FYS-2021 Machine Learning

Mandatory Assignment 1
Logistic Regression

Name: Christian Lund
Student ID: 587333
GitHub: https://github.com/lundchristian/ml_log_reg

UiT - The Arctic University of Norway
September 3, 2024

Contents

1 Problem 1

1.1 Problem 1a

In order to load the spotify dataset from the `.csv` file, I am using the `read_csv()` function in the *pandas* library.

As the dataset is a matrix, the number of samples (songs), are found by counting the number of rows, whilst the number of features are found by counting the number of columns.

- The number of songs are found to be 232725
- The number of features are found to be 18

1.2 Problem 1b

The dataset is rather large, and we only want to work with samples where the genre is pop or classical, see line 1 in code listing 1.

The *genre* feature will function as the label vector, therefore it must be converted to binary values, where pop is 1 and classical is 0, see line 2 in code listing 1.

Still, the dataset is large, and we only need the two features *liveness* and *loudness*, as well as the *genre* of course. These three columns are extracted from the dataset as seen in line 5 in code listing 1.

```
1 dataset = dataset[dataset["genre"].isin(["Pop", "Classical"])]
2 dataset["genre"] = dataset["genre"].map({"Pop": 1, "Classical": 0})
3 pop_samples: int = dataset[dataset["genre"] == 1].shape[0]
4 classical_samples: int = dataset[dataset["genre"] == 0].shape[0]
5 dataset = dataset.loc[:, ["genre", "liveness", "loudness"]]
```

Listing 1: Remove all samples where 'genre' is not equal to 'Pop' or 'Classical'

The number of samples for each class in the labels vector are found by sorting the vector for each binary class value and counting, see line 3 and 4 in code listing 1.

- The number of pop genre samples are found to be 9386
- The number of classical genre samples are found to be 9256

1.3 Problem 1c

Figure 1 shows a sketched plan for how we must reduce the dataset correctly into a training set and a testing set.

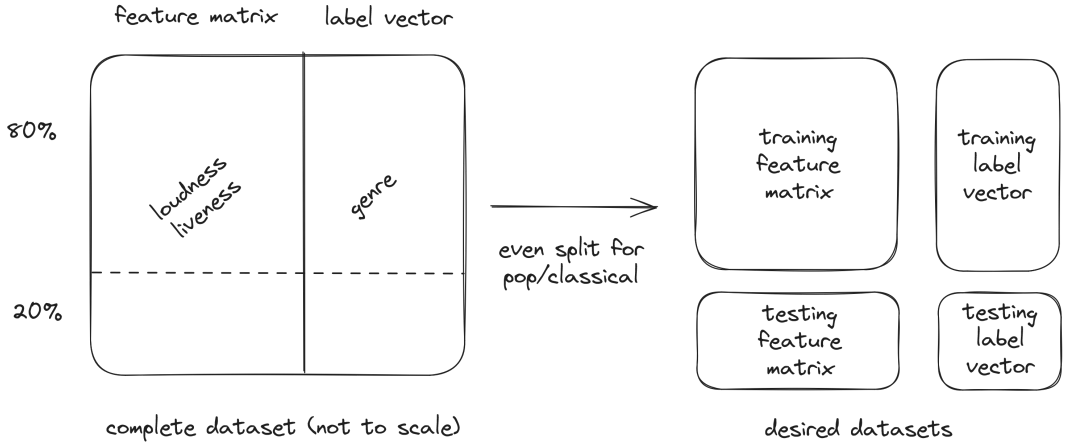

Figure 1: Sketch of splitting the dataset

As seen in lines 1-13 in code listing 2, I pick the samples for only one class, then sample pseudo-randomly 80% of it for training, and the rest goes to testing. By doing so, I maintain an even distribution of the classes in both the training and the testing set.

Subsequently, lines 15-18 in code listing 2, both classes are merged back together again into two numpy arrays, one for training and one for testing.

The feature matrices and the label vectors may now be extracted from the two sets as previously seen in code listing 1.

```
1  dataset_pop = dataset[dataset["genre"] == 1]
2  training_pop = dataset_pop.sample(
3      frac=0.8,
4      random_state=random.randint(1, 100)
5  )
6  testing_pop = dataset_pop.drop(training_pop.index)
7
8  dataset_classical = dataset[dataset["genre"] == 0]
9  training_classical = dataset_classical.sample(
10     frac=0.8,
11     random_state=random.randint(1, 100)
12 )
13 testing_classical = dataset_classical.drop(training_classical.index)
14
15 train_set = pd.concat([training_pop, training_classical])
16     .reset_index(drop=True)
17 test_set = pd.concat([testing_pop, testing_classical])
18     .reset_index(drop=True)
```

Listing 2: Split the dataset into a training set, and a testing set

1.4 Problem 1d

Figure 2 shows the data plotted on a *liveness* vs *loudness* plane, for visual ease the z-index is different between the two plots.

