# MACHINE LEARNING PROJECT

# REPORT

# INTRODUCTION

In general, our aim is to create a system in which we discriminate music genres by using the models we have created according to some characteristics of this data, after taking our data and subjecting it to certain regulations.

## STEP 1

After importing the necessary libraries, we took our data and read it and visualized it.

```
df = pd.read_csv("music_genre.csv")
df.head(50000)
```

| | instance_id | artist_name | track_name | popularity | acousticness | danceability | duration_ms | energy | instrumentalness | key | liveness | loudness | mode | sp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 32894.0 | Röyksopp | Röyksopp's Night Out | 27.0 | 0.004680 | 0.652 | -1.0 | 0.941 | 0.792000 | A# | 0.115 | -5.201 | Minor | |
| 1 | 46652.0 | Thievery Corporation | The Shining Path | 31.0 | 0.012700 | 0.622 | 218293.0 | 0.890 | 0.950000 | D | 0.124 | -7.043 | Minor | |
| 2 | 30097.0 | Dillon Francis | Hurricane | 28.0 | 0.003060 | 0.620 | 215613.0 | 0.755 | 0.011800 | G# | 0.534 | -4.617 | Major | |
| 3 | 62177.0 | Dubloadz | Nitro | 34.0 | 0.025400 | 0.774 | 166875.0 | 0.700 | 0.002530 | C# | 0.157 | -4.498 | Major | |
| 4 | 24907.0 | What So Not | Divide & Conquer | 32.0 | 0.004650 | 0.638 | 222369.0 | 0.587 | 0.909000 | F# | 0.157 | -6.266 | Major | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 49995 | 56911.0 | Too Short | Shake That Monkey | 55.0 | 0.000577 | 0.937 | 278707.0 | 0.695 | 0.000032 | C | 0.295 | -6.932 | Major | |
| 49996 | 20269.0 | Unknown Mortal Orchestra | So Good at Being in Trouble | 57.0 | 0.036300 | 0.829 | 230147.0 | 0.435 | 0.878000 | C | 0.119 | -10.136 | Major | |
| 49997 | 54580.0 | Tee Grizzley | First Day Out | 75.0 | 0.171000 | 0.587 | 254694.0 | 0.711 | 0.000000 | C# | 0.125 | -6.330 | Major | |
| 49998 | 64552.0 | Logic | Growing Pains III | 55.0 | 0.474000 | 0.514 | 246773.0 | 0.730 | 0.000000 | E | 0.511 | -8.491 | Minor | |
| 49999 | 28408.0 | Night Lovell | Barbie Doll | 56.0 | 0.133000 | 0.849 | 237667.0 | 0.660 | 0.000008 | C | 0.296 | -7.195 | Major | |

50000 rows × 18 columns

## STEP 2

With the df_describe() method, we saw our features in the data frame and the specific values (mean, stn, min,...) of our data here.

```
print(df.describe())
print(df.shape)
```

```
        instance_id    popularity  acousticness  danceability   duration_ms  \
count  50000.000000  50000.000000  50000.000000  50000.000000  5.000000e+04
mean   55888.396360     44.220420      0.306383      0.558241  2.212526e+05
std    20725.256253     15.542008      0.341340      0.178632  1.286720e+05
min    20002.000000      0.000000      0.000000      0.059600 -1.000000e+00
25%    37973.500000     34.000000      0.020000      0.442000  1.748000e+05
50%    55913.500000     45.000000      0.144000      0.568000  2.192810e+05
75%    73863.250000     56.000000      0.552000      0.687000  2.686122e+05
max    91759.000000     99.000000      0.996000      0.986000  4.830606e+06

              energy  instrumentalness      liveness      loudness  \
count  50000.000000      50000.000000  50000.000000  50000.000000
mean       0.599755          0.181601      0.193896     -9.133761
std        0.264559          0.325409      0.161637      6.162990
min        0.000792          0.000000      0.009670    -47.046000
25%        0.433000          0.000000      0.096900    -10.860000
50%        0.643000          0.000158      0.126000     -7.276500
75%        0.815000          0.155000      0.244000     -5.173000
max        0.999000          0.996000      1.000000      3.744000

        speechiness       valence
count  50000.000000  50000.000000
mean       0.093586      0.456264
std        0.101373      0.247119
min        0.022300      0.000000
25%        0.036100      0.257000
50%        0.048900      0.448000
75%        0.098525      0.648000
max        0.942000      0.992000
```

STEP 3

We removed the null values with the dropna() method in order to clean up my dataframe. We are preparing to process our data.

```
# drop na values
df = df.dropna()
df.describe()
```

| | instance_id | popularity | acousticness | danceability | duration_ms | energy | instrumentalness | liveness | loudness | speechiness | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 50000.000000 | 50000.000000 | 50000.000000 | 50000.000000 | 5.000000e+04 | 50000.000000 | 50000.000000 | 50000.000000 | 50000.000000 | 50000.000000 | 500 |
| mean | 55888.396360 | 44.220420 | 0.306383 | 0.558241 | 2.212526e+05 | 0.599755 | 0.181601 | 0.193896 | -9.133761 | 0.093586 | |
| std | 20725.256253 | 15.542008 | 0.341340 | 0.178632 | 1.286720e+05 | 0.264559 | 0.325409 | 0.161637 | 6.162990 | 0.101373 | |
| min | 20002.000000 | 0.000000 | 0.000000 | 0.059600 | -1.000000e+00 | 0.000792 | 0.000000 | 0.009670 | -47.046000 | 0.022300 | |
| 25% | 37973.500000 | 34.000000 | 0.020000 | 0.442000 | 1.748000e+05 | 0.433000 | 0.000000 | 0.096900 | -10.860000 | 0.036100 | |
| 50% | 55913.500000 | 45.000000 | 0.144000 | 0.568000 | 2.192810e+05 | 0.643000 | 0.000158 | 0.126000 | -7.276500 | 0.048900 | |
| 75% | 73863.250000 | 56.000000 | 0.552000 | 0.687000 | 2.686122e+05 | 0.815000 | 0.155000 | 0.244000 | -5.173000 | 0.098525 | |
| max | 91759.000000 | 99.000000 | 0.996000 | 0.986000 | 4.830606e+06 | 0.999000 | 0.996000 | 1.000000 | 3.744000 | 0.942000 | |

STEP 4

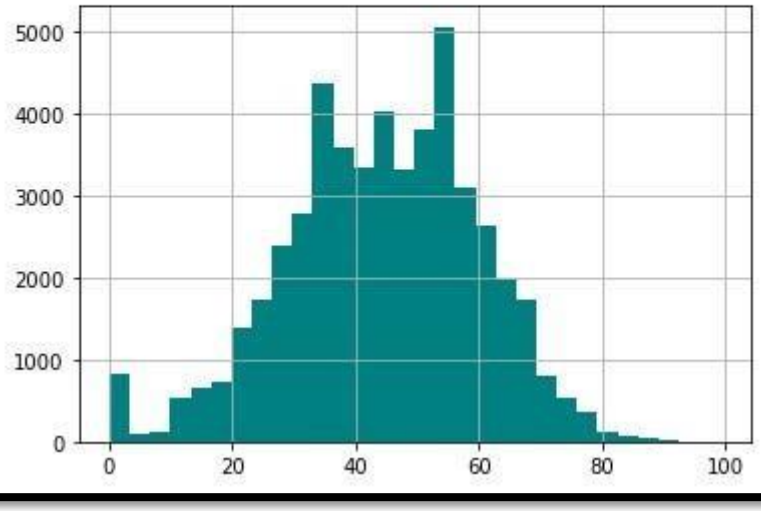
We manually select some features.

```
In [7]: # these features will be used in classification stage
        feature_names = ["popularity", "acousticness", "danceability", "duration_ms", "energy", "instrumentalness", "liveness",
                         "loudness", "speechiness", "valence"]
        feature_names
```

I examine these manually selected features on the histogram.

```
popularity = df.loc[:, "popularity"]
popularity.hist(bins=30, color='#008080')
```

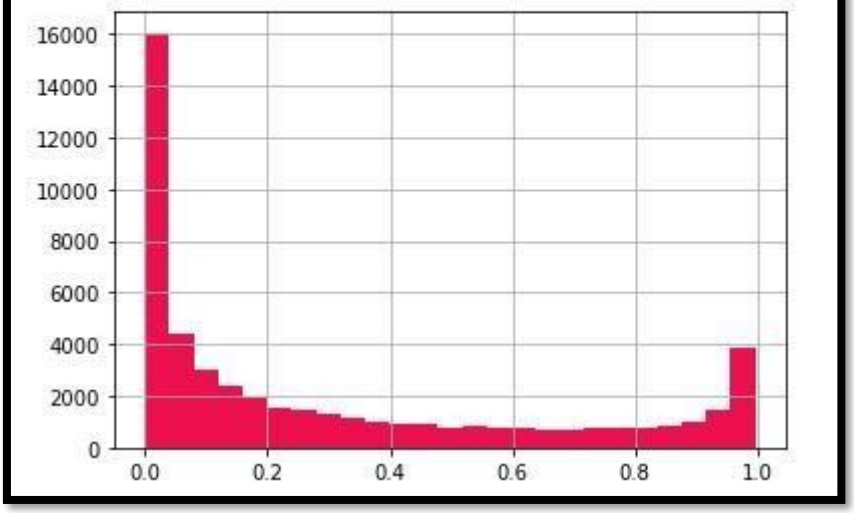
```
<AxesSubplot:>
```




I observed that this feature shows a distribution close to the symmetrical distributed. It can be concluded that mean and median values are close.

