Mandatory Assignment 1

Isak Steinmo Hansen

September 2024

`https://github.com/isakhansen00/MandatoryAssignment1/tree/main`

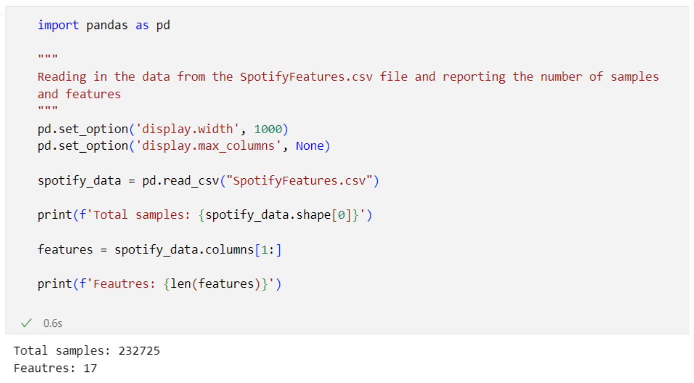
1 Introduction

This is the first mandatory assignment in the FYS-2021 Machine Learning course taught at UiT The Arctic University of Norway. I will be solving three problems ranging from some data pre-processing using pandas and numpy and present the results from that in problem 1. Furthermore in problem 2 I will be implementing a classifier from scratch using logistic regression and train it with the data i have prepared in the previous problem. In the same problem i will also be testing the classifier with the test set and compare the accuracy between the train and test set. In problem 3 I will be creating a confusion matrix based on the classification of the test set

2 Problem 1

2.1 (a)

For problem a i used pandas for reading in the data and reporting the number of rows (samples) and columns (features) that is contained in the entire dataset



```
import pandas as pd

"""
Reading in the data from the SpotifyFeatures.csv file and reporting the number of samples
and features
"""
pd.set_option('display.width', 1000)
pd.set_option('display.max_columns', None)

spotify_data = pd.read_csv("SpotifyFeatures.csv")

print(f'Total samples: {spotify_data.shape[0]}')

features = spotify_data.columns[1:]

print(f'Feautres: {len(features)}')
```

✓ 0.6s

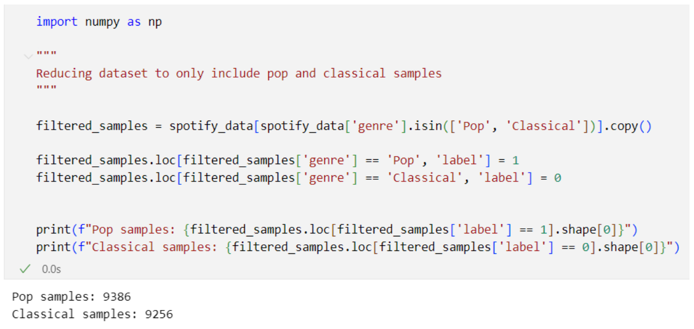
```
Total samples: 232725
Feautres: 17
```

Figure 1: total samples and features

The dataset contains 232725 samples and a total of 17 features

2.2 (b)

In this part of the problem i have a reduced the dataset down to only include samples where the genre is either pop or classical and reported the total samples of each of them. I also mapped the labels as pop = 1 and classical = 0



```
import numpy as np

"""
Reducing dataset to only include pop and classical samples
"""

filtered_samples = spotify_data[spotify_data['genre'].isin(['Pop', 'Classical'])].copy()

filtered_samples.loc[filtered_samples['genre'] == 'Pop', 'label'] = 1
filtered_samples.loc[filtered_samples['genre'] == 'Classical', 'label'] = 0


print(f"Pop samples: {filtered_samples.loc[filtered_samples['label'] == 1].shape[0]}")
print(f"Classical samples: {filtered_samples.loc[filtered_samples['label'] == 0].shape[0]}")
```

✓ 0.0s

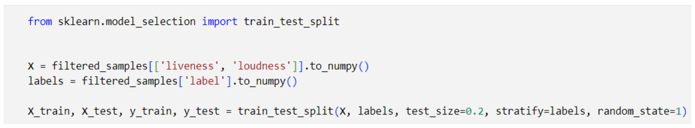
```
Pop samples: 9386
Classical samples: 9256
```

Figure 2: Pop and classical samples

We see a similar distribution of each class with 9386 pop samples and 9256 classical samples (only a 130 sample difference)

