Assignment 1: Text Classification with Logistic Regression

Instructor: Noah Smith *CSE 517 and 447 – Winter 2025*

Due at 11:59pm PT, January 22, 2025
50 points for CSE 447 (+ 5 points extra credit) / 60 points for CSE 517

In this assignment, you will learn how to implement logistic regression for binary and multi-class classifiers from scratch.

You will submit both your **code** and **writeup** (as PDF) via Gradescope. At the time of release of this homework (January 13, 2025), the submissions have yet not been enabled on gradescope. Remember to **specify your collaborators** (including AI tools like ChatGPT) and **how they contribute** to the completion of your assignment at the beginning of your writeup. If you work on the assignment independently, please specify so, too.

Required Deliverables

- **Code Notebook**: Submit the notebook as a `.ipynb` file. On Google Colab you can download it via `File` → `Download` → `Download .ipynb`. Make sure the name of the file is **exactly** `CSE447_Assignment1.ipynb` so that the autograder recognizes it.
- **CSV Files With Predictions and Adversarial Examples**: As you will go through the notebook, you will be asked to make predictions on a test dataset and save them in a CSV file. **Follow the exact names for these files as specified in the notebook** (`test_data_with_binary_predictions.csv` and `test_data_with_multiclass_predictions.csv`). For CSE 517, there is a question on generating adversarial examples and storing them in a csv file called `adversarial_examples.csv`. Upload all the three CSV files (or 2 if you are CSE 447 and not attempting the optional question) on gradescope along with your code.
- **Write-up**: For written answers and open-ended reports, produce a single PDF and submit it in Gradescope. We recommend using Overleaf to typeset your answers in LaTeX, but other legible typed formats are acceptable. We do not accept hand-written solutions because grading hand-written reports is incredibly challenging.

Recommended Reading

The homework is based on chapter 5 of Jurafsky and Martin. We provide all the details necessary to solve the homework in this handout and the notebooks, so it is not required to read the chapter to solve the exercises. However, we recommend going through these chapters if you are confused about any concepts that are covered in the homework.

Acknowledgment

This assignment is designed by Kabir Ahuja with invaluable feedback from Riva Gore, Khushi Khandelwal, Melissa Mitchell, and Kavel Rao. Kavel Rao also helped design autograder for the homework.

1 Text Classification Using Logistic Regression (50 pt for 447 and 60 pt for 517)

In this project, you will implement linear text classifiers for sentiment analysis. We will be working with the Stanford Sentiment Treebank (SST), which contains movie reviews with sentiment labels. We will consider both the binary version (only two labels positive and negative) as well as 5 label version (very negative, negative, neutral, positive, very positive). In particular you will learn:

- How to convert text inputs to features;
- How to train logistic regression models from scratch;
- How to evaluate classification models;
- How to interpret trained linear models;
- (Optional for 447) How to use insights from a model's decision making process to generate adversarial examples.

Notebook: You will use this Python notebook: CSE447_Assignment1.ipynb. Please make a copy for yourself by navigating to `File` → `Save a copy in Drive`. Alternatively, when attempting to save, Google Colab will prompt you to save a copy in your own drive. Make your way through the notebook and implement the classes and functions as specified in the instructions. All the data necessary for this project can be downloaded within the notebook itself.

Deliverables:

1. **Coding Exercises:** You should complete the code blocks denoted by `YOUR CODE HERE:` in the Python notebook. Do not forget to remove `raise NotImplementedError()` from the code blocks. To submit your code, download your notebook as a Python file (`CSE447a_Assignment1.py`).
2. **Files containing Predictions and Adversarial Examples** You should submit CSV files for test data predictions and generated adversarial examples. More details in the notebook and below.
3. **Write-up:** Your report for §1 should be **no more than four pages**. However, you will most likely be able to answer all questions within three pages. Note that the notebook also lists the same questions as in the write-up document, but those should be answered in the write-up pdf only and not in the notebook.