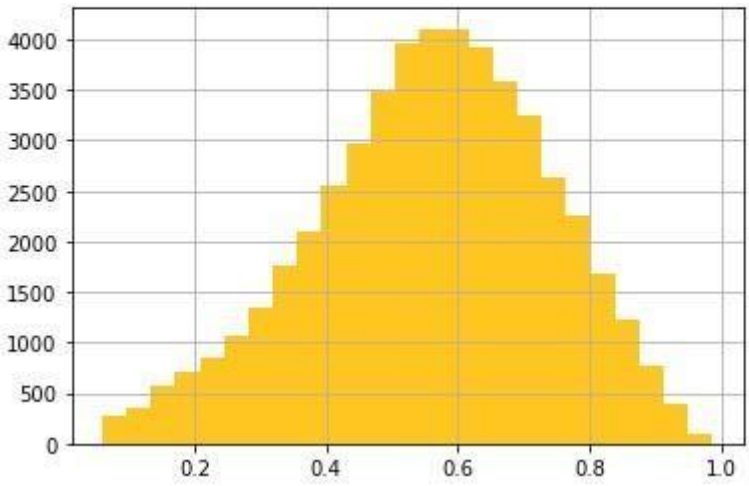
```
acousticness = df.loc[:, "acousticness"]
acousticness.hist(bins=25,color='#ee104e')
```

```
<AxesSubplot:>
```




```
danceability = df.loc[:, "danceability"]
danceability.hist(bins=25,color='#ffc621')
```

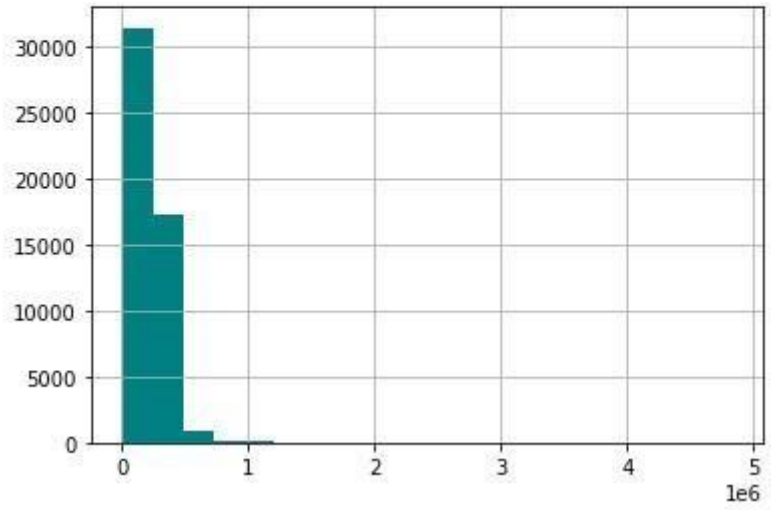
```
<AxesSubplot:>
```




I observed that the distribution of danceability also shows a distribution similar to popularity.

```
duration_ms = df.loc[:, "duration_ms"]
duration_ms.hist(bins=20 ,color='#008080')
```

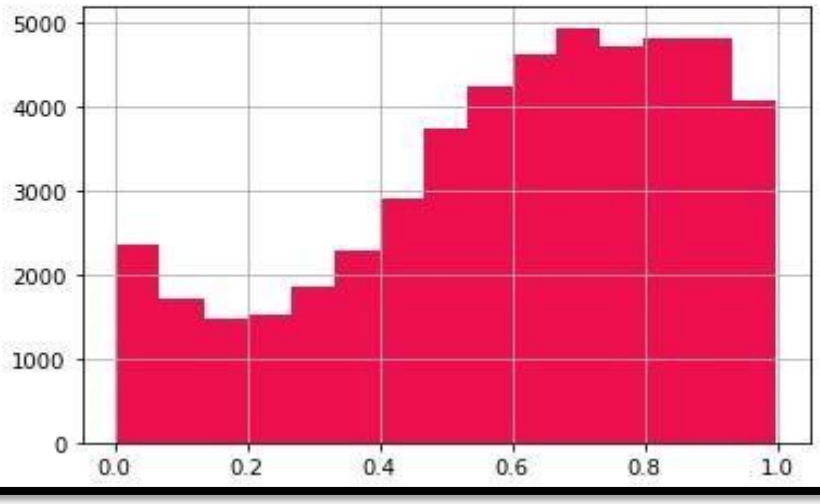
```
<AxesSubplot:>
```




Duration, on the other hand, spread in a narrower area in general. Accordingly, it can be concluded that the discrimination is low.

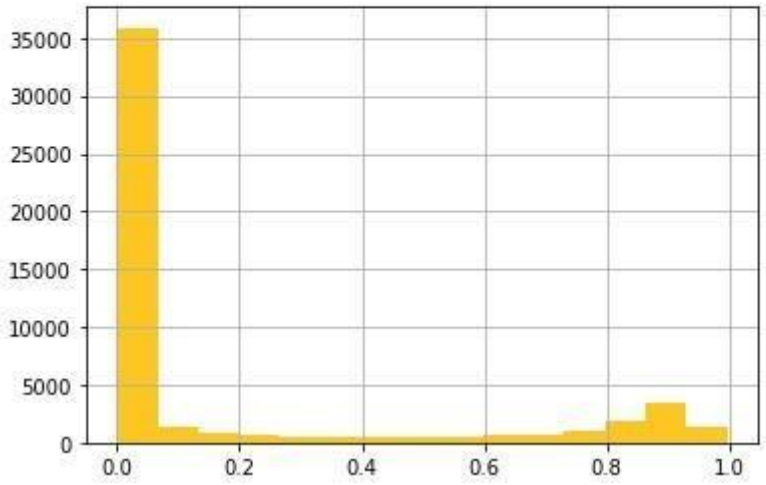
```
energy = df.loc[:, "energy"]
energy.hist(bins=15,color='#ee104e')
```

```
<AxesSubplot:>
```



```
instrumentalness = df.loc[:, "instrumentalness"]
instrumentalness.hist(bins=15,color='#ffc621')
```

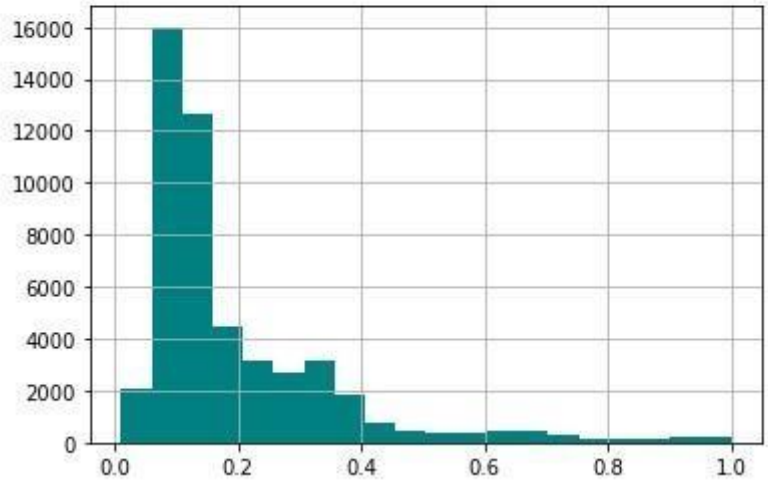
```
<AxesSubplot:>
```




We observe that the instrumentalness feature is only evident in a certain interval.

```
liveness = df.loc[:, "liveness"]
liveness.hist(bins=20 ,color='#008080')
```

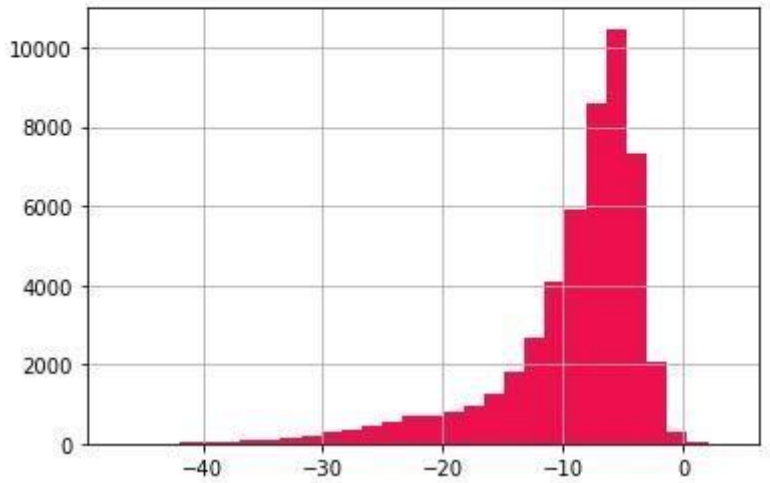
```
<AxesSubplot:>
```




It is seen that the liveness feature shows a skewed to right distribution. We see the mode > median > mean order. So the mean is also to the right of the peak. The distribution stacks up at high scores, that is, it stacks to the right. It has a negative belt. Instruction is sufficient. The learning level is high. The test is easy.

```
loudness = df.loc[:, "loudness"]
loudness.hist(bins=30,color='#ee104e')
```

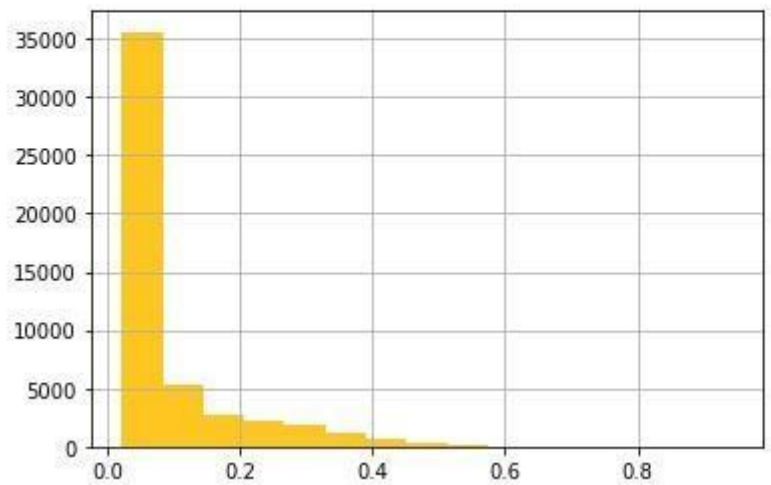
```
<AxesSubplot:>
```




I see that the loudness feature shows a skewed to left distribution. We see the order mean > median > mode. so the mean is also to the left of the peak. The scatter piles up on low scores. It has a positive belt. The test is difficult. Achievement is low and learning level is low.

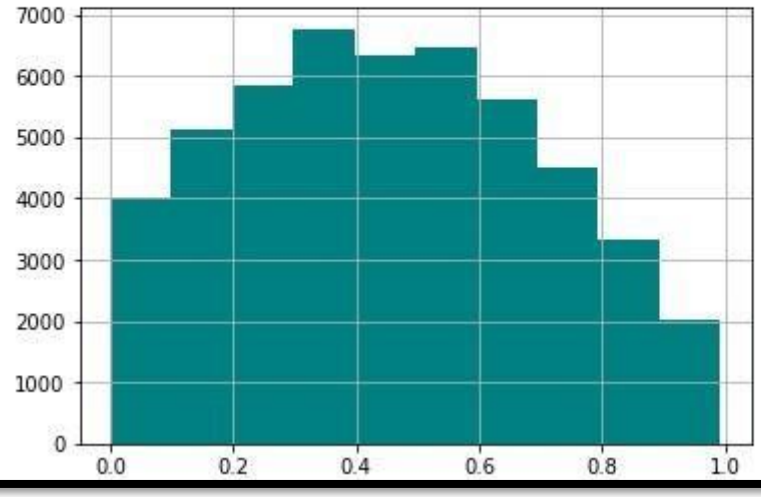
```
speechiness = df.loc[:, "speechiness"]
speechiness.hist(bins=15,color='#ffc621')
```

```
<AxesSubplot:>
```



```
valence = df.loc[:, "valence"]
valence.hist(bins=10 ,color='#008080')
```

```
<AxesSubplot:>
```




It is seen that the valence property shows a symmetrical distribution. In distributions close to this style, the median mode and mean values are close.

## STEP 5

I applied shuffling because shuffling data serves the purpose of reducing variance and making sure that models remain general and overfit less.

Then I determined our classes. Make sure it's unique.

