HW 1

Garrison Neel
CSCE 636

September 8, 2020

I Answers to the non-Programming Part

1 Data Preprocessing

(a) Explain what the function train_valid_split does and why we need this step.

The function train_valid_split separates the training data into two sections, one for training and one for validation. Ideally, this function ensures that the training data and validation data have similar distributions after the split. Keeping a separate validation set allows us to test the model on data it was not trained on, in order to monitor how training is progressing. Poor performance on the validation data is a sign of over-fitting. Good training and validation accuracy together are a sign that the model will work well on the test dataset. The test dataset is not used during training in order to later evaluate how the model will perform on data which did not influence the training.

(b) Before testing, is it correct to re-train the model on the whole training set? Explain your answer.

No. Retraining the model invalidates whatever metrics were used to tune the training process. Early stopping is also not possible when training on the full training set. The model can then potentially over-fit the data and perform worse than other models on the test data.

(d) In the function prepare_X, there is a third feature which is always 1. Explain why we need it.

The third feature is unity, and was added to simplify multiplication with the weight vectors. Technically, there is one weight per feature, and then a bias term. By adding the extra feature which is always 1, the math is simplified from $h(x) = \theta(w^T x + b)$ to $h(x) = \theta(w^T x)$.

(e) Visualize the training data from class 1 and 2.

There is a notable difference between the classes using the features we selected. Class '2' clearly has higher intensity and less symmetry on average. Class '1' is the opposite, with high symmetry and low intensity.

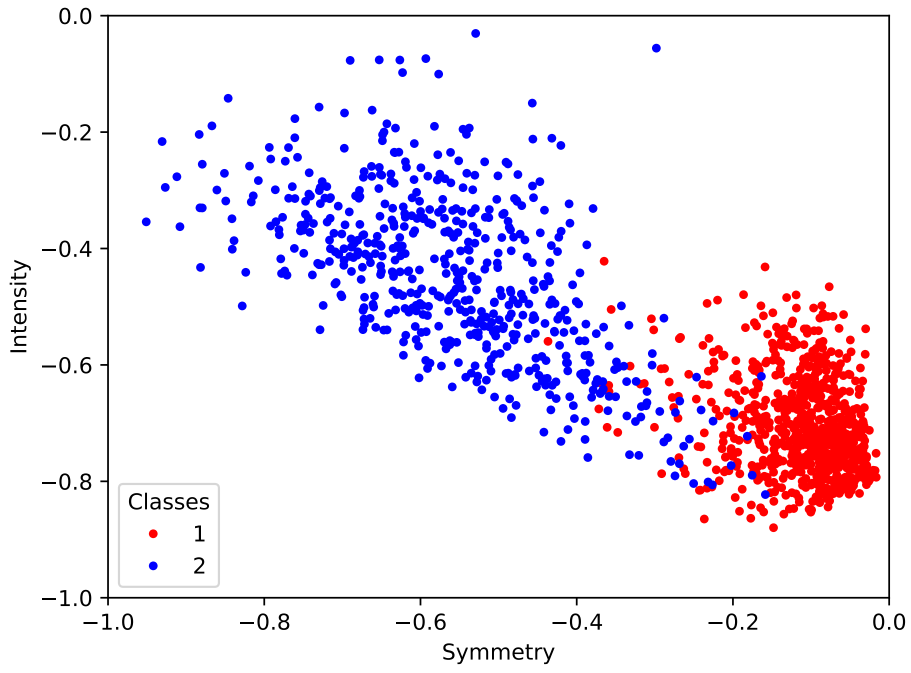

Figure 1: Training data visualized

2 Cross-Entropy Loss

(a) Write the loss function $E(w)$ for one training data sample (x,y).

$$E(w) = \ln(1 + e^{-yw^T x})$$

(b) Compute the gradient $\nabla E(w)$.

$$\begin{aligned}
\nabla E(w) &= \nabla \ln(1 + e^{-yw^T x}) \\
&= \frac{1}{1 + e^{-yw^T x}} \left(\nabla (1 + e^{-yw^T x})\right) \\
&= \frac{1}{1 + e^{-yw^T x}} \left(-yxe^{-yw^T x}\right) \\
&= -y \cdot \frac{e^{-yw^T x}}{1 + e^{-yw^T x}} \cdot x \\
&= -y \cdot \theta(yw^T x) \cdot x
\end{aligned} \tag{1}$$

(c) ...this is not the most efficient way since the decision boundary is linear. Why? Explain it. When will we need to use the sigmoid function in prediction?