1.1 Converting Text To Features (8 Points)

Typical ML models work on the data described using mathematical objects like vectors and matrices, which are often referred to as feature representations, elements of which are referred to as features. These features can be of different types depending upon the downstream application, like for building a classifier to predict whether to give credit to a customer, we might consider features like their age, income, employement status, etc. In the same way, to build a classifier for textual data, we need a way to describe (or represent) each text example in terms of numeric features which can then be fed to the classification algorithm of our choice.

Coding Exercises (8 points). Implement your code for the following classes in the notebook:

- `LinguisticVectorizer` (2 points)
- `BOWVectorizer` (3 points)

- `BOWVectorizerWNormalizer` (3 points)

Implementation Hints:

Warning: Failure to follow the steps may result in autotest failures.

- While normalizing documents, process them in following order:
  1. Lowercase
  2. Remove punctuation
  3. Remove stop words
  4. Convert low frequency words to <unk>
- Compute the word count with normalized words except <unk>
- While tokenizing the words out of the document, please use Python native String.split() method rather other "fancy" NLP packages (i.e. nltk.tokenize()).

1.2 Binary Logistic Regression (27 points for 447 and 37 points for 547)

We will now start building our first text classifier! We will start with logistic regression for binary classification, i.e., where each input is to be classified as one of the two labels. Consider an input $\boldsymbol{x} \in \mathbb{R}^d$ defining the feature vector and label $y \in \{0, 1\}$. Recall from the lectures that logistic regression has the following functional form:

$$\hat{y} = P(y = 1) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$$

where σ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

and the weight vector $\boldsymbol{w} \in \mathbb{R}^d$ and the bias term $b \in \mathbb{R}$ are the learnable parameters in the model. Note that during training, we use "soft" predictions (the probability the model assigns to label 1), rather than "hard" prediction (which we would get by rounding the output of the sigmoid function). When evaluating a model (test time), we would typically use hard predictions.

Binary Cross Entropy Loss To train a logistic regression model. we need to first define a loss function that quantifies the error between the model's prediction of the label ($\hat{y}$) and the ground truth label (y). For logistic regression with binary labels, we use binary cross entropy (BCE) loss. For a given prediction/ground truth pair, the BCE loss is given as:

$$\begin{aligned} L_{\text{BCE}}(\hat{y}, y) &= \begin{cases} -\log \hat{y} & \text{if } y = 1 \\ -\log(1 - \hat{y}) & \text{if } y = 0 \end{cases} \\ &= -\left[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\right] \end{aligned}$$

Plugging in the value of $\hat{y}$, we get the loss for a single instance:

$$L_{\text{BCE}}(\boldsymbol{w}, b \mid \boldsymbol{x}, y) = -[y \log \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b) + (1 - y) \log(1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b))]$$

To compute loss over the entire dataset $\mathcal{D}$, we average the loss for all examples:

$$L_{\text{BCE}}(\boldsymbol{w}, b \mid \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} L_{\text{BCE}}(\hat{y}_i, y_i)$$

where m is the number of examples in the dataset.

Gradient Descent for Logistic Regression The next step is find the values of the weight vector and bias term that minimize the binary cross entropy loss. One of the most commonly used optimization algorithms in machine learning is gradient descent. Recall from lectures that in gradient descent we iteratively update the parameters of a model in the opposite direction of the gradient of loss function with respect to the parameters, i.e.,

$$\theta^{t+1} = \theta^t - \frac{\eta}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x_i;\theta), y_i)$$

Note that for logistic regression, $L(f(x;\theta), y) = L_{\text{BCE}}(\hat{y}, y)$. θ denotes the parameters of the model, which for us is the weights $\boldsymbol{w}$ and bias b. η is called the learning rate, which determines the strength of every the update (too low then the convergence will be slow and too high we might overshoot the minimum).

