Tuan Tran
A20357888
CS584
Summer 2020

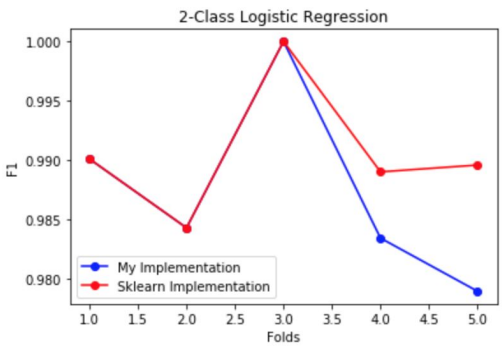
A. Implementation details:

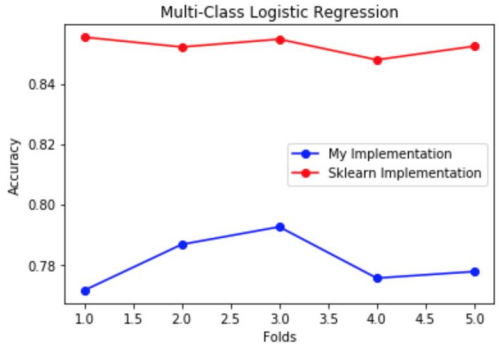
The program is divided into 4 total relevant files. There are three notebooks, each notebook represents one section in the implementation assignment. The content of each notebook has been organized such that it follows each of the instructions (part a, b, c, …) as stated in the prompt. Each cell in each notebook also has been fully documented and should run fine on any machine with the appropriate python modules. To use the program and check for correctness of the results stated, simply go into each notebook and run each of the cells. For notebook “Part 1” and “Part 2” specifically, please download the data following the links from the data folder, and place the data files in the same directory as the notebooks.

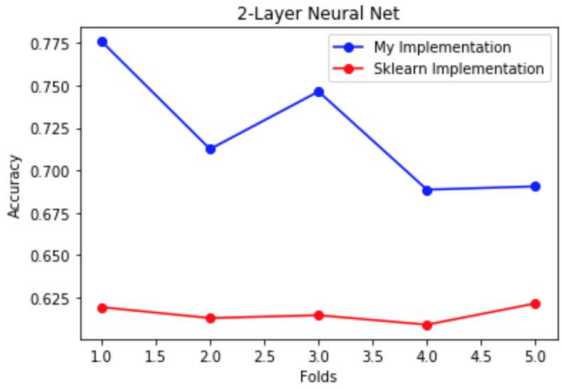
The implementation.py file contains the models implementations and helper functions. Inside are the functions for initializing weights, retaining subsets of classes for a dataset, and getting kfold results for Neural Net model, as well as three different models: 2-Class Logistic Regression, Multi-Class Logistic Regression, and 2-Layer Neural Net. Each of the models has 3 main functions which are fit(), predict() and predict_proba(). For all 3, the 0th feature (with value 1) is included into the input feature matrix. For the implementation of Multi-class models, we require that the labels be one hot encoded before feeding into the models, since we rely on one-hot-encoding representations to mimic the characteristics of the indicator functions when calculating gradients and loss.

B. Result and Discussion:

1. Correctness of Implemented Algorithms:
   In order to compare the implementations vs sklearn counterparts, we use 5-fold cross validation where in each fold we train, predict and evaluate both our models and the models from sklearn. For the scikit-learn models, we tried to set the hyperparameters as close to our implementation as possible in order to achieve a fair comparison. For example, for Logistic Regression, the number of training iterations or learning rates are set so that both our implementation and scikit-learn have the same values. We then obtain the following results:







We observed that in the case of 2-class, we obtained almost similar performance as the scikit-learn implementation. The very slight difference can come from many factors, such as weight initialization techniques (our implementation uses Xavier initialization).

In the case of Multi-class Logistic Regression however, scikit-learn implementation beats ours by 6-10%. We hypothesized that this is because multi-class is a much more complicated problem and our implementation is a very basic, vanilla version since it only allows varying the epochs and learning rate, while scikit-learn implementation allows for a much more variety of hyperparameters to tune.

