1. The Dataset (20 points)

Our dataset is a table of songs, each with a name, an artist, and a genre. For each song, we also know how frequently certain words occur in that song. More precisely, we have a list of approximately 5000 words. For each of these words, for each song, each item in the table describes the proportion of the song's lyrics that are the particular word.

For example, the lyrics of "In Your Eyes" is 168 words long. The word "like" appears twice: $\frac{2}{168} \approx 0.0119$ of the words in the song. Similarly, the word "love" appears 10 times: $\frac{10}{168} \approx 0.0595$ of the words.

Our dataset doesn't contain all information about a song. For example, it doesn't include the total number of words in each song, or information about the order of words in the song, let alone the melody, instruments, or rhythm. Nonetheless, you may find that word counts alone are sufficient to build an accurate genre classifier.

Run the cell below to read the `lyrics` table. **It may take up to a minute to load.**

In [2]:
```
import pandas as pd
```

In [6]:
```
df = pd.read_csv("lyrics_clean.csv")
df.head()
```

Out[6]:

	Title	Artist	Genre	i	the	you	to	and	a	me	...	writer	motivo	bake	insist	wel	santo	p
0	Slicker Than Your Average	Craig David	Hip-hop	0.049536	0.017028	0.035604	0.020124	0.007740	0.006192	0.058824	...	0.0	0.0	0.0	0.0	0	0.0	0
1	Right There	MF Grimm	Hip-hop	0.037825	0.054374	0.023641	0.049645	0.009456	0.016548	0.018913	...	0.0	0.0	0.0	0.0	0	0.0	0
2	Talkin' All That	Cashis	Hip-hop	0.056738	0.049645	0.051418	0.010638	0.026596	0.033688	0.007092	...	0.0	0.0	0.0	0.0	0	0.0	0

	Title	Artist	Genre	i	the	you	to	and	a	me	...	writer	motivo	bake	insist	wel	santo	p
3	It Only Hurts Me When I Cry	Raul Malo	Country	0.096491	0.074561	0.030702	0.017544	0.026316	0.017544	0.021930	...	0.0	0.0	0.0	0.0	0	0.0	0
4	Is It Too Late Now	Lester Flatt & Earl Scruggs	Country	0.043902	0.000000	0.073171	0.019512	0.000000	0.014634	0.034146	...	0.0	0.0	0.0	0.0	0	0.0	0

5 rows × 4979 columns

Question 1.1: Print the number of rows and columns in the dataset

In [7]:

```
print("There are", df.shape[0],"rows in the dataframe and", df.shape[1], "columns in the dataframe")
```

```
There are 1721 rows in the dataframe and 4979 columns in the dataframe
```

Question 1.2: Find the proportion of the word `like` in the song `In Your Eyes`

In [8]:

```
slist = df['Title'].tolist()
index = 0
sindex = 0
for i in slist:
    if(i == "In Your Eyes"):
        sindex = index
    index += 1
Likelist = df['like'].tolist()
index = 0
Like = 0
for i in Likelist:
    if(index == sindex):
        Like = i
    index += 1
print("The proportion of the word like in the song In your Eyes is", Like)
```

```
The proportion of the word like in the song In your Eyes is 0.01190476
```

Question 1.3: Set `expected_row_sum` to the number that you expect will result from summing all proportions in each row, excluding the first three columns.

In [9]:
```
# Set row_sum to a number that's the (approximate) sum of each row of word proportions.
expected_row_sum = 1
```

Verify your answer by doing sum along the columns for each row

In [10]:
```
newdf = df.loc[:, 'i'::1]
newdf.sum(axis=1)
```

Out[10]:
```
0       1.0
1       1.0
2       1.0
3       1.0
4       1.0
       ...
1716    1.0
1717    1.0
1718    1.0
1719    1.0
1720    1.0
Length: 1721, dtype: float64
```

Word Stemming

The columns other than Title, Artist, and Genre in the `lyrics` table are all words that appear in some of the songs in our dataset. Some of those names have been *stemmed*, or abbreviated heuristically, in an attempt to make different inflected forms of the same base word into the same string. For example, the column "manag" is the sum of proportions of the words "manage", "manager", "managed", and "managerial" (and perhaps others) in each song.

Stemming makes it a little tricky to search for the words you want to use, so we have provided another dataframe that will let you see examples of unstemmed versions of each stemmed word. Run the code below to load it.

Question 1.4: Read the vocabulary from the given file `mxm_reverse_mapping_safe.csv` and store it into a variale `vocab_mapping`

In [11]:
```
vocab_mapping = pd.read_csv("mxm_reverse_mapping_safe.csv")
vocab_mapping
```

Out[11]:

	Stem	Word

	Stem	Word
0	día	día
1	pido	pido
2	hatr	hatred
3	pide	pide
4	yellow	yellow
...	...	...
4971	yell	yell
4972	at	at
4973	confess	confess
4974	sincer	sincere
4975	richard	richard

4976 rows × 2 columns

Question 1.5: Compare if the number of stemmed words in the vocabulary is the same with one in the song lyrics dataset.

```
In [12]: df.loc[:, 'i':].shape[1] ==  vocab_mapping.shape[0]
```

```
Out[12]: True
```

Question 1.6: Assign `unchanged` to the **percentage** of words in `vocab_table` that are the same as their stemmed form.

```
In [15]: import numpy as np

         row_count = vocab_mapping.shape[0]
         unchanged = (np.count_nonzero(vocab_mapping['Stem'] == vocab_mapping['Word']) / row_count) * 100
         unchanged
```

```
Out[15]: 72.16639871382637
```

Question 1.7: Assign `stemmed_message` to the stemmed version of the word "message".

