FYS-2021: Report template — Machine learning

Assignment 1

Odne-Eirik Trøftbråten

September 08, 2024

1 Introduction

This report describes the design and implementation of a program that can classify the genre of a set of songs, by using the songs features. The songs this program uses are the dataset called “spotify-dataset.csv”. For this task I use logistic regression in order to classify the songs.

Code: https://github.com/Odne123/FYS-2021

2 Technical Background

2.1 Logistic regression

Logistic regression is a supervised algorithm used in machine learning used to classify. It is used in tasks predict if sample is a part of a certain category. It takes in variables and gives a binary output, either 1 or 0.

The logistic regression is defined as: $\frac{1}{1+e^{-z}}$, where z is the linear function: $y = mx + b$, where m is the weight (there can be several weights) and b is the bias.

2.2 Cost/Loss

Loss is the measurement of how good a model is at predicting values. It shows how far away from the actual value and the predicted value. This is then used to guide the model's learning process.[1] There are several different loss functions but, in this program, we use the cross-entropy loss function, which is a good function for classifying. For a single data point the function for cross-entropy loss is: $loss = -y * \log(p) - (1-y) * \log{(1-p)}$, for several datapoints you normally summarize the loss and divide it by the number of datapoints.[2]

In machine learning we use gradients. A gradient is a vector with both a direction and a magnitude. In this program, the gradient contains all the partial derivatives of the loss function. Since the gradient points in the steepest ascent of a function, we can utilize this and the partial derivatives to see how a change in one parameter affects the loss function, in order to minimize the loss.

Gradient descent is an algorithm that is used to minimize this loss. In this program I use the variant called stochastic gradient descent, which takes either a random single point or random mini-batch from the dataset and calculates the gradient.

[1] https://www.ibm.com/think/topics/loss-function#:~:text=It%20does%20so%20by%20quantifying,actual%20value%20or%20ground%20truth.

[2] https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning

3 Design

3.1 Preprocessing data

The data is first loaded with pandas. The dataset contains a huge amount of samples of songs, with its features, such as genre, loudness etc. The data is split into a new which only contain the genre classical and pop.

This dataset gets a new feature, called label, which indicates whether the song is of the genre pop, which gets the label 1, or classical, which gets the label 0. This is to prepare the data for classification.

Then we remove all features, except liveness and loudness, which is the features we will use to predict the genre later.

We split the dataset in to a 80/20 distribution, where we have 80 percent of the songs and labels which will be used to train the model, and the remaining 20 percent to test the model.

3.2 Implementing the model

First, we implement the sigmoid function for our logistic regression, as well as the loss function. These will be used to predict and change our parameters as the training goes. The sigmoid function ensures that we will not encounter value errors, if the input gets either too small or too high. The loss will quantify the error between the predictions and actual values.

The training function is then implemented, which uses stochastic gradient descent algorithm. We start with initializing the weights and bias. Then the training will start, going for how many epochs it gets told. For each epoch, the program takes a random small batch of the dataset and performs logistic regression on it. It then calculates gradients of the partial derivatives to weight and bias, which are then updated by a certain learning rate.

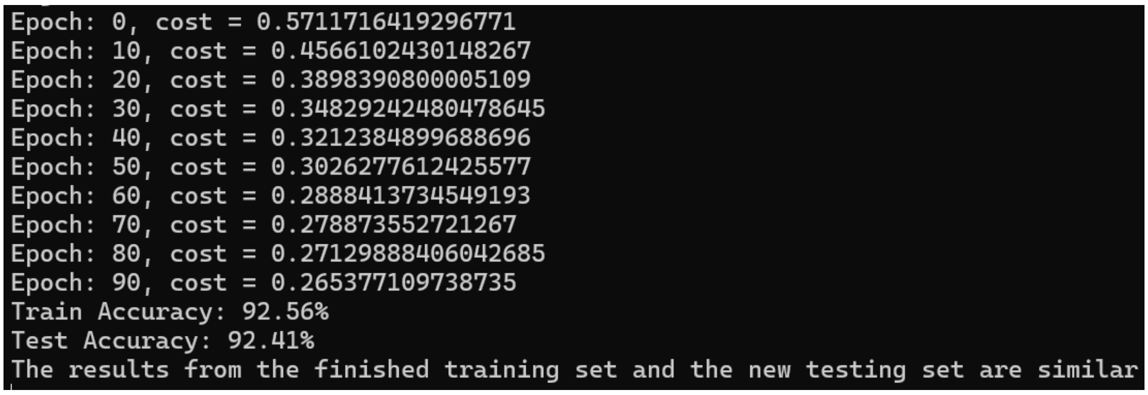
3.3 Model evaluation

We start with a prediction function which takes the features and the finished updated weights and bias and sends the linear function through the sigmoid function. Afterwards, the accuracy function will divide the number of correct guesses with all the true values, giving the accuracy of the model.

