Assignment 1, FYS-2021,
Logistic Regression Analysis on Spotify Data

Truls Larsen

GitHub repository link: https://github.com/tlar007/FYS-2021

UiT - Norges Arktiske Universitet

September 8, 2024

Problem 1: Data Preparation and Visualization

1a: Dataset Overview

The Spotify dataset consists of **232,725** songs, each with **18** features representing various musical attributes. The dataset includes a wide range of genres, e.g. Pop and Classical.

1b: Data Filtering and Labeling

The Spotfy dataset the dataset was filtered to include only songs categorized as 'Pop' or 'Classical'. After filtering:

- 'Pop' samples/songs labeled as 1: **9,386**.
- 'Classical' samples/songs labeled as 0: **9,256**.

1c: Data Splitting

The dataset is the filtered to only select the fatures features 'liveness' and 'loudness', for the classification task. And the dataset was split into training (80%) and test (20%) sets with from bulit in sklearn model in python, with maintaining an even distribution of genres in both sets with and picking random samples with within the class in each set.

Class	Genre	Training samples	Test samples
Class 1: '1'	Pop	7508	1878
Class 2: '0'	Classical	7405	1851

1d: Feature Visualization

To visualize the distribution of the selected features, a scatter plot of 'liveness' vs 'loudness' was created for the two genres. The samples from the two classes are overlapping and it could be difficult to optimize 100% correct linear classifier.

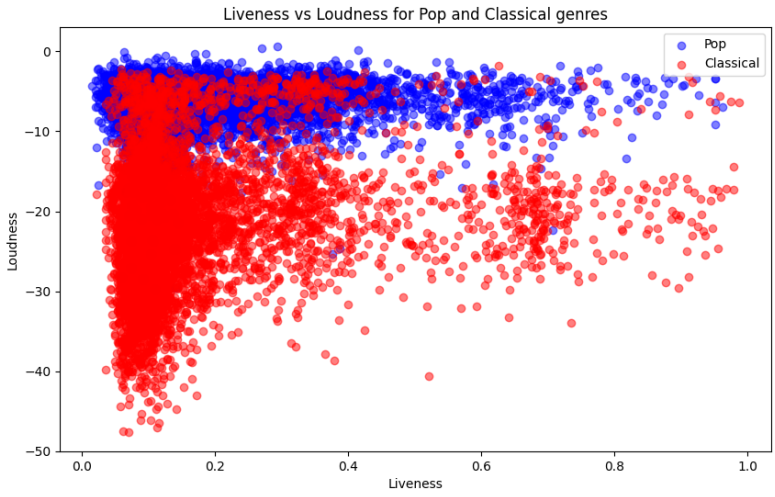

Figure 1: Scatter plot of Liveness vs Loudness for Pop and Classical genres.

Problem 2: Logistic Regression Model

2a: Model Training

Implementing a logistic discriminent classifier with a stochastic gradient decent.

$$\hat{y} = \begin{cases} 1, & \text{if } \sigma(z) \geq 0.5 \\ 0, & \text{if } \sigma(z) < 0.5 \end{cases} \tag{1}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

$$z = w_1 \cdot x_1 + w_2 \cdot x_2 + b \tag{3}$$

Where: $\hat{y}$: predicted y, σ: is the Sigmoid function, z: , x_i: features, w_i: weights, ϵ: constant to avoid $log(0)$.

The model uses the following predefined constants:

- learning rate: $lr = 0.001$
- number of epochs: 10
- $w_1 = [0, 0]$.
- $\epsilon = 10^{-8}$, and does not affect the the outcome to much.

Cross entrypy function for computing teh loss L:

$$L = -\sum_{i=1}^{m} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{4}$$

A logistic regression model was trained using stochastic gradient descent. The model parameters were optimized over 10 iterations with a learning rate of 0.001. The training process involved calculating gradients, updating weights, and minimizing the cost function.