It is not the most efficient because the decision point can be uniquely mapped to an equivalent decision point on $w^T x$ alone.

$$\begin{aligned}\text{Predicted class of } x &= \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.5 \\ -1 & \text{if } \theta(w^T x) < 0.5 \end{cases} \\ &= \begin{cases} 1 & \text{if } w^T x \geq 0 \\ -1 & \text{if } w^T x < 0 \end{cases}\end{aligned} \tag{2}$$

Using only $w^T x$ is necessarily faster since computing the sigmoid of the same value takes a nonzero amount of time. We use the sigmoid function in prediction when training the model (and when calculating cross-entropy loss) because it has a nice derivative which makes finding the gradient of the model easier for training.

(d) Is the decision boundary still linear if the prediction rule is changed to the following? Justify briefly.

$$\text{Predicted class of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.9 \\ -1 & \text{if } \theta(w^T x) < 0.9 \end{cases} \tag{3}$$

Yes. This new decision point of $\theta(w^T x) = 0.9$ still uniquely maps to a (different) value of $w^T x$.

(e) In light of your answers to the above two questions, what is the essential property of logistic regression that results in the linear decision boundary?

The decision boundary is linear because the sigmoid function is a monotonically increasing function of the input. So, for any decision point such as $\theta(w^T x) = 0.5$, there is an equivalent decision point using only $w^T x$. Since a decision boundary can be equivalently transferred to a threshold on $w^T x$, and $w^T x$ is linear, the decision boundary is also linear.

3 Sigmoid Logistic Regression

(d) Visualize the results of sigmoid logistic regression.

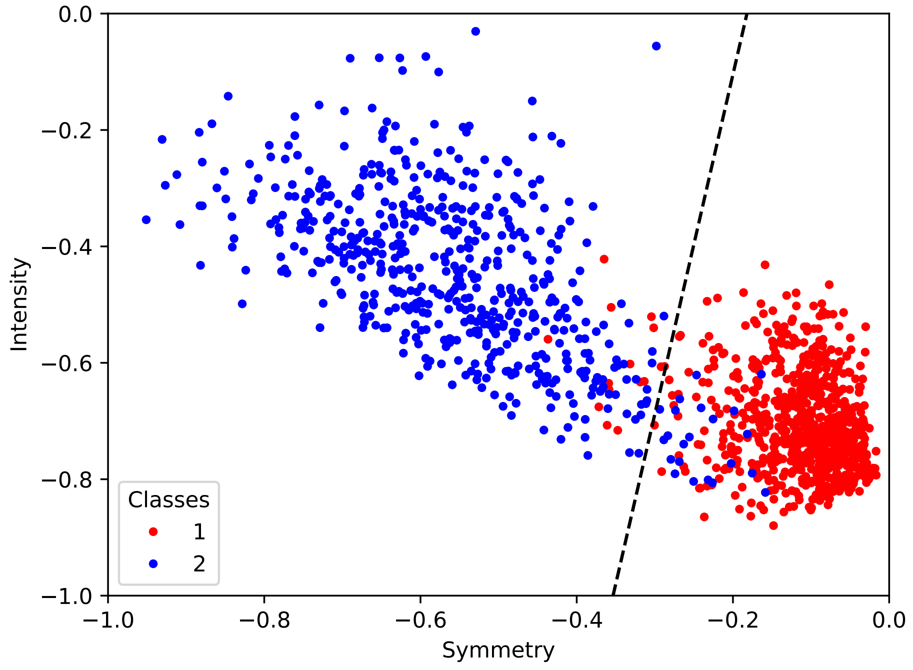

Figure 2: Sigmoid Logistic Regression Result. The decision boundaries are shown as a dashed black lines.

(e) Report the test accuracy of your best sigmoid logistic regression model.

On the test data, the model has an accuracy of 93.93%. More analysis is given in the results section.

4 Softmax Logistic Regression

(d) Visualize the results of softmax logistic regression.

