HOMEWORK 4: LOGISTIC REGRESSION

10-301/10-601 Introduction to Machine Learning (Fall 2022)
http://www.cs.cmu.edu/~mgormley/courses/10601/
OUT: Tuesday, Oct 4
DUE: Thursday, Oct 13 at 11:59 PM
TAs: Abhi, Jack, Jeneel, Lulu, Pranay

Summary In this assignment, you will build a sentiment polarity analyzer, which will be capable of analyzing the overall sentiment polarity (positive or negative) for movie reviews using logistic regression. In the written component, you will study linear and logistic regression.

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 3.9.12) and versions of permitted libraries (e.g. `numpy` 1.23.0) 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 and reference output 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?

- ● Henry Chai
- ○ 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 (37 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 (4 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:** 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. (1 point) 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.

   ◯ $w_2 > w_1$. Mean centering lowers the value of the bias term, which increases the slope of the regression line.

   ◯ $w_2 < w_1$. Mean centering decreases the variance of the data, so the regression line will not be as steep.

   ● $w_2 = w_1$. Mean centering data shifts the data and does not scale the coordinates; therefore, it does not change the fitted regression line's slope.

   ◯ Not enough information to decide. Mean centering can either increase or decrease our label values, so it may make our regression line either more or less steep.

3 Logistic Regression: Warm-Up (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 and parameter $\boldsymbol{\theta}$. (Note here we are using natural log, i.e., the base is e).

$J(\boldsymbol{\theta})$

0.7975

Work

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

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

θ_1	θ_2	θ_3
1.2956	2.0417	0.8291

Work

4 Logistic Regression: Analysis (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, assuming no feature engineering.
   - □ The sigmoid function is convex.
   - □ The negative log-likelihood function for logistic regression 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 *average* 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 η, which of the following is the correct expression for the update?

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

We assume that for From the plot of logistic function, we can see the plot goes to nearly plateaus when $z \geq 6$and $z \geq -6$.
This means that without regularization, it is possible that we achieve the plateau after several iterations of updating the weights, however, if we keep update to infinite iterations, the weights will become larger and larger, and therefore going to infinity 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.

5 Logistic Regression: Adversarial Attack (5 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.

A marine conservation group gathers data by asking participants to submit grayscale images of narwhals. Each pixel has an intensity value in the continuous range $[0, 1]$, zero being the darkest. The organization then runs a logistic regression model to predict if the photo actually contains a narwhal (mathematically, the "narwhal" prediction would correspond to the model predicting $\mathbf{y}^{(i)} = 1$). After training the model on a training dataset, the company achieves a mediocre test error. The organization wants to improve the model and offers monetary compensation to people who can submit photos that do not contain a narwhal but make the model predict "narwhal" (i.e., a false positive). In addition, 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. Specifically, we will use gradient descent to accomplish this.

1. (2 points) Given the company's model parameters $\boldsymbol{\theta}$ (i.e., the logistic regression coefficients), for a single image, you define the probability function for whether logistic regression predicts a narwhal as

$$p(y = 1|\mathbf{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^T\mathbf{x})$$

You wish to optimize $\mathbf{x}$ such that the model generates a "narwhal" prediction. Given this setup, Write (1) the gradient of the probability function with respect to the *feature* values and (2) the *gradient ascent* update rule. Use $\mathbf{x}$ as the input vector. *Hint*: You are updating $\mathbf{x}$ to produce an input that confuses the model.

Gradient of the Objective Function

$\nabla_{\mathbf{x}} J(\mathbf{x}) = \sigma(\boldsymbol{\theta}^T\mathbf{x})(1 - \sigma(\boldsymbol{\theta}^T\mathbf{x}))\boldsymbol{\theta} = \frac{\exp(\boldsymbol{\theta}^T\mathbf{x})}{(1+\exp(\boldsymbol{\theta}^T\mathbf{x}))^2}\boldsymbol{\theta}$

Update Rule

$\mathbf{x} \leftarrow \mathbf{x} + \eta\frac{\exp(\boldsymbol{\theta}^T\mathbf{x})}{(1+\exp(\boldsymbol{\theta}^T\mathbf{x}))^2}\boldsymbol{\theta}$, where η is the learning rate

2. (1 point) We require the feature values to be in the range $[0, 1]$ to generate an image. Using the setup outlined in 1, propose a different procedure subject to this constraint that does not require a gradient calculation.