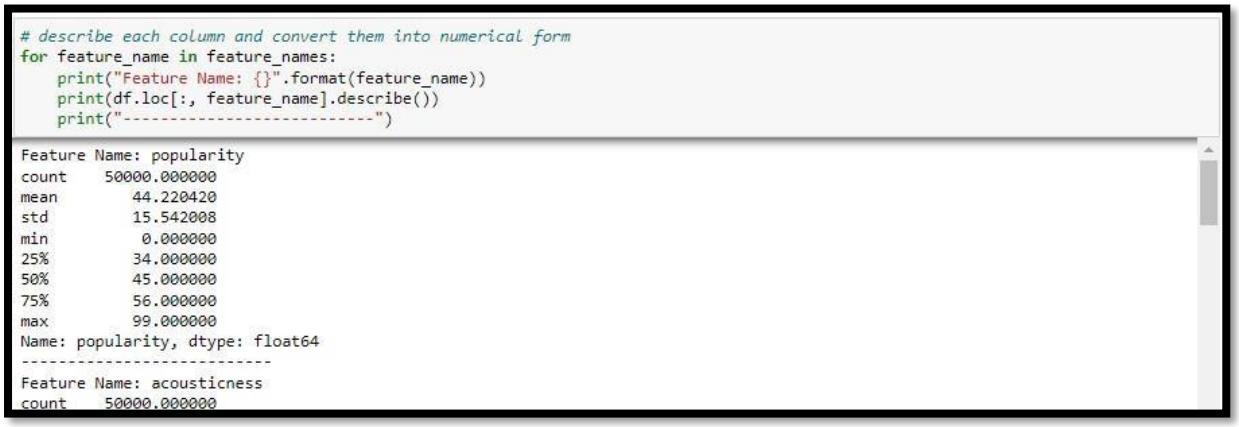
```
# shuffle the data
df = shuffle(df)
```

```
# music classes
music_classes = df.loc[:, "music_genre"].unique()
print(music_classes)
```

```
['Rap' 'Jazz' 'Hip-Hop' 'Electronic' 'Rock' 'Classical' 'Anime' 'Blues'
 'Country' 'Alternative']
```

## STEP 6

I converted each column to numeric value.



```
# describe each column and convert them into numerical form
for feature_name in feature_names:
    print("Feature Name: {}".format(feature_name))
    print(df.loc[:, feature_name].describe())
    print("---------------------------")
```

```
Feature Name: popularity
count    50000.000000
mean        44.220420
std         15.542008
min          0.000000
25%         34.000000
50%         45.000000
75%         56.000000
max         99.000000
Name: popularity, dtype: float64
---------------------------
Feature Name: acousticness
count    50000.000000
```

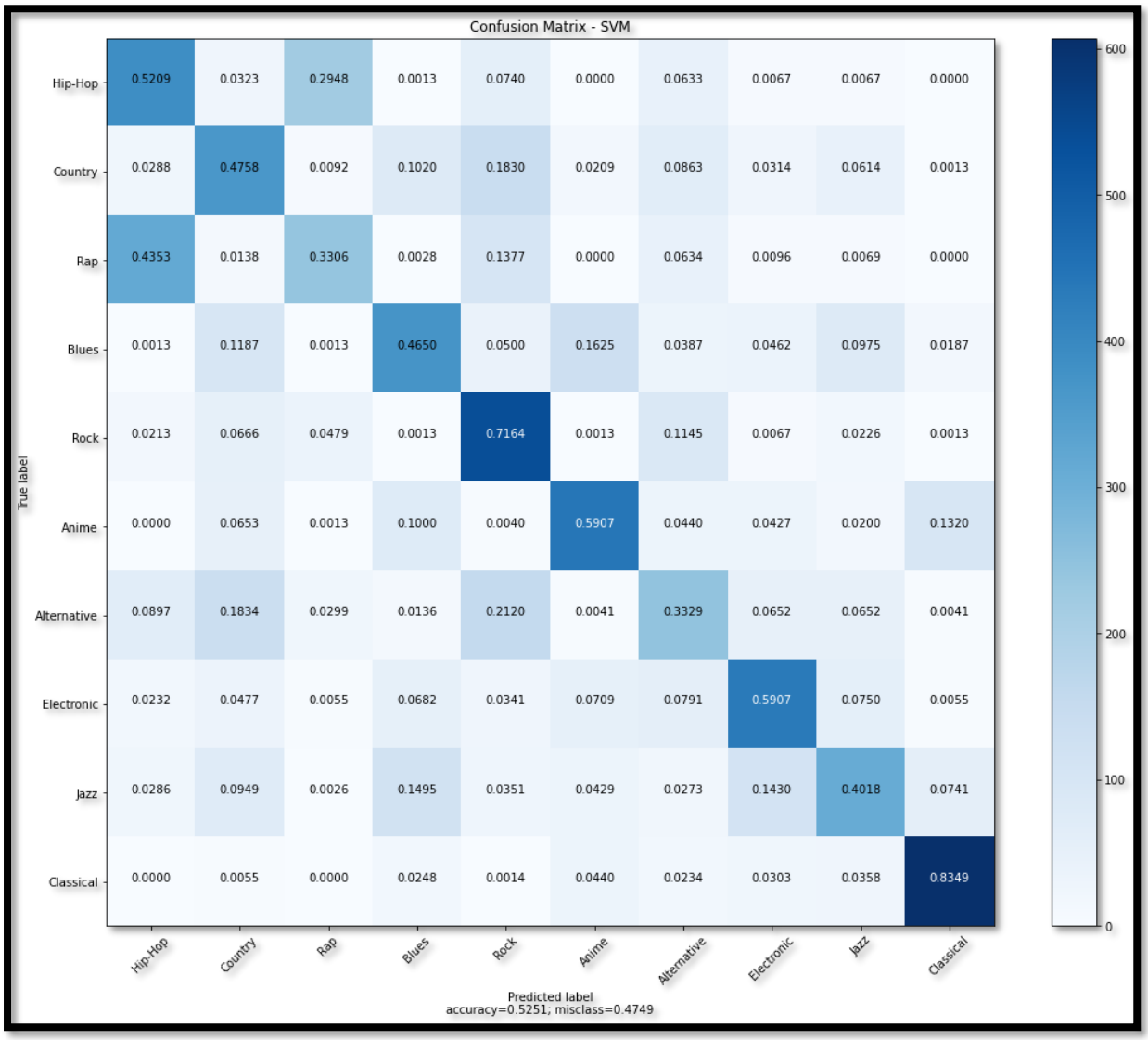
## STEP 7

At this stage, we will do feature elimination. I'm using thresholding to do this. We will observe that different tresholding values yield different results. We are sifting through 10 manually selected features. Let's examine the results we obtained by keeping the parameters of our SVM , Logistic Regression , ANN models constant and only changing the threshold values.

### STEP 7.1

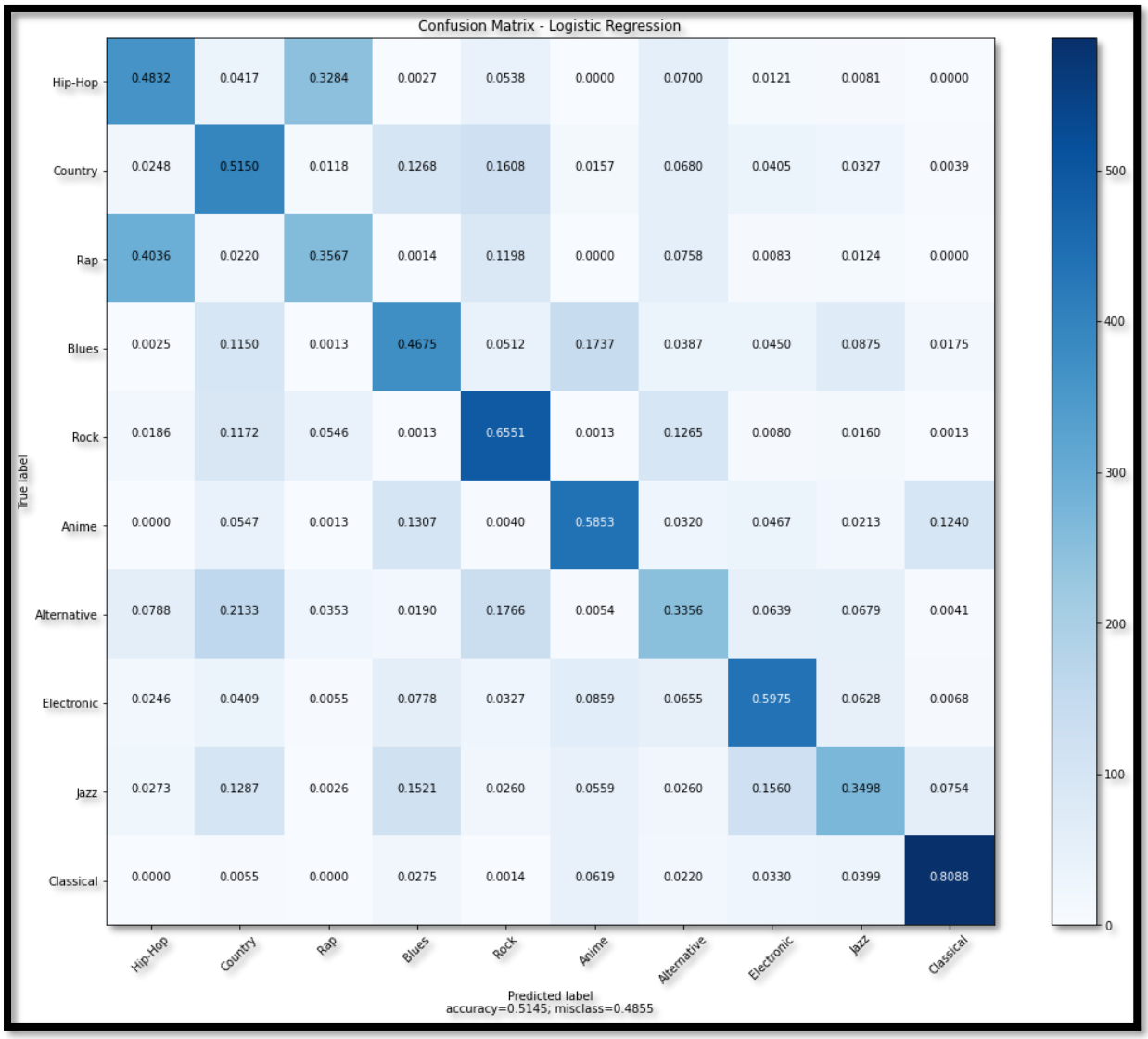
Threshold Value: .001 -- Chosen 9 out of 10 features.

```
print("-----------------------------------------------")

print(X.shape)

sel = VarianceThreshold(threshold=(.001))
sel.fit_transform(X)

X = sel.fit_transform(X)

print(X.shape)


print("-----------------------------------------------")
```