From just eyeballing the plot, about 85% of the classical music do not overlap with the pop music. However, the pop music is cluttered with samples from of the classical genre.

If we removed the obviously overlapping samples from the classical genre, then classification would be an easy task, as a straight line could be drawn intercepting the y axis at about -12.

This is sadly not the case, and the model will classify a lot of classical music as pop, leading to a good amount of false-positives.

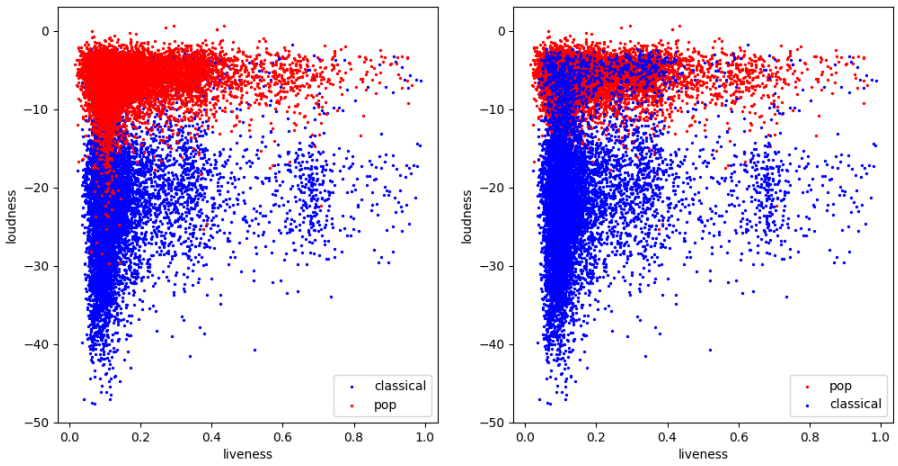

Figure 2: Plot of *liveness* vs *loudness*

2 Problem 2

2.1 Problem 2a

Before explaining the logistic discrimination classifier, I want to specify how I used the stochastic gradient descent. The gradient descent is used for finding the minimum of the loss function, but how often the parameters are updated depends on the batch size.

If the batch size is the entire dataset, then the parameters are updated once every epoch. And if the batch size is one sample, then the parameters are updated once for every sample in the dataset, and every sample in the dataset is looped once for every epoch. Then there is every combination in between, i.e. a batch size less than the number of samples, but greater than one[1].

Furthermore, the samples may be shuffled every epoch, perhaps aiding in the learning process as hinted to by the task description.

For this task I've chosen to randomly shuffle the training set every epoch, and I've set a batch size of one.

The model is trained by setting the weights and bias to zero, then every epoch the training set is pseudo-randomly shuffled. For every sample in the dataset, the *log odds* are calculated[2].

$$z = b + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Then the log odds are fed into the *sigmoid* function in order to gain a probability.

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

The gradient descent for the weights and the bias is calculated using the derived entropy loss function with respect to the weights, and with respect to the bias, respectively.

$$\nabla_{\mathbf{w}} \mathcal{L} = \mathbf{x}^T \cdot (\hat{y} - y) \quad \nabla_{\mathbf{b}} \mathcal{L} = (\hat{y} - y)$$

Loss, using the entropy loss function, is accumulated for every sample, and the mean is calculated at the end of the epoch in order to avoid an extra loop.

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \cdot \log(\hat{y}^{(i)}) + \left(1 - y^{(i)}\right) \cdot \log\left(1 - \hat{y}^{(i)}\right) \right)$$

Lastly, the accuracy is calculated with the final weights and bias with the overall accuracy formula.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

By experimenting with the three learning rates of 0.1, 0.01 and 0.001 and with 100 epochs, I found the results laid out in table 1. Based on these samples, the final one seemed like a good fit, even setting a learning rate of 0.0001 and running 500 epochs did not change the overall accuracy score.

LR	**0.1**	**0.01**	**0.001**
Accuracy	0.87	0.92	0.93
Weights	-6.44, 1.04	-1.83, 0.52	-1.29, 0.44
Bias	19.08	5.90	5.39

Table 1: Comparison of model performance with different learning rates (LR)

The plot in figure 3 shows the loss per epoch with a learning rate of 0.001.

