`, then $\phi_1\left(\mathbf{x}^{(i)}\right)$ would be a one-hot vector where the first entry is a 1 and the rest are 0.

9.2.2 Model 2: Word Embeddings

Rather than simply indicating which words are present, word embeddings represent each word by "embedding" it into a low-dimensional vector space, which may carry more information about the semantic meaning of the word. In this homework, we use the *word2vec* embeddings, a commonly used set of feature vectors. [3]

Embeddings `word2vec.txt` contains the *word2vec* embeddings of `15k` words. Note that not every word in the movie review examples will be included in the provided `word2vec.txt` file. Each line consists of a word and its embedding separated by tabs: `word\tfeature1\tfeature2\t...feature300\n`. Each word's embedding is always a 300-dimensional vector. As an example, here are the first few lines of `word2vec.txt`:

```
films    -0.598   -0.622   -0.637    4.742    4.323   -5.980   ...
adapted   0.175   -0.399   -2.337   -0.299   -4.781   -2.029   ...
from     -0.114   -2.072   -0.874   -0.483    0.354   -2.205   ...
comic    -0.119   -1.952   -0.226   -0.825    5.625   -0.266   ...
```

Words in `word2vec.txt` are listed in the same order as in the dictionary `dict.txt`.

Using Word Embeddings For this model, there will be two steps in the feature engineering process:

1. First, we would like to exclude words from the movie review that are not included in the word2vec dictionary. Let $\mathbf{x}\_\text{trim}^{(i)} = \text{TRIM}(\mathbf{x}^{(i)})$, where $\text{TRIM}(\mathbf{x}^{(i)})$ trims the list of words $\mathbf{x}^{(i)}$ by only including words of $\mathbf{x}^{(i)}$ present in `word2vec.txt`.

2. Second, we want to take the trimmed vector $\mathbf{x}\_\text{trim}^{(i)}$ and convert it to the final feature vector by averaging the word2vec embeddings of its words:

$$\phi_2\left(\mathbf{x}^{(i)}\right) = \frac{1}{J}\sum_{j=1}^{J} word2vec(\mathbf{x}\_\text{trim}_j^{(i)})$$

[3] For more details on how these embeddings were trained, see the original paper at https://arxiv.org/pdf/1301.3781.pdf

where J denotes the number of words in $\mathbf{x}\text{\_trim}^{(i)}$ and $\mathbf{x}\text{\_trim}_j^{(i)}$ is the j-th word in $\mathbf{x}\text{\_trim}^{(i)}$.

In the given equation, $word2vec(\mathbf{x}\text{\_trim}_j^{(i)}) \in \mathbb{R}^{300}$ is the *word2vec* feature vector for the word $\mathbf{x}\text{\_trim}_j^{(i)}$.

The following **example** provides a reference for Model 2:

- Let $\mathbf{x}^{(i)}$ denote the sentence "`a hot dog is not a sandwich because it is not square`".
- A toy *word2vec* dictionary is given as follows:

```
hot          0.1    0.2    0.3
not         -0.1    0.2   -0.3
sandwich     0.0   -0.2    0.4
square       0.2   -0.1    0.5
```

- Then, $\mathbf{x}\text{\_trim}^{(i)}$ denotes the trimmed review "`hot not sandwich not square`". In this trimmed text, the words that are not in the *word2vec* dictionary are excluded. Also note that we keep the order of words and do not de-duplicate words in the trimmed text. [4]
- The feature for $\mathbf{x}^{(i)}$ can be calculated as

$$\begin{aligned}\phi_2(\mathbf{x}^{(i)}) &= \frac{1}{5}\big(word2vec(\text{hot}) + 2\cdot word2vec(\text{not}) + word2vec(\text{sandwich}) + word2vec(\text{square})\big)\\ &= \begin{bmatrix}0.02 & 0.06 & 0.12\end{bmatrix}^T.\end{aligned}$$

The motivation of this model is that pre-trained feature representations such as word2vec embeddings may provide richer information about semantics of the sentence. You will observe whether using pre-trained word embeddings to build feature vectors will improve or degrade accuracy over the bag-of-words features.

9.3 `feature.py`

`feature.py` implements bag-of-words and word embeddings (described above in 9.2) to transform raw training examples (a label and a list of English words) to formatted training examples (a label and a feature vector, which may be created either through bag-of-words or through word embeddings).

Inputs

- **Input data** for training, validation, and testing. Each data point contains a label and an English movie review in the format described in 9.1.
- **Dictionary and word2vec embeddings** to use for the bag-of-words and word embedding feature extraction methods, respectively.
- **A feature flag** that indicates whether to use Model 1 (bag-of-words) or Model 2 (word2vec).

