Classifying albums based on dynamic range values

Sander de Jong
s2096943

January 14, 2020

1 Introduction

This project tries to classify albums based on the loudness of the songs on the album. The loudness is measured by the dynamic range values of the songs. The reason why dynamic range values could be a good classifier is found in a phenomenon called the "Loudness War". In order to make a song stand out from all others, producers and mastering engineers use all kinds of tricks to make songs sound louder when they are played at similar volume levels. This is an old phenomenon that originated in the 40s of the last century when 7-inch singles were cut to sound louder when played in jukeboxes or when songs were proposed to A&R managers of record labels[1]. When a song is louder than the competition, it makes it stand out and research has shown that humans think louder songs sound better[2].

Although it is an old phenomenon, the options to make vinyl records sound louder are limited. Excessive compression applied to vinyl records render the medium unplayable, because the needle pops out of the groove[3]. With the introduction of the compact disc (CD), these limitations were not an issue anymore and digital signal processing made it easier to apply compression to music. This made the issue more prominent, with the peak of the loudness war in the 00s. This might also be one of the reasons for the resurgence of vinyl in recent years. The perception that vinyl sounds better might have been caused by differences in mastering rather than differences in media used.

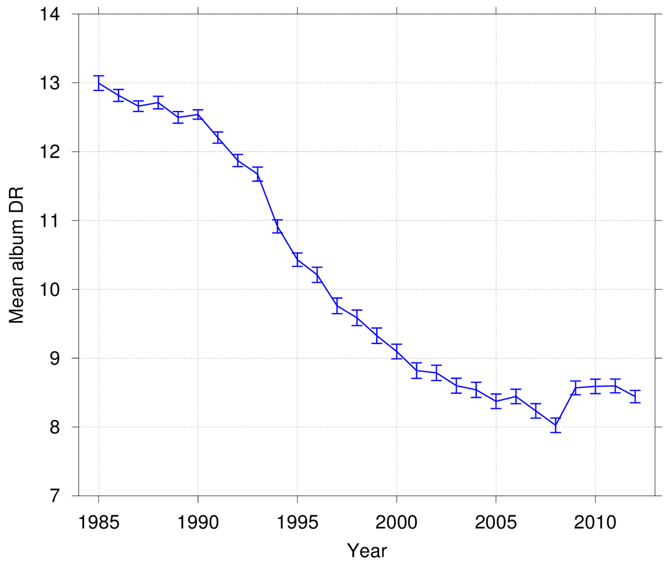
It became a topic of discussion following the release of Metallica's Death Magnetic album, which was heavily compressed[1]. The album was featured in the rhythm game Guitar Hero in which players try to play along with the record. The game used the unmastered multi-tracks to provide the sound of the different instruments that could be played in game. The files were ripped from the game and mixed together, resulting in a version of the album with far less compression. The excessive compression used and the fact that it concerned a popular band made the issue apparent to a larger audience. The downward trend seems to have stopped after that, seeing wider dynamic range in records from the last ten years.

Because the phenomenon became more apparent after the introduction of the CD and was most prevalent in the 90s and 00s, it is interesting to see the trend from 1985 onwards. Inspiration for this research was drawn from an article written in 2013[4] in which Jasper van Dorp visualised this trend based on 14680 values. The goal of this project is to see whether these values can be used as a classifier to predict the year an album was released in. Further inspection of the database raised the suspicion that the values of albums in different genres might vary enough for them to be reliable classifiers.

Therefore this project consists of two experiments with the following hypotheses:

- H1: The database of dynamic range values can be used as classifier to determine the year an album was released in based on its dynamic range value.
- H2: The dynamic range values of albums in different genres vary enough to use it as a classifier to determine the genre an album with a specific value belongs to.