In [17]:
```
# Set stemmed_message to the stemmed version of "message" (which
# should be a string).
stemmed_message = vocab_mapping[vocab_mapping["Word"]=='message']
stemmed_message
```

Out[17]:

	Stem	Word
4151	messag	message

In [18]:
```
stemmed_message.loc[4151, "Stem"]
```

Out[18]:
```
'messag'
```

Question 1.8: Assign `unstemmed_singl` to the word in `vocab_table` that has "singl" as its stemmed form. (*Note that multiple English words may stem to "singl", but only one example appears in* `vocab_table`.)

In [20]:
```
# Set unstemmed_singl to the unstemmed version of "single" (which
# should be a string).
unstemmed_singl = vocab_mapping[vocab_mapping["Stem"]=='singl']
unstemmed_singl
```

Out[20]:

	Stem	Word
4254	singl	single

In [21]:
```
unstemmed_singl.loc[4254, "Word"]
```

Out[21]:
```
'single'
```

Question 1.9: What word in `vocab_table` was shortened the most by this stemming process? Assign `most_shortened` to the word. *hint: function len(str) will return the length of the input string* `str`. *You will do a loop over rows of the vocabulary to compute the length of each word.*

In [22]:
```
length_of_stems = []
length_of_words = []
for index, row in vocab_mapping.iterrows():
    st_length = len(row['Stem'])
    length_of_stems.append(st_length)
    w_length = len(row['Word'])
    length_of_words.append(w_length)
```

```
vocab_mapping["Stem length"] = length_of_stems
vocab_mapping["Word length"] = length_of_words
vocab_mapping
```

Out[22]:

	Stem	Word	Stem length	Word length
0	día	día	3	3
1	pido	pido	4	4
2	hatr	hatred	4	6
3	pide	pide	4	4
4	yellow	yellow	6	6
...	...	...	...	...
4971	yell	yell	4	4
4972	at	at	2	2
4973	confess	confess	7	7
4974	sincer	sincere	6	7
4975	richard	richard	7	7

4976 rows × 4 columns

In [23]:
```
vocab_mapping["Difference"] = (vocab_mapping["Word length"] - vocab_mapping["Stem length"]).abs()
vocab_mapping
```

Out[23]:

	Stem	Word	Stem length	Word length	Difference
0	día	día	3	3	0
1	pido	pido	4	4	0
2	hatr	hatred	4	6	2
3	pide	pide	4	4	0
4	yellow	yellow	6	6	0
...	...	...	...	...	...

	Stem	Word	Stem length	Word length	Difference
4971	yell	yell	4	4	0
4972	at	at	2	2	0
4973	confess	confess	7	7	0
4974	sincer	sincere	6	7	1
4975	richard	richard	7	7	0

4976 rows × 5 columns

In [24]:
```
vocab_mapping[vocab_mapping["Difference"] == vocab_mapping["Difference"].max()]
```

Out[24]:

	Stem	Word	Stem length	Word length	Difference
983	intern	international	6	13	7

Splitting the dataset

We're going to use our `lyrics` dataset for three purposes. First, we want to *train* various song genre classifiers. Second, we want to *validate* which classifier is most effective. Finally, we want to *test* the performance of our final classifier. Hence, we need three different datasets: *training*, *validation*, and *test*.

The purpose of a classifier is to generalize to unseen data that is similar to the training data. Therefore, we must ensure that there are no songs that appear in two different sets. We do so by splitting the dataset randomly. The dataset has already been permuted randomly, so it's easy to split. We just take the top for training, the next part for validation, and the last for test.

Question 1.10: Split the data with the ratio `80%` for training and `20%` for testing.

In [25]:
```
training_proportion = 0.8
num_songs = df.shape[0]

num_train = int(num_songs * training_proportion)
num_test = num_songs - num_train
```

```
print("Num song:", num_songs)

print("Num train:", num_train)
print("Num test:", num_test)
```

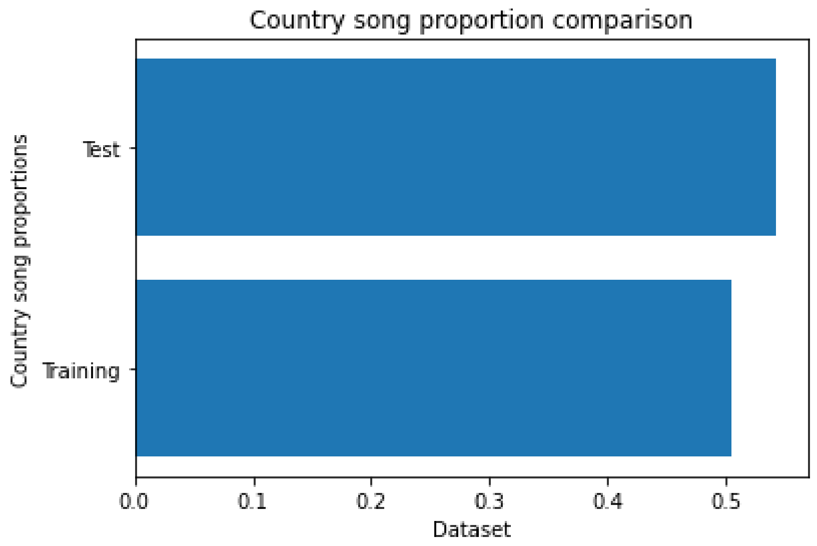
```
Num song: 1721
Num train: 1376
Num test: 345
```

In [26]:
```
train_lyrics = df[:num_train]
test_lyrics = df[num_train:]

print("Training: ", train_lyrics.shape[0], "; Test: ", test_lyrics.shape[0])
```

```
Training:  1376 ; Test:  345
```

Question 1.11: Draw a horizontal bar chart with three bars that shows the proportion of Country songs in each of the training and testing datasets.