Outputs

- **Formatted data** for training, validation, and testing. You should perform feature extraction on *each* of the training, validation, and test sets. Each data point contains a label and a feature vector, which is either the length of the vocabulary (when using bag-of-words) or length 300 (when using word2vec).

[4] Keeping duplicates is equivalent to weighting words by their frequency. If `"good"` appears 3 times as often as `"bad"`, the movie review is more likely to be positive than negative.

Output Format Each output file (one for training data, one for validation, and one for testing) should contain the formatted presentation of each example printed on a new line. Use `\n` to create a new line. The format for each line should exactly match `label\tvalue1\tvalue2\tvalue3\t...valueM\n`.

Each line corresponds to a particular movie review, where the first entry is the label and the rest are the features in the feature vector. If bag-of-words is used, each feature is 1 or 0 depending on whether the corresponding dictionary word is present in the review. If word embeddings are used, the rows are the summed up word2vec vectors for all the words present in the dictionary. All entries are separated with a tab character. The handout folder contains example formatted outputs; they are partially reproduced below for your reference. For Model 2, please round your outputs to 6 decimal places.

For Model 1 (bag-of-words):

```
1       0       0       1       0       0       1       0       ...
0       0       0       1       0       0       1       1       ...
1       0       0       0       0       1       0       1       ...
1       1       0       0       0       0       1       1       ...
```

For Model 2 (word embeddings):

```
1.000000        -0.213174        0.135342       -0.254539       ...
1.000000        -0.202170        0.294211       -0.374678       ...
1.000000        -0.222277        0.299419       -0.524649       ...
1.000000        -0.229423       -0.219987       -0.374537       ...
```

9.4 `lr.py`

`lr.py` implements a logistic regression classifier that takes in formatted training data and produces a label (either 0 or 1) that corresponds to whether each movie review was negative or positive. See the Logistic Regression Review section (9.6) for the stochastic gradient descent loss function (and for more details on how to train the classifier).

Inputs

- **Formatted data** for training, validation, and testing. Each data point contains a label and a corresponding feature vector.
- **Dictionary and word2vec embeddings** to use for the bag-of-words and word embedding feature extraction methods, respectively.
- **The number of epochs** to train for, which will be passed in as a command line argument.

Note that we do *not* need to indicate whether the input feature vectors are bag-of-words or word2vec, since in either case we have a set of real-valued vectors to perform logistic regression on.

Requirements

- Include an intercept term in your model. You can either treat the intercept term as a separate variable, or fold it into the parameter vector. In either case, make sure you update the intercept parameter correctly.
- Initialize all model parameters to 0.
- Use stochastic gradient descent (SGD) to train the logistic regression model. Details on the loss function and gradient update rule are provided in are provided in the Logistic Regression Review (9.6) section.
- Perform SGD updates on the training data **in the order that the data is given in the input file**. While we would normally shuffle training examples in SGD, we need training to be deterministic in order to autograde this assignment (otherwise, we would not know whether your answer differs from ours because it is incorrect or because your model saw the data in a different order). **Do not shuffle the training data.**

Outputs

- **Labels** for the training and testing data.
- **Metrics** for the training and testing error.

Output Labels Format Your `lr` program should produce two output `.labels` files containing the predictions of your model on training data and test data. Each file should contain the predicted labels for each example printed on a new line. The name of these files will be passed as command line arguments. Use `\n` to create a new line. An example of the labels is given below.

```
0
0
1
0
```

Output Metrics Format Your program should generate a `.txt` file where you report the final training and testing error after training has completed. The name of this file will be passed as a command line argument.

All of your reported numbers should be within 0.00001 of the reference solution, and round the error values to 6 decimal places. The following example is the reference solution for the large dataset with Model 1 after 500 training epochs. See `model1_metrics_out.txt` in the handout.

```
error(train): 0.042500
error(test): 0.150000
```

Each line in the output file should be terminated by a newline character `\n`. There is a whitespace character after the colon.

