Overview of [the name of your ML algorithm] Give an overview of the algorithm and describe its advantages and disadvantages. Representation: describe how the feature values are converted into a single number prediction. Loss: describe the metric used to measure the difference between the model's prediction and the target variable. Optimizer: describe the numerical algorithm used to find the model parameters that minimize the loss for a training set. Use markdown in the jupyter notebook, add equations to explain math, and use pseudo-code to explain how numerical algorithms work.

Week of October 28th: You should complete the markdown section of the report and make sure everyone understands the math and numerical methods behind the algorithm.

DATA2060 Final Project

Jessica Wan, Michael Lu, Angela Zhu, Qiming Fang

October 2024

1 Overview of Multi-class Classification

Multi-class classification involves the use of a binary classification algorithm to determine the most probable decision from multiple class possibilities. Two main techniques utilizing logistical regression for multi-class classification are the one-vs-all and all-pairs algorithms.

1.1 Representation

Both one-vs-all and all-pairs multi-class classification algorithms are built using an ensemble of simple binary classification models. Any type of binary classification model can be used for the basic models. For our project we will be using the logistic regression binary classification model we completed in homework 3. The hypothesis of multiclass logistic regression of class j is given below.

$$h_w(x)_j = \frac{e^{\langle w_j, x\rangle}}{\sum_{s=1}^{k} e^{\langle w_s, x\rangle}}$$

The weight matrix w has size $k \times d$, where k is the number of classes and d is the number of features.

1.1.1 One-vs-all

This algorithm trains n binary classification models where n is the number of classes. Each of these model predicts whether a sample belongs to the n^{th} class. When predicting, the model selects the class with the highest predicted probability. For example, if we have 3 classes A, B, C, we would train 3 binary classification models as follows:

- Model 1: A vs. not A (B or C).
- Model 2: B vs. not B (A or C).
- Model 3: C vs. not C (A or B).

For a given sample, if the binary classification predictions are 0.76, 0.43, and 0.5 for classifiers 1, 2, and 3 respectively, the model would predict class A since classifier 1 had the highest predicted probability of the sample belonging to its predicted class.

Here the pseudo code for the algorithm:

input:
training set $S = (x_1, y_1), ..., (x_m, y_m)$
algorithm for binary classification model
foreach $i \in \mathcal{Y}$
let $S_i = (x_1, (-1)^{\mathbb{1}[y_1 \neq i]}), ..., (x_m, (-1)^{\mathbb{1}[y_m \neq i]}$
let $h_i = A(S_i)$
output:
the multi-class hypothesis defined by $h(x) \in \text{argmax}_{i \in \mathcal{Y}} h_i(x)$

One-vs-all is a very simple algorithm to implement and can provide explainable results compared to some more complex models. However, for tasks with large numbers of classes, this approach can be computationally expensive as each class would require its own binary classification model. Additionally, large numbers of classes will lead to increasingly imbalanced data for each binary classification model. If we have five classes with a perfectly balanced distribution, each binary classification model will have to train on a dataset with a 20/80 imbalance.

1.1.2 All-pairs

This algorithm uses binary classification on all pairs of possible classes, hence if we have n classes, then the algorithm would train $\binom{n}{2}$ binary classification models. For each pair of classes, call this (i, j) with i and j as different classes/labels, we would take all samples with label either i or j, then train our binary classification model on this subset of data to predict whether or not a sample belongs to class i or class j. In our binary classification model we can label class i as 1 and class j as -1. When predicting, for each class i we will essentially see how often each binary classification for i against every other class j will predict i, and the class i with the highest winning rate will be our final prediction.

Here is the pseudocode for the algorithm:
input:
training set $S = (x_1, y_1), ..., (x_m, y_m)$
algorithm for binary classification A
foreach $i, j \in \mathcal{Y}, i < j$:
$S_{i,j}$ initialize to be empty
for t=1,...,m:
if $y_t = i$, add $(x_t, 1)$ to $S_{i,j}$
if $y_t = j$, add $(x_t, -1)$ to $S_{i,j}$ let $h_{i,j} = A(S_{i,j})$

output:

the multi-class hypothesis defined by $h(x) \in argmax_{i \in \mathcal{Y}}(\sum_{j \in \mathcal{Y}} sign(j - i)h_{i,j}(x))$

Compared to the one-vs-all approach, the all-pairs algorithm is faster to train on each binary classification model as it trains on a smaller portion of data than one-vs-all, however all-pairs takes longer to compute our final prediction. With increasing number of classes, this method can also be computationally expensive, as we have to train $\binom{n}{2}$ models, where n is our number of classes.

1.2 Loss

We use multiclass log loss, also known as cross-entropy loss for multi-class classification. Cross entropy loss is given by the equation below.

$$L_s(h_w) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{k} \mathbf{1}[y_i = j] \log h_w(x_i)_j$$

where m is the number of observations and k is the number of classes. For each observation, we compute the softmax probability for the true class j. If the true label of y_i matches class j, we add the negative logarithm of this probability to the total loss. After calculating this for each observation, we sum the individual losses and take the mean across all observations.

1.3 Optimizer

In order to optimize the weights to minimize loss with logistic regression, we must perform stochastic gradient descent (SGD) for each batch of training data used to learn our model. Stochastic gradient descent follows the form:

$$w = w - \alpha \nabla L_s(h_w)$$

where w represents the weighting coefficient vector for each feature, α determines the step size of our descent, and $\nabla L_s(h_w)$ is the gradient of the loss function associated with the probability that a data point x belongs to a class s. The batch size, which ranges from (1, size of training set), can be determined according to the size of the data set and what is optimal for the model's time-space expectations. Smaller batch sizes result in noisy estimates, but faster convergence. Larger batch sizes are slower, but more accurate.

We continue the descent until reaching a convergence threshold ϵ, at which point we can assume we have reached the local minimum of the loss function.

The algorithm for SGD to find the minimum loss until convergence is described below:

input:

training examples X,Y of size S

step size α
batch size $b < S$

```
converge = False
while not converge:
    shuffle X,Y
    prev_epoch_loss = loss(X,Y)
    for each batch:
        Lw = zeros((num_classes, num_features + 1))
        for x,y in batch:
            probabilities = softmax(x)
            for class j in classes:
                h_wx = probabilities[j]
                if y == j: Lw[j] += (h_wx - 1)*x
                else: Lw[j] += (h_wx)*x
        w -= alpha * Lw / num_examples_in_batch
    curr_epoch_loss = loss(X,Y)
    converge = diff(curr_epoch_loss, prev_epoch_loss) < threshold
```