In [27]:
```
training_country_song_count = train_lyrics[train_lyrics["Genre"] == "Country"].shape[0]
training_country_song_proportion = training_country_song_count / train_lyrics.shape[0]
test_country_song_count = test_lyrics[test_lyrics["Genre"] == "Country"].shape[0]
test_country_song_proportion = test_country_song_count / test_lyrics.shape[0]
import matplotlib.pyplot as plt


datasets = ['Training', 'Test']
country_song_proportions = [training_country_song_proportion, test_country_song_proportion]

plt.barh(datasets, country_song_proportions)
plt.xlabel("Dataset")
plt.ylabel("Country song proportions")
plt.title("Country song proportion comparison")

plt.show()
```



2. K-Nearest Neighbors (20 points)

K-Nearest Neighbors (k-NN) is a classification algorithm. Given some *features* of an unseen example, it decides whether that example belongs to one or the other of two categories based on its similarity to previously seen examples.

A feature we have about each song is *the proportion of times a particular word appears in the lyrics*, and the categories are two music genres: hip-hop and country. The algorithm requires many previously seen examples for which both the features and categories are known: that's the `train_lyrics` table.

We're going to visualize the algorithm, instead of just describing it. To get started, let's pick colors for the genres.

In [28]:
```
# Just run this cell to define genre_color.

def genre_color(genre):
    """Assign a color to each genre."""
    if genre == 'Country':
        return 'gold'
    elif genre == 'Hip-hop':
        return 'blue'
    else:
        return 'green'
```

```
In [29]: genre_color('Country')
```

```
Out[29]: 'gold'
```

```
In [30]: genre_color('Hip-hop')
```

```
Out[30]: 'blue'
```

Classifying a song

In k-NN, we classify a song by finding the `k` songs in the *training set* that are most similar according to the features we choose. We call those songs with similar features the "neighbors". The k-NN algorithm assigns the song to the most common category among its `k` neighbors.

Let's limit ourselves to just 2 features for now, so we can plot each song. The features we will use are the proportions of the words "like" and "love" in the lyrics. Taking the song "In Your Eyes" (in the test set), 0.0119 of its words are "like" and 0.0595 are "love". This song appears in the test set, so let's imagine that we don't yet know its genre.

First, we need to make our notion of similarity more precise. We will say that the *dissimilarity*, or *distance* between two songs is the straight-line distance between them when we plot their features in a scatter diagram. This distance is called the Euclidean ("yoo-KLID-ee-un") distance.

For example, in the song *Insane in the Brain* (in the training set), 0.0203 of all the words in the song are "like" and 0 are "love". Its distance from *In Your Eyes* on this 2-word feature set is $\sqrt{(0.0119 - 0.0203)^2 + (0.0595 - 0)^2} \approx 0.06$. (If we included more or different features, the distance could be different.)

A third song, *Sangria Wine* (in the training set), is 0.0044 "like" and 0.0925 "love".

Question 2.1: Define a function that creates a plot to display a test song and some training songs in a two-dimensional space defined by two features. Utilize the function to visualize the songs *In Your Eyes*, *Sangria Wine*, and *Insane in the Brain*.

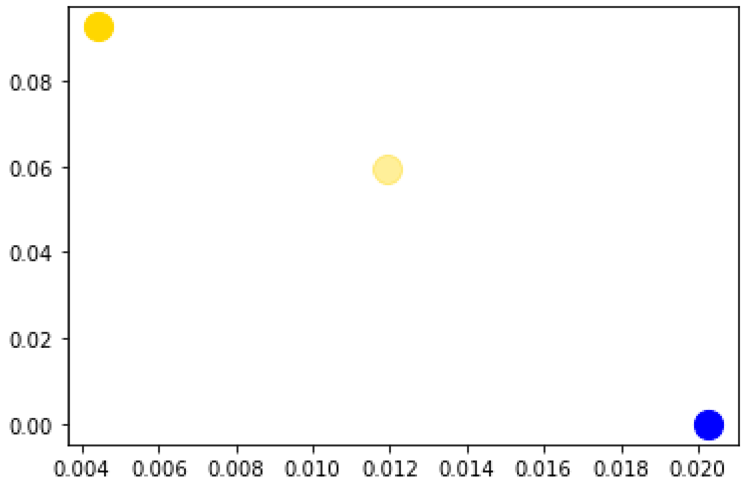
hint: the function has four arguments and it does not return anything but it plots the songs in 2D space:

- test_song: has string datatype, is the name of a song

- training_songs: has list datatype, is a list of songs
- x_feature: has string datatype, is the name of a feature.
- y_feature: has string datatype, is the name of another feature.

