HOMEWORK 4: LOGISTIC REGRESSION

10-301/10-601 Introduction to Machine Learning (Summer 2024)
https://www.cs.cmu.edu/~hchai2/courses/10601/
OUT: Tuesday, June 4th
DUE: Tuesday, June 11th

START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 2.1"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section on the course site for more information: https://www.cs.cmu.edu/~hchai2/courses/10601/#Syllabus
- **Late Submission Policy:** See the late submission policy here: https://www.cs.cmu.edu/~hchai2/courses/10601/#Syllabus
- **Submitting your work:**
  - **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 a **total of 10 Gradescope programming submissions.** Use them wisely. In order to not waste code submissions, we recommend debugging your implementation on your local machine (or the linux servers) and making sure your code is running correctly first before any Gradescope coding submission.
  - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, we will be using Gradescope (https://gradescope.com/). Please use the provided template. Submissions must be written in LaTeX. Regrade requests can be made, however this gives the TA the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted. Each derivation/proof should be completed on a separate page. For short answer questions you **should not** include your work in your solution. If you include your work in your solutions, your assignment may not be graded correctly by our AI assisted grader.
- **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.

For multiple choice or select all that apply questions, shade in the box or circle in the template document corresponding to the correct answer(s) for each of the questions. For LaTeX users, replace `\choice` with

`\CorrectChoice` to obtain a shaded box/circle, and don't change anything else.

Instructions for Specific Problem Types

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

Select One: Who taught this course?

- ● Henry Chai
- ○ 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?

$\boxed{\text{10-601}}$ $\boxed{\text{10-}\cancel{6}\text{301}}$

Written Problems (42 points)

1 Linear Regression (6 points)

1. Consider the following dataset:

x	9.0	2.0	6.0	1.0	8.0
y	1.0	0.0	3.0	0.0	1.0

Let $\boldsymbol{x}$ be the vector of datapoints and $\boldsymbol{y}$ be the label vector. Here, we are fitting the data using gradient descent, and our objective function is $\frac{1}{N}\sum_{i=1}^{N}(wx_i + b - y_i)^2$ where N is the number of data points, w is the weight, and b is the intercept.

(a) (2 points) If we initialize the weight as 3.0 and intercept as 0.0, what is the gradient of the loss function with respect to the weight w, calculated over all the data points, in the first step of the gradient descent update?

Gradient:

Work

(b) (2 points) What is the gradient of the loss function with respect to the intercept b, calculated over all the data points, in the first step of the gradient descent update?

Gradient:

Work

(c) (2 points) Let the learning rate be 0.01. Perform one step of gradient descent on the data. Fill in the following blanks with the value of the weight and the value of the intercept after this step.

Weight:

Intercept:

2 MLE/MAP (4 points)

1. (1 point) **True or False:** Suppose you place a Beta prior over the Bernoulli distribution, and attempt to learn the parameter w of the Bernoulli distribution from data. Further suppose an adversary chooses "bad" but finite hyperparameters for your Beta prior in order to confuse your learning algorithm. As the number of training examples grows to infinity, the MAP estimate of w can still converge to the MLE estimate of w.

   ◯ True

   ◯ False

2. (1 point) **Select one:** Let Γ be a random variable with the following probability density function (pdf):

$$f(\gamma) = \begin{cases} 2\gamma & \text{if } 0 \leq \gamma \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

   Suppose another random variable Y, which is conditioning on Γ, follows an exponential distribution with $\lambda = 3\gamma$. Recall that the exponential distribution with parameter λ has the following pdf:

$$f_{exp}(y) = \begin{cases} \lambda e^{-\lambda y} & \text{if } y \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

   What is the MAP estimate of γ given $Y = \frac{2}{3}$ is observed?

Your Answer

3. (2 points) Neural the Narwhal found a mystery coin and wants to know the probability of landing on heads by flipping this coin. He models the coin toss as sampling a value from Bernoulli(w) where w is the probability of heads. He flips the coin three times and the flips turned out to be heads, tails, and heads. An oracle tells him that $w \in \{0, 0.25, 0.5, 0.75, 1\}$, and *no other values of w should be considered.*

   Find the MLE and MAP estimates of w. Use the following prior distribution for the MAP estimate:

$$p(w) = \begin{cases} 0.9 & \text{if } w = 0 \\ 0.04 & \text{if } w = 0.25 \\ 0.03 & \text{if } w = 0.5 \\ 0.02 & \text{if } w = 0.75 \\ 0.01 & \text{if } w = 1 \end{cases}.$$

   Again, remember that $w \in \{0, 0.25, 0.5, 0.75, 1\}$, so the MLE and MAP should also be one of them.