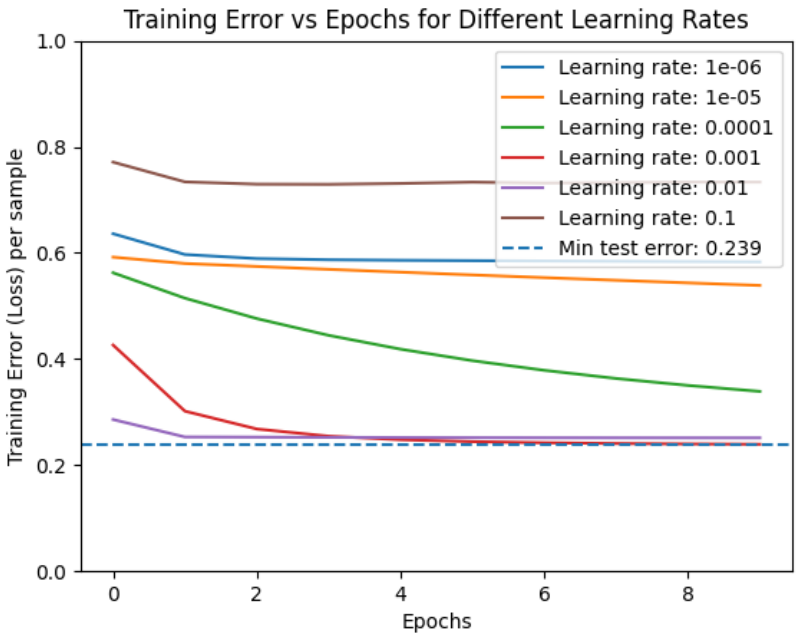

Figure 2: Learning rates for 10 epochs

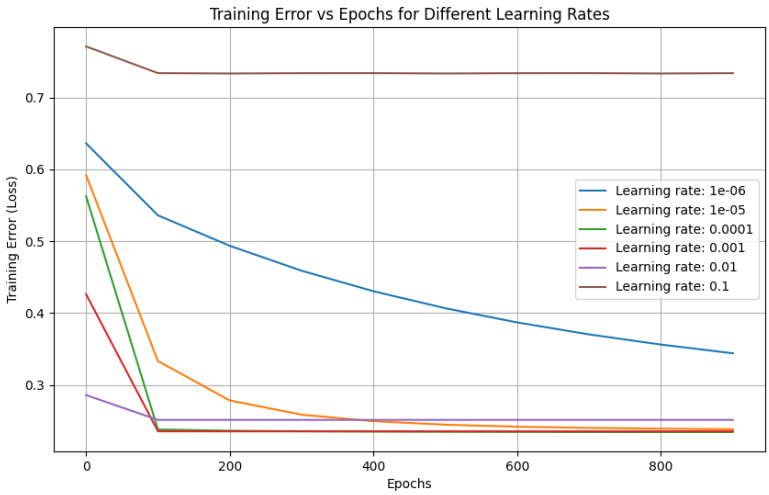

Figure 3: Learning rates for 1000 epochs, with 100 epoch resolution

2b: Test Set Evaluation

The trained model was evaluated on the test set, achieving an accuracy of **92.28%**. This result indicates that the model was able to classify the majority of the songs correctly.

2c: Decision Boundary Visualization

The decision boundary given by w and b from the regression model.

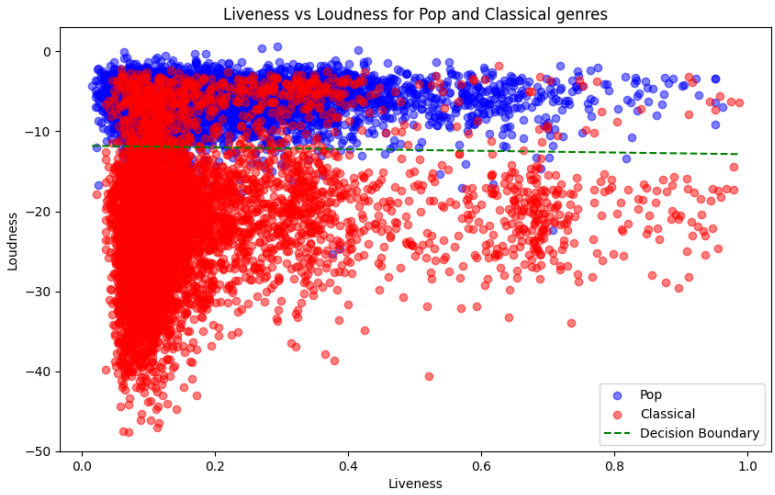

Figure 4: Decision Boundary on the Liveness vs Loudness plot.