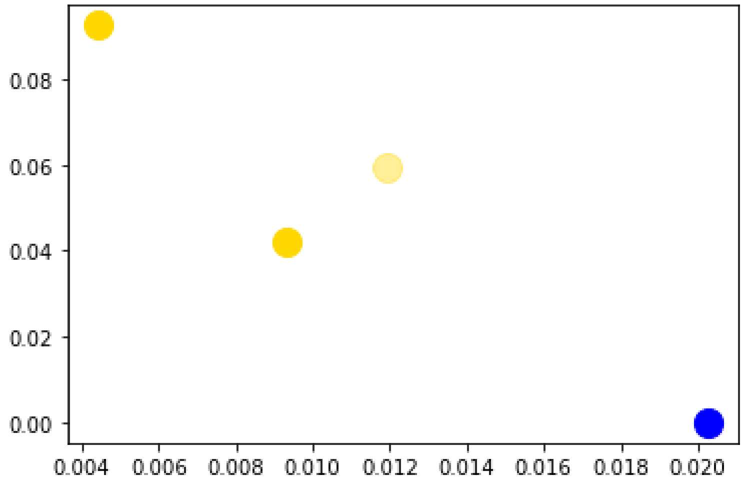
In [31]:
```python
import matplotlib.pyplot as plt

def plot_with_two_features(test_song, training_songs, x_feature, y_feature):
    """Plot a test song and training songs using two features."""
    like_prob = df.loc[df["Title"] == test_song, "like"].values[0]
    love_prob = df.loc[df["Title"] == test_song, "love"].values[0]
    genre = df.loc[df["Title"] == test_song, "Genre"].values[0]
    shown_color = genre_color(genre)
    plt.scatter(x=like_prob, y=love_prob, color=shown_color, s=200, alpha=0.4)
    for song in training_songs:
        like_prob = df.loc[df["Title"] == song, "like"].values[0]
        love_prob = df.loc[df["Title"] == song, "love"].values[0]
        genre = df.loc[df["Title"] == song, "Genre"].values[0]
        shown_color = genre_color(genre)
        plt.scatter(x=like_prob, y=love_prob, color=shown_color, s=200)
```

In [37]:
```python
# visualize the distances of the songs In Your Eyes, Sangria Wine, and Insane in the Brain.
training = ["Sangria Wine", "Insane In The Brain"]
test_song = "In Your Eyes"
plot_with_two_features(test_song, training, "like", "love")
```



Question 2.2: Utilize the `plot_with_two_features` function and plot the positions of the three songs *Sangria Wine*, *Lookin' for Love*, *Insane In The Brain* together with the song *In Your Eyes*. Which one is closer to *In Your Eyes* and what is its genre?

In [38]:
```
training = ["Sangria Wine", "Lookin' for Love",  "Insane In The Brain"]
plot_with_two_features("In Your Eyes", training, "like", "love")
```



Question 2.3. Complete the function `distance_two_features` that computes the Euclidean distance between any two songs, using two features. Utilize the function `distance_two_features` to show that *Lookin' for Love* is closer to *In Your Eyes* than *Insane In The Brain*.

In [39]:
```
import math as m

def distance_two_features(title0, title1, x_feature, y_feature):
    """Compute the distance between two songs, represented as rows."""
    x1_prob = df.loc[df["Title"] == title0, x_feature].values[0]
    y1_prob = df.loc[df["Title"] == title0, y_feature].values[0]

    x2_prob = df.loc[df["Title"] == title1, x_feature].values[0]
    y2_prob = df.loc[df["Title"] == title1, y_feature].values[0]

    distance = m.sqrt( (x1_prob-x2_prob)**2 + (y1_prob-y2_prob)**2 )
    return distance
for song in ["Lookin' for Love", "Insane In The Brain"]:
    song_distance = distance_two_features(song, "In Your Eyes", "like", "love")
    print(song, 'distance:\t', song_distance)
```

```
Lookin' for Love distance:       0.017854025951587398
Insane In The Brain distance:    0.060108782340654685
```

The nearest neighbor to a song is the example in the training set that has the smallest distance from that song.

Question 2.4. What are the names and genres of the 7 closest songs to "In Your Eyes" in `train_lyrics`, by Euclidean distance for the 2 features "like" and "love"? To answer this question, make a dataframe named `close_songs` containing those 7 songs with columns "Title", "Artist", "Genre", "like", and "love" from the `lyrics` dataframe, as well as a column called `distance` that contains the distance from "In Your Eyes" **sorted in ascending order**.

In [40]:
```
like_love = train_lyrics[["Title", "Artist", "Genre", "like", "love"]]
close_songs = like_love.copy()

distances = []
for index, row in close_songs.iterrows():
    distance_to_in_your_eyes = distance_two_features(row['Title'], "In Your Eyes", "like", 'love')
    distances.append(distance_to_in_your_eyes)
#     break

close_songs["Distance"] = distances

close_songs = close_songs.nsmallest(7, 'Distance')
close_songs
```

Out[40]:

	Title	Artist	Genre	like	love	Distance
828	If This Isn't Love	Jennifer Hudson	Hip-hop	0.008869	0.053215	0.007001
1108	Big Red Rocket Of Love	Reverend Horton Heat	Hip-hop	0.000000	0.057692	0.012045
1106	In the Middle of a Heartache	Wanda Jackson	Country	0.000000	0.063953	0.012702
160	The Hardest Part	Allison Moorer	Country	0.000000	0.064286	0.012822
1063	One Time	Justin Bieber	Hip-hop	0.000000	0.053030	0.013561
627	This Tornado Loves You	Neko Case	Country	0.000000	0.052846	0.013650
1201	Mama Knew Love	Anthony Hamilton	Hip-hop	0.020619	0.048969	0.013687

Question 2.5 . Find the most common value in the column `Genre` of the dataframe `close_songs`. In case of a tie, it can return any of the most common values.

In [41]:
```
close_songs['Genre'].mode().values[0]
```

Out[41]:
```
'Hip-hop'
```

Congratulations are in order -- you've classified your first song!

