CS429/529 - Project 2: Logistic Regression

Names: Kaneez Fatima, Sathvik Quadros

1 Effect of Learning Rate/Number of Epochs on Accuracy of Logistic Regression

All testing was conducted using the entire training set for training the model, and then determining the accuracy using the Kaggle score to due having a limited training set, with the regularization parameter (λ) as 0.01. As there may be correlation between the learning rate and the number of epochs, we tested various learning rate values along with changes in the number of epochs to see what gave us the best accuracy. We did not continue testing higher numbers of epochs if the same level of accuracy was reached after increasing the number of epochs for the same learning rate, as that showed that further increases would have no quantifiable effect on the accuracy.

Epochs / LR	10k	50k	100k	250k	500k
0.01	0.55	0.64	0.66	0.67	0.68
0.05	0.64	0.67	0.68	0.68	0.66
0.1	0.66	0.68	0.68	0.66	0.66
0.8	0.68	0.66	0.66	0.66	skipped
2.0	0.67	0.66	0.66	skipped	skipped
10.0	0.62	0.64	0.65	skipped	skipped

Things to note about the above table are:

- We went up to 10 on the learning rate as we wanted to find the optimal learning rate which would allow us to have to go through the least number of epochs in order to train our model.
- We expected that at some point that we would see some sort of obvious decrease in accuracy, but we did not expect that we would have to go up to 10 to see this.
- Additionally, we did not expect that a learning rate of 0.8 would still be viable, even leading to convergence at only 10k epochs.

Out of curiosity, we trained a model with a learning rate of 10 over 2 million epochs, getting an accuracy of 0.65, which is the same as 100k epochs for this learning rate.

2 Effect of Learning Rate/Number of Epochs on Train Time of Logistic Regression

All training was conducted using the entire training set, with the regularization parameter (λ) as 0.01. We did not continue testing higher numbers of epochs if the same level of accuracy was reached after increasing the number of epochs for the same learning rate, as that showed that further increases would have no quantifiable effect on the accuracy, and so by extension we do not have the training time information for those cases. However, as the purpose of collecting the training time is to see which combination of lr and number of epochs will give us the highest accuracy with the least training time, the fact that we don't have those data points is inconsequential. All training times are in seconds, with those that reached the highest accuracy in the previous table colored green.

Epochs / LR	10k	50k	100k	250k	500k
0.01	1.887	10.816	18.455	46.547	98.292
0.05	1.888	9.159	18.715	46.990	92.594
0.1	1.892	9.517	18.851	46.411	93.781
0.8	1.880	9.671	18.628	47.020	skipped
2.0	1.908	9.506	18.774	skipped	skipped
10.0	1.881	9.555	18.854	skipped	skipped

Things to note about the above table are:

- Learning rate has little to no effect on the time it takes to train the model.
- The number of epochs, however, is directly proportional to the amount of time it takes to train the model.
- According to my estimates, the average amount of time it takes to complete 1 epoch is just $1.9 * 10^{-4}$ of a second.

The time it took to run the 10 learning rate 2 million epoch model was 375.776 seconds.

3 Choice of Learning Rate and Number of Epochs

As we were able to get surprisingly accurate results with a learning rate of 0.8 at just 10k epochs, making it the fastest model we tested by far which met the maximum accuracy cap, we went ahead and decided to use that as a baseline and then go from there. After a bit of experimentation, we decided to go with a learning rate of 0.8 and 5k epochs, giving us an accuracy of 0.68 (which is still the maximum accuracy) and having a training time of only 0.951 seconds.

4 Comparison of Logistic Regression with Other Models

As all the models are different, the only metrics they can really be universally evaluated against are train time and accuracy. Due to time constraints we were unable to do the level of optimization for each of these additional models that we did for logistic regression, but we've put the hyperparameters we used for these models below.

- Logistic Regression: We used the optimized model we decided on in "Choice of Learning Rate and Number of Epochs" here.
- Random Forest: We used sklearn's RandomForestClassifier with 1000 trees.
- Gaussian Naive Bayes: We used sklearn's GaussianNB model, which does not take any unique hyperparameters.
- SVM: We used sklearn's SVM with a RBF kernel, as that performed the best out of all of them.

Model	Train Time	Accuracy
Logistic Regression	0.951	0.68
Random Forest	2.876	0.60
Gaussian Naive Bayes	0.00236	0.47
SVM	1.708	0.67

From the above table we can clearly see that for the specific features that we extracted from our dataset, our Logistic Regression model out performs all our other selections.

5 Dealing with Misclassified Samples

The most straightforward method to identify misclassified samples is to simply listen to parts of each of the samples, and then either reclassify or remove the samples. This method has many advantages for a small dataset like ours, one which contains samples of music, as music is something which is highly human centric. Unfortunately, due to the fact that neither of us listen to many of these genres of music (I only know what reggae is because I looked it up and I still can't tell the difference between rock and metal), we had to come up with alternative ways of detecting misclassified instances.

In order to deal with misclassified samples, we tried to use a 3D PCA visualization of the features for all of the samples of each of the classes. Unfortunately, the captured variance was relatively low (0.52-0.58).

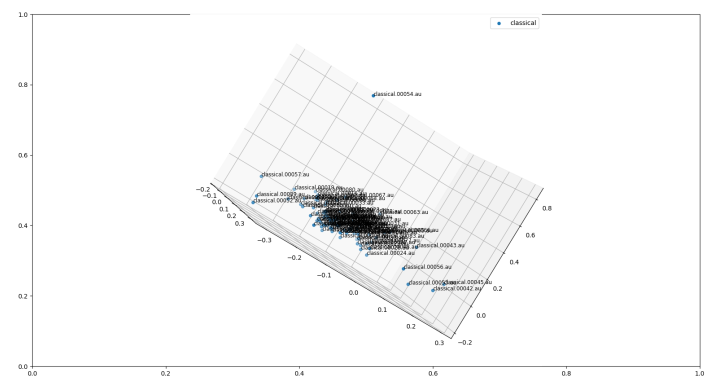

Figure 1: Example of 3D PCA used

While we were able to find some outliers this way, we may have ended up making a mistake. Here are the ones we believe were actually misclassified after seeing that removing the others reduced accuracy:

- rock.00063
- classical.00054

Doing this bumped our accuracy up from the 0.68 we had previously to 0.69.

6 Choosing Features from Samples

A wide variety of features were chosen for the purpose of training the AI model. Many were taken directly from the recommendations given in the google document, while others, like tempo and beat count, seemed generally important (e.g., tempo is a large part of determining the genre of a musical piece). A few other features were chosen simply due to the ease in which they could be implemented, as we are always able to remove them without much effort if they are shown to be problematic.

7 Optimizations

The only significant optimizations we did were finding a high learning rate that allowed us to significantly reduce the number of epochs, as shown in the previous sections, and use multithreading in feature extraction, as there weren't many other places where multithreading would be implemented.

Using multithreading in feature extraction reduced the amount of time it took to collect the features by a significant amount, as shown in the table below (all time in seconds, on a 16 core 32 thread system):

Threads	1	4	16	32
Extract Time	4716.342	638.994	316.360	288.186

8 Other Features/Things to Note

A variety of other features have also been implemented for this project:

- The logistic regression model has been written in an incredibly generic way, allowing for each reuse in other situations.
- The code for training and evaluating has been split into two functions, allowing anyone to reuse them in any other situation, or simply evaluate without retraining.
- Another implementation of gradient descent with unit tests is available in gradient_descent.py due to us accidentally writing all the code for this twice as it was relatively simple.
- the pickle library has been used for easy saving and loading of models.