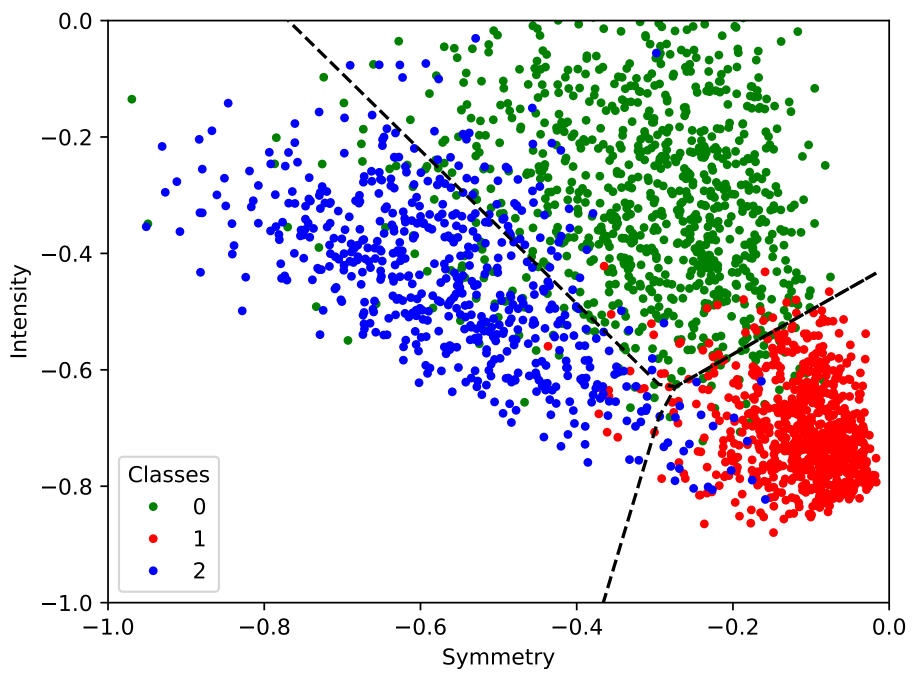

Figure 3: Softmax logistic regression result. The decision boundary is shown as a dashed black line.

(e) Report the test accuracy of your best softmax logistic regression model.

The accuracy of the best model is 86.6% on the test data. More analysis is given in the results section.

5 Softmax Logistic vs Sigmoid Logistic

(a) Train the softmax logistic classifier and the sigmoid logistic classifier using the same data until convergence. Compare these two classifiers and report your observations and insights.

The classifiers have the same accuracy after convergence. The two models were each trained for 10,000 iterations to ensure convergence. The models both score 93.07% on the test dataset, and the weights are:

Sigmoid	[9.30235953 29.04619607 1.24630182]
Softmax	[[-4.65117976 -14.52309804 -0.62315091] [4.65117976 14.52309804 0.62315091]]

Table 1: Model weights

(b) How can we set the learning rates so that $w_1 - w_2 = w$ holds for all training steps?

By setting the learning rate of the sigmoid regression to double that of the softmax regression, the two maintain a relationship such that the difference in the softmax weights is equal to the sigmoid weights. This is shown in the lecture notes and empirically using by training the two using the same training data for the same number of iterations. From experimentation, a learning rate ratio of 2 is needed.

II Analysis and Results of the Programming Part

All necessary portions of the code were implemented and appear to be working properly. The output of the gradient descent

1 Sigomoid Logistic Regression

Batch gradient descent seemed to give the best results. The best model used a learning rate of 0.1, with 1000 max iterations, and a batch size of 10. Batches of 10 seemed to give rapid progress but maintained stability compared to SGD for the same number of iterations. Training accuracy was 97.18% and validation accuracy was 97.62%, so there wasn't an indication of over-fitting during training. The test accuracy ended up being considerably lower than the validation accuracy which suggests a need for a more representative training dataset.

learn_rate	max_iter	batch size	Weights	Test Accuracy
0.1	1000	10	[3.94984525, 21.69510653, -3.71881575]	93.93%

Table 2: Binary sigmoid regression results

2 Softmax Multiclass Logistic Regression

The multi-class regression had difficulty achieving 89% accuracy on the training set. The best model was trained using a learning rate of 0.1, maximum of 1000 iterations, and a batch size of 10. Again, the small batchsize increased the convergence rate while maintaining stability by averaging over the batch.

This model achieved an accuracy of 89.6% on the training data, 87.14% on the validation data. Since the training accuracy is higher than the validation accuracy, there is a sign of over-fitting which indeed was present when the model was evaluated on the test data.

Plenty of tweaking resulted in similar over-fitting, so the result presented could potentially be improved with early stopping.

learn_rate	max_iter	batch size	Weights	Test Accuracy
0.1	1000	10	[[6.71712878, 0.52647702, 10.29469115] [-1.20732903,14.8411791, -8.52388117] [-5.50979975, -15.36765611, -1.77080998]]	86.60%

Table 3: multi-class softmax regression results