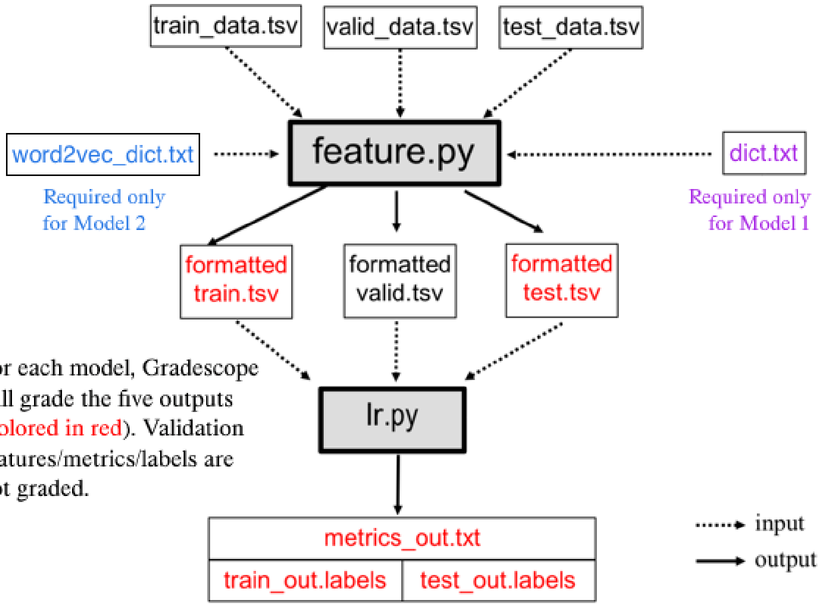


Figure 1: Programming pipeline for sentiment analyzer based on binary logistic regression

9.5 Command Line Arguments

The autograder runs and evaluates the output from the files generated, using the following command (note `feature` will be run before `lr`):

```
$ python feature.py [args1...]
$ python lr.py [args2...]
```

Where above `[args1...]` is a placeholder for nine command-line arguments: `<train_input>` `<validation_input>` `<test_input>` `<dict_input>` `<feature_dictionary_input>` `<formatted_train_out>` `<formatted_validation_out>` `<formatted_test_out>` `<feature_flag>`. These arguments are described in detail below:

1. `<train_input>`: path to the training input `.tsv` file (see Section 9.1)
2. `<validation_input>`: path to the validation input `.tsv` file (see Section 9.1)
3. `<test_input>`: path to the test input `.tsv` file (see Section 9.1)
4. `<dict_input>`: path to the dictionary input `.txt` file (see Section 9.1)
5. `<feature_dictionary_input>`: path to the word2vec feature dictionary `.tsv` file (see Section 9.2)
6. `<formatted_train_out>`: path to output `.tsv` file to which the feature extractions on the *training* data should be written (see Section 9.3)
7. `<formatted_validation_out>`: path to output `.tsv` file to which the feature extractions on the *validation* data should be written (see Section 9.3)
8. `<formatted_test_out>`: path to output `.tsv` file to which the feature extractions on the *test* data should be written (see Section 9.3)

9. `<feature_flag>`: integer taking value 1 or 2 that specifies whether to construct the Model 1 feature set or the Model 2 feature set (see Section 9.2)—that is, if `feature_flag == 1` use Model 1 features; if `feature_flag == 2` use Model 2 features

Likewise, `[args2...]` is a placeholder for eight command-line arguments: `<formatted_train_input>` `<formatted_validation_input>` `<formatted_test_input>` `<train_out>` `<test_out>` `<metrics_out>` `<num_epoch>` `<learning_rate>`. These arguments are described in detail below:

1. `<formatted_train_input>`: path to the formatted training input `.tsv` file (see Section 9.3)
2. `<formatted_validation_input>`: path to the formatted validation input `.tsv` file (see Section 9.3)
3. `<formatted_test_input>`: path to the formatted test input `.tsv` file (see Section 9.3)
4. `<train_out>`: path to output `.labels` file to which the prediction on the *training* data should be written (see Section 9.4)
5. `<test_out>`: path to output `.labels` file to which the prediction on the *test* data should be written (see Section 9.4)
6. `<metrics_out>`: path of the output `.txt` file to which metrics such as train and test error should be written (see Section 9.4)
7. `<num_epoch>`: integer specifying the number of times SGD loops through all of the training data (e.g., if `<num_epoch>` equals 5, then each training example will be used in SGD 5 times).
8. `<learning_rate>`: float specifying the learning rate; 0.00001 for the **large** dataset, and 0.00003 for the **small** dataset

As an example, the following two command lines would run your programs on the large dataset in the handout for 500 epochs using the features from Model 1.

```
$ python feature.py largedata/train_data.tsv largedata/valid_data.tsv
largedata/test_data.tsv dict.txt word2vec.txt
largeoutput/formatted_train.tsv largeoutput/formatted_valid.tsv
largeoutput/formatted_test.tsv 1

$ python lr.py largeoutput/formatted_train.tsv
largeoutput/formatted_valid.tsv largeoutput/formatted_test.tsv
largeoutput/train_out.labels largeoutput/test_out.labels
largeoutput/metrics_out.txt 500 0.00001
```

Note that **the learning rate is different for the large dataset and small dataset** as specified above.

