Making Your Own Christmas Song Classifier

Machine learning with Spotipy

By Iris Kotsinas and Ina Eriksson

Classification of Christmas songs

Introduction

The present music industry includes a great variety of genres and whoever you are, you are most likely to find a genre of your own taste. In order to match a given song to a genre, one can study the way it is composed or its musical style, since the basic genres have a few principle aspects that make it easier to identify them. However, manually classifying songs is time consuming and requires full attention. Therefore, the use of machine learning for this purpose would be of great interest.

This project report explains how a given song can be classified based on its audio features. More specifically, it examines how a program can identify whether the given song is a christmas song or not. Concerning model selection for christmas song classification, the three machine learning models **Decision Tree Classifier**, **K-Nearest Neighbor Classifier** and **Random Forest Classifier** are used. Performance comparisons between models can provide insights into the advantages and drawbacks of each method and help us decide which model is the most suitable for similar problems. The training and test data consist of two playlists with 216 and 218 songs respectively. The christmas playlist contains both classical and modern christmas songs, and the ordinary playlist contains various genres, such as pop, rock, rap and house. Some of the music albums can be seen in Figure 1.

GUNS N' ROSES USE YOUR ILLUSION II
KHALID
TALK
I hate being Bi-Polar its awesome
michael bublé
christmas
deluxe special edition
White Christmas
a winter romance
Dean Martin
ORCHESTRA CONDUCTED BY GUS LEVENE

Figure 1. Some of the selected music albums.

Background

Spotify's audio features

Spotify offers developers the possibility to read calculated audio features of tracks [1]. The features are divided into three categories: **mood**, **properties** and **context**. Mood includes the features **danceability**, **valence**, **energy** and **tempo**. Danceability is based on rhythm stability, beat strength, and overall regularity and describes how suitable a track is for dancing. The danceability is given as a value between 0 and 1 where 0 is the least danceable and 1 is the most danceable. Valence describes the musical positiveness by giving a score between 0 and 1, where a track with a low score sounds more negative, depressive and angry and a track with a high valence score sounds more positive and cheerful. Energy is a measure of the perceptual intensity and a track with high energy is often experienced as loud and noisy. Tempo describes the pace of a track and gives the overall tempo of a track as beats per minute.

The category properties includes the features **loudness**, **speechiness** and **instrumentalness**. Loudness is the average loudness of a track given in decibel. Speechiness provides the number of spoken words in a track, a large value of speechiness indicates that it is a talk show, audio book or similar.

The category context includes **liveness** and **acousticness**. Liveness indicates whether the song was recorded with an audience or not. High liveness value indicates that the song was performed live. Acousticness describes how acoustic the song is, a value close to 1 indicates that it is likely to be an acoustic song.

Machine learning models

Decision Tree Classifier

Decision Tree Classifier is a supervised learning method used for classification and regression. Decision trees learn from data to approximate a sine curve with a set of decision rules. The deeper the tree, the more complex the decision rules and the fitter the model. Classification or regression models of decision trees are structured like a tree. They break down datasets into smaller and smaller subsets and the final result is a tree with decision nodes and leaf nodes [2].

K-Nearest Neighbor Classifier

The K-nearest neighbor (KNN) algorithm is a simple supervised machine learning algorithm that can be used to solve both classification and regression problems. KNN algorithms are based on the feature similarity approach. The value k represents the number of nearest neighbors. This value is the core deciding factor of the algorithm [2].

Random Forest Classifier

Random Forest Classifier is one of many powerful tools in machine learning. Similar to KNN and Decision Trees Classifier, Random Forest Classifier is used as a classification algorithm. A positive outcome from using Random Forest is the minimized risk of overfitting. Random Forest creates multiple decision trees during the training process and the final decision is the resulting outcome from the majority of the trees. One of the big drawbacks with Random Forest is the computational cost [2].

Data collection

The data is retrieved from Spotify with the library Spotipy [3]. Spotify is a python library for Spotify Web API, which gives full access to the music data from Spotify. Based on simple REST principles, the Spotify Web API endpoints return JSON metadata about music artists, albums, and tracks, directly from the Spotify Data Catalogue.

The dataset contains a mix of christmas songs as well as ordinary songs. 216 of the songs are Christmas songs and 218 are ordinary songs. The dataset essentially contains information about each song such as track name, artist name, danceability, key of the song, acousticness, speech, tempo, liveness, valence, popularity and decade along with other factors that can help deduce meaningful information in determining if a song can be classified as a christmas song or not.