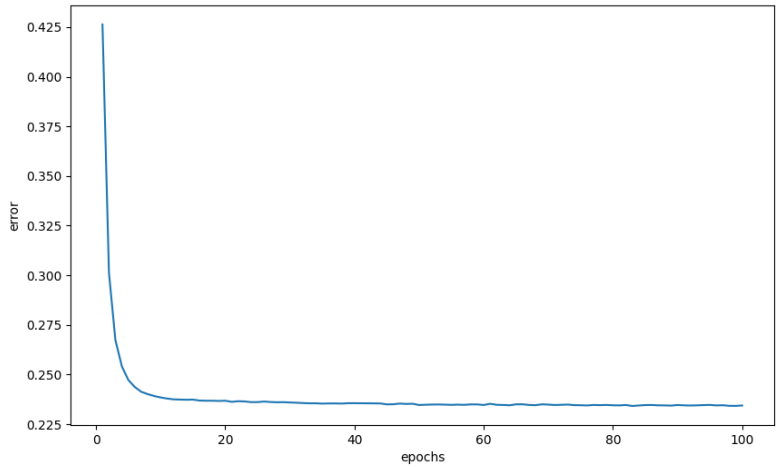

Figure 3: Plot showing the loss per epoch

2.2 Problem 2b

The difference in overall accuracy between the training set and the testing set is negligible, indicating that the model performs well on both sets and is able to generalize.

The threshold for what counts as a positive sample is in this case set to inclusive 0.5.

- Overall accuracy on training set: 0.925
- Overall accuracy on testing set: 0.926
- Difference in overall accuracy: 0.001

2.3 Problem 2c

Figure 4 shows the linear line drawn using the learned parameters. As assumed in problem 1d, a large portion of the classical music will be classified as pop. However, it would be difficult to improve the model without either filtering the dataset or by experimenting with different features.

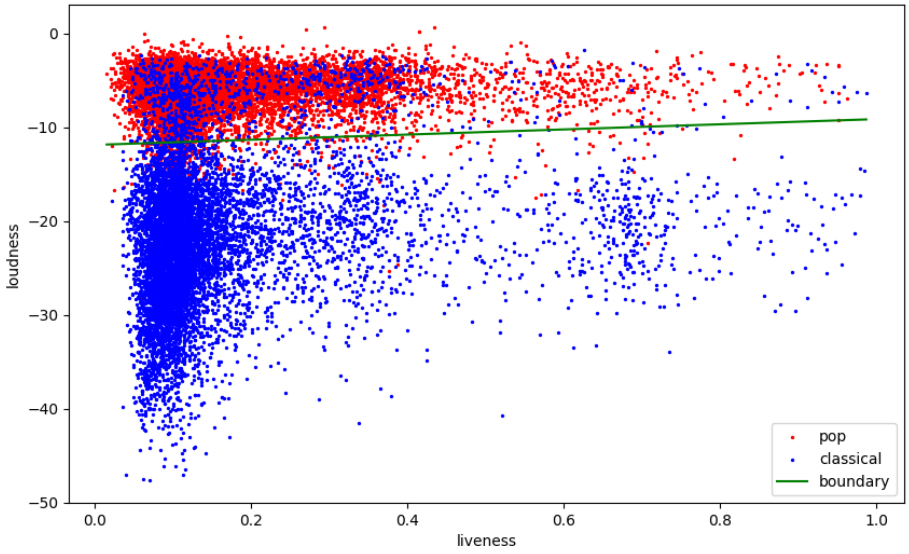

Figure 4: Plot of the models prediction boundary

3 Problem 3

3.1 Problem 3a

The confusion matrix is created simply by running through all of the labels, and testing different equality combinations between the true and predicted value, an excerpt is shown in code listing 3

```
if    y[i] == 1 and y_hat[i] == 1:
    TP += 1
elif y[i] == 0 and y_hat[i] == 1:
    FP += 1
elif y[i] == 0 and y_hat[i] == 0:
    TN += 1
elif y[i] == 1 and y_hat[i] == 0:
    FN += 1
```

Listing 3: Confusion matrix sorting

After all combinations of positive and negative are counted for, then the matrix itself may be easily constructed by placing the counted combinations in the correct place in the matrix. The result in this case is displayed in figure 5, disregard the tick marks as they are inverted.