Your Answer

We could instead use probabilistic approach to find the optimal features:
1. To find features that the model generates a "narwhal" prediction, we need to solve $p(y = 1|\mathbf{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^T\mathbf{x}) \geq 0.5 \Rightarrow \boldsymbol{\theta}^T\mathbf{x} \geq 0$.
2. To let feature values be in range $[0, 1]$, we could add an inequality constraint: $0 \leq \mathbf{x} \leq 1$.
Thus we could instead solve $\mathbf{x}$ such that $\boldsymbol{\theta}^T\mathbf{x} \geq 0$, and $0 \leq \mathbf{x} \leq 1$. This procedure does not require a gradient calculation.

3. (2 points) **Select all that apply:** Now let's consider whether logistic regression is well-suited for this task. Suppose the exact same white narwhal in a dark ocean background was used to generate the training set. The training photos were captured with the side view of the narwhal 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 "narwhal"?

   - ☐ A new photo with the same narwhal in the upper right corner of the image.
   - ■ Identical to one of the training photos, but the narwhal replaced with an equal size white cardboard cutout of the narwhal.
   - ☐ Identical to one of the training photos, but the background changed to white.
   - ☐ None of the above.

6 Vectorization and Pseudocode (6 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 produce the same result?

- ■ `u = X @ v`
- □ `u = v @ X`
- ■ `u = np.matmul(X, v)`
- □ `u = np.matmul(v, X)`
- □ `u = X * v`
- □ `u = v * X`
- ■ `u = np.dot(X, v)`
- □ `u = np.dot(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) Select the line(s) of valid Python code that constructs a matrix whose i-th row is $\left(\mathbf{x}_i - \overline{\mathbf{x}_i}\right)^T$.

   - □ `(X - np.mean(X, axis=0)).T`
   - ■ `X - np.mean(X, axis=1, keepdims=True)`
   - □ `X - np.mean(X, axis=0, keepdims=True)`
   - ■ `X - np.expand_dims(np.mean(X, axis=1), 1)`
   - □ None of the above.

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

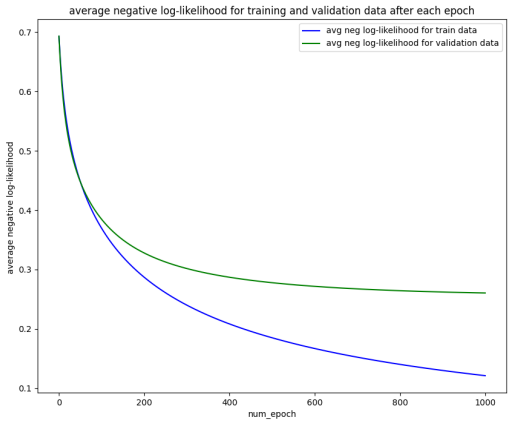
- ■ `np.matmul(w * M.T, M)`
- □ `np.matmul(w * M, M.T)`
- □ `np.dot(w * M, M.T)`
- □ `w * np.dot(M.T, M)`
- □ None of the above.

7 Programming Empirical Questions (9 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**. For all the questions below, unless otherwise specified, use the constant learning rate 0.001.

1. (2 points) 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 1,000 epochs. The y-axis should show the negative log-likelihood and the x-axis should show the number of epochs.