After this, we check if the model has been overfitted, by checking the difference of output from the already reviewed training set and the new testing set, which will give a message if the difference is too big.

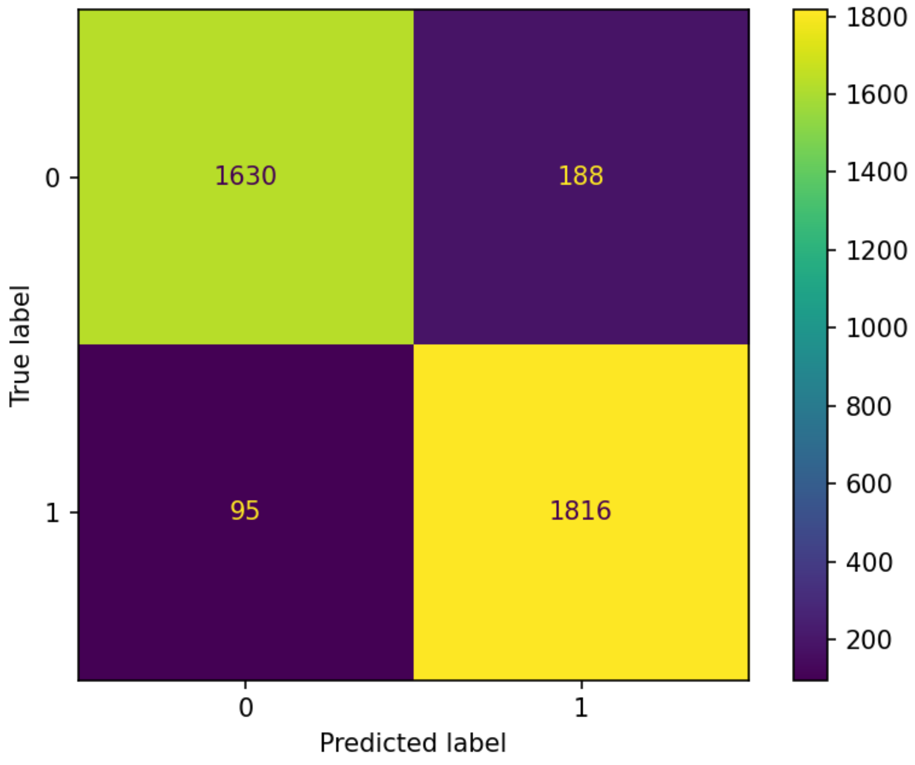
At the end, a confusion matrix is implemented to better visualize the performance of the model.