Method

First a client credentials manager must be initialized. Authorized requests to the Spotify platform requires that you are granted permission to access data. Therefore, client credentials must be used together with server-to-server authentication. Only endpoints that do not access user information can be accessed. The client credentials can be found with the Spotify Web API, where one can receive a personal Client ID and Secret Key. The Client ID and Secret Key are thereafter used to obtain an access token, granting permission to access the wanted data.

```
cid =''
secret = ''
username=''
client_credentials_manager = SpotifyClientCredentials(client_id=cid,
client_secret=secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

Thereafter, the playlists are declared. The playlist identification number can be found in the URL of a playlist and is declared as a variable in the code. It is later used as an input to a function that extracts the URI of all songs in the playlist.

```
playlist1 = "1ZmU9yhNg0Oee9LH3E9sOR"
playlist2 = "0xBmP4bvMO7CbnAaKeGvb2"

list_tracks1 = get_playlist_URIs(username, playlist1)
list_tracks2 = get_playlist_URIs(username, playlist2)

audio_features_df1 = get_audio_features(list_tracks1)
audio_features_df2 = get_audio_features(list_tracks2)
```

playlist1 contains christmas songs and **playlist2** contains ordinary songs. Then the playlist URIs are retrieved. The function that extracts the URI of all songs returns a list with identification numbers used as input to a function that returns the features of the songs.

```
def get_playlist_tracks(username, playlist_id):
  tracks_list = []
  results = sp.user_playlist(username, playlist_id,
  fields="tracks,next")
  tracks = results['tracks']
  while tracks:
     tracks_list += [ item['track'] for (i, item) in
     enumerate(tracks['items']) ]
     tracks = sp.next(tracks)
  return tracks_list

def get_playlist_URIs(username, playlist_id):
  return [t["uri"] for t in get_playlist_tracks(username,
  playlist_id)]
```

The audio features of each track are then extracted. The function that extracts features from Spotify can only handle a maximum 50 songs at the time and the playlists are therefore split up using the function splitlist.

```
def get_audio_features(track_URIs) :
  features = []
  r = splitlist(track_URIs,5)
  for pack in range(len(r)):
     features = features + (sp.audio_features(r[pack]))
  df = pd.DataFrame.from_dict(features)
  df["uri"] = track_URIs
  return df

def splitlist(track_URIs,step):
    return [track_URIs[i::step] for i in range(step)]
```

A column called target is added to both playlists with a label 1 if the song belongs to christmas songs and label 0 if the song belongs to ordinary songs. The two lists are added together and are called training data. All of this is repeated for a set of test songs.

```
audio_features_df1["target"] = 1
audio_features_df2["target"] = 0

training_data = pd.concat([audio_features_df1,audio_features_df2], axis=0,
join='outer', ignore_index=True)
```

Using the built in function of sklearn, the data is split up into training and test data. 80 % of the data is used as training data and 20 % of the data is used as test data. Only the features which best describe the difference between christmas and ordinary songs are used, referring to the first line of code below.

```
selected_features = ['danceability','acousticness','energy','speechiness']
train, test = train_test_split(training_data, test_size = 0.2)
x_train = train[selected_features]
y_train = train['target']
x_test = test[selected_features]
y_test = test['target']
```

The Decision Tree model is built from sklearn **DecisionTreeClassifier**. The accuracy is calculated with the function accuracy_score of the sklearn library.

```
# Decision tree classifier
dtc = DecisionTreeClassifier()
dt = dtc.fit(x_train,y_train)
y_pred = dtc.predict(x_test)
score_dt = accuracy_score(y_test, y_pred) * 100
score_dt_.append(score_dt)
```

The KNN model is built from sklearn **KNeighborsClassifier**. In our model we have used 5 neighbors.

```
# KNN classifier
knc = KNeighborsClassifier(5)
knc.fit(x_train,y_train)
knn_pred = knc.predict(x_test)
score_knn = accuracy_score(y_test, knn_pred) * 100
score_knn_.append(score_knn)
```

The Random Forest Classifier is built from sklearn **RandomForestClassifier**. In our Random Forest model we have set the number of trees to 100 and the maximum depth of the tree to two.