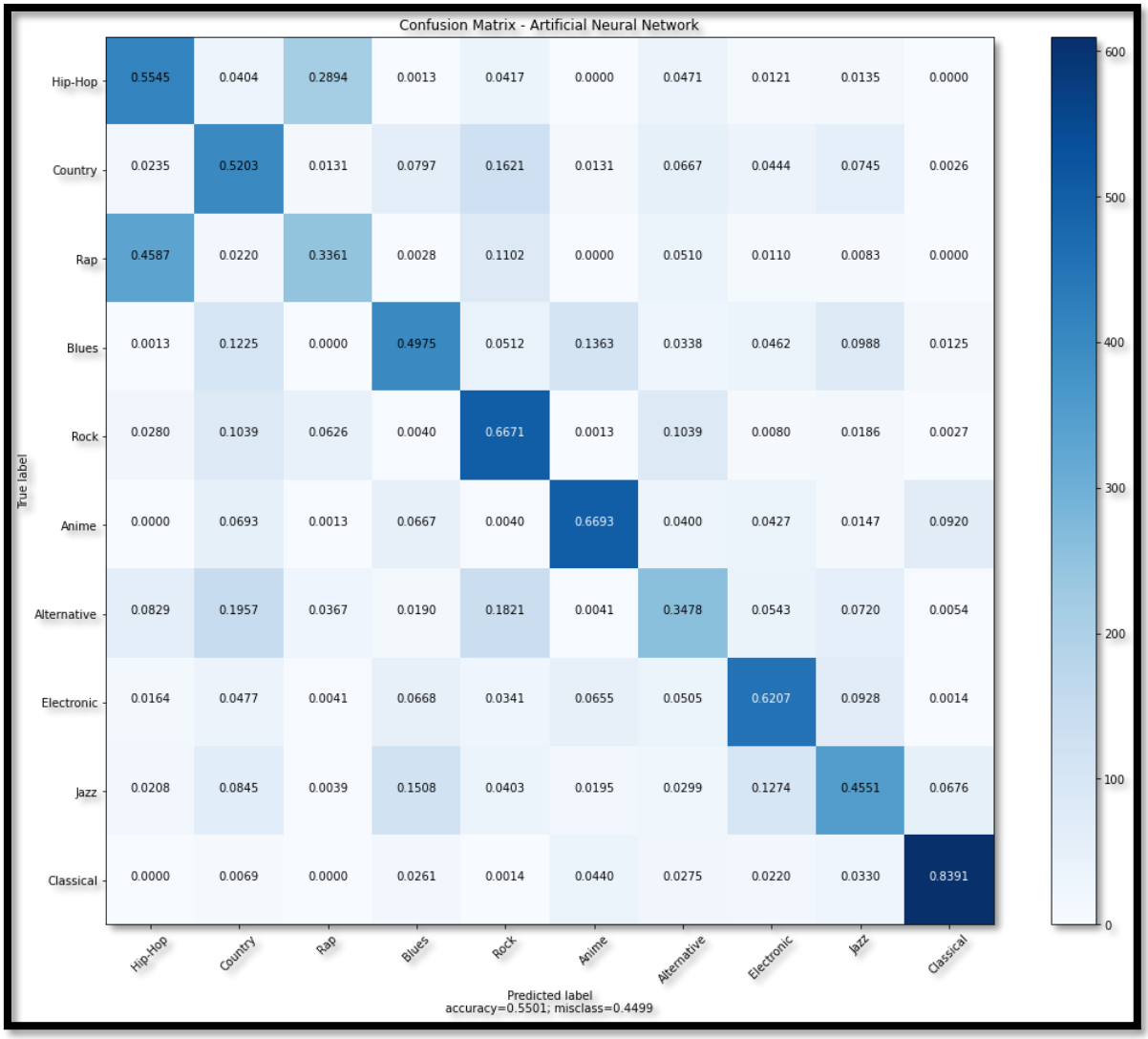
Predicted label
accuracy=0.5251; misclass=0.4749

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Hip-Hop | 0.46 | 0.52 | 0.49 | 743 |
| Country | 0.43 | 0.48 | 0.45 | 765 |
| Rap | 0.45 | 0.33 | 0.38 | 726 |
| Blues | 0.52 | 0.47 | 0.49 | 800 |
| Rock | 0.50 | 0.72 | 0.59 | 751 |
| Anime | 0.62 | 0.59 | 0.61 | 750 |
| Alternative | 0.38 | 0.33 | 0.35 | 736 |
| Electronic | 0.60 | 0.59 | 0.59 | 733 |
| Jazz | 0.51 | 0.40 | 0.45 | 769 |
| Classical | 0.77 | 0.83 | 0.80 | 727 |
| accuracy | | | 0.53 | 7500 |
| macro avg | 0.52 | 0.53 | 0.52 | 7500 |
| weighted avg | 0.52 | 0.53 | 0.52 | 7500 |



Predicted label
accuracy=0.5145; misclass=0.4855

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Hip-Hop | 0.46 | 0.48 | 0.47 | 743 |
| Country | 0.41 | 0.52 | 0.46 | 765 |
| Rap | 0.44 | 0.36 | 0.39 | 726 |
| Blues | 0.48 | 0.47 | 0.47 | 800 |
| Rock | 0.51 | 0.66 | 0.57 | 751 |
| Anime | 0.59 | 0.59 | 0.59 | 750 |
| Alternative | 0.39 | 0.34 | 0.36 | 736 |
| Electronic | 0.58 | 0.60 | 0.59 | 733 |
| Jazz | 0.51 | 0.35 | 0.41 | 769 |
| Classical | 0.77 | 0.81 | 0.79 | 727 |
| accuracy | | | 0.51 | 7500 |
| macro avg | 0.51 | 0.52 | 0.51 | 7500 |
| weighted avg | 0.51 | 0.51 | 0.51 | 7500 |



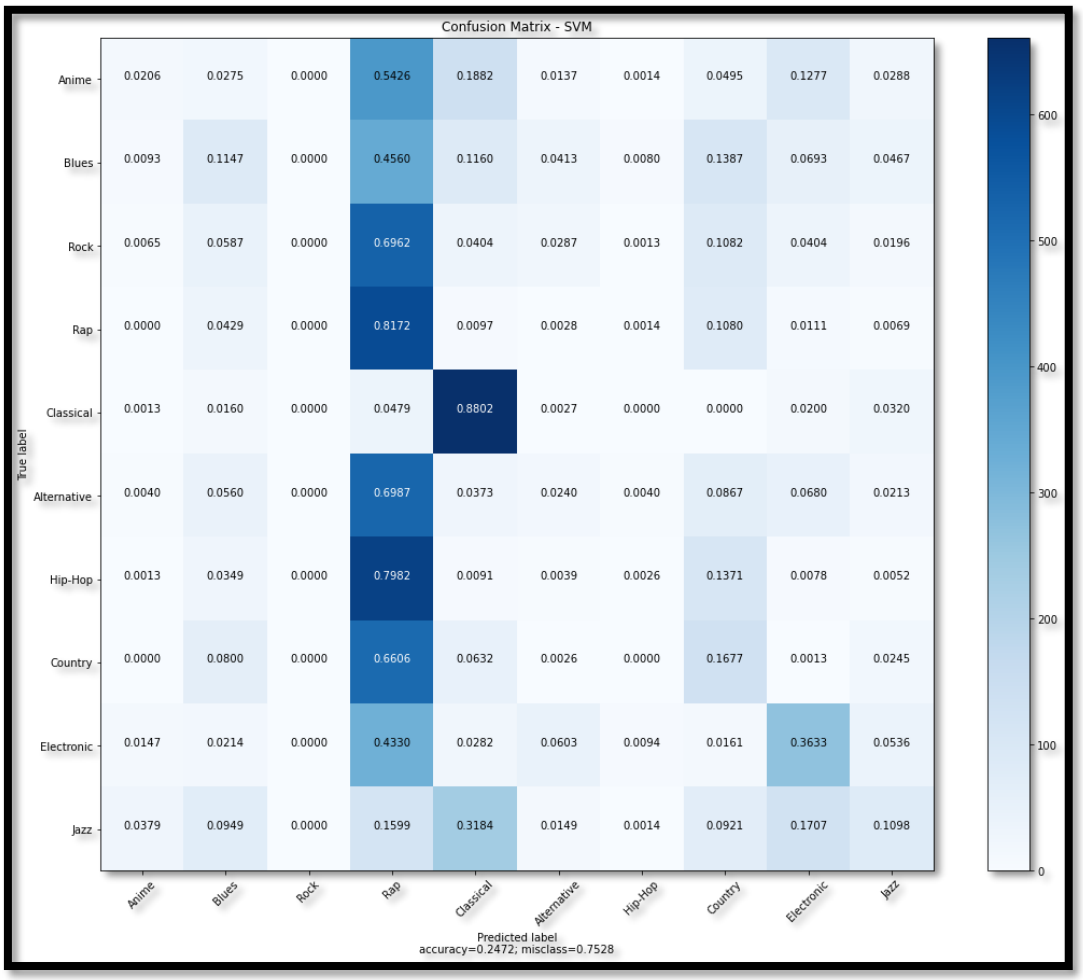
Predicted label
accuracy=0.5501; misclass=0.4499

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Hip-Hop | 0.47 | 0.55 | 0.51 | 743 |
| Country | 0.43 | 0.52 | 0.47 | 765 |
| Rap | 0.44 | 0.34 | 0.38 | 726 |
| Blues | 0.56 | 0.50 | 0.53 | 800 |
| Rock | 0.52 | 0.67 | 0.58 | 751 |
| Anime | 0.70 | 0.67 | 0.68 | 750 |
| Alternative | 0.43 | 0.35 | 0.38 | 736 |
| Electronic | 0.62 | 0.62 | 0.62 | 733 |
| Jazz | 0.52 | 0.46 | 0.49 | 769 |
| Classical | 0.81 | 0.84 | 0.83 | 727 |
| accuracy | | | 0.55 | 7500 |
| macro avg | 0.55 | 0.55 | 0.55 | 7500 |
| weighted avg | 0.55 | 0.55 | 0.55 | 7500 |

