Assignment1 - FYS-2021

Lars Moen Storvik

September 2024

Here is the link to my GitHub repository: GitHub Repository.

1

a

I used *dataframe.info()* to find there are 232725 songs(samples) and 18 features in the file.

b

c

Since task b and c overlaps in my code, I join them together in this report. This is a quick rundown of how I ended up with my train and test numpy arrays:

1. Read data from file
2. Add labels
3. Get two data sets from all songs:
   (a) Data set with Pop songs
   (b) Data set with classical songs
4. Split the data sets into two train sets and two test sets (with ratio 80-20)
5. Merge the training and test pop and classical sets. And also shuffle them randomly
6. End with splitting the training and test sets making four numpy arrays:
   (a) training set of liveness and loudness features
   (b) training set of labels
   (c) test set of liveness and loudness features
   (d) test set of labels

```
# Read data file
df = pd.read_csv("SpotifyFeatures.csv")

# 1a)
df.info()

# Add labels - Pop: 1    Classical: 0
df.loc[:, 'label'] = df['genre'].apply(lambda x: 1 if x == 'Pop' else 0)

# "Reduce data frame size" by making two seperate df
samples_classic = df.loc[df['genre'].isin(['Classical'])]
samples_pop = df.loc[df['genre'].isin(['Pop'])]
```

```
13
14
15 def train_and_test_set(df):
16     """ Returns a 80-20% ratio of train and test data """
17     ratio = 0.8
18     total_rows = df.shape[0]
19     train_size = int(total_rows*ratio)
20     train = df[0:train_size]
21     test = df[train_size:]
22     return train, test
23
24 # Get dataframes of train and test data
25 train_class, test_class = train_and_test_set(samples_classic)
26 train_pop, test_pop = train_and_test_set(samples_pop)
27
28 # Merge classical and pop songs (of the training and test sets) and shuffle them (using
       sample(frax=1))
29 train = pd.concat([train_class, train_pop]).sample(frac=1)
30 test = pd.concat([test_class, test_pop]).sample(frac=1)
31
32 # Convert to numpy arrays
33 train_features = train[['liveness', 'loudness']].to_numpy()
34 train_label = train['label'].to_numpy()
35
36 test_features = test[['liveness', 'loudness']].to_numpy()
37 test_label = test['label'].to_numpy()
```