```
# Random forest classifier
sc = StandardScaler()
X_train = sc.fit_transform(x_train)
X_test = sc.transform(x_test)
classifier = RandomForestClassifier(n_estimators=100, max_depth=2,
random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
score_rf = accuracy_score(y_test, y_pred)*100
score_rf_.append(score_rf)
```

Since the splitting of the data is random and affects the results, the training and classification of the model is repeated 30 times.

Data analysis

The data is analyzed with the use of visualizations such as radar charts and histograms. Radar charts can be used as a general comparison of all features. For this project, the radar chart gives an overview of the data to visually determine the differences between christmas songs and ordinary songs, see Figure 2.

In Figure 2, the eight features in the dataset can be seen at the axises. The points are the mean value of the features which spans an area. From the radar chart it can be seen that christmas songs have an average high value of the features acousticness and instrumentalness whereas the other features are on an average lower than ordinary songs.

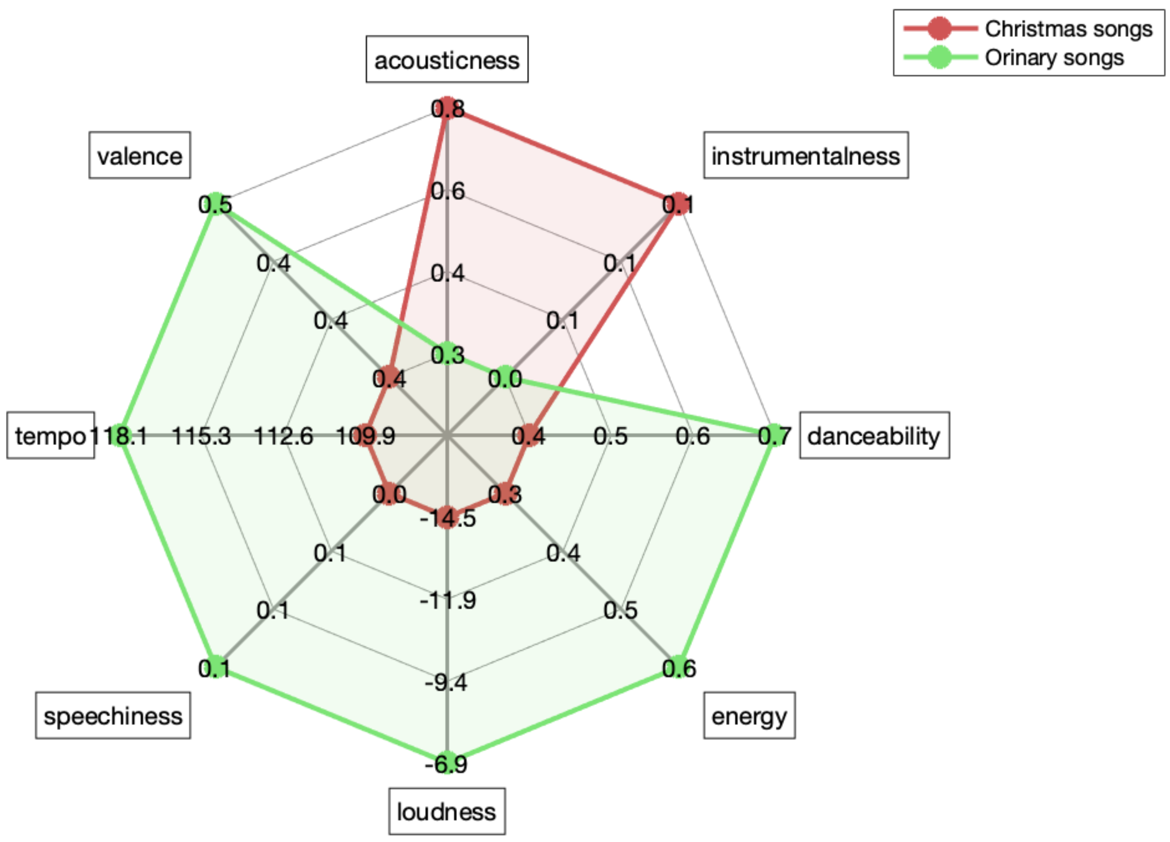

Figure 2. The eight audio features in a radar chart.

The histogram enables the viewer to visually verify which features have significant differences between christmas songs and ordinary songs. Figure 3 shows the distribution of each feature of both christmas songs and ordinary songs.

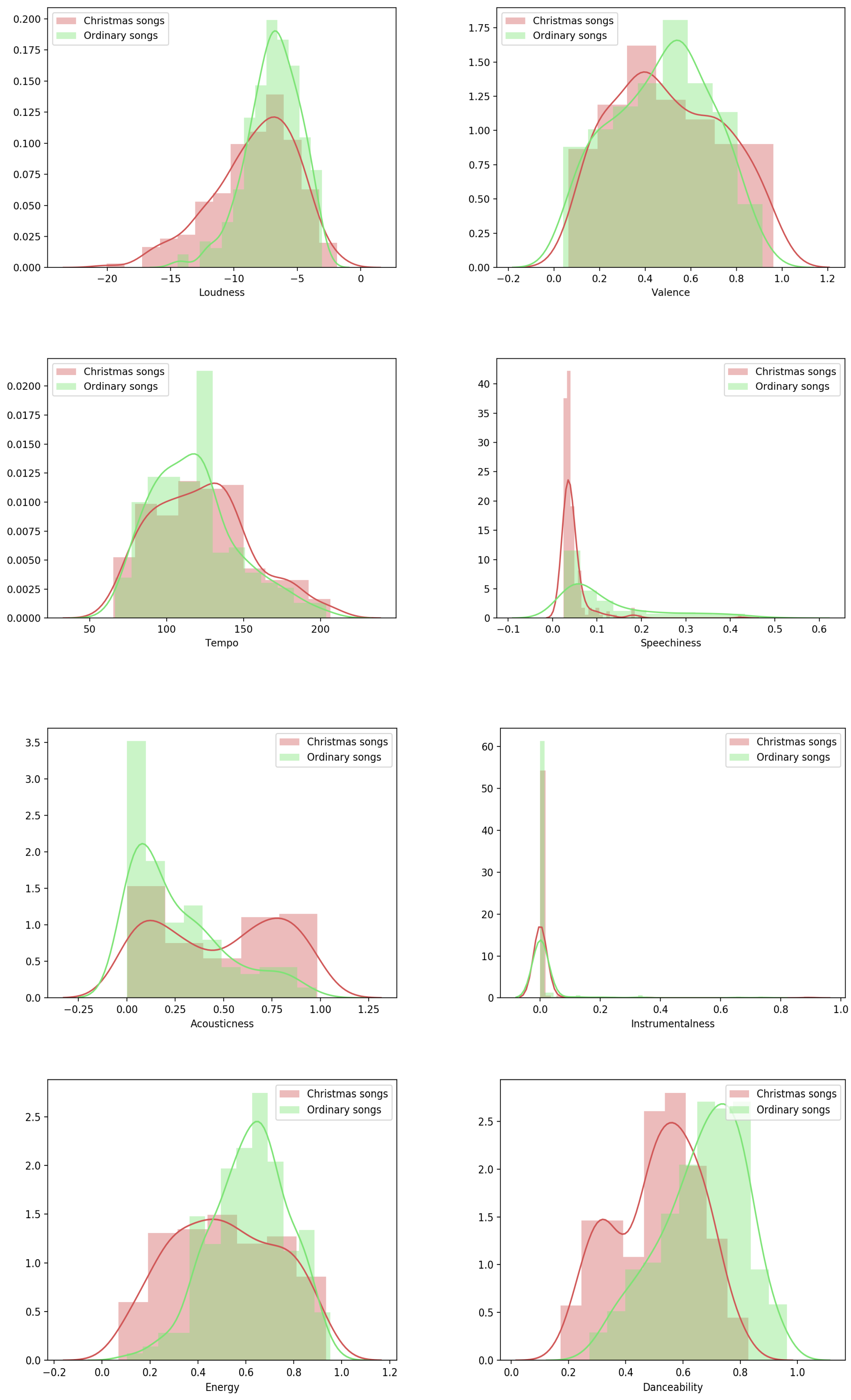

Figure 3. Distribution of each audio feature.

The red curve in Figure 3 shows the distribution of christmas songs and the green curve shows the distribution of the ordinary songs. From a qualitative analysis of the histograms and the radar chart, the selected features in the model are danceability, acousticness, energy and speechiness.

Result

The accuracy of the models can be seen in Figure 4.

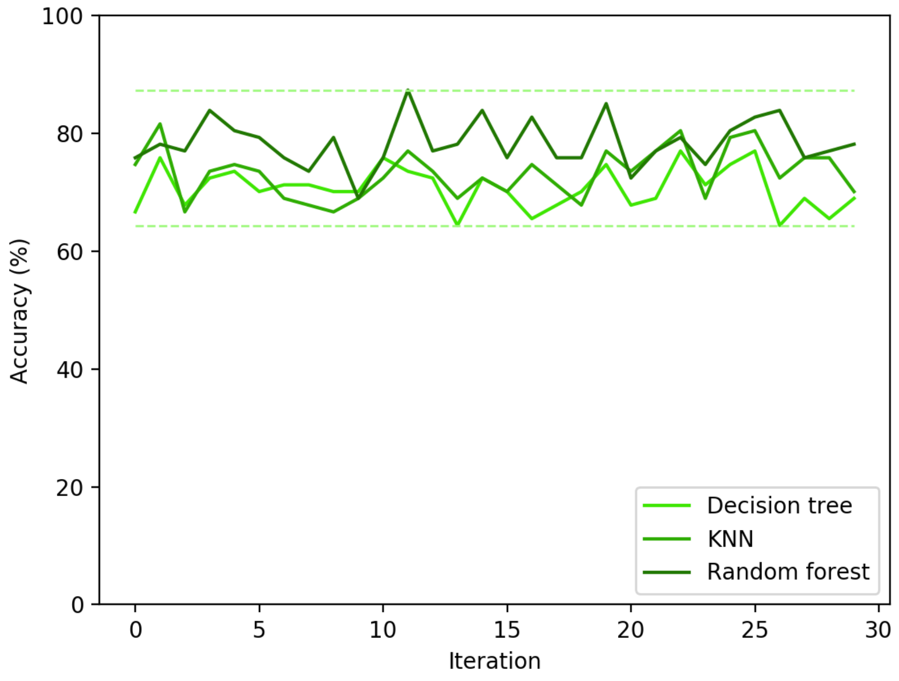

Figure 4. Accuracy of the machine learning models.