STEP 7.1

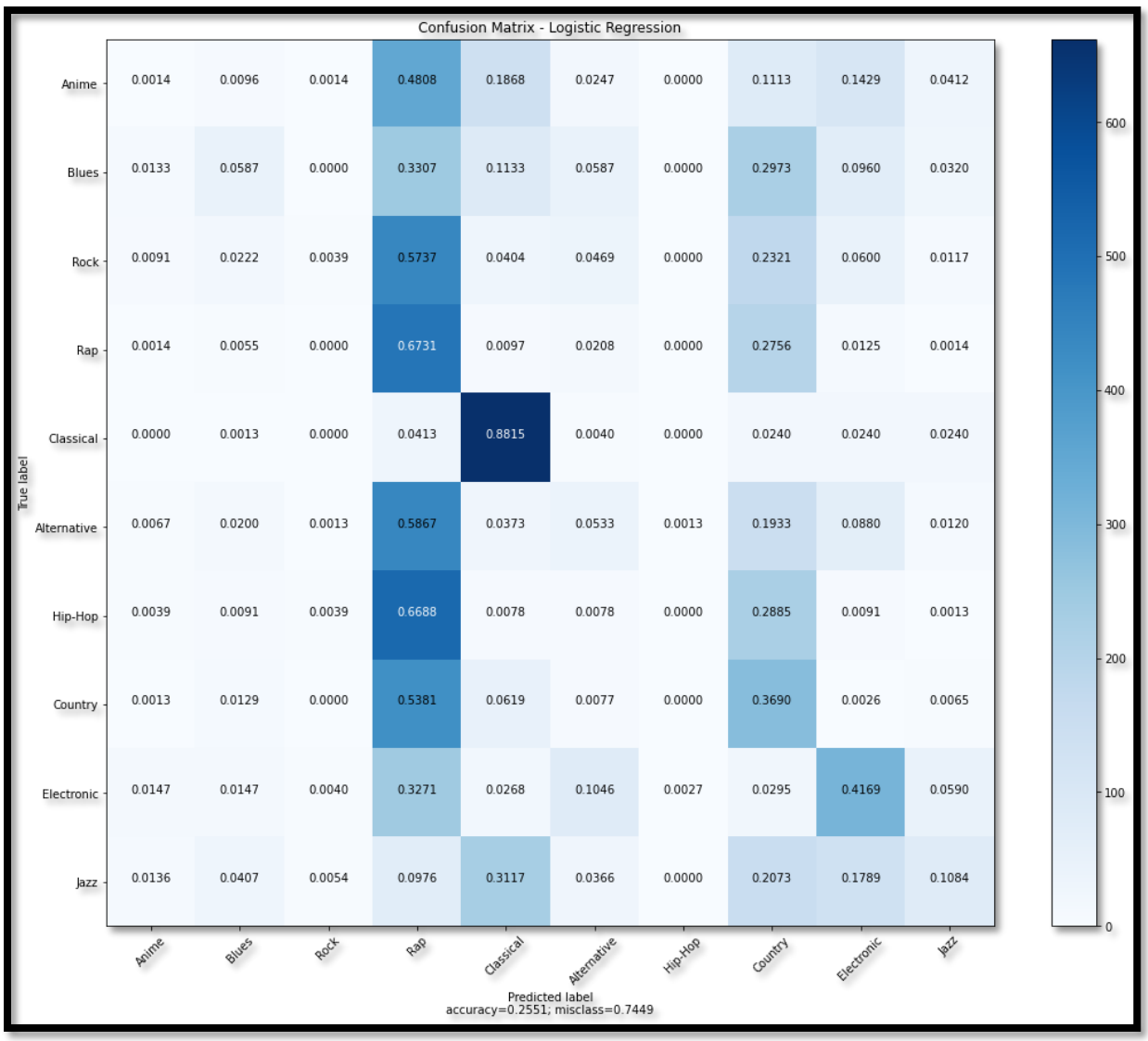
Threshold Value: .1 -- Chosen 2 out of 10 features.

```
print("------------------------------------------------")

print(X.shape)

sel = VarianceThreshold(threshold=(.1))
sel.fit_transform(X)

X = sel.fit_transform(X)

print(X.shape)


print("------------------------------------------------")
```


Confusion Matrix - SVM
Anime 0.0206 0.0275 0.0000 0.5426 0.1882 0.0137 0.0014 0.0495 0.1277 0.0288
Blues 0.0093 0.1147 0.0000 0.4560 0.1160 0.0413 0.0080 0.1387 0.0693 0.0467
Rock 0.0065 0.0587 0.0000 0.6962 0.0404 0.0287 0.0013 0.1082 0.0404 0.0196
Rap 0.0000 0.0429 0.0000 0.8172 0.0097 0.0028 0.0014 0.1080 0.0111 0.0069
Classical 0.0013 0.0160 0.0000 0.0479 0.8802 0.0027 0.0000 0.0000 0.0200 0.0320
Alternative 0.0040 0.0560 0.0000 0.6987 0.0373 0.0240 0.0040 0.0867 0.0680 0.0213
Hip-Hop 0.0013 0.0349 0.0000 0.7982 0.0091 0.0039 0.0026 0.1371 0.0078 0.0052
Country 0.0000 0.0800 0.0000 0.6606 0.0632 0.0026 0.0000 0.1677 0.0013 0.0245
Electronic 0.0147 0.0214 0.0000 0.4330 0.0282 0.0603 0.0094 0.0161 0.3633 0.0536
Jazz 0.0379 0.0949 0.0000 0.1599 0.3184 0.0149 0.0014 0.0921 0.1707 0.1098
True label
Anime Blues Rock Rap Classical Alternative Hip-Hop Country Electronic Jazz
Predicted label
accuracy=0.2472; misclass=0.7528
600 500 400 300 200 100 0

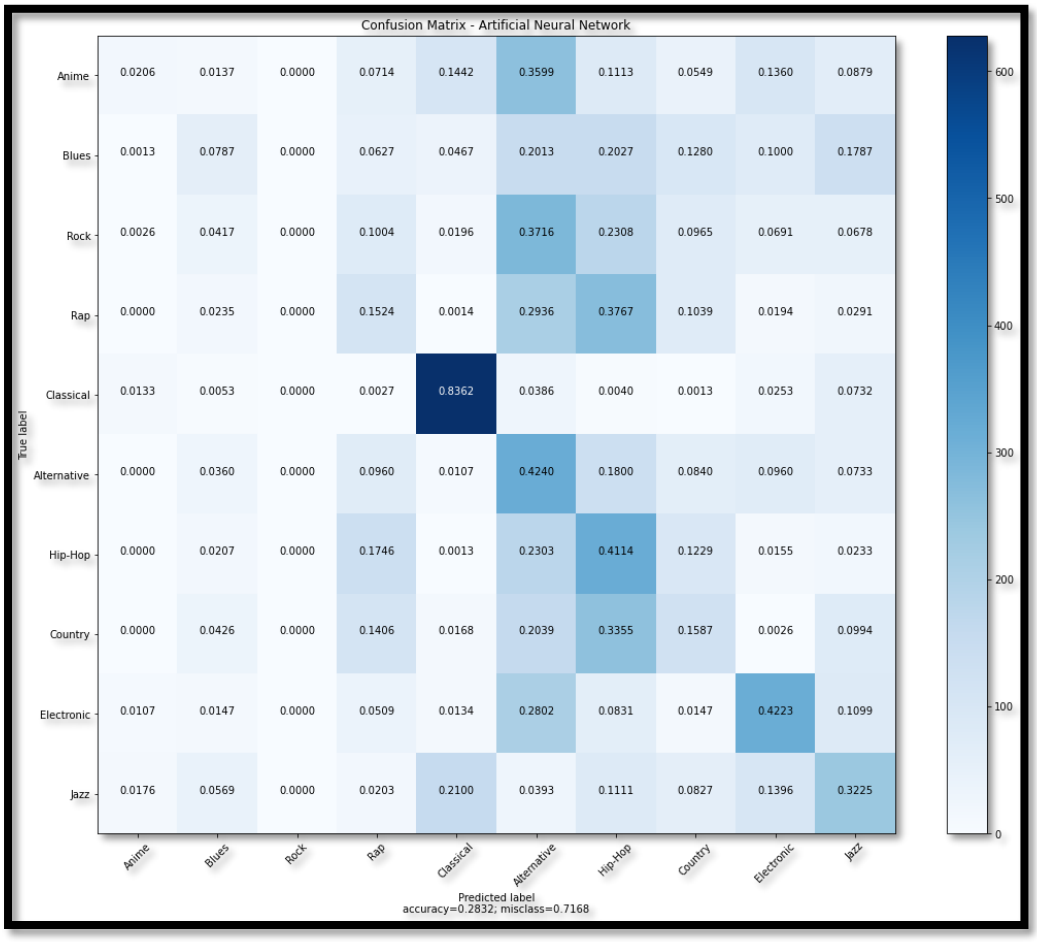
accuracy=0.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anime | 0.21 | 0.02 | 0.04 | 728 |
| Blues | 0.21 | 0.11 | 0.15 | 750 |
| Rock | 0.00 | 0.00 | 0.00 | 767 |
| Rap | 0.15 | 0.82 | 0.25 | 722 |
| Classical | 0.52 | 0.88 | 0.66 | 751 |
| Alternative | 0.12 | 0.02 | 0.04 | 750 |
| Hip-Hop | 0.09 | 0.00 | 0.01 | 773 |
| Country | 0.19 | 0.17 | 0.18 | 775 |
| Electronic | 0.41 | 0.36 | 0.39 | 746 |
| Jazz | 0.31 | 0.11 | 0.16 | 738 |
| accuracy | | | 0.25 | 7500 |
| macro avg | 0.22 | 0.25 | 0.19 | 7500 |
| weighted avg | 0.22 | 0.25 | 0.19 | 7500 |



Predicted label
accuracy=0.2551; misclass=0.7449

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anime | 0.02 | 0.00 | 0.00 | 728 |
| Blues | 0.30 | 0.06 | 0.10 | 750 |
| Rock | 0.20 | 0.00 | 0.01 | 767 |
| Rap | 0.15 | 0.67 | 0.25 | 722 |
| Classical | 0.53 | 0.88 | 0.66 | 751 |
| Alternative | 0.15 | 0.05 | 0.08 | 750 |
| Hip-Hop | 0.00 | 0.00 | 0.00 | 773 |
| Country | 0.19 | 0.37 | 0.25 | 775 |
| Electronic | 0.41 | 0.42 | 0.41 | 746 |
| Jazz | 0.36 | 0.11 | 0.17 | 738 |
| accuracy | | | 0.26 | 7500 |
| macro avg | 0.23 | 0.26 | 0.19 | 7500 |
| weighted avg | 0.23 | 0.26 | 0.19 | 7500 |



Predicted label
accuracy=0.2832; misclass=0.7168

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anime | 0.31 | 0.02 | 0.04 | 728 |
| Blues | 0.24 | 0.08 | 0.12 | 750 |
| Rock | 0.00 | 0.00 | 0.00 | 767 |
| Rap | 0.17 | 0.15 | 0.16 | 722 |
| Classical | 0.65 | 0.84 | 0.73 | 751 |
| Alternative | 0.17 | 0.42 | 0.25 | 750 |
| Hip-Hop | 0.21 | 0.41 | 0.27 | 773 |
| Country | 0.19 | 0.16 | 0.17 | 775 |
| Electronic | 0.41 | 0.42 | 0.42 | 746 |
| Jazz | 0.30 | 0.32 | 0.31 | 738 |
| accuracy | | | 0.28 | 7500 |
| macro avg | 0.26 | 0.28 | 0.25 | 7500 |
| weighted avg | 0.26 | 0.28 | 0.25 | 7500 |

STEP 7.1 and 7.2 - RESULT:

When we change the Threshold value, we see that the efficiency we get decreases, as can be seen when the current outputs are examined. When we reduce the number of features, efficient learning does not occur.

Therefore, I will continue to the next stages, provided that the threshold value is .001, that is, it is evaluated over 9 features.