For the 2-layer Neural Net model, our implementation actually beats scikit-learn by around the same margin as Multi-class LR. We hypothesized that this is because the scikit-learn model uses Stochastic Gradient Descent whereas our implementation uses Ordinary Gradient Descent, which is more accurate than SGD.

Thus, via the results collected, we are confident that our implementation is correct.

2. Part 1:
   For part 1, we choose the banknote authentication problem for 2-class classification and the fashion mnist problem for multi-class classification. Both datasets are image data. The banknote dataset features are extracted by applying different Wavelet Transforms on 400x400 images and the goal is to identify if the image is a banknote or not. The

fashion mnist data are 28x28 grayscale images of fashion garments (for example shirts, trousers, etc) and the features are raw pixels, so 784 features in total per instance. The goal is to classify the correct clothing piece in the image.

In addition to fitting a Logistic Regression model on the original features of the banknote data, we also experimented with mapping the features to higher dimensional space with degree 2 and 3 polynomials in order to increase the model complexity. Using a learning rate of 1e-4 and training for 30 epochs, we use 5-fold cross validation and we obtain the following results:

```
Performing 5-fold cross validation across 3 different models:
Logistic Regression with original features:
        Average training f1 score: 0.8590160986821302
        Average testing f1 score: 0.8277957926131062
Logistic Regression with Degree 2 polynomial mapping of features:
        Average training f1 score: 0.8944071472551494
        Average testing f1 score: 0.9115824172132593
Logistic Regression with Degree 3 polynomial mapping of features:
        Average training f1 score: 0.9577456723177298
        Average testing f1 score: 0.9613551343884282
```

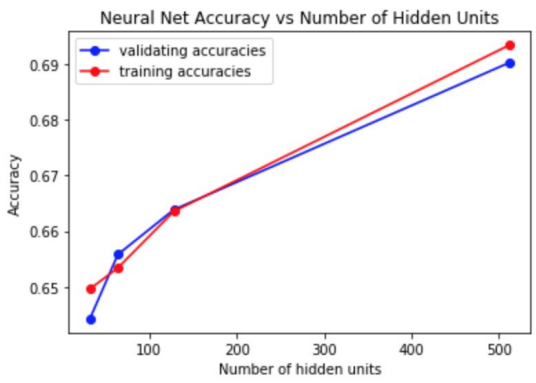
The result is intuitive as more complex models gave better performance. A bit surprisingly, the model with degree 3 polynomial mapping has both very high training and testing F1 score, indicating that it learns the data very well while not overfitting at all.

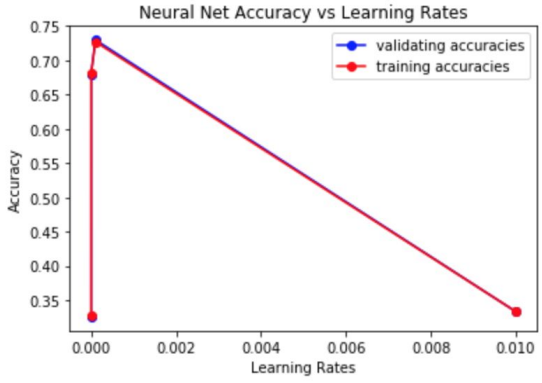
For the fashion mnist dataset, the 28x28 images have been vectorized to 784 dimensional vectors in order to make them suitable for training Logistic Regression. In addition to vectorization, we also had to normalize the pixel values which is **critical** to correct training of our Logistic Regression implementation. The reason being without normalization, the range of pixel values are large enough that it will lead to a softmax result where some of the K values may be **extremely small**. Taking the log of such small values leads to negative infinity and thus leads to a NaN loss value. As a result, we need to normalize the pixels in order for the loss to attain real values and the model to start learning.

This dataset is more complicated than the banknote data since it has 5 times more classes and the features (raw pixels) are not as fine-grained as the banknote data. Using a learning rate of 0.0001 and training the model for 200 epochs, we obtain an average training and testing accuracy of 75% with 5-fold cross validation. Training the final model using the same config, we obtain a 77% accuracy on the final test set. The mediocre results make sense, since Logistic Regression is not exactly suitable for image data. The reason being we need to vectorize the images to feed data into Logistic Regression, and by vectorizing the data we are essentially collapsing and destroying potentially very important spatial features.