d

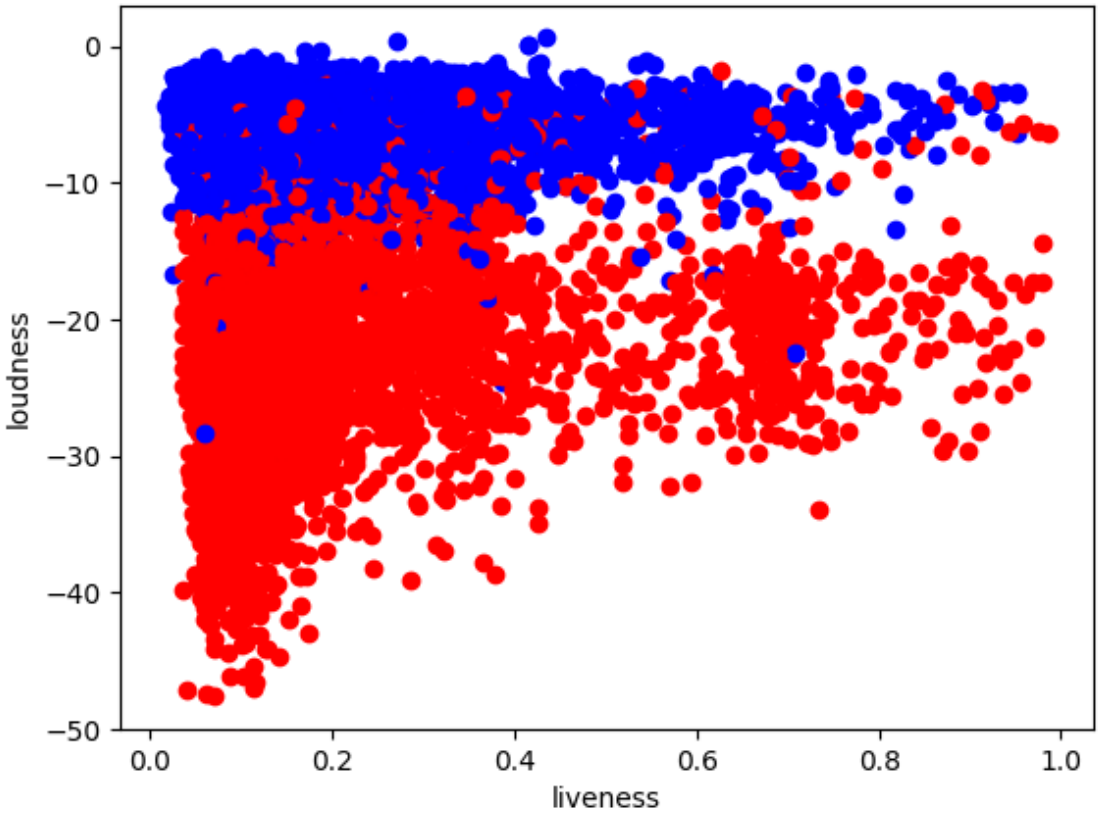

Figure 1: Plot of liveness VS loudness. Pop samples are red and classic samples are blue

As seen in figure 1 The two classes has a okey seperation with some overlap. The liveness variables are similar in all samples, but they are seperated on the loudness variable. I think the classification will be alright, but not perfect considering there are some overlap.

2

a

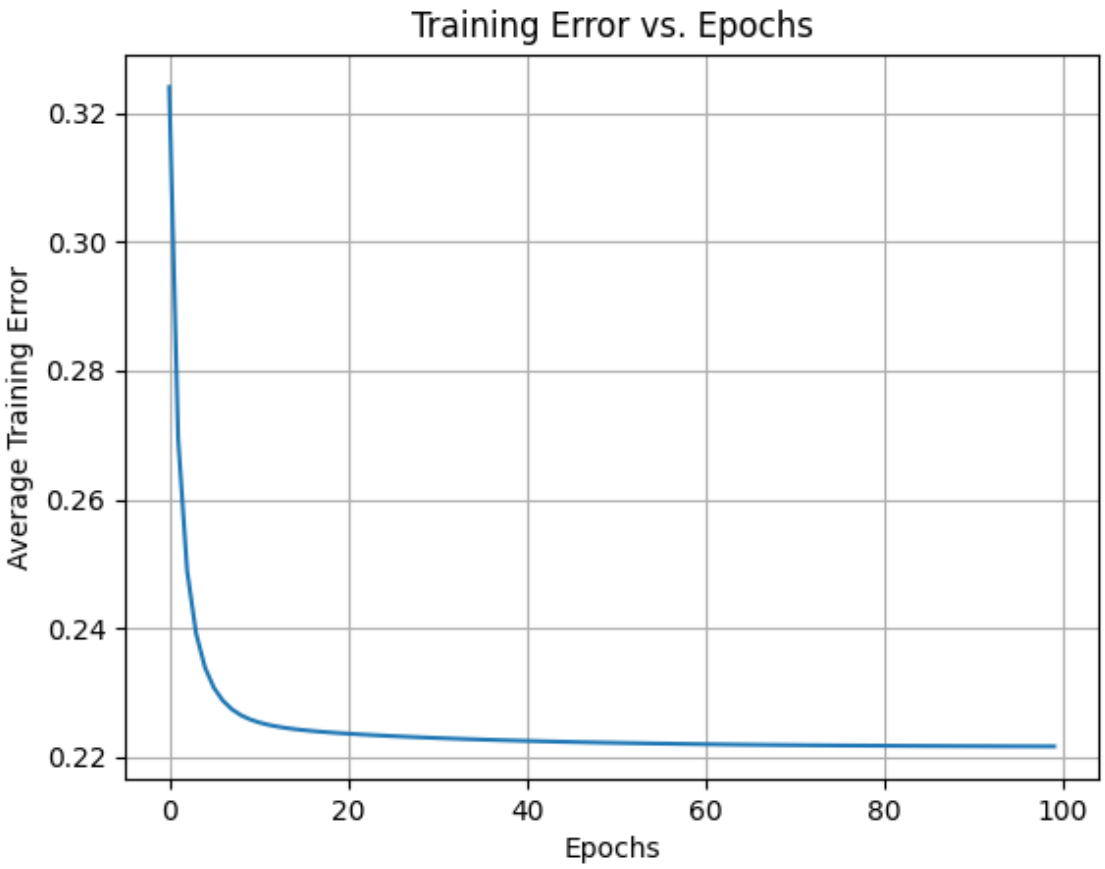

Figure 2: Plot of training error as a function of epochs

As seen in figure 2 the training error decreases for each epoch and converges to a certain point over a number of epochs. If the epochs are increased, the training error will continue to converge. If the learning rate is decreased/increased the training error will converge slower/faster. In the figure above I trained the algorithm with 100 epochs and a learning rate of 0.001.
The training error converges to somewhere around 0.222

b

Accuracy training set: 93.23%
Accuracy for test set: 90.22%
The accuracy for both the training and test set differs a bit each time the program is run: I think this is because the samples are shuffled randomly each time.
There is a small difference between the training and test set, where the training set has a better accuracy. This is because these are the samples the algorithm is trained on.