## STEP 8

We labeled our music genres.

```
# labels
df_y = df.loc[:, "music_genre"].values

# encode each label to the integer value
# ex; Blues --> 0 etc.
le = preprocessing.LabelEncoder()
y = le.fit_transform(df_y)
y_text = df_y.tolist()

print(y)
print(y.shape)
for value in list(set(y)):
    print("{} --> {}".format(value, le.inverse_transform([value])))
```

STEP 9

We have reserved part of our dataset for testing. We will not be training with this part.

```
# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y_text, test_size=0.15, random_state=42)
```

```
print(X_train.shape, X_test.shape, len(y_train), len(y_test))
```

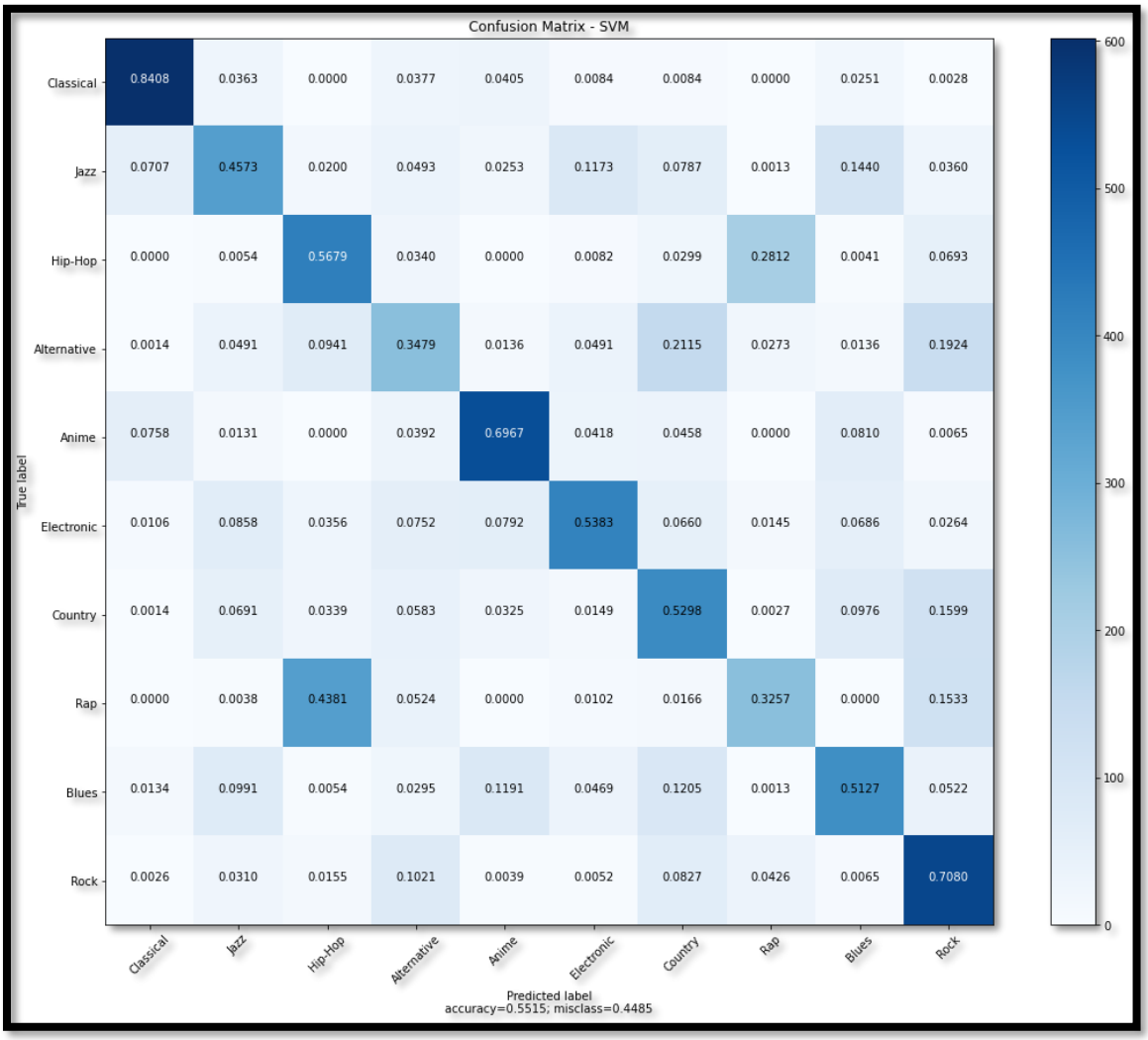
```
(42500, 9) (7500, 9) 42500 7500
```

STEP 10

Training part. I observed that we obtained different results by changing the parameters in the models we used.

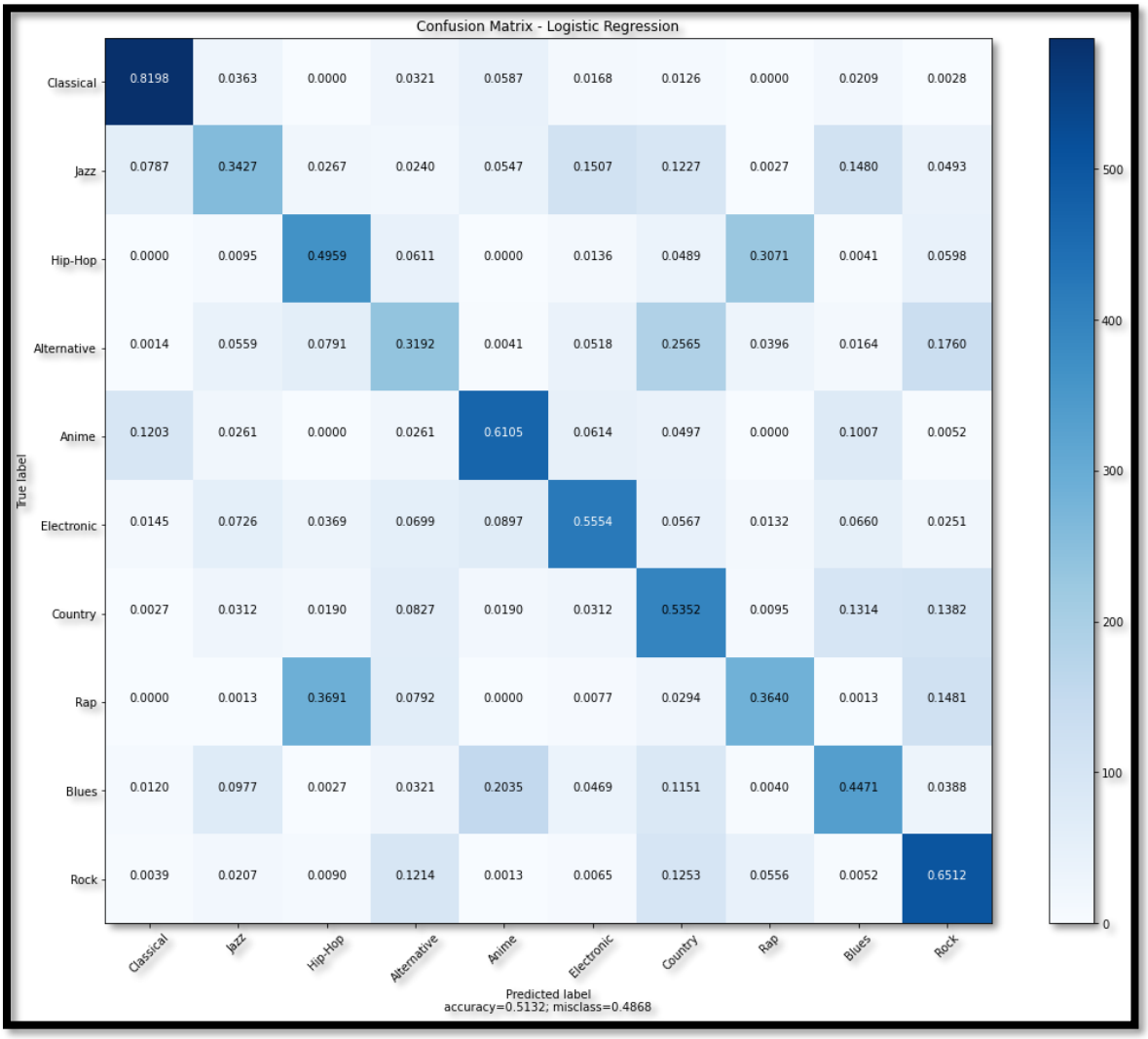
STEP 10.1

```
In [25]: # Model training
         ###############
         print("SVM Model is starting")
         svm_model = SVC(kernel='poly',degree=4).fit(X_train, y_train)
         print("Logistic Regression Model is starting")
         logistic_model = LogisticRegression().fit(X_train, y_train)
         print("ANN Model is starting")
         mlp_model = MLPClassifier(activation='identity', solver='lbfgs', alpha=1e-5).fit(X_train, y_train)
```