3. Features (20 points)

Now, we're going to extend our classifier to consider more than two features at a time.

Euclidean distance still makes sense with more than two features. For `n` different features, we compute the difference between corresponding feature values for two songs, square each of the `n` differences, sum up the resulting numbers, and take the square root of the sum.

Question 3.1

Write a function to compute the Euclidean distance between two **arrays** of features of *arbitrary* (but equal) length. Use it to compute the distance between the first song in the training set and the first song in the test set, *using all of the features*. (Remember that the title, artist, and genre of the songs are not features.)

Hint: The function has two arguments which are two arrays representing the two lists of features:

In [42]:
```
import numpy as np

def distance(features1, features2):
    """The Euclidean distance between two arrays of feature values."""
    abs_diff = np.abs(features1 - features2)
    squared = np.square(abs_diff)

    sum_squared = np.sum(squared)

    return np.sqrt(sum_squared)
```

In [43]:
```
first_song_features = train_lyrics.loc[0, 'i':].values
first_song_features
```

Out[43]:
```
array([0.0495356, 0.01702786, 0.03560372, ..., 0.0, 0.0, 0], dtype=object)
```

```
In [44]: second_song_features = test_lyrics.loc[num_train, 'i':].values
         second_song_features
```

```
Out[44]: array([0.17529879999999998, 0.003984064, 0.0438247, ..., 0.0, 0.0, 0],
               dtype=object)
```

```
In [45]: distance_first_to_first = distance(first_song_features, second_song_features)
         distance_first_to_first
```

```
Out[45]: 0.20977496424035585
```

Creating your own feature set

Unfortunately, using all of the features has some downsides. One clear downside is *computational* -- computing Euclidean distances just takes a long time when we have lots of features. You might have noticed that in the last question!

So we're going to select just 20. We'd like to choose features that are very *discriminative*. That is, features which lead us to correctly classify as much of the test set as possible. This process of choosing features that will make a classifier work well is sometimes called *feature selection*, or more broadly *feature engineering*.

Question 3.2

Look through the list of features (the labels of the `lyrics` table after the first three). Choose 20 common words that you think might let you distinguish between country and hip-hop songs. Make sure to choose words that are frequent enough that every song contains at least one of them. Don't just choose the 20 most frequent, though... you can do much better.

The first time you answer this question, spend some time looking through the features, but not more than 15 minutes.

```
In [ ]: words = ['i', 'the', 'you', 'to', 'and', 'a', 'me', 'it', 'not', 'in', 'my',
                'is', 'of', 'your', 'that', 'do', 'on', 'are', 'we', 'am']
```

Question 3.3

In two sentences or less, describe how you selected your features.

I selected these features based on their usage in country and hip hop to where they are used in both but there are distinguishable between the two different genres.

Question 3.4

Use the `distance` function developed above to compute the distance from the first song in the test set to all the songs in the training set, **using your set of 20 features**. Make a new dataframe called `genre_and_distances` with one row for each song in the training set and two columns:

- The `"Genre"` of the training song
- The `"Distance"` from the first song in the test set

Ensure that `genre_and_distances` is **sorted in increasing order by distance to the first test song**.

In [46]:
```
feature_labels = test_lyrics.columns[3:23].values
feature_labels
```

Out[46]:
```
array(['i', 'the', 'you', 'to', 'and', 'a', 'me', 'it', 'not', 'in', 'my',
       'is', 'of', 'your', 'that', 'do', 'on', 'are', 'we', 'am'],
      dtype=object)
```

In [48]:
```
test_song_features = test_lyrics.loc[num_train, feature_labels]
test_song_features
```

Out[48]:
```
i          0.175299
the      0.00398406
you       0.0438247
to       0.00796813
and       0.0119522
a        0.00398406
me        0.0358566
it        0.0358566
not       0.0119522
in       0.00398406
my       0.00796813
is       0.00398406
of       0.00796813
your              0
that              0
do                0
on       0.00398406
are               0
we                0
```

```
am         0.00398406
Name: 1376, dtype: object
```

In [49]:
```
test_song_features = test_lyrics.loc[num_train, feature_labels]

genre_and_distances = train_lyrics[["Genre"]].copy()
distances_to_test_song = []
for index, row in train_lyrics.iterrows():
    features = row.loc[feature_labels]
    the_distance = distance(test_song_features, features)
    distances_to_test_song.append(the_distance)

genre_and_distances["Distance"] = distances_to_test_song
genre_and_distances = genre_and_distances.sort_values(['Distance'], ascending=True)
genre_and_distances
```

Out[49]:

	Genre	Distance
1294	Hip-hop	0.068603
820	Country	0.070478
1102	Country	0.071549
639	Country	0.074927
37	Hip-hop	0.078204
...	...	...
1299	Country	0.247216
73	Hip-hop	0.254622
1097	Country	0.257257
696	Country	0.272240
64	Hip-hop	0.281842

1376 rows × 2 columns

Question 3.5

Now compute the 5-nearest neighbors classification of the first song in the test set. That is, decide on its genre by finding the most common genre among its 5 nearest neighbors, according to the distances you've calculated. Then check whether your classifier chose the right genre. (Depending on the features you chose, your classifier might not get this song right, and that's okay.)

```
In [50]: genre_and_distances = genre_and_distances.nsmallest(5, "Distance")
         genre_and_distances
```

Out[50]:

	Genre	Distance
1294	Hip-hop	0.068603
820	Country	0.070478
1102	Country	0.071549
639	Country	0.074927
37	Hip-hop	0.078204

```
In [51]: genre_and_distances['Genre'].mode().values[0]
```

```
Out[51]: 'Country'
```

A classifier function

Now it's time to write a single function that encapsulates this whole process of classification.