3. Part 2:
   In part 2, we use the same fashion mnist dataset, but we choose the subset of 3 classes only - class “t-shirt”, “coat” and “shirt”. We choose these 3 classes since they are all upper body clothing pieces and thus it will be interesting to see if the model can learn to differentiate between them.

   We first examine how the model will perform under different numbers of hidden units (32, 64, 128 and 512) for the hidden layer and under different learning rates (1e-8, 1e-6, 1e-4 and 1e-2). For the varying number of units experiment, we set the learning rate at 1e-6 and for varying the learning rate experiment, we set the number of hidden units at 128. We then trained for 100 epochs:





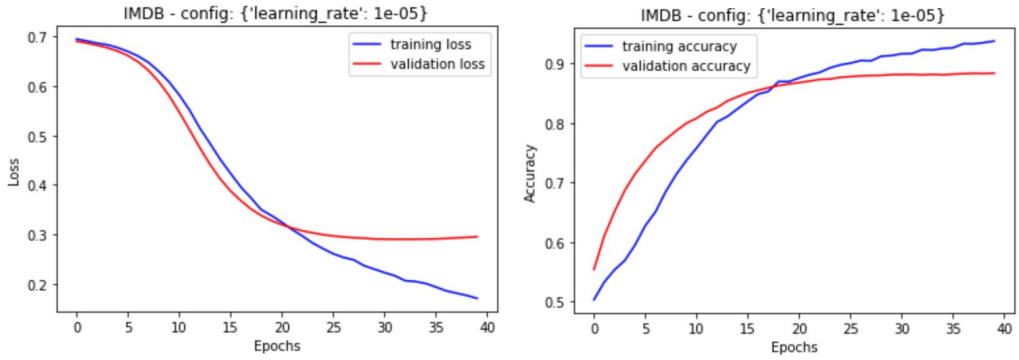
   For the first graph, we observe that we get better performance with a bigger number of hidden units. This is reasonable because more hidden units lead to a more complex model with more parameters, thus leading to a better performance. In the second graph, we can see that the performance peaks for learning rate = 1e-4 but dips very low at 1e-8 and 1e-2. We hypothesize that for learning rate = 1e-2, the learning rate is too high and thus the optimizer seems to overshoot the minima, leading to very poor performance. For learning rate = 1e-8, it seems like the learning rate value simply is too low, thus we may need to train for 200 or even 300 epochs for it to reach the same performance.

4. Part 3:
   For the imdb dataset, we initially built a 3-layer Neural Net with 128 as well as 512 units each, 1e-5 learning rate. We trained the models for 30 epochs and observed that the 128-unit model overfitted at 10th epoch while the 512-unit model overfitted much more quickly at the 5th epoch. For the 512-unit model, overfitting also becomes more pronounced as training accuracy was at ~100% but validation accuracy is stuck at ~86%. Moreover, we observed that the validation loss even started to increase at around 20th epoch.

Because of this behavior, we included dropout into the 512-unit model and increased the number of epochs to 40. We found that dropout at 40% rate was very effective in preventing overfitting for our problem. Overfit only started at around 17th epochs and we observed a higher validation accuracy of 89% with training accuracy standing at 93%.

We then make the model even more complex to see how it behaves by increasing from 3-layer to a 5-layer model, with everything kept the same, and then a 8-layer, 1024-unit model trained for 10 epochs. Both of these complex models yielded a similar result to the 512-unit model with dropout. Thus we decided to choose the 512-unit model with dropout as the final model since it is not worth going the extra mile of adding that much more complexity, which added a lot of extra parameters with no visible performance gain. We also tried changing activation function and optimizer but found that reLU hidden activation, softmax output activation and Adam optimizer yielded the best result.

We obtained the following validation graphs from the 512-unit model with dropout:



Training the final model on the train set and evaluate on the entire test set, we obtained the following full report:

	precision	recall	f1-score	support
0	0.89	0.88	0.89	12500
1	0.88	0.89	0.89	12500
accuracy			0.89	25000
macro avg	0.89	0.89	0.89	25000
weighted avg	0.89	0.89	0.89	25000

F1 score: 0.886958401863102