The accuracy varies with each iteration, and after 30 iterations the lowest accuracy value was 64% and the highest accuracy value was 87%. Random Forest gives an overall higher accuracy than Decision Tree and KNN. As the data is split up into 80% training data and 20% test data, the models are tested with 88 songs.

The performance of the models are tested with an additional test, with the use of songs not included in either of the playlists as input. Ten new songs, of which the models had never seen before, were introduced and the results can be seen in Table 1.

Model	Accuracy
Decision tree classifier	100 %
KNN classifier	80 %
Random forest classifier	80 %

Table 1. Accuracy of the machine learning models.

All ten songs are correctly classified by the Decision Tree Classifier. 80% of the songs are correctly classified with KNN and Random Forest Classifier. The model performance in the second evaluation does not agree with the results in Figure 4. However, the most reliable results are given in Figure 4, since more songs are used to evaluate the models. This demonstrates that the choice of model is dependent on the data. In order to find the best suited model, the recommendation is to create several models and evaluate the accuracy.

Discussion and conclusion

An important aspect of this research project is to consider what kind of data was chosen to be included in the dataset. The first playlist contains various christmas songs and the second playlist contains songs of different genres, such as pop, rap, rock and house. The christmas playlist consists of both modern and classical christmas songs, which could decrease the accuracy values, as modern christmas songs can be mistaken for ordinary pop songs. It was however of great interest to include the modern christmas songs for a more interesting result, since christmas pop songs are growing more popular.