Question 3.6. Write a function called `classify`. It should take the following arguments:

- An array of features for a song to classify ,
- A dataframe has similar structure of the original dataset,
- `k`, the number of neighbors to use in classification.

It should return the class your classifier picks for the given row of features (e.g., `'Country'` or `'Hip-hop'`). Test if the function works by classifying the first song in the test set using k=5.

```
In [77]: def classify(test_features, train_dataframe, k):
             """Return the most common class among k nearest neigbors to test_row."""
```

```
    feature_labels = train_dataframe.columns[3:23].values

    genre_and_distances = train_dataframe[["Genre"]].copy()
    distances_to_test_song = []
    for index, row in train_dataframe.iterrows():
        features = row.loc[feature_labels]
        the_distance = distance(test_features, features)
        distances_to_test_song.append(the_distance)

    genre_and_distances["Distance"] = distances_to_test_song
    genre_and_distances = genre_and_distances.nsmallest(k, "Distance")

    return genre_and_distances['Genre'].mode().values[0]
```

In [53]:
```
feature_labels = test_lyrics.columns[3:23].values
test_song_features = test_lyrics.loc[num_train, feature_labels]
test_song_features
```

Out[53]:
```
i         0.175299
the     0.00398406
you      0.0438247
to      0.00796813
and      0.0119522
a       0.00398406
me       0.0358566
it       0.0358566
not      0.0119522
in      0.00398406
my      0.00796813
is      0.00398406
of      0.00796813
your             0
that             0
do               0
on      0.00398406
are              0
we               0
am      0.00398406
Name: 1376, dtype: object
```

In [54]:
```
classify(test_song_features, train_lyrics, k=5)
```

Out[54]:
```
'Country'
```

Question 3.7. Assign `grandpa_genre` to the genre predicted by your classifier for the song "Grandpa Got Runned Over By A John Deere", using 9 neigbors.

In [59]:
```
feature_labels = test_lyrics.columns[3:23].values
test_song = df.loc[df["Title"] == "Grandpa Got Runned Over By A John Deere"]
test_song
```

Out[59]:

	Title	Artist	Genre	i	the	you	to	and	a	me	...	writer	motivo	bake	insist	wel	santo
1412	Grandpa Got Runned Over By A John Deere	Cledus T. Judd	Country	0.015707	0.015707	0.010471	0.005236	0.036649	0.026178	0.010471	...	0.0	0.0	0.0	0.0	0	0.0

1 rows × 4979 columns

In [63]:
```
test_song_features = test_song.loc[1412, feature_labels]
test_song_features
```

Out[63]:
```
i        0.0157068
the      0.0157068
you      0.0104712
to       0.0052356
and      0.0366492
a         0.026178
me       0.0104712
it       0.0157068
not      0.0052356
in       0.0104712
my       0.0052356
is       0.0052356
of       0.0104712
your             0
that     0.0104712
do               0
on       0.0209424
are      0.0052356
we        0.026178
```

```
am                0
Name: 1412, dtype: object
```

In [64]:
```
classify(test_song_features, train_lyrics, k=9)
```

Out[64]:
```
'Hip-hop'
```

Evaluating your classifier

Now that it's easy to use the classifier, let's see how accurate it is on the whole test set. But we will reduce the test set to 20 songs only to save computing power.

Question 3.8. Generate a new test set of 20 songs from your current test set

In [65]:
```
test_lyrics = test_lyrics.sample(20)
test_lyrics
```

Out[65]:

	Title	Artist	Genre	i	the	you	to	and	a	me	...	writer	motivo	bake	insist	wel
1519	Up Up & Away	Kid Cudi	Hip-hop	0.044061	0.040230	0.001916	0.007663	0.044061	0.015326	0.017241	...	0.0	0.0	0.0	0.0	0
1379	Dear Life	Anthony Hamilton	Hip-hop	0.054795	0.018265	0.059361	0.004566	0.004566	0.022831	0.018265	...	0.0	0.0	0.0	0.0	0
1455	Back To The Wall	Steve Earle	Country	0.064846	0.047782	0.017065	0.040956	0.023891	0.013652	0.010239	...	0.0	0.0	0.0	0.0	0
1441	Do You Feel Me	Anthony Hamilton	Hip-hop	0.058824	0.006920	0.114187	0.017301	0.020761	0.000000	0.079585	...	0.0	0.0	0.0	0.0	0
1653	Shuttin' Detroit Down	John Rich	Country	0.011494	0.061303	0.015326	0.019157	0.045977	0.007663	0.019157	...	0.0	0.0	0.0	0.0	0
1589	Old Blue Mountain	John Rich	Country	0.049505	0.024752	0.000000	0.039604	0.054455	0.000000	0.004950	...	0.0	0.0	0.0	0.0	0
1426	Grandma Got Run Over By A Reindeer	Cledus T. Judd	Country	0.013123	0.031496	0.020997	0.007874	0.057743	0.028871	0.015748	...	0.0	0.0	0.0	0.0	0