Figure 1: The development of dynamic range values from 1985 to 2014 as measured by Jasper van Dorp



2 Methods

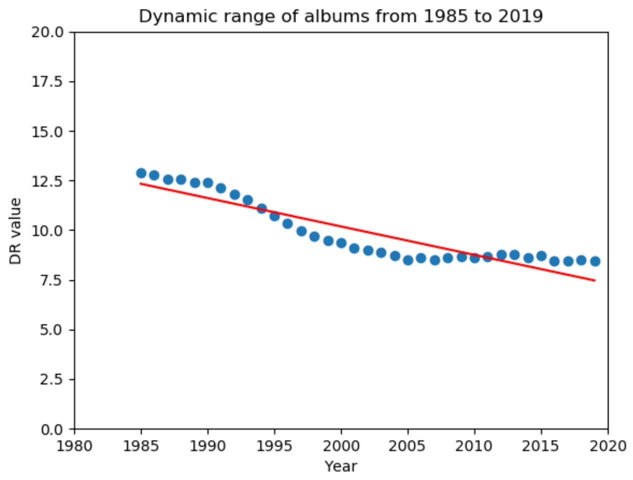
2.1 Algorithm used to calculate dynamic range

The algorithm that is used to calculate dynamic range values to be submitted to the database is DROffline[5]. It calculates the peak-to-loudness ratio or crest factor. It evaluates this crest factor throughout the whole song, resulting in a single value for the song. When the algorithm is presented with an album of songs, it outputs the mean dynamic range values of the songs on the album along with the individual song values.

2.2 Dataset

Hi-fi enthusiasts have collaboratively maintained a complete database of dynamic range values consisting of over 140000 albums (`http://dr.loudness-war.info`). Even though the database has thrived due to submissions by volunteers, the maintainer of the website doesn't provide an API to use the information and is not reachable through email or social media. Therefore the values had to be obtained by building a custom web scraper that saves the dynamic range values of all entries (for the code of the web scraper see the appendix). The data used in these experiments were scraped on 29 November 2019.

Figure 2: The development of dynamic range values from 1985 to 2019, based on the database entries



The mean DR values of the albums in the years from 1985 until 2019 are plotted in figure 2 using blue dots. It shows roughly the same curve as Jasper van Dorp's, although this database consists of a lot more values. There are some problems with the dataset that influence the results. Although over half of the changes in values throughout the years are statistically significant (see appendix A), the spread is not that large. Moreover, the values submitted to the database are rounded to integers instead of the decimal values that result from the algorithm used to obtain the dynamic range values.

Figure 3: The number of dynamic range values present in the database for the various years

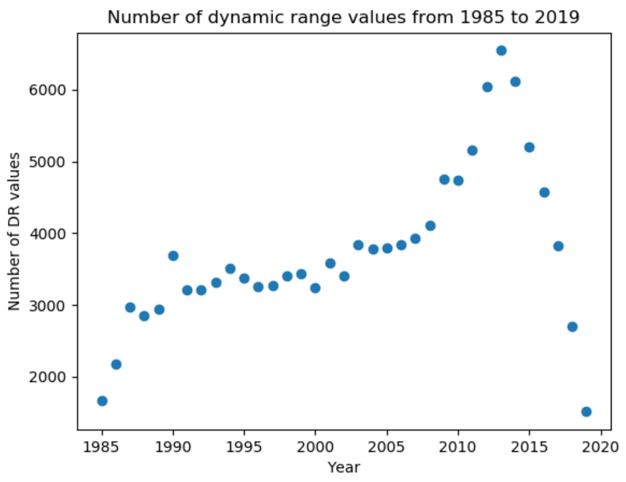
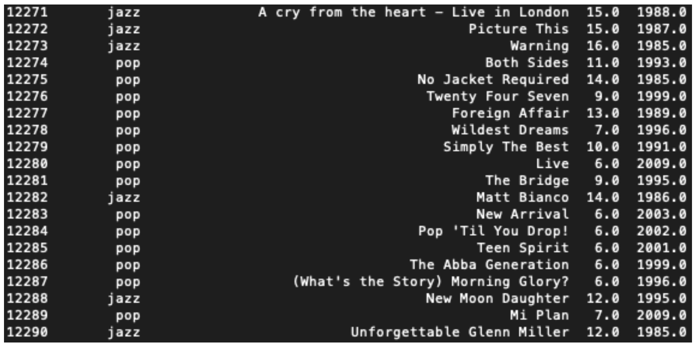


Figure 3 shows that although there are more values for recent years, all years contain enough data points for this experiment. Apart from the limitation in years used, vinyl records were excluded due to several reasons. As mentioned in the introduction, the loudness war isn't as prevalent as it is with CDs. Furthermore, the algorithm used doesn't work as well for vinyl records[6]. By only using CDs, the format is kept constant to test the other parameters.