2.3 (c)

For this task i have further reduced the dataset to only include two features, liveness and loudness. The dataset is also now split into two numpy arrays, where one includes two columns one for liveness and one for loudness, the other array contrains the known labels of all samples. Lastly the dataset has been split into a training and test set using sklearn.



```
from sklearn.model_selection import train_test_split


X = filtered_samples[['liveness', 'loudness']].to_numpy()
labels = filtered_samples['label'].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, stratify=labels, random_state=1)
```

Figure 3: splitting the dataset

To maintain the class distribution between pop and classical samples in the train and test set i have made sure to set the stratify parameter in the train_test_split function to my labels. this makes sure the dataset is properly stratified.

2.4 (d)

For this bonus task i have plotted all the samples on a liveness vs loudness plane and seperated the samples by color.

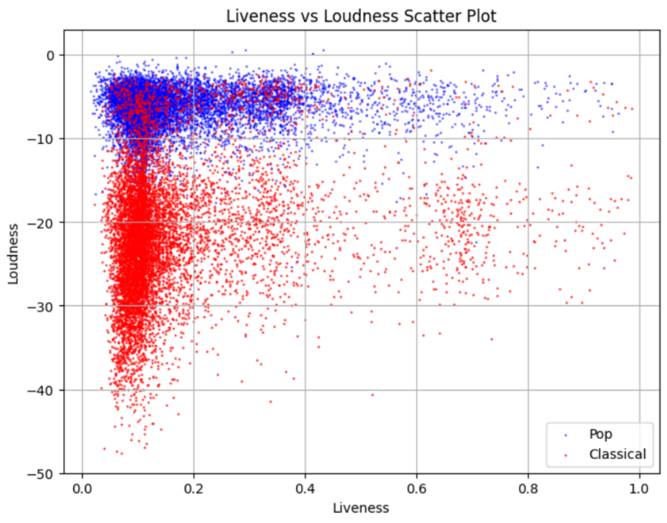


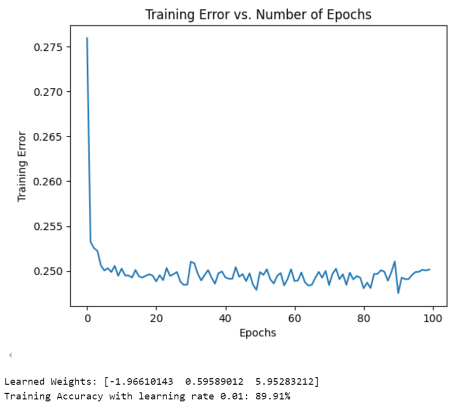
Figure 4: loudness vs liveness plane

Based on this plot we can see that there generally is a clear separation between the samples, mostly for the loudness aspect. We see that the majority of the classical samples lie between -15 and -40 loudness and the majority of the pop samples lie between 0 and -10 loudness. There are some samples that are mixed in with eacother, especially the classical samples we see in the upper left corner, but I would still predict a decent result on the classification since the majority of the classical samples is further down on the loudness scale compared to the pop samples, and we can see a good seperation between the samples

3 Problem 2

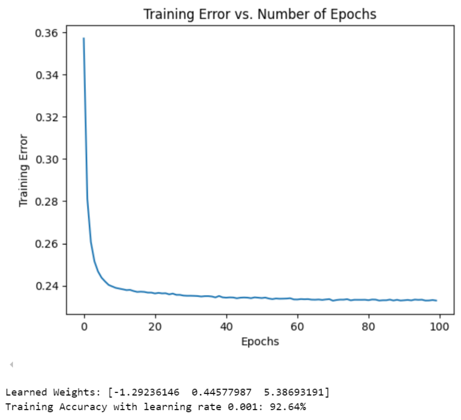
3.1 (a)

When running my implementation of logistic regression with a few different learning rates we see some differences in the way the loss curve acts, mainly that the higher we set the learning rate the quicker it seems to converge, but it also seems to oscillate quite a bit more. This is mainly because with a higher learning rate the adjustment it makes after every epoch is quite big causing it never completely reach the minimum. The lower we set the learning rate we see the opposite, it converges slightly slower but it seems more stable when it does converge. Figure 5-7 shows the loss curve and training accuracy of three different learning rates