	Title	Artist	Genre	i	the	you	to	and	a	me	...	writer	motivo	bake	insist	wel
1654	Me And Those Dreamin' Eyes Of Mine	D'Angelo	Hip-hop	0.040609	0.022843	0.053299	0.007614	0.015228	0.005076	0.038071	...	0.0	0.0	0.0	0.0	0
1591	Old Time Sake	Kathleen Edwards	Country	0.037975	0.031646	0.082278	0.012658	0.018987	0.006329	0.018987	...	0.0	0.0	0.0	0.0	0
1474	DDT	Jurassic 5	Hip-hop	0.036364	0.036364	0.036364	0.036364	0.054545	0.027273	0.009091	...	0.0	0.0	0.0	0.0	0
1675	My Little Girl in Tennessee	Lester Flatt & Earl Scruggs	Country	0.053719	0.028926	0.008264	0.016529	0.004132	0.004132	0.033058	...	0.0	0.0	0.0	0.0	0
1466	Whatta You Know	Cypress Hill	Hip-hop	0.049080	0.092025	0.113497	0.012270	0.036810	0.015337	0.006135	...	0.0	0.0	0.0	0.0	0
1657	Creepy Crawl	Necro	Hip-hop	0.000000	0.034483	0.045977	0.022989	0.107280	0.022989	0.015326	...	0.0	0.0	0.0	0.0	0
1595	It Doesn't Matter	Alison Krauss & Union Station	Country	0.085938	0.015625	0.054688	0.023438	0.000000	0.015625	0.000000	...	0.0	0.0	0.0	0.0	0
1427	Chambre De Gosses	Passi	Hip-hop	0.000000	0.000000	0.000000	0.000000	0.000000	0.002331	0.000000	...	0.0	0.0	0.0	0.0	0
1503	Tequila (Tequila Sunrise)	Cypress Hill	Hip-hop	0.000000	0.000000	0.000000	0.000000	0.000000	0.048276	0.034483	...	0.0	0.0	0.0	0.0	0
1533	Drop In A Bucket	Mary Gauthier	Country	0.076923	0.030100	0.070234	0.016722	0.023411	0.056856	0.010033	...	0.0	0.0	0.0	0.0	0
1707	Buckingham Palace	Canibus	Hip-hop	0.045959	0.049128	0.025357	0.025357	0.020602	0.026941	0.011093	...	0.0	0.0	0.0	0.0	0
1684	These Days	Kasey Chambers	Country	0.026316	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0
1431	Crows	Allison Moorer	Country	0.057143	0.068571	0.000000	0.034286	0.017143	0.022857	0.028571	...	0.0	0.0	0.0	0.0	0

20 rows × 4979 columns

Question 3.9. Classify every song in the newly generated test set, then compute the proportion of correct classifications. (It may take some minutes to complete the classification of these 20 songs)

In [78]:
```python
numCorrect = 0
numSongs = 20
test_song_features = test_lyrics.loc[1519, feature_labels]
print("The classification of",test_lyrics.loc[1519, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1519, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1379, feature_labels]
print("The classification of",test_lyrics.loc[1379, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1379, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1455, feature_labels]
print("The classification of",test_lyrics.loc[1455, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1455, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1441, feature_labels]
print("The classification of",test_lyrics.loc[1441, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1441, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1653, feature_labels]
print("The classification of",test_lyrics.loc[1653, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1653, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1589, feature_labels]
print("The classification of",test_lyrics.loc[1589, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1589, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1426, feature_labels]
print("The classification of",test_lyrics.loc[1426, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1426, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1654, feature_labels]
print("The classification of",test_lyrics.loc[1654, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1654, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1591, feature_labels]
print("The classification of",test_lyrics.loc[1591, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
```

```python
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1591, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1474, feature_labels]
print("The classification of",test_lyrics.loc[1474, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1474, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1675, feature_labels]
print("The classification of",test_lyrics.loc[1675, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1675, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1466, feature_labels]
print("The classification of",test_lyrics.loc[1466, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1466, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1657, feature_labels]
print("The classification of",test_lyrics.loc[1657, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1657, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1595, feature_labels]
print("The classification of",test_lyrics.loc[1595, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1595, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1427, feature_labels]
print("The classification of",test_lyrics.loc[1427, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1427, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1503, feature_labels]
print("The classification of",test_lyrics.loc[1503, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1503, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1533, feature_labels]
print("The classification of",test_lyrics.loc[1533, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1533, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1707, feature_labels]
print("The classification of",test_lyrics.loc[1707, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1707, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1684, feature_labels]
print("The classification of",test_lyrics.loc[1684, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1684, "Genre"]):
    numCorrect += 1
```

```
test_song_features = test_lyrics.loc[1431, feature_labels]
print("The classification of",test_lyrics.loc[1431, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1431, "Genre"]):
    numCorrect += 1
print("The proportion of correct classifications is",(numCorrect/numSongs))
```

```
The classification of Up Up & Away is Country
The classification of Dear Life is Hip-hop
The classification of Back To The Wall is Hip-hop
The classification of Do You Feel Me is Hip-hop
The classification of Shuttin' Detroit Down is Country
The classification of Old Blue Mountain is Country
The classification of Grandma Got Run Over By A Reindeer is Hip-hop
The classification of Me And Those Dreamin' Eyes Of Mine is Country
The classification of Old Time Sake is Hip-hop
The classification of DDT is Hip-hop
The classification of My Little Girl in Tennessee is Country
The classification of Whatta You Know is Hip-hop
The classification of Creepy Crawl is Country
The classification of It Doesn't Matter is Country
The classification of Chambre De Gosses is Hip-hop
The classification of Tequila (Tequila Sunrise) is Hip-hop
The classification of Drop In A Bucket is Hip-hop
The classification of Buckingham Palace is Hip-hop
The classification of These Days is Hip-hop
The classification of Crows is Hip-hop
The proportion of correct classifications is 0.55
```

At this point, you've gone through one cycle of classifier design. Let's summarize the steps:

1. From available data, select test and training sets.
2. Choose an algorithm you're going to use for classification.
3. Identify some features.
4. Define a classifier function using your features and the training set.
5. Evaluate its performance (the proportion of correct classifications) on the test set.