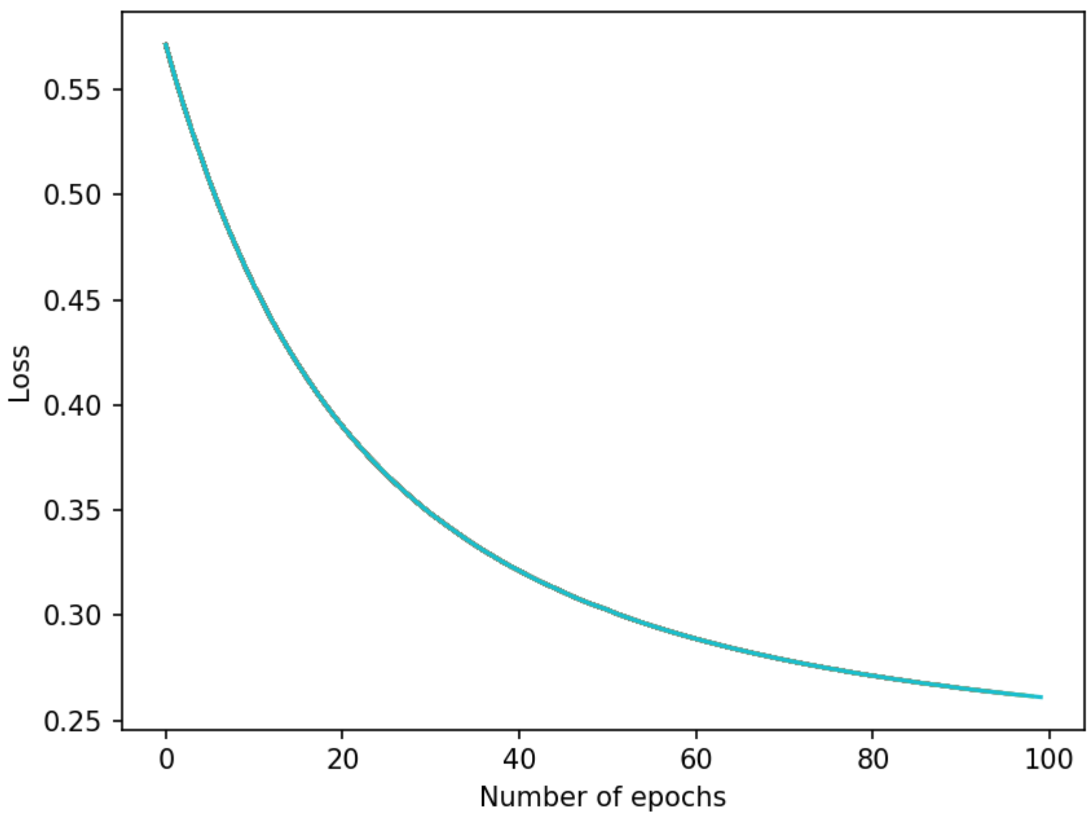
4 Experiments and Results



```
Epoch: 0, cost = 0.5711716419296771
Epoch: 10, cost = 0.4566102430148267
Epoch: 20, cost = 0.3898390800005109
Epoch: 30, cost = 0.34829242480478645
Epoch: 40, cost = 0.3212384899688696
Epoch: 50, cost = 0.3026277612425577
Epoch: 60, cost = 0.2888413734549193
Epoch: 70, cost = 0.278873552721267
Epoch: 80, cost = 0.27129888406042685
Epoch: 90, cost = 0.265377109738735
Train Accuracy: 92.56%
Test Accuracy: 92.41%
The results from the finished training set and the new testing set are similar
```

Confusion matrix:




0.55
0.50
0.45
0.40
0.35
0.30
0.25
Loss
0
20
40
60
80
100
Number of epochs

5 Discussion

The model got a 92.41% accuracy with a learning rate of 0.01 and a 100 epochs. This seems be the peak of how high it could get, as I also tried to reduce the learning rate all the way down to 0.0001 with 40 000 epochs, with no significant improvement. This may be because of the limitations of only 2 features, adding more features could give the model more parameters to use to classify.

I also researched a bit in normalizing the features, since some of the values in one of the features could be way bigger than another value in the other feature, in case this would add to a better model. It could perhaps aid in the gradient descent, as larger values contribute more to the loss function.[3] However, this did not change the output in any observable way, so I removed it from the program. However, in my humble and in great part uneducated guess, this is good accuracy.

6 Conclusion

The model achieved a accuracy of 92.42% with a learning rate of 0.1 and 100 epochs. This did not change when lowering the learning rate and increasing the epochs, likely due to the limitation of using only 2 features. Normalizing the features values did not aid so it was removed as well. Overall, the model performed well, given its restraints.

6.1 Problem 3b

The confusion matrix gives additional information. It shows how the true positives, meaning correctly predicting a class(pop song predicted as a pop song), true negatives(classical songs predicted as classical songs), false positives(classical songs predicted as pop songs) and false negatives(pop songs predicted as classical songs). The loss graph however, shows the loss reduction as the training goes.

7 Sources

References

[1] MarkovML. (2024). Normalization in machine learning. MarkovML. Retrieved September 8, 2024, from https://www.markovml.com/blog/normalization-in-machine-learning.

[3] https://www.markovml.com/blog/normalization-in-machine-learning

[2] IBM. (2024). Loss function in machine learning. IBM Think. Retrieved September 8, 2024, from https://www.ibm.com/think/topics/loss-function#:~:text=It%20does%20so%20by%20quantifying,actual%20value%20or%20ground%

[3] DataCamp. (2024). *The cross-entropy loss function in machine learning*. DataCamp. Retrieved September 8, 2024, from https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning