Prediction

After looking into and experimenting with the dataset, we quickly came to an argument: Using features of music to perform linear regression in hope of predicting another qualitative trait of music is not valuable. Therefore, we turned our heads to KNN classification, since it is way more meaningful to classify music based on its qualitative characteristics. For instance, if people are having a hard time to recognize the genre of the music they are listening to, and they do not possess professional music knowledge, they can pick up our classifier and it will help them tremendously. About 85% of our total dataset is used to construct the training set, and the left 15% is reserved for testing. Using the KNN classification with k=3, we are able to classify between Rock and Electronic through these features: acousticness, danceability, energy and instrumentalness. We select these features based on the reasoning in the aggreated table section above.

In [11]:

```python
# set the random seed so that results are reproducible
np.random.seed(109)

def su(column):
   return (column - np.mean(column)) / np.std(column)

# Function that transform data of two genres into standard units
def into_su(table, genre_1, genre_2):
   table = table.where('track_genre', are.contained_in(genre_1 + " " + genre_2))
   return Table().with_columns(
       'track_genre', table.column('track_genre'),
       'track_title', table.column('track_title'),
       'acousticness', su(table.column('acousticness')),
       'danceability', su(table.column('danceability')),
       'energy', su(table.column('energy')),
       'instrumentalness', su(table.column('instrumentalness')),
       'liveness', su(table.column('liveness')),
       'speechiness', su(table.column('speechiness')),
       'tempo', su(table.column('tempo')),
       'valence', su(table.column('valence'))
   )

# Distance between two rows of features
def distance(test, train):
   return np.sqrt(np.sum((np.array(list(test)) - np.array(list(train)))**2))

# Return the most common class among k nearest neigbors to test_row
def classify(test_row, train_rows, train_labels, k):
   distances = np.array([distance(test_row, train_row) for train_row in train_rows.rows])
   genre_and_distances = Table().with_columns('Genre', train_labels, 'Distance', distances).sort('Distance')
   return genre_and_distances.take(np.arange(k)).group('Genre').sort('count', descending = True).column('Genre').item(0)

# Takes one test row and return the most commonn class among k nearest neighbors to it
def classify_feature_row(row):
   return classify(row, train_features, train_tracks.column('track_genre'), 3)
```

In [12]:

```python
# Transform data into standard units
rock_electronic_su = into_su(t_f, 'Rock', 'Electronic')

# We choose 85% of our dataset to construct the training set and the left 15% to construct the test set
training_proportion = 275/325

shuffeled_table = rock_electronic_su.sample(with_replacement=False)
num_tracks = shuffeled_table.num_rows
num_train = int(num_tracks * training_proportion)
num_test = num_tracks - num_train

train_tracks = shuffeled_table.take(np.arange(num_train))
test_tracks = shuffeled_table.take(np.arange(num_train, num_tracks))

print("Training set size: ",   train_tracks.num_rows, ";",
     "Test set size: ",       test_tracks.num_rows)

# Choose features
prediction_features = make_array('acousticness', 'danceability', 'energy', 'instrumentalness')

train_features = train_tracks.select(prediction_features)
```

```
test_features = test_tracks.select(prediction_features)
```

```
Training set size:  275 ; Test set size:  50
```

In [13]:

```
# Perform prediction using k-value of 3
test_guesses = test_features.apply(classify_feature_row)
proportion_correct = np.count_nonzero(test_guesses == test_tracks.column('track_genre')) / test_tracks.num_rows
proportion_correct
```

Out[13]:

```
0.9
```

In [14]:

```
# Show the table of correctness and explore patterns in mistakes
test_track_correctness = test_tracks.select('track_title', 'track_genre').with_column('Was correct', test_guesses ==
test_tracks.column('track_genre'))
test_tracks.join('track_title', test_track_correctness).select(['track_genre'] + list(prediction_features) + ['Was
correct']).sort('Was correct', descending=False).show(5)
```

track_genre	acousticness	danceability	energy	instrumentalness	Was correct
Electronic	0.981998	-0.675465	-1.29283	0.62478	False
Rock	1.10619	1.62896	-1.88411	0.48764	False
Electronic	-0.525448	-0.0270166	-0.997525	0.243931	False
Electronic	1.24785	-0.301917	-0.562653	-0.794205	False
Electronic	0.560187	0.947889	-1.29486	-0.761794	False

... (45 rows omitted)

From the value of proportion_correct, we get to know that our classifier has an accuracy of
90%
90%. By pulling out the table of mistakes, we can't really directly tell any patterns in them. However, we did notice that music could be classified into multiple genres and yet the data we operated upon only has music classified in a single genre. We know that KNN classifier tends to make mistakes along the decision boundary, and music that are labeled with multiple genres tends to live on such boundary. In this case, the mistakes that our classifier made could've been classified with genre Rock / Electronic in the original dataset, but the sample data cannot reflect this.

Conclusion

With p-value of 5%, we failed to reject our null hypothesis, meaning that the small discrepancies appeared in the dataset are due to a chance model. The accuracy of our KNN classfier with k=3 is

90%

90%, which is good enough for a small dataset with 325 Rock and Electronic tracks in total.

We did notice some limitations in our explorations with the sampled dataset. In our observations of the dataset, we noticed an abnormality that the sum of #tracks in genres table does not add up to the number of total tracks mentioned in the project introduction. We think this might be caused by duplicates in the original dataset, since a piece of music can be simultaneously classified into different genres. However, we only see one genre being assigned to each track in tracks table, and this will introduce bias towards our understanding of the sample's population distribution. We would love to perform **Bootstrapping** on tracks table to get a better sense at how the original distribution looks like. Nevertheless, we already lost the information of whether the track is being labeled in multiple genres. Therefore, our produced estimation would still be flawed since it comes from a potentially biased data sample.