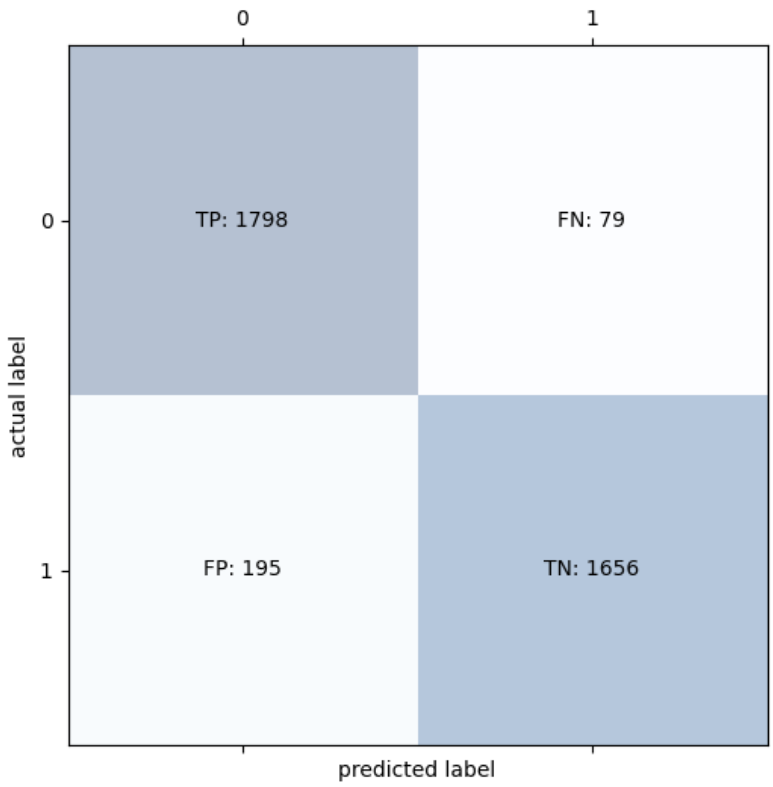

Figure 5: Confusion matrix

Assessing the results, the number of false-positives are doubled that of the false-negatives.

3.2 Problem 3b

In addition to the evaluation metrics already calculated, I've also calculated the precision, recall and F1 scores. Apart from the confusion matrix, the evaluation metrics of the models performance are displayed in table 2.

Metric	Value
Accuracy	0.92
F1 Score	0.92
Precision	0.90
Recall	0.93

Table 2: Model Performance Metrics

As previously mentioned, the number of false-positives are doubled that of the false-negatives. This makes sense considering how much more of the classical samples overlap into the *positive zone*, compared with the other way around. The accuracy score does not capture this information as well as the confusion matrix.

3.3 Problem 3c

Considering which songs that are difficult to classify is a bit difficult without knowledge of how the data was collected, and what the *loudness* and *liveness* represent.

Assuming that loudness refers to the amplitude, and a lower value is a greater amplitude. And assuming that liveness refers to the *tone* of the song in regards to being more or less *lively*. Then I would have no idea, these genres do not stir my musical joy, best of luck.

4 Reading material

It is often not easy to pinpoint an exact source to cite when solving problems like these ones. Therefore I will provide a short list here of heavily used resources.

- The assignment and related dataset[5][8]
- The relevant lecture slide[4][6][7]
- The documentation for the pandas library[3]

List of Figures

Listings

References

[1] Wikipedia contributors. *Stochastic Gradient Descent*. https://en.wikipedia.org/wiki/Stochastic_gradient_descent. Accessed: 2024-09-03. 2024. URL: https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[2] Google Developers. *Logistic Regression - Machine Learning Crash Course*. https://developers.google.com/machine-learning/crash-course/logistic-regression. Accessed: 2024-09-03. 2024. URL: https://developers.google.com/machine-learning/crash-course/logistic-regression.

[3] The Pandas Development Team. *Pandas Documentation*. Accessed: 2024-09-03. 2024. URL: https://pandas.pydata.org/docs/index.html.

[4] UiT. *Evaluating ML Results*. Accessed: 2024-09-03. 2024. URL: canvas/modules/lectureslides/EvaluatingMLresults.pdf.

[5] UiT. *FYS 2021 Assignment1*. Accessed: 2024-09-03. 2024. URL: canvas/modules/mandatoryassignments/FYS_2021_Assignment1.pdf.

[6] UiT. *Linear And Logistic Regression*. Accessed: 2024-09-03. 2024. URL: canvas/modules/lectureslides/Linear_logistic_regression.pdf.

[7] UiT. *Optimization And Gradient Descent*. Accessed: 2024-09-03. 2024. URL: canvas/modules/lectureslides/optimizationandgradientdescent.pdf.

[8] UiT. *Spotify Features*. Accessed: 2024-09-03. 2024. URL: canvas/modules/mandatoryassignments/Spotify%20Features.csv.zip.