Problem 3: Results and Analysis

3a: Confusion Matrix

The confusion matrix was manually calculated to provide a detailed breakdown of the model's performance on the test set. This matrix reveals the number of true positives, true negatives, false positives, and false negatives.

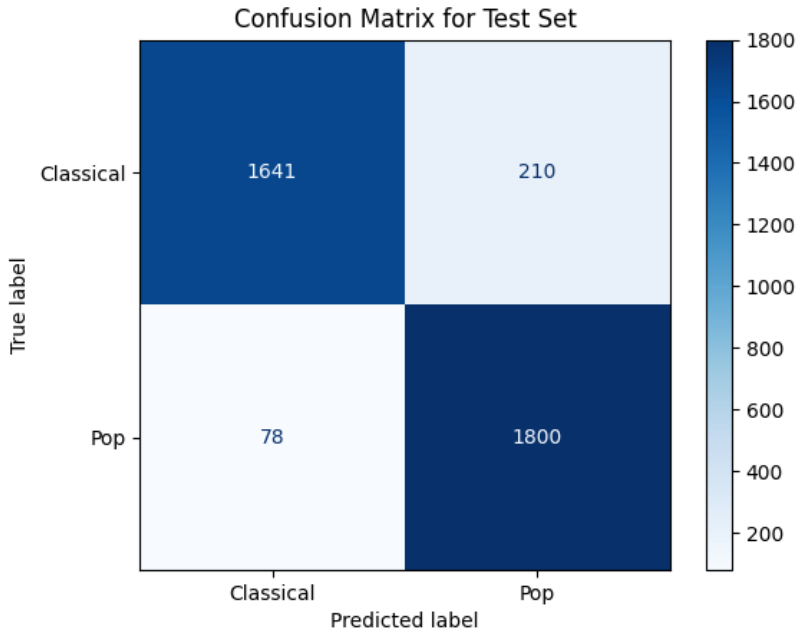

Figure 5: Confusion Matrix for the Test Set.

3b: Performance Analysis

While the accuracy metric gives an overall sense of the model's performance, the confusion matrix offers deeper insights. It highlights the specific areas where the model performed well and where it struggled. For example, the model's ability to correctly classify 'Pop' songs as 'Pop' and 'Classical' songs as 'Classical' is evident from the high values along the diagonal of the matrix.

References

[1] Alpaydin, Ethem. *Introduction to Machine Learning*. MIT Press, 2014.

[2] Lecture notes and slides from Canvas in course FYS-2021.

[3] Spotify Dataset from Canvas in course FYS-2021.

[4] Python Documentation, *Numpy*, *Pandas*, *Matplotlib*. Available at: `https://numpy.org/doc/`, `https://pandas.pydata.org/pandas-docs/stable/`, `https://matplotlib.org/stable/contents.html`

Appendix

Python code Problem 1

1a

```
import pandas as pd
import numpy as np

# Load the dataset
df = pd.read_csv('SpotifyFeatures.csv')

# Problem 1a: Report the number of samples and features.
num_samples = df.shape[0]
num_features = df.shape[1]
#print(f"Number of samples: {num_samples}, Number of features: {
                                num_features}")
```

1b

```
# Problem 1b: Filter the dataset for 'Pop' and 'Classical' genres
                                and create labels.
filtered_df = df[df['genre'].isin(['Pop', 'Classical'])].copy()  #
                                .copy() to avoid
                                SettingWithCopyWarning
filtered_df['label'] = (filtered_df['genre'] == 'Pop').astype(int)
pop_count = filtered_df[filtered_df['genre'] == 'Pop'].shape[0]
classical_count = filtered_df[filtered_df['genre'] == 'Classical'].
                                shape[0]
#print(f"Number of 'Pop' samples: {pop_count}, Number of 'Classical
                                ' samples: {classical_count}")
```

