FYS-2021-Assignment-1

Tor Opdahl
email: top010@uit.no

September 2024

1 Link to Github:

`https://github.com/toropdahl/FYS-2021-MAssignment-1`

2 Problem 1

The end point of this problem, is to extract data from a csv file, and format the data into training and test sets for the machine learning model to use in problem 2.

2.1 Extracting and Filtering

I used the pandas library functions to extract and filter the data from the csv, using the read_csv()([1]) function. Then i filtered the data to only contain samples from the genres "pop" and "classical" music, to then only keep the attributes "liveness", "loudness", "genre". I achieved this through simple index filtering operations on the data-frame.

2.2 Changing Label and Splitting

I then used the .apply() attribute of the data-frame to change the label inside the "genre" field to either 0 or 1 depending on if it is "pop" or "classical", where "pop" entries received the number 1 and "classical" 0.
Then i split the data-frame into 2 parts, one holding the genre label, and one containing the 2 other attributes, "liveness" and "loudness".
All that was left then was to throw these arrays in sklearn's train_test_split() function, to achieve a training set of 80% and test set of 20% of the total number of entries in the data([2]).

3 Problem 2

3.1 Implementing the Sigmoid Function

The Sigmoid function i implemented used the following formula:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where:

$$z = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

Here, the w's in the formula represents the different weights, the x'es represent the different attributes and b the bias. The Sigmoid function i implemented in python calculated the probability for 1 sample, and returns this probability.

3.2 Cost Calculation Function

To calculate the cost of a single prediction, I used the following formula:

$$\text{Cost}(y_i, \hat{y}) = \begin{cases} -\log(\hat{y}) & \text{if } y_i = 1 \\ -\log(1 - \hat{y}) & \text{if } y_i = 0 \end{cases}$$

This formula represents the binary cross-entropy loss, which is well-suited for binary classification problems. It measures the difference between the predicted probability ($\hat{y}$) and the actual label (y_i). The farther away the model is from the correct result, the higher the cost.

3.3 Stochastic gradient Descent

I implemented the following formulas to update the weights and bias of the model ([3]):

$$w_1 = w_1 - \alpha \cdot (y_i - \hat{y}) \cdot x_1$$

$$w_2 = w_2 - \alpha \cdot (y_i - \hat{y}) \cdot x_2$$

$$b = b - \alpha \cdot (y_i - \hat{y})$$

Here, it updates the weights by calculating the error of the prediction, multiplying it with the corresponding attribute and the learning rate, to then subtract this value from the current value of the weight. The Bias is updated by just multiplying the error with the learning rate, and then subtracting this value from the bias.

3.4 Training Loop

The implementation uses a training loop that represent 1 epoch in the training, such that the function just needs to be called multiple times to achieve multiple epochs.
The training loop first extracts the attributes from the training data set and then makes a prediction using the Sigmoid function, then it uses the prediction to update the weights an bias of the model, to then calculate the cost of this sample, adding the cost of the sample to a variable holding the total cost of the epoch.
Finally, it calculates the average cost of each sample in the epoch, appending this value to a list. This list will later be used to plot the training error.

3.5 Shuffling and Training

To achieve multiple epochs, the model shuffles the training data before starting a new epoch. Meaning each epoch uses the whole training data set, only a different permutation of it. This makes it so the model can easily experiment with different numbers of epochs in its training.

3.6 Test Loop

The test loop goes though the training dataset. On each sample, it uses the Sigmoid function calculate a probability, and then calculates the cost of this sample to add to the total cost of the test. It then uses the probability to produce a prediction. It then appends the predicted label to one list, and the true label to another.
When the model has run though each sample, it then uses the different lists holding the predicted label and actual label to calculate the accuracy of the test, and prints the result along with the confusion matrix.

3.7 Plotting the Training Error

This part of the implementation uses the list of average cost each epoch had during the training of the model, and plots it using matplotlibs pyplot library functions. It then displays the result in form of a graph.

3.8 Results

In the end, the model consistently produces an accuracy between 92 and 93%. This is though experimenting with different learning rates and epochs, where the model performs the most consistently and best with a learning rate of **0.001** and **20** as the number of epochs. When experimenting with the learning rates, it becomes clear that the model converges with fewer epochs when the learning rate is high, and slower when the learning rate is low.
In testing of the model, the main metric used to see the accuracy between the training and testing of the model is the cost each of them produce. Seeing that

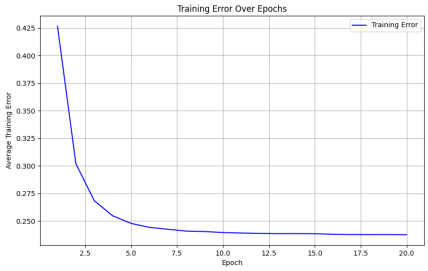

Figure 1: Graph of Training Error

the average cost of the last epoch is close to the average cost of the training data, shows that the model is as accurate on the training dataset and the testing dataset.

4 Problem 3

4.1 Confusion Matrix

This is the confusion matrix produced on the last test run on the model:

	Predicted Negative (0)	**Predicted Positive (1)**
Actual Negative (0)	1640	211
Actual Positive (1)	74	1804

Table 1: Confusion Matrix

The confusion matrix tells me that the model is more accurate at predicting classical songs than it is pop songs, as it has more correct predictions and fewer false positives. This either means that it is generally more difficult to classify pop songs, or simply that the model is more precise when it comes to classifying these types of songs.

References

[1] Pandas Contributors. pandas documentation. https://pandas.pydata.org/docs/, 2024. [Online; accessed 8-September-2024].

[2] SkLearn Contributors. Documentation of scikit-learn 0.21.3. https://scikit-learn.org/0.21/documentation.html, 2024. [Online; accessed 8-September-2024].

[3] Wikipedia Contributors. Stochastic gradient descent. https://en.wikipedia.org/wiki/Stochastic_gradient_descent, 2024. [Online; accessed 8-September-2024].