2.2.1 Genre data

Because the dynamic range database doesn't contain information for the genres of the albums, the scraped data had to be cross-referenced with another database containing this information. After evaluating APIs of different services, Last.fm was chosen because it is quite lenient concerning rate limiting[7]. It specifies a limit of one API call per second, but the API was used with a higher rate without any problems. Another reason why this service was chosen is because it contains genre data for artists instead of individual songs. To reduce the time it takes to get the data, the decision was made to obtain genre data for artists instead of for individual albums. Although this is a simplification because artists can make albums in various genres, this limited the amount of items to be requested to 30000 instead of 140000. Even with this simplification, obtaining these 3000 artist-genre combinations took 8 hours. The genre data Last.fm provides is submitted by users. It maintains a list of genres assigned by users and a count of how many users chose this specific genre. The most popular genre for each artist was linked to its albums.

Figure 4: An extract of the database cross-referenced with the genre data from Last.fm



```
12271      jazz          A cry from the heart - Live in London  15.0  1988.0
12272      jazz                                   Picture This  15.0  1987.0
12273      jazz                                        Warning  16.0  1985.0
12274       pop                                     Both Sides  11.0  1993.0
12275       pop                            No Jacket Required  14.0  1985.0
12276       pop                             Twenty Four Seven   9.0  1999.0
12277       pop                                Foreign Affair  13.0  1989.0
12278       pop                                Wildest Dreams   7.0  1996.0
12279       pop                             Simply The Best  10.0  1991.0
12280       pop                                          Live   6.0  2009.0
12281       pop                                    The Bridge   9.0  1995.0
12282      jazz                                   Matt Bianco  14.0  1986.0
12283       pop                                   New Arrival   6.0  2003.0
12284       pop                            Pop 'Til You Drop!   6.0  2002.0
12285       pop                                  Teen Spirit   6.0  2001.0
12286       pop                          The Abba Generation   6.0  1999.0
12287       pop          (What's the Story) Morning Glory?   6.0  1996.0
12288      jazz                             New Moon Daughter  12.0  1995.0
12289       pop                                       Mi Plan   7.0  2009.0
12290      jazz                   Unforgettable Glenn Miller  12.0  1985.0
```

2.3 Experiments

2.3.1 Experiment I: Determining the date of an album based on dynamic range values

The values of the dataset are used as a classifier to find the year an album was released. This is done by creating a regression line using the mean dynamic range value per year. The regression line can be seen in red in figure 2. The means are used as training data, while the values of the individual albums are used as test data.

Furthermore, a feed-forward neural network model was used to predict the year based on the available dynamic range values. The network consists of two dense layers with ReLu activation and a last dense layer with softplus activation. Several activation functions were tested, but the outcomes were similar. For the neural network, a different split in training and validation data is used. The means that were used by the linear regression are not needed here and therefore the individual values are separated in training and validation data. The full code for the linear regression and neural network can be found in the appendix.

2.3.2 Experiment II: Determining the genre of an album based on its dynamic range values

The genre data from Last.fm contains quite specific genres. To limit the number of genres compared, several subgenres are combined to create logical categories. For example, all subgenres with "Rock" in the genre name are combined into one category. The same for pop, blues, metal etc. Several of these categories are combined to see whether the dynamic range values can be used to distinguish them from each other.

The resulting input data is split into training and test data. Subsequently, several machine learning classifiers are used to see which method works best for this experiment. Finally, a feed-forward neural network is used to see whether this gives better results than the built-in classifiers. The network consists of three layers, two dense layers with 512 nodes and a final dense layer with nodes equal to the amount of genres that are evaluated. The intermediate layers use ReLu activation and the output layer uses softmax activation, because this works best for multi-class classification problems. Different amount of nodes and different activation functions were tested, but these didn't make an impact or even made the performance worse.

The neural network is trained with two different datasets. One with just the genre and dynamic range data and one test with genre, dynamic range and year data.

3 Results

3.1 Experiment I

The results of the linear regression show a Root Mean Squared Error of 15 years, which is not very useful as classifier. This is not that surprising considering the aforementioned problems with the dataset and the fact that linear regression can result in values outside of the specified range. It can even predict years in the future, therefore the large error rate is not surprising.

Results of testing regression:
Mean Absolute Error: 11.25
Mean Squared Error: 230.9
Root Mean Squared Error: 15.20

The neural network predicts the year with an accuracy of 6% which is higher than the chance value of 3%, but is not very useful. The flaws with the dataset are too big to get impressive results using this method. Tweaking the network didn't change the accuracy. Because the results of experiment II were more promising, the focus shifted towards that experiment.

Results of the feed-forward neural network:
Training accuracy: 0.0616
Validation accuracy: 0.0631

3.2 Experiment II

The various classifiers work very well to predict the genre of an album based on its dynamic range (see table 1). The classifiers give similar results and the neural network doesn't bring improvements over the classifiers. This is to be expected since the classifiers are built specifically for these kinds of problems. Tweaks to the neural network didn't provide better results and more advanced types of neural networks seem like an overkill for this problem. The prediction tree method gives the best scores, although the differences are small. The SVM takes significantly longer to calculate the result and performs worse, this is due to the fact that SVMs are best used for problems with multiple input variables. The addition of year data gives far worse results compared to just using the genre and dynamic range values, therefore these results have been omitted and this part of the experiment has not been pursued further. It shows once again that the combination of year and dynamic range values is not very useful as classifier.

Genre data does provide promising results. Genres of completely different music work best. Especially classical music can be discriminated from the other genres with near-perfect accuracy. When genres of more similar music are classified together, the performance is worse but still quite impressive. Rock and metal can sound quite similar, but a combination of the two still scores 68%.

Genres	KNN	Prediction Tree	Naive Bayes	SVM	Feed-forward NN
Pop, Rock, Jazz	0.6188	0.7048	0.6961	0.6961	Train: 0.6953 Test: 0.7005
Pop, Rock, Metal, Jazz	0.5020	0.5354	0.5293	0.5072	Train: 0.5376 Test: 0.5361
Pop, Rock, Metal, Jazz, Classical	0.5371	0.5411	0.5336	0.5085	Train: 0.5304 Test: 0.5444
Pop, Rock, Metal, Jazz, Classical, Blues, Hip-hop	0.5067	0.5071	0.5055	0.4831	Train: 0.5133 Test: 0.5104
Pop, Jazz, Classical	0.6931	0.7410	0.7314	0.7317	Train: 0.7384 Test: 0.7410
Rock, Metal	0.6826	0.6826	0.6812	0.6500	Train: 0.6865 Test: 0.6829
Pop, Jazz	0.6506	0.7489	0.7450	0.7450	Train: 0.7478 Test: 0.7450
Metal, Jazz	0.7593	0.7708	0.7730	0.7730	Train: 0.7716 Test: 0.7688
Rock, Jazz	0.7269	0.8110	0.8031	0.8031	Train: 0.7965 Test: 0.8110
Rock, Classical	0.9942	0.9944	0.9941	0.9942	Train: 0.9946 Test: 0.9944
Pop, Classical	0.9779	0.9779	0.9764	0.9764	Train: 0.9718 Test: 0.9779

Table 1: The results for the different combinations of genres using various classifiers.