Important Note: You will not be writing out the predictions on validation data, only on train and test data. The validation data is *only* used to give you an estimate of held-out negative log-likelihood at the end of each epoch during training. You are asked to graph the negative log-likelihood vs. epoch of the validation and training data in Programming Empirical Questions section. [a]

[a] For this assignment, we will always specify the number of epochs. However, a more mature implementation would monitor the performance on validation data at the end of each epoch and stop SGD when this validation log-likelihood appears to have converged. You should ***not*** implement such a convergence check for this assignment.

9.6 Logistic Regression Review

Assume you are given a dataset with N training examples and M features. We first write down the $\frac{1}{N}$ times the *negative* conditional log-likelihood of the training data in terms of the design matrix $\mathbf{X}$, the labels $\mathbf{y}$, and the parameter vector $\boldsymbol{\theta}$. This will be your objective function $J(\boldsymbol{\theta})$ for gradient descent. (Recall that i-th row of the design matrix $\mathbf{X}$ contains the features $\mathbf{x}^{(i)}$ of the i-th training example. The i-th entry in the vector $\mathbf{y}$ is the label $y^{(i)}$ of the i-th training example. Here we assume that each feature vector $\mathbf{x}^{(i)}$ contains an intercept *feature*, e.g. $x_0^{(i)} = 1 \; \forall i \in \{1, \ldots, N\}$. As such, **the intercept parameter is folded into our parameter vector $\boldsymbol{\theta}$.**

Taking $\mathbf{x}^{(i)}$ to be a $(M+1)$-dimensional vector where $x_0^{(i)} = 1$, the likelihood $p\left(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}\right)$ is:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta}) = \prod_{i=1}^{N} \left(\frac{e^{\boldsymbol{\theta}^T \mathbf{x}^{(i)}}}{1 + e^{\boldsymbol{\theta}^T \mathbf{x}^{(i)}}} \right)^{y^{(i)}} \left(\frac{1}{1 + e^{\boldsymbol{\theta}^T \mathbf{x}^{(i)}}} \right)^{\left(1 - y^{(i)}\right)} \tag{1}$$

$$= \prod_{i=1}^{N} \frac{\left(e^{\boldsymbol{\theta}^T \mathbf{x}^{(i)}} \right)^{y^{(i)}}}{1 + e^{\boldsymbol{\theta}^T \mathbf{x}^{(i)}}} \tag{2}$$

Hence, $J(\boldsymbol{\theta})$, that is $\frac{1}{N}$ times the negative conditional log-likelihood, is:

$$J(\boldsymbol{\theta}) = -\frac{1}{N} \log p\left(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}\right) = \frac{1}{N} \sum_{i=1}^{N} \underbrace{\left[-y^{(i)} \left(\boldsymbol{\theta}^T \mathbf{x}^{(i)} \right) + \log \left(1 + e^{\boldsymbol{\theta}^T \mathbf{x}^{(i)}} \right) \right]}_{J^{(i)}(\boldsymbol{\theta})} \tag{3}$$

The partial derivative of $J(\boldsymbol{\theta})$ with respect to θ_j , $j \in \{0, ..., M\}$ is:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{N} \sum_{i=1}^{N} \underbrace{\left[-x_j^{(i)} \left(y^{(i)} - \frac{e^{\boldsymbol{\theta}^T \mathbf{x}^{(i)}}}{1 + e^{\boldsymbol{\theta}^T \mathbf{x}^{(i)}}} \right) \right]}_{\frac{\partial J^{(i)}(\boldsymbol{\theta})}{\partial \theta_j}} \tag{4}$$

The gradient descent update rule for binary logistic regression for parameter element θ_j is:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} \tag{5}$$

Then, the stochastic gradient descent update for parameter element θ_j using the i-th datapoint $(\mathbf{x}^{(i)}, y^{(i)})$ is:

$$\theta_j \leftarrow \theta_j + \alpha x_j^{(i)} \left[y^{(i)} - \frac{e^{\boldsymbol{\theta}^T \mathbf{x}^{(i)}}}{1 + e^{\boldsymbol{\theta}^T \mathbf{x}^{(i)}}} \right] \tag{6}$$

9.7 Starter Code

To help you start this assignment, we have provided starter code in the handout.

9.8 Gradescope Submission

You should submit your `feature.py` and `lr.py` to Gradescope. *Note*: please do not zip them or use other file names. This will cause problems for the autograder to correctly detect and run your code. Gradescope will also provide **hints for common bugs**; Ctrl-F for HINT if you did not receive a full score.