MLE of w	MAP of w

3 Logistic Regression: Analysis & Practice (12 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 $\ell(\mathbf{w})$ for binary logistic regression can be expressed as

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

   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, $\mathbf{w} \in \mathbb{R}^{M+1}$ 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

$$f(\mathbf{w}) = \ell(\mathbf{w}) + \lambda\frac{1}{2}\sum_{j=0}^{M} w_j^2$$

   where λ is the regularization weight, w_j is the jth element in the weight vector $\mathbf{w}$. Suppose we are updating w_k with learning rate η, which of the following is the correct expression for the update?

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

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

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

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

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

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. 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 $\mathbf{w}$ after training is

$$\mathbf{w} = \begin{bmatrix} 1.5 & 2 & 1 \end{bmatrix}^T.$$

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.

(a) (2 points) Calculate $\ell(\mathbf{w})$, $\frac{1}{N}$ times the negative log-likelihood over the given data and parameter $\mathbf{w}$. (Note here we are using natural log, i.e., the base is e).

$\ell(\mathbf{w})$

Work

(b) (2 points) Calculate the gradients $\frac{\partial \ell(\mathbf{w})}{\partial w_j}$ with respect to w_j for all $j \in \{1, 2, 3\}$.

$\partial \ell(\mathbf{w})/\partial w_1$	$\partial \ell(\mathbf{w})/\partial w_2$	$\partial \ell(\mathbf{w})/\partial w_3$

Work

(c) (1 point) Update the parameters following the parameter update step $w_j \leftarrow w_j - \eta \frac{\partial \ell(\mathbf{w})}{\partial w_j}$ and write the updated (numerical) value of the vector $\mathbf{w}$. Use learning rate $\eta = 1$.

w_1	w_2	w_3

Work

4 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.

Consider a logistic regression model whose purpose is to identify narwhals in grayscale images. Each pixel has an intensity value in the continuous range $[0, 1]$, zero being the darkest. The model outputs $\mathbf{y}^{(i)} = 1$ when it predicts the input image contains a narwhal. So we can represent the probability function for whether logistic regression predicts a narwhal as

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

where $\mathbf{w}$ is the vector of learned coefficients and $\mathbf{x}$ is a vector of pixels representing the input image. For example, consider a small picture that consists of 4 pixels with the corresponding intensity values

0.0	0.1
0.5	1.0

In order to input this into our model, we could create the vector $\mathbf{x} = \begin{bmatrix} 0.0 & 0.5 & 0.1 & 1.0 \end{bmatrix}$ where each element in the vector corresponds to a specific pixel in the image, arranged in column-major order.

1. (2 points) After training the model on a training dataset, we arrive at a set of parameters $\mathbf{w}$. Given the model parameters $\mathbf{w}$ and the probability function as defined above, we wish to find an input vector $\mathbf{x}$ that causes the model to make a false prediction of a narwhal. In this subpart, we will do this by using gradient ascent *on our input* $\mathbf{x}$, keeping $\mathbf{w}$ fixed, to optimize the probability that $\mathbf{x}$ is assigned to the narwhal class.

   Given this setup, write (1) the gradient of the probability function with respect to $\mathbf{x}$ and (2) the *gradient ascent* update rule for $\mathbf{x}$. Define the learning rate to be η.

Gradient of the probability function with respect to x

Update Rule

2. (1 point) Using the parameter values $\mathbf{w}$, directly define in closed form a vector $\mathbf{x}$ that maximizes the probability of the narwhal class. You should not use any gradient calculations, nor should you use any iterative updates to compute $\mathbf{w}$. Assume that none of the elements of $\mathbf{w}$ are 0. Remember that we require the feature values to be in the range $[0, 1]$ to generate an image.

Your Answer

3. (2 points) **Select one:** Now let's consider whether logistic regression is well-suited for this task. Suppose photos of 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 of the below descriptions of a **test image**, if any, would the model be most likely to 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.

5 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 $\mathbf{w}$, respectively. `X.shape == (N, M)`, `w.shape == (N,)`.

   (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.

6 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 0.1.

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

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

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, $\eta \in \{10^{-1}, 10^{-2}, 10^{-3}\}$. 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

7 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

8 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.

8.1 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 (8.2).

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.