The gradient of the BCE loss with respect to $\boldsymbol{w}$ is given by:

$$\nabla_w L_{\text{BCE}}(\hat{y}_i, y_i) = \left[\frac{\partial L_{\text{BCE}}(\hat{y}_i, y_i)}{\partial w_1}, \cdots, \frac{\partial L_{\text{BCE}}(\hat{y}_i, y_i)}{\partial w_d}\right]^\top$$

$$\frac{\partial L_{\text{BCE}}(\hat{y}_i, y_i)}{\partial w_j} = -(y_i - \hat{y}_i)x_{ij}$$

Note that x_{ij} refers to the jth feature of the ith example. Similarly, for the bias term we get:

$$\nabla_b L_{\text{BCE}}(\hat{y}_i, y_i) = \frac{\partial L_{\text{BCE}}(\hat{y}_i, y_i)}{\partial b} = -(y_i - \hat{y}_i)$$

Plugging these into our gradient descent equation we get:

$$w_j^{t+1} = w_j^t - \frac{\eta}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)x_{ij}$$

$$b^{t+1} = b^t - \frac{\eta}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)$$

Before we begin our implementation, there are two points to note. First you must have noticed that we sum over m examples in our update equation. These are the all the examples in our training data. In practice, with large datasets, it can get very expensive to compute gradients with respect to all examples. In such cases, it is common to use stochastic or mini-batch gradient descent, where for each update we only use a small batch of training data to update the weights (we use a different batch for every update till we exhaust all the training data), with optional repeated passes over the training dataset. An extreme case of this is where we only use one example at a time to update the weights:

$$w_j^{t+1} = w_j^t - \eta(\hat{y}_i - y_i)x_{ij}$$

$$b^{t+1} = b^t - \eta(\hat{y}_i - y_i)$$

Second, we wrote the above equations in terms of scalar variables and their summations. In practice, it can be much more efficient to vectorize these equations and use matrix operations which can make use of parallel computation. The vectorized version of our update rule will look like:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \frac{\eta}{m}\boldsymbol{X}^\top(\hat{\boldsymbol{y}} - \boldsymbol{y})$$

$$b^{t+1} = b^t - \frac{\eta}{m}\mathbf{1}_d^\top(\hat{\boldsymbol{y}} - \boldsymbol{y})$$

Here, $\boldsymbol{X} \in \mathbb{R}^{m\times d}$ is the input matrix where each row is a feature vector for an input example. Similarly, $\boldsymbol{y} \in \{0,1\}^m$ is the vector containing labels for each example and $\hat{\boldsymbol{y}} \in \mathbb{R}^m$ is the vector of predicted labels. $\mathbf{1}_d \in \mathbb{R}^d$ is a d-length vector of all ones.

1.2.1 Coding Exercises (20 points).

Implement the following classes and functions in the notebook:

- class `LogisticRegression` (2 points)
- function `bce_loss` (1 points)
- function `gradient_descent_update_vanilla` (2 points)
- function `gradient_descent_update_vectorized` (2 points)
- function `train_logistic_regression` (5 points)
- functions `get_accuracy`, `get_precision`, `get_recall`, and `get_f1_score` (2 points)
- function `evaluate_logistic_regression` (3 points)
- function `interpret_logistic_regression` (3 points)

1.2.2 Write-Up Questions (7 points + 5 points extra credits for 447 and 17 points for 547).

Check section 1 of CSE447_Assignment1_Writeup.pdf for detail.

1.3 Multinomial Logistic Regression (15 points)

We will now move to the multi-class case, i.e., where the text is to be classified into more than 2 classes. We will be working with the 5-label version of the SST dataset where the labels are: very negative, negative, neutral, positive, very positive.

To extend logistic regression to multi-class classification, we can use the softmax function. For a vector $\boldsymbol{z} = [z_1, z_2, \cdots, z_K]$ of K arbitrary real numbers, the softmax function maps them to a probability distribution, with each value between 0 and 1 and summing to 1. The softmax function is defined as:

$$\texttt{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \quad \forall i \in \{1, \ldots, K\}$$

In multinomial logistic regression, we consider a weight vector $\boldsymbol{w}_k$ for each of the K classes, take the dot product with input features (also adding a bias term unique for each class) to obtain scores or *logits* for each of the classes. We then apply the softmax function to get the probability distribution over the classes. Formally, this is given by:

$$\hat{y}_k = P(y_k = 1 \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j)}$$

where $P(y_k = 1 \mid \boldsymbol{x})$ is the probability of the input $\boldsymbol{x}$ belonging to class k, $\boldsymbol{w}_k$ is the weight vector for class k, and b_k is the bias term for class k. Note that, with multinomial logistic regression, you cannot calculate the probability of any label without calculating the logits for all of the labels.