4 Conclusion

The initial goal of the project to date albums based on dynamic range values was not successful. This can be attributed to problems with the dataset. The website it was scraped from rounds the values to integers, making it less precise than the algorithm provides. But even if the values would be available as floats, the spread is still quite small making it hard to use it as a classifier. Furthermore, linear regression might not be the best approach to this problem because it can result in values outside of the specified range. If the experiment would be repeated with better data and other approaches, the expectation is that the results would be better but still not very impressive.

During the research of dynamic range values, it became apparent that genres have different mean values and that this might be a better classifier. The experiments showed that this is the case, classifying a combination of classical music and rock or pop music with near perfect scores. Although classical music is completely different from other genres, even more similar genres like rock and metal can be distinguished from each other very accurately.

References

[1] All Things Considered show. The Loudness Wars: Why Music Sounds Worse. 31 December 2009. Accessed 13 January 2020. https://www.npr.org/2009/12/31/122114058/the-loudness-wars-why-music-sounds-worse?sc=nl&cc=mn-20100102&t=1578856508222

[2] Welch, D. and Fremaux, G. (2017). Why Do People Like Loud Sound? A Qualitative Study. Int J Environ Res Public Health. 2017 Aug; 14(8): 908. Published online 2017 Aug 11. doi:10.3390/ijerph14080908

[3] Gross, J. (2006). Everything Louder Than Everything Else: Have the loudness wars reached their final battle? XL Recording Studio Guide 2006. Accessed at https://web.archive.org/web/20061019013037/http://www.austin360.com/music/content/music/stories/xl/2006/09/28cover.html on 13 January 2020

[4] Van der Dorp, J. (2013). The Loudness War: Dynamic Range over the years. TheMindGap. Accessed at http://www.themindgap.nl/?p=114 on 13 January 2020.

[5] DROffline. MAAT. Accessed at https://www.maat.digital/droffline/ on 14 January 2020

[6] Sheperd, I (2013). Why you can't measure vinyl with the TT Meter. Production advice. accessed at https://productionadvice.co.uk/tt-meter-not-for-vinyl/ on 14 January 2020.

[7] Last FM API. Accessed at https://www.last.fm/api/ on 14 January 2020.

Appendix A: p-values

Year	p-value	Significant
1985	0.0922	False
1986	0.0	True
1987	0.8663	False
1988	0.0001	True
1989	0.3855	False
1990	0.0	True
1991	0.0	True
1992	0.0	True
1993	0.0	True
1994	0.0	True
1995	0.0	True
1996	0.0	True
1997	0.0001	True
1998	0.0	True
1999	0.0626	False
2000	0.0	True
2001	0.0216	True
2002	0.2059	False
2003	0.0017	True
2004	0.0001	True
2005	0.0313	True
2006	0.0421	True
2007	0.0858	False
2008	0.1411	False
2009	0.465	False
2010	0.5137	False
2011	0.1187	False
2012	0.4518	False
2013	0.0008	True
2014	0.0308	True
2015	0.0	True
2016	0.8161	False
2017	0.4694	False
2018	0.9208	False

Table 2: The p-values for the comparison between subsequent years. The p-value is the outcome of a t-test between the dynamic range values of the year mentioned and the year after

Appendix B: code

```python
from selenium import webdriver
from BeautifulSoup import BeautifulSoup
import pandas as pd

driver = webdriver.Chrome()
results = [[]]
# nr_pages should contain the amount of pages of http://dr.loudness-war.info/album/list/
# While scraping the data this value was 1391
nr_pages = 1391
def add_page(url):
  driver.get(url)
  content = driver.page_source

  # Use the BeautifulSoup package to find the right column and save its values in results
  bs=BeautifulSoup(content)
  div = bs.find('div', {'class': 'table-responsive-sm'})
  if div is not None:
    rows  = div.findAll('tr')
    #print(rows)
    for row in rows:
      tds=row.findAll('td')
      result = []
      for td in tds:
        if td:
          data = td.text.strip()
          if data:
              result.append(data)
      results.append(result)


for i in range(1,nr_pages+1):
  add_page("http://dr.loudness-war.info/album/list/" + str(i))
# Create a dataframe from results and save it to disc.
df = pd.DataFrame(results, columns=["Artist", "Album", "Year", "avg DR", "min DR", "max DR", "codec", "source"])
print(df)
df.to_csv('dr_data.csv', encoding="utf-8")
```