4. Feature design (15 points)

One way to interpret the accuracy of a classifier is to compare it to another classifier.

Question 4.1. Below we've provided 10 features selected by the staff `["come", "do", "have", "heart", "make", "never", "now", "wanna", "with", "yo"]`. Build a 5-nearest-neighbor classifier using these features and compute its accuracy on the test set.

In [79]:
```
def classify(test_features, train_dataframe, k):
    """Return the most common class among k nearest neigbors to test_row."""
    feature_labels = ["come", "do", "have", "heart", "make", "never", "now", "wanna", "with", "yo"]

    genre_and_distances = train_dataframe[["Genre"]].copy()
    distances_to_test_song = []
    for index, row in train_dataframe.iterrows():
        features = row.loc[feature_labels]
        the_distance = distance(test_features, features)
        distances_to_test_song.append(the_distance)

    genre_and_distances["Distance"] = distances_to_test_song
    genre_and_distances = genre_and_distances.nsmallest(k, "Distance")

    return genre_and_distances['Genre'].mode().values[0]
```

In [80]:
```
numCorrect = 0
numSongs = 20
test_song_features = test_lyrics.loc[1519, feature_labels]
print("The classification of",test_lyrics.loc[1519, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1519, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1379, feature_labels]
print("The classification of",test_lyrics.loc[1379, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1379, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1455, feature_labels]
print("The classification of",test_lyrics.loc[1455, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1455, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1441, feature_labels]
print("The classification of",test_lyrics.loc[1441, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1441, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1653, feature_labels]
print("The classification of",test_lyrics.loc[1653, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1653, "Genre"]):
    numCorrect += 1
```

```python
test_song_features = test_lyrics.loc[1589, feature_labels]
print("The classification of",test_lyrics.loc[1589, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1589, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1426, feature_labels]
print("The classification of",test_lyrics.loc[1426, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1426, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1654, feature_labels]
print("The classification of",test_lyrics.loc[1654, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1654, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1591, feature_labels]
print("The classification of",test_lyrics.loc[1591, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1591, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1474, feature_labels]
print("The classification of",test_lyrics.loc[1474, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1474, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1675, feature_labels]
print("The classification of",test_lyrics.loc[1675, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1675, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1466, feature_labels]
print("The classification of",test_lyrics.loc[1466, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1466, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1657, feature_labels]
print("The classification of",test_lyrics.loc[1657, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1657, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1595, feature_labels]
print("The classification of",test_lyrics.loc[1595, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1595, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1427, feature_labels]
print("The classification of",test_lyrics.loc[1427, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1427, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1503, feature_labels]
print("The classification of",test_lyrics.loc[1503, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
```

```
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1503, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1533, feature_labels]
print("The classification of",test_lyrics.loc[1533, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1533, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1707, feature_labels]
print("The classification of",test_lyrics.loc[1707, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1707, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1684, feature_labels]
print("The classification of",test_lyrics.loc[1684, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1684, "Genre"]):
    numCorrect += 1
test_song_features = test_lyrics.loc[1431, feature_labels]
print("The classification of",test_lyrics.loc[1431, "Title"],"is",classify(test_song_features, train_lyrics, k=5))
if((classify(test_song_features, train_lyrics, k=5)) == test_lyrics.loc[1431, "Genre"]):
    numCorrect += 1
print("The proportion of correct classifications is",(numCorrect/numSongs))
```

```
The classification of Up Up & Away is Country
The classification of Dear Life is Country
The classification of Back To The Wall is Hip-hop
The classification of Do You Feel Me is Country
The classification of Shuttin' Detroit Down is Country
The classification of Old Blue Mountain is Country
The classification of Grandma Got Run Over By A Reindeer is Country
The classification of Me And Those Dreamin' Eyes Of Mine is Hip-hop
The classification of Old Time Sake is Country
The classification of DDT is Country
The classification of My Little Girl in Tennessee is Country
The classification of Whatta You Know is Country
The classification of Creepy Crawl is Country
The classification of It Doesn't Matter is Country
The classification of Chambre De Gosses is Country
The classification of Tequila (Tequila Sunrise) is Country
The classification of Drop In A Bucket is Country
The classification of Buckingham Palace is Country
The classification of These Days is Country
The classification of Crows is Hip-hop
The proportion of correct classifications is 0.45
```

Question 4.2. Are the features you chose better or worse than the staff features at classifying the test set? Why do you think this is so?

The features I chose were better because they had a higher proportion of correct classification.

Question 4.3. Is there anything random about a classifier's accuracy measured in this way? Is it possible that the difference in classifier performance is due to chance? If so, describe (in 2-3 sentences) how you would investigate that.

I do not think that there is any randomness in these classifiers since all it does is test whether the words are in the songs and based on that it determines whether the song is hip-hop or country.

5. Computational thinking (15 points)

The following questions are answered via a video of no more than 5 minutes. Everybody must speak. You will provide the link to that video in the answer box.

Question 5.1: Specifically refer to some lines of code, or the thought processes that you made in all the above solutions to elaborate computational concepts which are used in solving the project.

```
In [ ]:
```

Question 5.2: How did you work as a team to complete the project?

```
In [ ]:
```

Question 5.3 (Optional - no credit): Draw a picture (or better yet, a data visualization) of life before, during, and/or after taking Computational Thinking with Data Science course.

```
In [ ]:
```