c

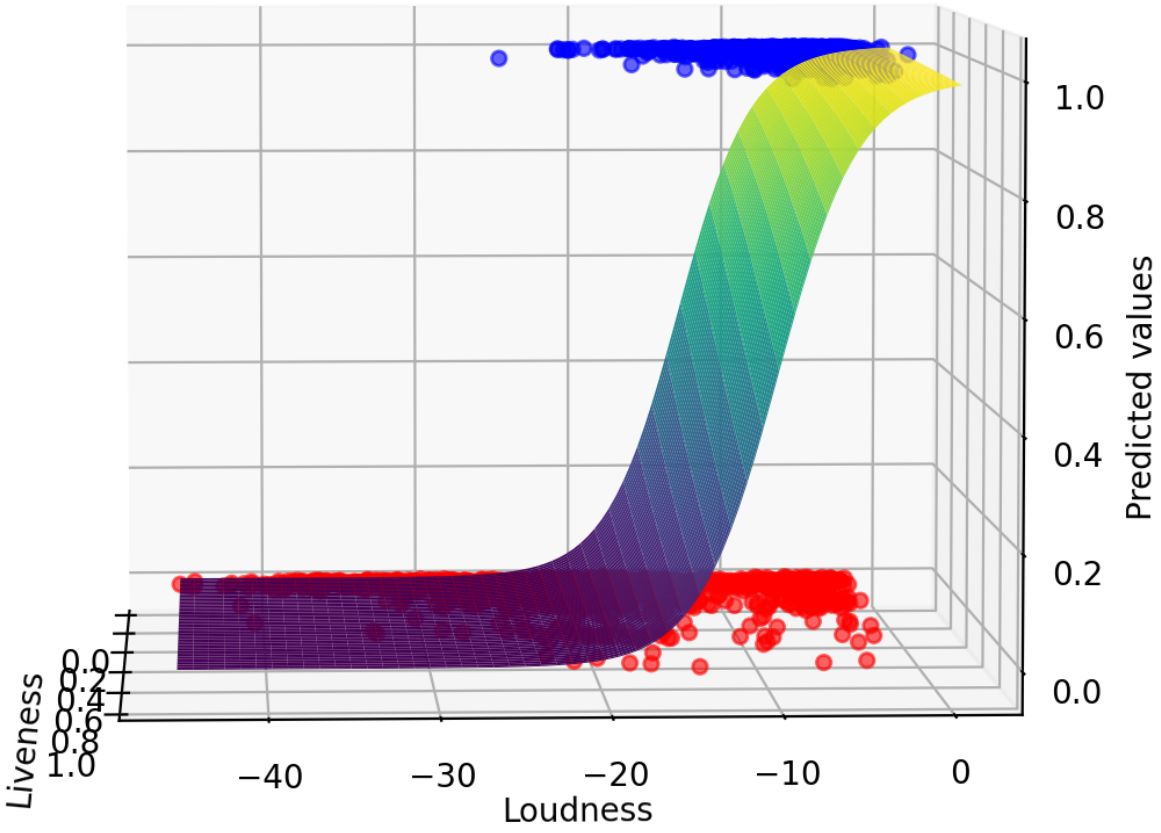

Figure 3: Plot of predicted decision boundery. Pop: red. Classic: blue

As shown in figure **??** the predicted boundery is not perfect,but because there are so much overlap I can't see how it would be much better.

3

a

	Predicted Pop	Predicted Classic
Actial Pop	1802	289
Actual Classic	76	1563

Figure 4: Confusion matrix

b

The confusion matrix shows how well the model performs for each class, while the accuracy does not. From the confusion matrix I can see the precision of the model for each class:

Precision pop:

$$\frac{1802}{1802+76} = 96,0\%$$

Precision Classic

$$\frac{1563}{1563+289} = 84,4\%$$

Of the predicted pop and classic values, a lot more of the pop values where actually pop, than the classic ones.
But when seeing how many of the true values that where correctly predicted we see that the model is a lot better at classifying classic songs than pop songs:
Pop:

$$\frac{1802}{1802+289} = 86,2\%$$

Classic

$$\frac{1563}{1563+76} = 95,4\%$$