1c

```
from sklearn.model_selection import train_test_split

# Problem 1c: Select features 'liveness' and 'loudness' and split
                                    the data
features = filtered_df[['liveness', 'loudness']].values
labels = filtered_df['label'].values
X_train, X_test, y_train, y_test = train_test_split(features,
                                    labels, test_size=0.2, stratify=
                                    labels, shuffle=True,
                                    random_state=42)

data = X_train, X_test, y_train, y_test

""" # Counting, Max and Min value of features
for i in data:
    print(i[0:10])

X_1_max = 0
X_1_min = 100

for i in data[:2]:
    print(i[:10])

    for X in i:
        # X_1
        if X[1] > X_1_max:
            X_1_max = X[1]
        if X[1] < X_1_min:
            X_1_min = X[1]

print(X_1_max, X_1_min)


for i in data[2:]:
    pop_count_array = sum(i)
    classical_count_array = len(i)-sum(i)
    print(f'pop: {pop_count_array}')
    print(f'classical: {classical_count_array}') """
```

1d

```
import matplotlib.pyplot as plt

# Problem 1d: Visualize the features with a scatter plot
def plot_scatterplot(X_train, y_train):
    plt.figure(figsize=(10, 6))
    plt.scatter(X_train[y_train == 1][:, 0], X_train[y_train == 1][
                                    :, 1], color='blue', label='
                                    Pop', alpha=0.5)
    plt.scatter(X_train[y_train == 0][:, 0], X_train[y_train == 0][
                                    :, 1], color='red', label='
                                    Classical', alpha=0.5)
    plt.xlabel('Liveness')
    plt.ylabel('Loudness')
    plt.legend()
```

```
    plt.title('Liveness vs Loudness for Pop and Classical genres')
    plt.show()

plot_scatterplot(X_train, y_train)
```

Problem 2

2a

```
import numpy as np

# Problem 2a: Implement your own logistic discrimination classifier
                                        .
# Logistic Regression Functions
def logistic_regression_model(X, y, epochs=10, learning_rate=0.001,
                                  epsillon = 0.00001):
    """
    Found optimal values after plotting for different lr
    X: features
    y: labels
    epochs=10
    learning_rate=0.001
    epsillon = 0.00001
    """
    # Initial values for w and b
    number_of_features = X.shape[1]
    w = np.zeros(number_of_features)
    b = 0

    number_of_samples = len(X)
    epoch_list = []
    training_error_list = []

    print(f'learning rate: {learning_rate}')

    for epoch in range(epochs):
        ## SHUFFLE???
        training_error_sum = 0
        for i in range(number_of_samples):
            # step-1 (linear combination) z value (z = np.dot(w.T,
                                                  X[i]) + b):
            # X[i] as an array is in correct shape even w/o
                                                  transposing
            z = np.dot(w, X[i]) + b

            # step-2 sigmoid_value for y_hat
            y_hat = 1 / (1 + np.exp(-z))

            # step-3 find the deviance from y to get the gradients
            differance = y_hat - y[i]
            dw = differance * X[i]
            db = differance

            # step-4  update the weights w and bias b
            w = w - learning_rate * dw
```

```
            b = b - learning_rate * db

            # Training error - only for plotting
            # to avoid log(x=0), use log(x+epsilon), epsilon = 10
                                                (^-5)
            training_error_i = - (y[i] * np.log(y_hat+epsillon) + (
                                                1-y[i]) * np.log(1-
                                                y_hat+epsillon))
            training_error_sum = training_error_sum +
                                                training_error_i

        # Appending training error after an epochs
        #L_avg = L_sum / number_of_samples
        epoch_list.append(epoch)
        training_error_sum = training_error_sum / number_of_samples
        training_error_list.append(training_error_sum)

        #if epoch == 1:
        print(f'Learning-rate: {learning_rate}, Epoch: {epoch},
                                        Training error L: {
                                        training_error_sum}')
        if epoch % 100 == 0:
            print(f'Learning-rate: {learning_rate}, Epoch: {epoch},
                                         Training error L: {
                                        training_error_sum}')

    # returning the trained weights w, trained bias b and the
                                    errors for each epoch:
                                    training_error_list
    return w, b, training_error_list

# plot the training array
def plot_errors_with_lr(error_nested):
    learning_rates = [1e-06, 1e-05, 1e-04, 1e-03, 1e-02, 1e-01]
    epochs = 10

    training_error_for_different_learning_rates = []
    weights_for_different_learning_rates  = []
    bias_for_different_learning_rates= []

    for lr in learning_rates:
        w, b, training_error = logistic_regression_model(X_train,
                                        y_train, epochs=epochs,
                                        learning_rate=lr)

        # append to lists for plotting
        weights_for_different_learning_rates.append(w)
        bias_for_different_learning_rates.append(b)
        training_error_for_different_learning_rates.append(
                                        training_error)
    min_value = min(min(inner_list) for inner_list in error_nested)

    plt.figure()
    for i in range(len(error_nested)):
        plt.plot(range(0, len(error_nested[i])), error_nested[i],
                                        label=f'Learning rate: {
                                        learning_rates[i]}')
```

```
    plt.axhline(y=min_value, label = f'Min training error: {
                                        min_value:.3f}', linestyle =
                                        '--')
    plt.xlabel('Epochs')
    plt.ylabel('Training Error (Loss) per sample')
    plt.title('Training Error vs Epochs for Different Learning
                                        Rates')
    plt.ylim(0, 1)
    plt.legend()
    plt.show()

#def plot_errors_with_lr(
                                    training_error_for_different_learning_rates
                                    )

w, b, training_error = logistic_regression_model(X_train, y_train)
```