Listing 1: scrapedr.py

```python
import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
import matplotlib.pyplot as plt
import discogs_client
import pylast
import os, time
import requests


# Read the dr data obtained in scrape_dr and find all the artists.
data = pd.read_csv("dr_data.csv")
data = data[data['Year'] >= 1985.0]
artists = data['Artist'].unique()
# A Last.fm API key is needed to use this script.
API_KEY =
URL = "http://ws.audioscrobbler.com/2.0/"
# Find an artist and return its most popular genre.
def get_genre(artist, PARAMS):
  r = requests.get(url = URL, params = PARAMS).json()

  try:
    genre = list(list(r.values())[0].values())[0][0]['name']
    return genre
  except:
    return "Not found"

genres = np.empty([len(artists), 2], dtype="U200")
for i in range(0, len(artists)):
  artist = artists[i]
  PARAMS = {'method': 'artist.gettoptags', 'artist' : artist, 'api_key':API_KEY, 'format':'json'}
  genre = get_genre(artist, PARAMS)
  if(genre != "Not found"):
    genres[i] = [artist, genre]
    np.savetxt('genres.txt', genres, fmt='%s', delimiter=',')
```

Listing 2: getgenres.py

```python
import csv, pandas as pd

# Open the dr data obtained in scrape_dr.py and the genre data obtained in get_genre.py
data = pd.read_csv("dr_data.csv")
data = data[data['Year'] >= 1985.0]
data = data[data['source'] != "Vinyl"]


with open('genres.csv', mode='r') as infile:
    reader = csv.reader(infile)
    genres = {rows[0]:rows[1] for rows in reader}

temp_df = [[]]
# Iterate over the albums and combine all subgenres into logical categories.
for index, row in data.iterrows():
  try:
    if "pop" in genres[row['Artist']]:
      temp_df.append(["pop", row['Album'], row['avg DR'], row['Year']])
    if "rock" in genres[row['Artist']]:
      temp_df.append(["rock", row['Album'], row['avg DR'], row['Year']])
    if "metal" in genres[row['Artist']]:
      temp_df.append(["metal", row['Album'], row['avg DR'], row['Year']])
    if "jazz" in genres[row['Artist']]:
      temp_df.append(["jazz", row['Album'], row['avg DR'], row['Year']])
    if "classical" in genres[row['Artist']]:
      temp_df.append(["classical", row['Album'], row['avg DR'], row['Year']])
    if "hiphop" in genres[row['Artist']] or "rap" in genres[row['Artist']]:
      temp_df.append(["rap", row['Album'], row['avg DR'], row['Year']])
    if "blues" in genres[row['Artist']]:
      temp_df.append(["blues", row['Album'], row['avg DR'], row['Year']])
  except:
    print("Artist not in database")
# Save the combination of album, dr and genre data
data  = pd.DataFrame(temp_df, columns = ["Genre", "Album", "DR", "Year"])
data.to_csv('genre_data.csv', encoding="utf-8")
```