Your Answer



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

Your Answer

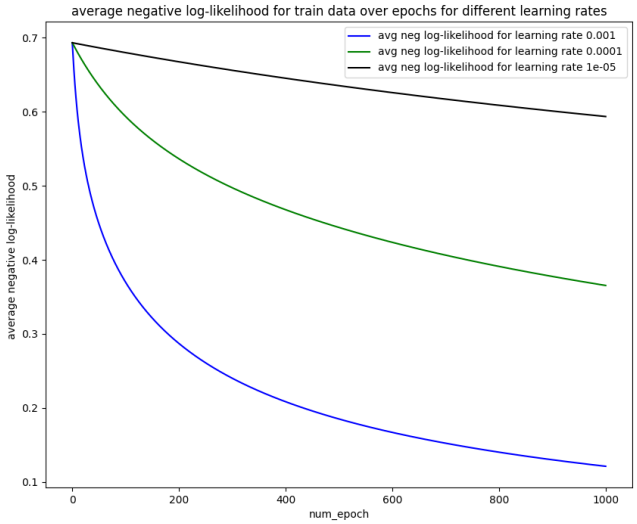
The training and validation log-likelihood curves look different. The validation dataset has larger average negative log-likelihood than train dataset after each of 1000 training epochs.
The validation dataset is used to tune the hyper-parameters. The validation error is generally lager than train dataset, and the difference gets smaller as we tune our hyper-parameters better. In this experiment, we didn't do the experiment to tune hyper-parameters using validation dataset. Therefore, there could be an obvious difference between validation and train error.
The negative log-likelihood is our train objective, and we want to minimize it in order to minimize the prediction errors, hence if validation has larger error than train dataset, validation dataset should also has a larger average negative log-likelihood.

3. (2 points) Report your train and test error for the large data set (found in the `largedata` folder in the handout) after running for 1,000 epochs. Please round to the fourth decimal place, e.g., 0.1234.

Train Error	Test Error
0.0100	0.1625

4. (2 points) 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, $\eta \in \{10^{-3}, 10^{-4}, 10^{-5}\}$. The y-axis should show the *average* negative log-likelihood, the x-axis should show the number of epochs (from 0 to 1,000 epochs), and the plot should contain three curves corresponding to the three values of η. Provide a legend that indicates the learning rate η for each curve.

Your Answer



5. (1 point) Compare how quickly each curve in the previous question converges.

Your Answer

The training curve for learning rate 10^{-5} has the slowest convergence rate.
The training curve for learning rate 10^{-4} has a faster convergence rate than 10^{-5} curve.
The training curve for learning rate 10^{-3} has the fastest convergence rate, and much faster than 10^{-5} and 10^{-4} curves.

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 (75 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 restaurant review is positive or negative.

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 Yelp dataset.[1] In the data files, each line is a single example that consists of a label (0 for negative reviews and 1 for positive ones) and a set of words. The format of each 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.

Examples of the data are as follows:

```
1   i will never forget this single breakfast experience in mad...
0   the search for decent chinese takeout in madison continues ...
0   sorry but me julio fell way below the standard even for med...
1   so this is the kind of food that will kill you so there s t...
```

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 review has variable length and is not real-valued.

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 a word embeddings model, 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 Model

In order to transform a set of words into vectors, we rely on a popular method of feature engineering: word embeddings.

We use ϕ to denote a feature engineering method and $\mathbf{x}^{(i)}$ to denote a training example (a set of English words as seen in 9.1).

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. [2]

Embeddings `word2vec.txt` contains the *word2vec* embeddings of `15k` words. Not every word in each review is present in the provided `word2vec.txt` file. We treat such missing words as "out-of-vocabulary" and ignore them. Each line consists of a word and its embedding separated by tabs:

[1] For more details, see https://www.yelp.com/dataset.

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

`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   ...
```

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 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:

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

   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:

- 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. [3]
- The feature for $\mathbf{x}^{(i)}$ can be calculated as

$$\begin{aligned}\boldsymbol{\phi}_2(\mathbf{x}^{(i)}) &= \frac{1}{5}\big(word2vec(\text{hot}) + 2\cdot word2vec(\text{not}) + word2vec(\text{sandwich}) + word2vec(\text{square})\big)\\ &= \begin{bmatrix}0.02 & 0.06 & 0.12\end{bmatrix}^T.\end{aligned}$$

[3] 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.

9.3 `feature.py`

`feature.py` implements 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).

Inputs

- **Input data** for training, validation, and testing. Each data point contains a label and an English restaurant review in the format described in 9.1.
- **word2vec embeddings** to use for the word embedding feature extraction methods.

Outputs

- **Formatted data** for training, validation, and testing. You should perform feature extraction on *each* of the training, validation, and test sets.

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 restaurant review, where the first entry is the label and the rest are the features in the feature vector. 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 on the small dataset; they are partially reproduced below for your reference. Please round your outputs to 6 decimal places.