2b

```
from sklearn.metrics import accuracy_score

# Problem 2b: Testing my models w and b on the test set
def predict(X, w, b):
    z = np.dot(w, X.T) + b
    y_hat = 1 / (1 + np.exp(z))  # Sigmoid function
    y_predict = np.where(y_hat >= 0.5, 0, 1) #Threshold to classify
                                        as 0 or 1
    return y_predict

#Function for evaluation
def evaluate(X, y, w, b):
    y_predicted = predict(X, w, b)
    accuracy = accuracy_score(y, y_predicted) # using sklearn
    return accuracy

testing_accuracy = evaluate(X_test, y_test, w, b)
print('The accuracy of our model on our training set is:',
                                    testing_accuracy)
```

2c

```
import matplotlib.pyplot as plt

# Problem 2c: Visualize the decision boundary
def plot_decision_boundary(X_train, y_train):
    # Generate x values for the decision boundary
    x_values = np.linspace(X_train[:, 0].min(), X_train[:, 0].max()
                                        , 100)

    # Compute corresponding y values using the decision boundary
                                        equation
    y_values = -(w[0] * x_values + b) / w[1]
```

```
    # Plot the decision boundary
    plt.figure(figsize=(10, 6))
    plt.scatter(X_train[y_train == 1][:, 0], X_train[y_train == 1][
                                    :, 1], color='blue', label='
                                    Pop', alpha=0.5)
    plt.scatter(X_train[y_train == 0][:, 0], X_train[y_train == 0][
                                    :, 1], color='red', label='
                                    Classical', alpha=0.5)
    plt.plot(x_values, y_values, 'g--', label='Decision Boundary')
    plt.xlabel('Liveness')
    plt.ylabel('Loudness')
    plt.legend()
    plt.title('Liveness vs Loudness for Pop and Classical genres')
    plt.show()

plot_decision_boundary(X_train, y_train)
```

Problem 3

3a

```
from sklearn.metrics import confusion_matrix,
                                ConfusionMatrixDisplay

# Problem 3a: Create Confusion Matrix for the Test Set
def plot_confusion_matrix(y_true, y_pred):
    # Generate the confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    # Create the confusion matrix plot
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                    display_labels=['Classical',
                                    'Pop'])
    disp.plot(cmap='Blues')
    plt.title('Confusion Matrix for Test Set')
    plt.show()

# Predict using the trained model
y_pred_test = predict(X_test, w, b)

# Plot the confusion matrix
plot_confusion_matrix(y_test, y_pred_test)
```

3b

```
# Problem 3b: Accuracy of the model
test_accuracy = accuracy_score(y_test, y_pred_test)
print(f"Accuracy of the model on the test set: {test_accuracy * 100
                                :.2f}%")

# Additional explanation of confusion matrix
print("Confusion Matrix gives more insight into:")
print("1. True Positives (Pop correctly classified)")
```

```
print("2. True Negatives (Classical correctly classified)")
print("3. False Positives (Classical misclassified as Pop)")
print("4. False Negatives (Pop misclassified as Classical)")
```