Listing 3: combinedata.py

```python
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

# Read the genre data from combine_data.py and extract the columns that are needed
dataframe = pandas.read_csv("genre_data", header=None)
dataframe = dataframe.dropna()
dataset = dataframe.values
X = dataset[:, [3]].astype(float)
Y = dataset[:,1]
print(X)
print(X.shape)
print(Y.shape)

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 0)

# Use the K-nearest-Neighours classifier and return the
# resulting accuracy and confusion matrix
knn = KNeighborsClassifier(n_neighbors = 7).fit(X_train, y_train)
accuracy = knn.score(X_test, y_test)
print(accuracy)
knn_predictions = knn.predict(X_test)
cm = confusion_matrix(y_test, knn_predictions)
print(cm)


# Use the Decision Tree classifier and return the
# resulting accuracy and confusion matrix
dtree_model = DecisionTreeClassifier(max_depth = 2).fit(X_train, y_train)
```

```python
dtree_predictions = dtree_model.predict(X_test)
accuracy = accuracy_score(y_test, dtree_predictions)
print(accuracy)

cm = confusion_matrix(y_test, dtree_predictions)
print(cm)



# Use the Gaussian Naive Bayes classifier and return the
# resulting accuracy and confusion matrix
gnb = GaussianNB().fit(X_train, y_train)
gnb_predictions = gnb.predict(X_test)

accuracy = gnb.score(X_test, y_test)
print(accuracy)

cm = confusion_matrix(y_test, gnb_predictions)
print(cm)


# Use the Support Vector Machine classifier and return the
# resulting accuracy and confusion matrix
svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
svm_predictions = svm_model_linear.predict(X_test)

accuracy = svm_model_linear.score(X_test, y_test)
print(accuracy)

cm = confusion_matrix(y_test, svm_predictions)
print(cm)

#Convert the potential genres outcomes using one-hot encoding
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
dummy_y = np_utils.to_categorical(encoded_Y)

X_train, X_test, y_train, y_test = train_test_split(X, dummy_y, random_state = 0)
def baseline_model():
  model = Sequential()
```

```
    model.add(Dense(512, input_dim=1, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# Build a simple feed-forward neural network and test its performance on the genre and dr data
model = baseline_model()
model.summary()
model.fit(X_train, y_train, batch_size=100, nb_epoch=25, validation_data=(X_test, y_test))
```

Listing 4: classifier.py

```
import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier

# Read the DR data that was scraped using scrape_dr.py and
# prepare the means as training data and the individual values
# as test values.
data = pd.read_csv("dr_data.csv")
data = data[data['source'] != 'Vinyl']
data = data[['Year','avg DR']]
data = data[data['Year'] >= 1985.0]
data_mean = data.groupby(['Year'])['avg DR'].mean()
data_mean = pd.DataFrame(data_mean, columns = ['avg DR'])
dr_train = np.array(data_mean.iloc[:,0]).reshape((-1, 1))
year_train = np.array(data_mean.index.values).reshape((-1, 1))
```

```python

dr_test = np.array(data['avg DR']).reshape(-1,1)
year_test = np.array(data['Year'])

# Perform the linear regression and print the outcome
model = LinearRegression().fit(dr_train, year_train)
r_sq = model.score(dr_train, year_train)

print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('slope:', model.coef_)

y_pred = model.predict(dr_test)
df = pd.DataFrame({'Actual': year_test.flatten(), '
    Predicted': y_pred.flatten()})
print(df)

# Plot the means per year and the regression line
plt.scatter(year_train, dr_train)
axes = plt.gca()
axes.set_xlim([1980,2020])
axes.set_ylim([0,20])
axes.set_title("Dynamic range of albums from 1985 to
    2019")
axes.set_xlabel("Year")
axes.set_ylabel("DR value")
y_pred = model.predict(year_train)
plt.plot(year_train, y_pred, color='red')
plt.show()

print('Mean Absolute Error:', metrics.
    mean_absolute_error(year_train, y_pred))
print('Mean Squared Error:', metrics.
    mean_squared_error(year_train, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.
    mean_squared_error(year_train, y_pred)))

# Encode the potential output years with one-hot
    encoding
encoder = LabelEncoder()
encoder.fit(year_test)
encoded_Y = encoder.transform(year_test)
dummy_y = np_utils.to_categorical(encoded_Y)
X_train, X_test, y_train, y_test = train_test_split(
    dr_test, dummy_y, random_state = 0)

```

```

def baseline_model():
  model = Sequential()
  model.add(Dense(512, input_dim=1, activation='relu'))
  model.add(Dense(512, activation='relu'))
  model.add(Dense(35, activation='softmax'))
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model

# Build a simple feed-forward neural network and test its performance on the year and dr data
model = baseline_model()
model.summary()
model.fit(X_train, y_train, batch_size=100, nb_epoch=25, validation_data=(X_test, y_test))
```
Listing 5: predictor.py