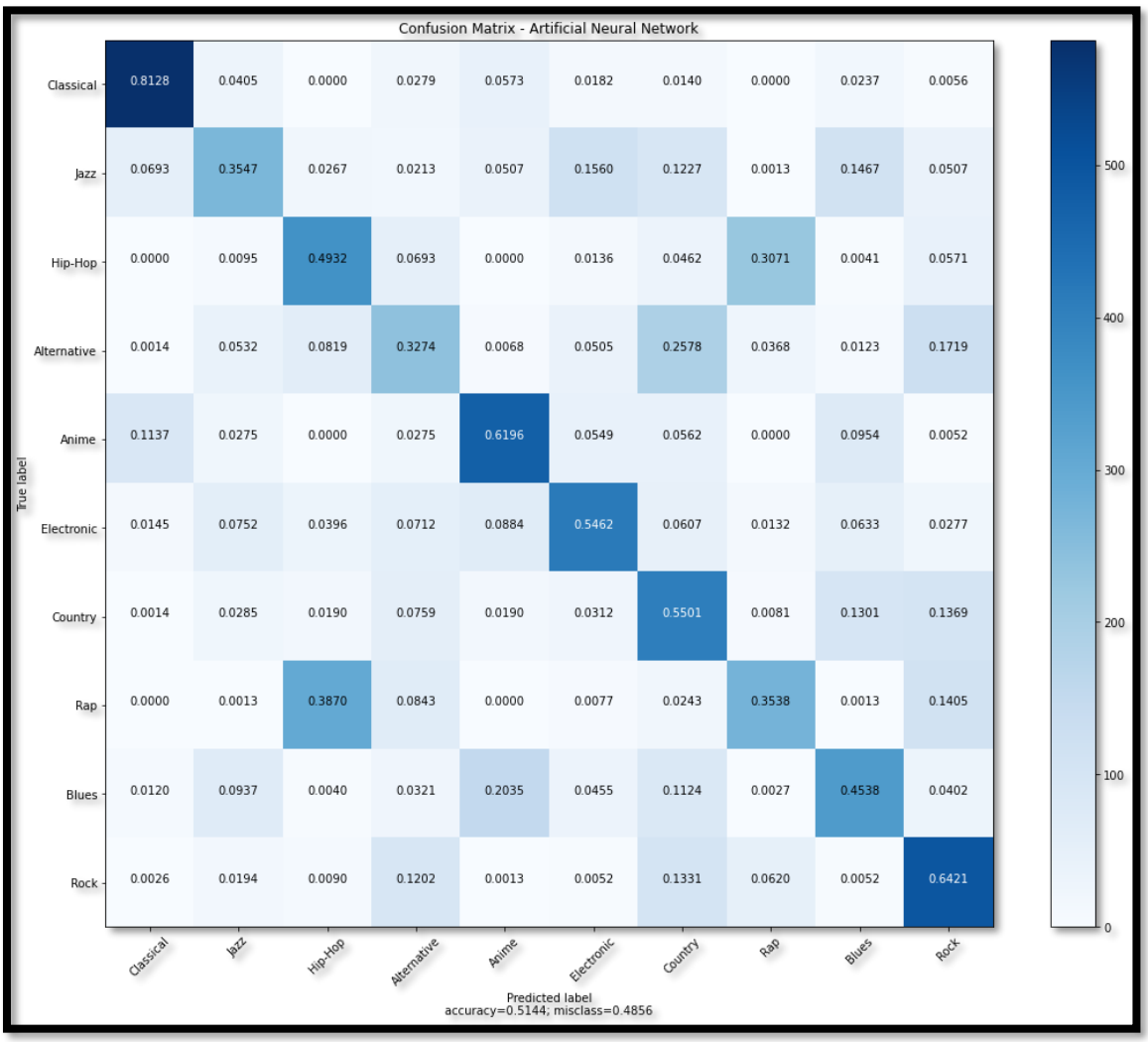

accuracy=0.5515; misclass=0.4485

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Classical | 0.82 | 0.84 | 0.83 | 716 |
| Jazz | 0.54 | 0.46 | 0.49 | 750 |
| Hip-Hop | 0.46 | 0.57 | 0.51 | 736 |
| Alternative | 0.41 | 0.35 | 0.38 | 733 |
| Anime | 0.69 | 0.70 | 0.70 | 765 |
| Electronic | 0.64 | 0.54 | 0.59 | 758 |
| Country | 0.44 | 0.53 | 0.48 | 738 |
| Rap | 0.48 | 0.33 | 0.39 | 783 |
| Blues | 0.54 | 0.51 | 0.52 | 747 |
| Rock | 0.51 | 0.71 | 0.59 | 774 |
| accuracy | | | 0.55 | 7500 |
| macro avg | 0.55 | 0.55 | 0.55 | 7500 |
| weighted avg | 0.55 | 0.55 | 0.55 | 7500 |




Predicted label
accuracy=0.5132; misclass=0.4868

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Classical | 0.77 | 0.82 | 0.79 | 716 |
| Jazz | 0.50 | 0.34 | 0.41 | 750 |
| Hip-Hop | 0.47 | 0.50 | 0.48 | 736 |
| Alternative | 0.37 | 0.32 | 0.34 | 733 |
| Anime | 0.59 | 0.61 | 0.60 | 765 |
| Electronic | 0.59 | 0.56 | 0.57 | 758 |
| Country | 0.39 | 0.54 | 0.45 | 738 |
| Rap | 0.47 | 0.36 | 0.41 | 783 |
| Blues | 0.47 | 0.45 | 0.46 | 747 |
| Rock | 0.51 | 0.65 | 0.57 | 774 |
| accuracy | | | 0.51 | 7500 |
| macro avg | 0.51 | 0.51 | 0.51 | 7500 |
| weighted avg | 0.51 | 0.51 | 0.51 | 7500 |

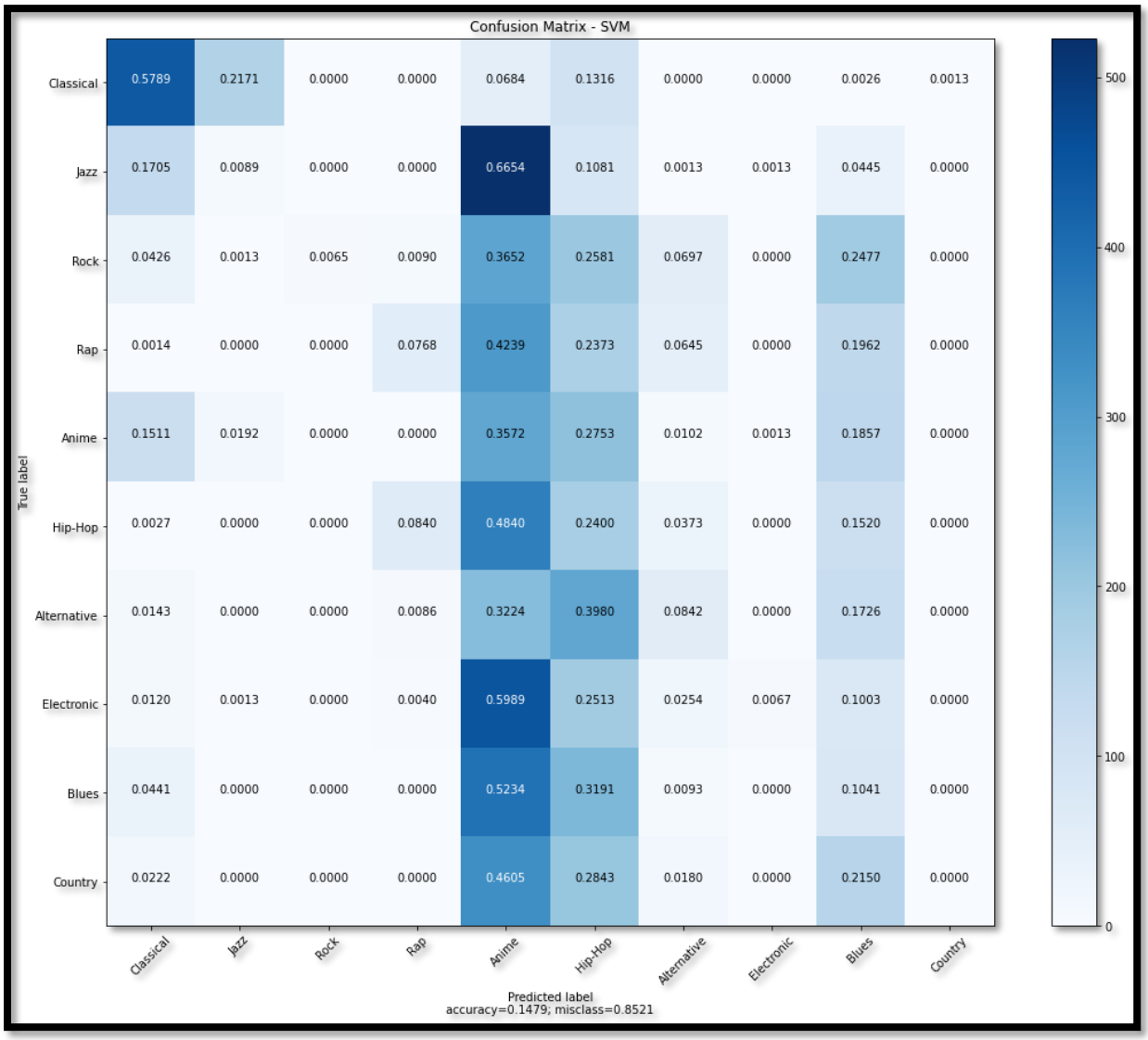



Predicted label
accuracy=0.5144; misclass=0.4856

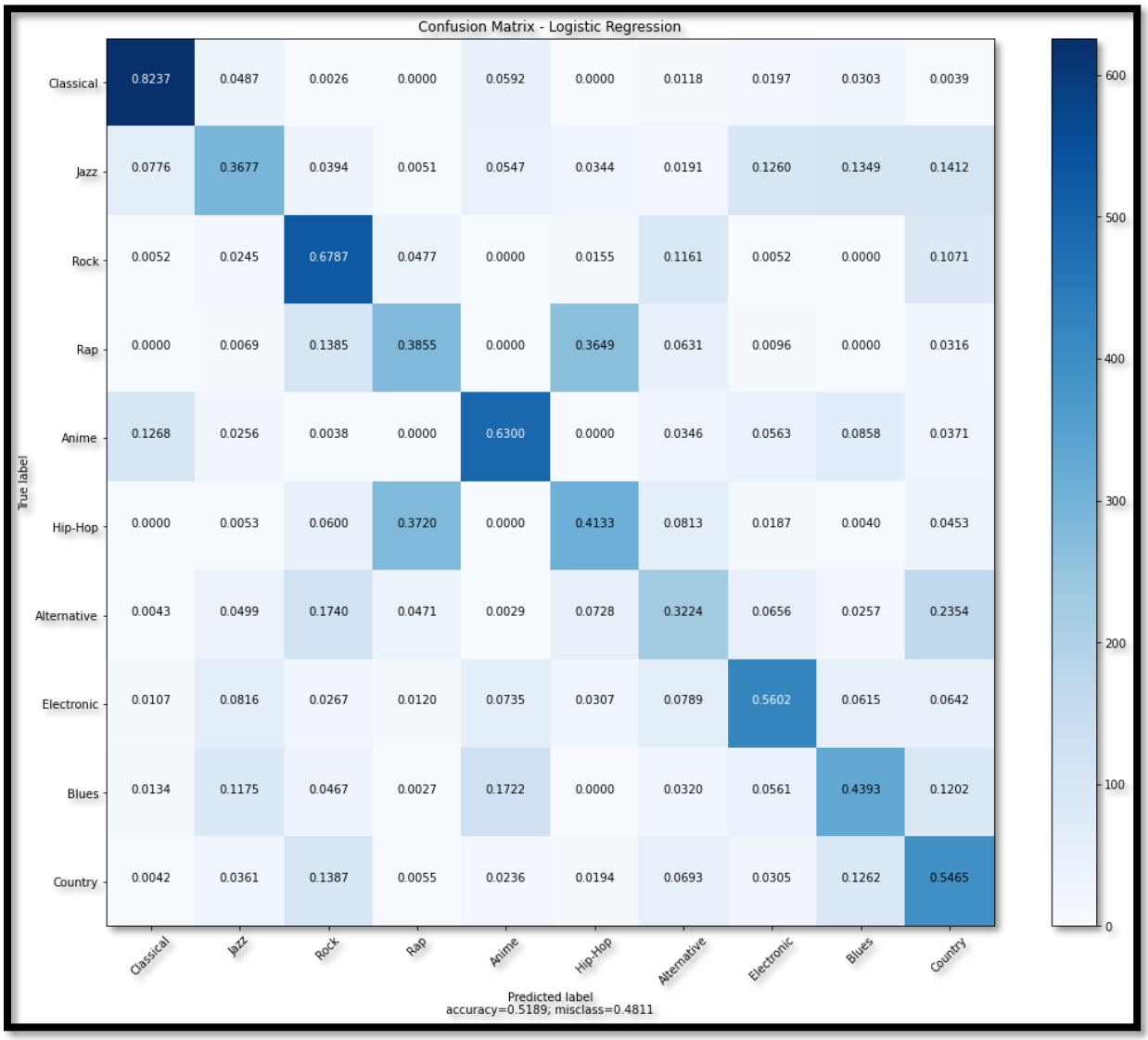
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Classical | 0.78 | 0.81 | 0.80 | 716 |
| Jazz | 0.51 | 0.35 | 0.42 | 750 |
| Hip-Hop | 0.45 | 0.49 | 0.47 | 736 |
| Alternative | 0.37 | 0.33 | 0.35 | 733 |
| Anime | 0.60 | 0.62 | 0.61 | 765 |
| Electronic | 0.59 | 0.55 | 0.57 | 758 |
| Country | 0.40 | 0.55 | 0.46 | 738 |
| Rap | 0.46 | 0.35 | 0.40 | 783 |
| Blues | 0.48 | 0.45 | 0.47 | 747 |
| Rock | 0.51 | 0.64 | 0.57 | 774 |
| accuracy | | | 0.51 | 7500 |
| macro avg | 0.52 | 0.52 | 0.51 | 7500 |
| weighted avg | 0.52 | 0.51 | 0.51 | 7500 |