Note: In practice, we often use a trick to make the computation of softmax more numerically stable. We subtract the maximum value from the logits before applying softmax. This does not change the output of softmax but can prevent numerical overflow. The softmax function with this trick is given by:

$$\texttt{softmax}(z_i) = \frac{\exp(z_i - \max(\boldsymbol{z}))}{\sum_{j=1}^{K} \exp(z_j - \max(\boldsymbol{z}))}, \quad \forall i \in \{1, \ldots, K\}$$

where here "max" takes the elementwise maximum from the vector.

Cross Entropy Loss for Multinomial Logistic Regression. The loss function for multinomial logistic regression is the cross entropy loss. Here it is useful to represent the ground truth label as a K-dimensional "one-hot" vector, $\boldsymbol{y} = \langle y_1, \ldots, y_K \rangle$ such that:

$$y_k = \begin{cases} 1 & \text{if } k \text{ is the correct label} \\ 0 & \text{otherwise} \end{cases}$$

(This is analogous to the way we treated $\hat{\boldsymbol{y}}$ above.) For a given input $\boldsymbol{x}$ and ground truth label $\boldsymbol{y}$, the cross entropy loss is given by:

$$L_{\text{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

$$L_{\text{CE}}(\boldsymbol{w}, \boldsymbol{b} \mid \boldsymbol{x}, \boldsymbol{y}) = -\sum_{k=1}^{K} y_k \log \frac{\exp(\boldsymbol{w}_k^\top \boldsymbol{x} + b_k)}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j)}$$

where $\hat{\boldsymbol{y}}$ is the predicted probability distribution over the classes and $\boldsymbol{y}$ is the one-hot encoded ground truth label, such that $y_k = 1$ if the input belongs to class k and 0 otherwise. We can compute the loss over the entire dataset $\mathcal{D}$ by averaging the loss over all examples:

$$L_{\text{CE}}(\boldsymbol{w}, \boldsymbol{b} \mid \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} L_{\text{CE}}(\hat{\boldsymbol{y}}_{\boldsymbol{i}}, \boldsymbol{y}_{\boldsymbol{i}})$$

Gradient Descent for Multinomial Logistic Regression The gradient of the cross entropy loss with respect to the weights and biases can be computed as follows:

$$\nabla_{\boldsymbol{w}_k} L_{\text{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{x}$$

$$\nabla_{b_k} L_{\text{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \hat{\boldsymbol{y}} - \boldsymbol{y}$$

The gradient descent update rule for the weights and biases is given by:

$$\boldsymbol{w}_k^{t+1} = \boldsymbol{w}_k^t - \frac{\eta}{m} \sum_{i=1}^{m} (\hat{\boldsymbol{y}}_{\boldsymbol{i}} - \boldsymbol{y}_{\boldsymbol{i}})^T \boldsymbol{x}_{\boldsymbol{i}}$$

$$b_k^{t+1} = b_k^t - \frac{\eta}{m} \sum_{i=1}^{m} (\hat{\boldsymbol{y}}_{\boldsymbol{i}} - \boldsymbol{y}_{\boldsymbol{i}})$$

1.3.1 Coding Exercises (11 points).

Implement the following functions and classes in Part 3 (Multinomial Logistic Regression) of the notebook:

- class `MultinomialLogisticRegression` (2 points)
- function `ce_loss` (1 points)
- function `gradient_descent_update_multiclass` (3 points)
- function `train_multinomial_logistic_regression` (2 points)
- functions `get_precision_multiclass`, `get_recall_multiclass`, `get_f1_score_multiclass`, and `get_confusion_matrix` (2 points)
- function `evaluate_multiclass_logistic_regression` (1 points)

1.3.2 Write-Up Questions (4 points)

Check section 2 of CSE447_Assignment1_Writeup.pdf for detail.