8.2 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 8.1).

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 *GloVe* embeddings, a commonly used set of feature vectors. [2]

Embeddings `glove_embeddings.txt` contains the *GloVe* embeddings of `6792` words. Not every word in each review is present in the provided `glove_embeddings.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 work at https://nlp.stanford.edu/projects/glove/

`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 `glove_embeddings.txt`, with values rounded to 3 decimal places:

```
deserves  0.175  -0.153  -0.208  0.092  0.222   0.202  ...
butter    0.357   0.469  -0.021  0.024 -0.168  -0.213 ...
staffing -0.076   0.212  -0.384  0.552 -0.193  -0.052 ...
weird     0.110   0.090   0.139  0.340 -0.098  -0.113 ...
```

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 *GloVe* 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 `glove_embeddings.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 *GloVe* embeddings of its words:

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

   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, $\mathit{GloVe}(\mathbf{x}\_\text{trim}_j^{(i)}) \in \mathbb{R}^{300}$ is the *GloVe* 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 *GloVe* 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 *GloVe* 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

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

[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.

8.3 `feature.py`

`feature.py` implements word embeddings (described above in 8.2) 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 8.1.
- ***GloVe* 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 *GloVe* 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       ...
```

8.4 `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 0.1.

```
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.

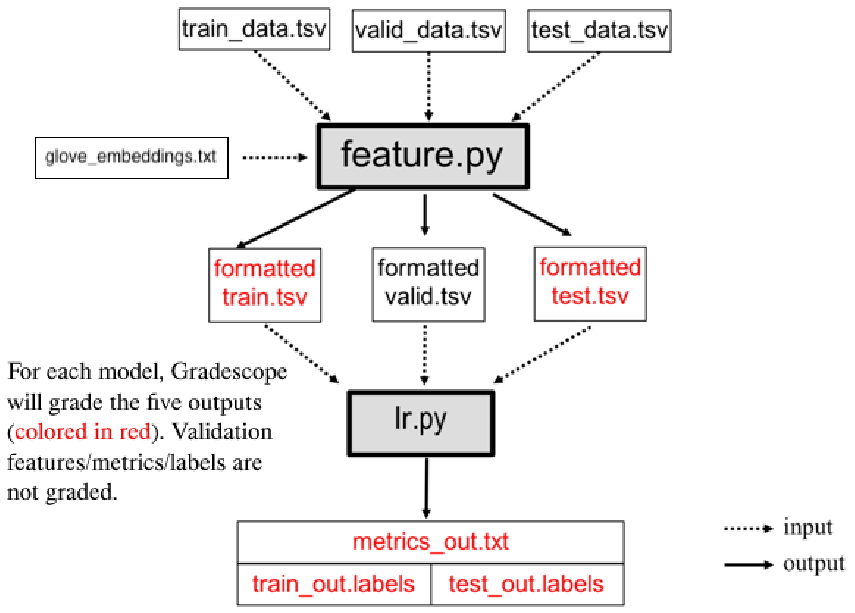

Figure 1: Programming pipeline for sentiment analyzer based on binary logistic regression

8.5 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 8.1)
2. `<validation_input>`: path to the validation input `.tsv` file (see Section 8.1)
3. `<test_input>`: path to the test input `.tsv` file (see Section 8.1)
4. `<feature_dictionary_input>`: path to the *GloVe* feature dictionary `.txt` file (see Section 8.2)
5. `<formatted_train_out>`: path to output `.tsv` file to which the feature extractions on the *training* data should be written (see Section 8.3)
6. `<formatted_validation_out>`: path to output `.tsv` file to which the feature extractions on the *validation* data should be written (see Section 8.3)
7. `<formatted_test_out>`: path to output `.tsv` file to which the feature extractions on the *test* data should be written (see Section 8.3)

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 8.3)
2. `<formatted_validation_input>`: path to the formatted validation input `.tsv` file (see Section 8.3)
3. `<formatted_test_input>`: path to the formatted test input `.tsv` file (see Section 8.3)
4. `<train_out>`: path to output `.txt` file to which the prediction on the *training* data should be written (see Section 8.4)
5. `<test_out>`: path to output `.txt` file to which the prediction on the *test* data should be written (see Section 8.4)
6. `<metrics_out>`: path of the output `.txt` file to which metrics such as train and test error should be written (see Section 8.4)
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 0.1 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 \
glove_embeddings.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 \
0.1
```

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.

8.6 Starter Code

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

8.7 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.