STEP 10.2

```
# Model training
################
print("SVM Model is starting")
svm_model = SVC(kernel='sigmoid').fit(X_train, y_train)
print("Logistic Regression Model is starting")
logistic_model = LogisticRegression().fit(X_train, y_train)
print("ANN Model is starting")
mlp_model = MLPClassifier(activation='relu', solver='lbfgs', alpha=1e-5).fit(X_train, y_train)
```

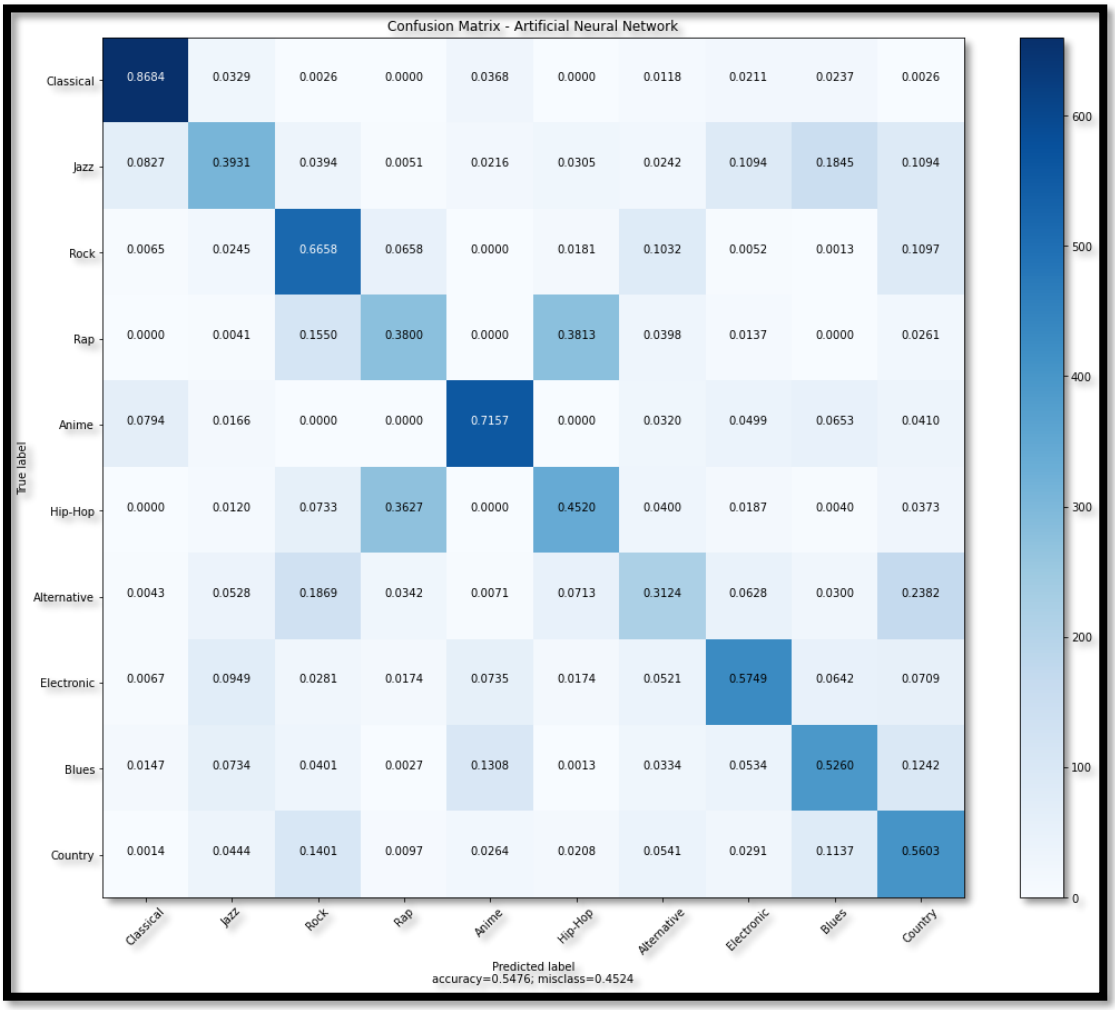


| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Classical | 0.55 | 0.58 | 0.57 | 760 |
| Jazz | 0.04 | 0.01 | 0.01 | 786 |
| Rock | 1.00 | 0.01 | 0.01 | 775 |
| Rap | 0.41 | 0.08 | 0.13 | 729 |
| Anime | 0.09 | 0.36 | 0.14 | 781 |
| Hip-Hop | 0.10 | 0.24 | 0.14 | 750 |
| Alternative | 0.25 | 0.08 | 0.13 | 701 |
| Electronic | 0.71 | 0.01 | 0.01 | 748 |
| Blues | 0.07 | 0.10 | 0.09 | 749 |
| Country | 0.00 | 0.00 | 0.00 | 721 |
| accuracy | | | 0.15 | 7500 |
| macro avg | 0.32 | 0.15 | 0.12 | 7500 |
| weighted avg | 0.32 | 0.15 | 0.12 | 7500 |



Predicted label
accuracy=0.5189; misclass=0.4811

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Classical | 0.77 | 0.82 | 0.80 | 760 |
| Jazz | 0.49 | 0.37 | 0.42 | 786 |
| Rock | 0.53 | 0.68 | 0.60 | 775 |
| Rap | 0.43 | 0.39 | 0.41 | 729 |
| Anime | 0.63 | 0.63 | 0.63 | 781 |
| Hip-Hop | 0.44 | 0.41 | 0.43 | 750 |
| Alternative | 0.37 | 0.32 | 0.35 | 701 |
| Electronic | 0.59 | 0.56 | 0.57 | 748 |
| Blues | 0.48 | 0.44 | 0.46 | 749 |
| Country | 0.40 | 0.55 | 0.46 | 721 |
| accuracy | | | 0.52 | 7500 |
| macro avg | 0.51 | 0.52 | 0.51 | 7500 |
| weighted avg | 0.52 | 0.52 | 0.51 | 7500 |



Predicted label
accuracy=0.5476; misclass=0.4524

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Classical | 0.81 | 0.87 | 0.84 | 760 |
| Jazz | 0.54 | 0.39 | 0.45 | 786 |
| Rock | 0.52 | 0.67 | 0.58 | 775 |
| Rap | 0.43 | 0.38 | 0.40 | 729 |
| Anime | 0.72 | 0.72 | 0.72 | 781 |
| Hip-Hop | 0.46 | 0.45 | 0.46 | 750 |
| Alternative | 0.43 | 0.31 | 0.36 | 701 |
| Electronic | 0.61 | 0.57 | 0.59 | 748 |
| Blues | 0.52 | 0.53 | 0.52 | 749 |
| Country | 0.42 | 0.56 | 0.48 | 721 |
| accuracy | | | 0.55 | 7500 |
| macro avg | 0.54 | 0.54 | 0.54 | 7500 |
| weighted avg | 0.55 | 0.55 | 0.54 | 7500 |

!!! Model comparison will be made in the result part.

## STEP 11

I make specific edits to show the results on confusion_matrix.

## STEP 12

We obtain the confusion_matrix with the relevant codes and finish our process.

SVM:

```
cm_svm = confusion_matrix(y_test, y_pred_svm, labels=list(music_classes))
plot_confusion_matrix(cm_svm, list(music_classes), "Confusion Matrix - SVM")

print(classification_report(y_test, y_pred_svm, labels=list(music_classes)))
```

LOGISTIC

```
cm_lr = confusion_matrix(y_test, y_pred_lr, labels=list(music_classes))
plot_confusion_matrix(cm_lr, list(music_classes), "Confusion Matrix - Logistic Regression")

print(classification_report(y_test, y_pred_lr, labels=list(music_classes)))
```

ANN

```
cm_ann = confusion_matrix(y_test, y_pred_ann, labels=list(music_classes))
plot_confusion_matrix(cm_ann, list(music_classes), "Confusion Matrix - Artificial Neural Network")

print(classification_report(y_test, y_pred_ann, labels=list(music_classes)))
```

## RESULT

First of all, when we over-eliminate as a result of the threshold adjustments made for feature elimination, we see that it has a negative effect on all models. The model is underfitting because it cannot complete its learning. The results of the tests we made on our model with the test set give bad results compared to the models made with more features.

After determining the Threshold value as a constant variable, we will examine our models. We need to know some of the explanations under the con_matix we use to make these reviews.

Accuracy is a metric that is widely used to measure the success of a model but does not appear to be sufficient on its own.

$$\frac{TP + TN}{TP + FP + TN + FN}$$

Precision, on the other hand, shows how many of the values we estimated as Positive are actually Positive.

$$\textbf{\textit{Precision}} = \frac{TP}{TP + FP}$$

Recall, on the other hand, is a metric that shows how much of the operations we need to estimate as Positive, we estimate as Positive.

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1 Score value shows us the harmonic mean of Precision and Recall values.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Accordingly, I will make our general evaluation over the f1 score.

10.1 :

The values we get when we use the 4th degree polynomial function for SVM and the identity function for ANN:

SVM: 0.55

LOG: 0.51

ANN: 0.51

We can say that it provides the most successful result for svm under these conditions. In this context, let's make an experiment with how the change of the polynomial function degree will result for SVM. The process takes too long, so it is inefficient..

```
In [*]: # Model training
        ################
        print("SVM Model is starting")
        svm_model = SVC(kernel='poly',degree=9).fit(X_train, y_train)
        print("Logistic Regression Model is starting")
        logistic_model = LogisticRegression().fit(X_train, y_train)
        print("ANN Model is starting")
        mlp_model = MLPClassifier(activation='identity', solver='lbfgs', alpha=1e-5).fit(X_train, y_train)

        SVM Model is starting
```

10.2 :

Let's compare the data we get when we choose sigmoid for SVM and Relu function for ANN.

SVM: 0.12

LOG: 0.51

ANN: 0.54

As can be seen, when sigmoid is used for svm, it gives an extremely bad result. because while the sigmoid function has two distinctions, 0 and 1, ours is more than two. It seems that it is not suitable for our data. In this combination, we achieved the most successful result on our ANN model.

## GENERALLY

When we compare the SVM models, we see that the distribution of our current data is more suitable for the polynomial distribution.

When we compare our ANN Models, it has been observed that the accuracy value of the model has increased since it shows a distribution suitable for the ReLU function rather than the unit function.

We kept the logistic regression values constant at the last stage. It was observed that when we enlarged the trheshold values only, a decrease from 0.51 to 0.19 was observed. This shows that when applying this teaching model, it is efficient when more features are used.

It was a project in which we prepared the data beforehand, then performed the ferature elimination and normalization processes and divided the data, first applying the learning and then testing stages, observing the results on the matrix and presenting this matrix with different evaluation criteria.