The size of the datasets might also affect the prediction accuracy of the models. In this project, the playlists contain almost 220 songs each. We chose to keep the two datasets of christmas songs and non-christmas songs balanced, with an even number of entries. The use of larger datasets could affect the performance of the models and the operations would be more computationally expensive. Therefore, the size of the datasets were moderate. Including too few songs in the datasets could possibly lead to histograms with less accurate representation of the playlists. The aim is to find balance between including enough songs to give a good representation of typical christmas songs and ordinary songs, and at the same time keep the computational costs low.

It is also of importance to discuss the audio features included in the datasets. The audio features were chosen with regard to Figure 3, where each feature was visualized in a histogram. The best fitted audio features were chosen by comparing the plots of the christmas and ordinary playlists (see Figure 3). The vaster the difference, the better the feature, since it would increase the accuracy score. The final selected features in the model were danceability, acousticness, energy and speechiness. If the audio features with similar plots had been chosen, the accuracy score would have been lower. The audio features were selected with respect to this project's specific purpose, that is comparing christmas songs with ordinary songs. Had the purpose and data of this study been different, other audio features could have been chosen.

The specific settings of the models used in this project are optimized for this study. Other data could possibly require different settings and values. In conclusion, all three machine learning methods included in this study can be used to create a christmas song classifier with sufficient results. The best classifier is however Random Forest Classifier, which resulted in an accuracy of 87%.

References

[1] Plantinga B. What do Spotify's audio features tell us about this year's Eurovision Song Contest? Medium. [cited: 14-01-21]

[2] James G, Witten D, Hasite T, Tibshirani R. An Introduction to Statistical Learning. Springer Science; 2013.doi: 10.1007/978-1-4614-7138-7.

[3] Spotipy. Welcome to Spotipy. [cited: 2020] https://spotipy.readthedocs.io/en/2.16.1/