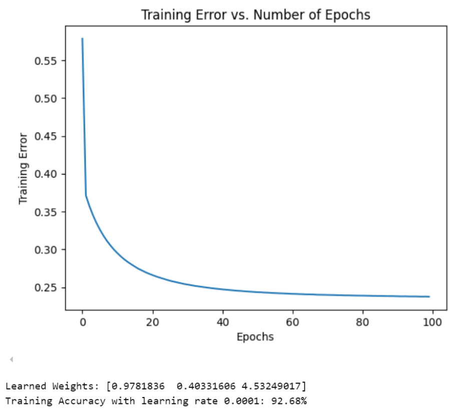
```
Learned Weights: [-1.96610143  0.59589012  5.95283212]
Training Accuracy with learning rate 0.01: 89.91%
```

Figure 5: Loss and accuracy with learning rate 0.01



```
Learned Weights: [-1.29236146  0.44577987  5.38693191]
Training Accuracy with learning rate 0.001: 92.64%
```

Figure 6: Loss and accuracy with learning rate 0.001

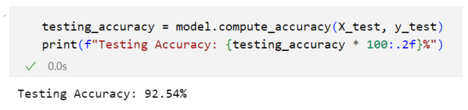


```
Learned Weights: [0.9781836  0.40331606 4.53249017]
Training Accuracy with learning rate 0.0001: 92.68%
```

Figure 7: Loss and accuracy with learning rate 0.0001

3.2 (b)

In the previous task i trained using several learning rates, so for the test set i am running the last trained model (which is learning rate of 0.0001) and for this i get a test accuracy between 92.5 and 92.6 percent



```
testing_accuracy = model.compute_accuracy(X_test, y_test)
print(f"Testing Accuracy: {testing_accuracy * 100:.2f}%")
```

```
Testing Accuracy: 92.54%
```

Figure 8: Testing accuracy

When we compare the test accuracy to the training accuracy as seen in figure 7 we see that the test accuracy is only slightly worse than the training accuracy, this is expected since the test set is entirely unknown to the model when we make predictions based on its features and compute the accuracy. However, considering we still get a accuracy that is really close to the training accuracy it might indicate that our model learns the general patterns and variances in the data quite well, and does not overfit to the training set.

3.3 (c)

Figure 9 shows the plot from 1d with a decision boundary line plotted based on the learned weights. Intuitively this seems like a decent line that seperates the classes quite well.

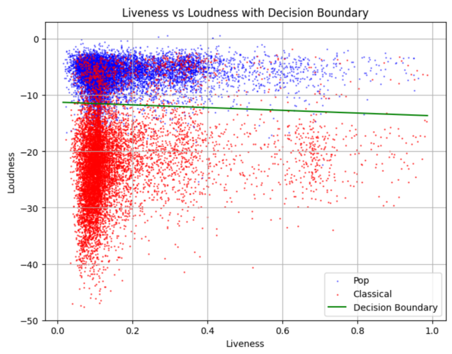


Figure 9: Decision Boundary line

4 Problem 3

4.1 (a)

Figure 10 shows the confusion matrix

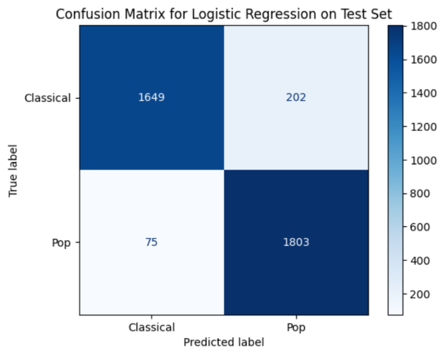


Figure 10: Confusion Matrix

4.2 (b)

The confusion matrix gives us a bit more granular details about what exactly is causing reduced accuracy of our model. In our case we can see that the reduced accuracy is mostly because we are more likely to classify classical samples as pop even though they are not (false positive) than to classify pop samples as classical (false negative).

4.3 (c)

Generally it seems that classical songs with a high loudness value is the hardest songs to classify correctly, based on the fact that on the loudness vs liveness plot the classes seems to seperate mostly based on loudness.

To find a classical song that a pop fan might like i decided to find the classical song with the most loudness (because the pop songs generally have more loudness) and based on that i would recommend the song "Go Faster" by Richie Kotzen. However, this seems like it is more of a rock song. This could perhaps mean there is some mislabeling in the dataset that could negatively affect the classification accuracy.