```
1.000000        -0.166646       0.641027        -0.064805       ...
0.000000        -0.224874       0.461526        -0.215232       ...
0.000000        -0.222178       0.437475        -0.083073       ...
1.000000        -0.215923       0.612535         0.061671       ...
```

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 restaurant review was negative or positive. **Inputs**

- **Formatted data** for training, validation, and testing. Each data point contains a label and a corresponding feature vector. These files are the ones produced by `feature.py`.
- **The number of epochs** to train for, which will be passed in as a command line argument.
- **The learning rate**, also passed in via the command line.

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 (recommended). 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.
- 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. **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 `.txt` 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.

```
1
0
0
1
```

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 you should round the error values to 6 decimal places. The following example is the reference solution for the small dataset after 500 training epochs with learning rate 0.001.

```
error(train): 0.000000
error(test): 0.625000
```

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

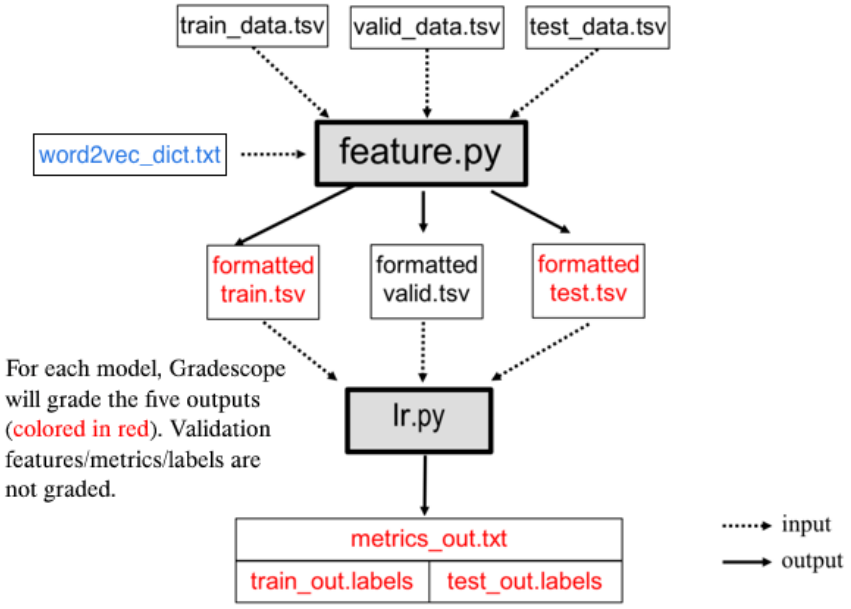


Figure 1: Programming pipeline for sentiment analyzer based on binary logistic regression

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 seven command-line arguments: `<train_input>` `<validation_input>` `<test_input>` `<feature_dictionary_input>` `<formatted_train_out>` `<formatted_validation_out>` `<formatted_test_out>` . 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. `<feature_dictionary_input>`: path to the word2vec feature dictionary `.txt` file (see Section 9.2)
5. `<formatted_train_out>`: path to output `.tsv` file to which the feature extractions on the *training* data should be written (see Section 9.3)
6. `<formatted_validation_out>`: path to output `.tsv` file to which the feature extractions on the *validation* data should be written (see Section 9.3)
7. `<formatted_test_out>`: path to output `.tsv` file to which the feature extractions on the *test* data should be written (see Section 9.3)

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 `.txt` file to which the prediction on the *training* data should be written (see Section 9.4)
5. `<test_out>`: path to output `.txt` 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; in the reference output, we set the learning rate to be 0.001 for all datasets

As an example, the following two command lines would run your programs on the large dataset in the handout for 500 epochs. You are given the output of this command and the equivalent command on the small dataset in the handout directories `largeoutput` and `smalloutput`.

```
$ python feature.py \
largedata/train_large.tsv \
largedata/val_large.tsv \
largedata/test_large.tsv \
word2vec.txt \
largeoutput/formatted_train_large.tsv \
largeoutput/formatted_val_large.tsv \
largeoutput/formatted_test_large.tsv

$ python lr.py \
largeoutput/formatted_train_large.tsv \
largeoutput/formatted_val_large.tsv \
largeoutput/formatted_test_large.tsv \
largeoutput/formatted_train_labels.txt \
largeoutput/formatted_test_labels.txt \
largeoutput/formatted_metrics.txt \
500 \
0.001
```

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 Starter Code

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

9.7 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.