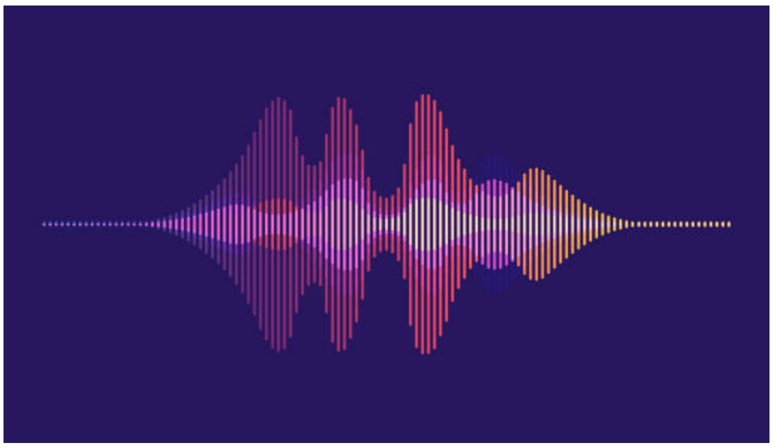
TTT4275 Lab Report

Daniel Borger Larsen
Inger Juni Lærdal

April, 2023



Contents

1 Introduction

In this project we use different classifiers to classify music in the right genre. The dataset we will be using is the GTZAN dataset which consists of 1000 audio tracks with 30, 10 and 5 second duration, it is divided in 10 genres which each consists of 100 tracks. The 10 genres is *pop, metal, disco, blues, reggae, classical, rock, hip hop, country* and *jazz*. The features we will be using is extracted by using the python LIBROSA package in Python [1] and is further described in the Theory section.

People generally prefer some genres over the other, and having an automated way of classifying music into genres will make it easier for people to discover music better suited to their taste, and having their music categorized will make it easier for artists to establish a fanbase.

The report is structured as follows

- The relevant theory is described is the Theory section
- The task is described in the Task section
- The implementation and results are presented in Implementation and results
- And finally the conclusion is given in Conclusion

2 Theory

This section is a presentation of the relevant theory for the report. The relevant classifier for the subsequent implementation is the k-Nearest Neighbor classifier.

The k-Nearest Neighbor (kNN) classifier is a template based classifier, which means that it matches an input x with a set of reference values (templates), which are of the same form as x. Then it chooses the class which is most similar to x, and classify x to that class. This decision rule is called "Nearest neighbor". K-Nearest neighbor is when the algorithm chooses the k most similar references to x, and the class with the most references in this set, gets to assign x to its class. There are multiple ways to construct template based classifiers by choosing different approaches among the following:

- Decision rule (such as NN and K-NN)
- Distance measure between input and references
- Choice of reference

There are a few distance measurements that can be used, but in the following project the *Euclidian distance* (1) and the *relative distance* (2) has been used.

$$d = (x - \mu)^\top (x - \mu) \tag{1}$$

$$d = \left(\frac{(x - \mu)}{\mu}\right)^\top \left(\frac{(x - \mu)}{\mu}\right) \tag{2}$$

Creating a K-NN classifier essentially boils down to choice of reference. A good choice of reference would be seperable features, i.e. features that have little to no overlap among each other. The four most important features, especially in the first two parts in this project, has been; $spectral\_rolloff\_mean$, $mfcc\_1\_mean$, $spectral\_centroid\_mean$ and $tempo$. These features are described as follows:

- $spectral\_roloff\_mean$ is the mean frequency where the signal energy is contained below a certain percent [2], in this case it is 85%.
- $mfcc\_1\_mean$ The $Mel\ Frequency\ Cepstral\ Coefficient$ ($MFCC$) is the few coefficients that represent the shape of the *cepstrum*, where the *Mel cepstrum* where the cepstrum is computed on the Mel bands instead of the Fourier spectrum [2] and [3]. The $mfcc\_1\_mean$ Is the mean value of those coefficients in the first bin.

- *spectral_centroid_mean* Is the mean of the *spectral centroid* where the *spectral centroid* is described with

$$C_f = \Sigma_{k \in K_+} kX(k) \tag{3}$$

  where K_+ is a set containing only non-negatvie frequency indices, k is frequency index, $X(k)$ is the normalised discrete Fourier Transform, see [3] page 136.

- *tempo* is the number of beats per minute, measured in BPM [4].

- $Spectral\_roloff\_var$: the standard deviation of the *spectral* $roloff\ frequency$ at 85%.

3 Task

The task at hand is divided into four distinct parts. The first part entail the design, implementation and evalutaion of a k-NN classifier with $k = 5$. The classifier should classify tracks with the data provided by the dataset *GenreClassData_30s.txt* by evaluating the four features: *spectral rolloff mean, mfcc 1 mean, spectral centroid mean* and *tempo.* The classifier will classify the tracks into one of the following ten genres: *pop, metal, disco, blues, reggae, classical, rock, hip hop, country* and *jazz.* The performance is to be evaluated by discussing its confusion matrix, and error rate for the test set.

The second part is focused on the seperability and its effects on performance. This is done by producing a histogram for each of the listed classes and features, where the classes has been reduced to the following four genres: *pop, disco, metal* and *classical.* The histograms are then analyzed to find the feature with the most overlap between the classes. The classifier is then evaluated with the remaining three features, and compared to the previous iteration.

Third part focuses on feature selection. Design a k-NN classifier using only four features, where at least three of them among the original four. Discuss and compare the results with previous iterations.

Fourth, and final part, is to design any classifier that classifies the audio tracks for all of the ten genres. No limit on what features to select, or how many. All of the data sets are available. The performance is to be evaluated and discussed in comparison to previous designs with error rates and confusion matrices.

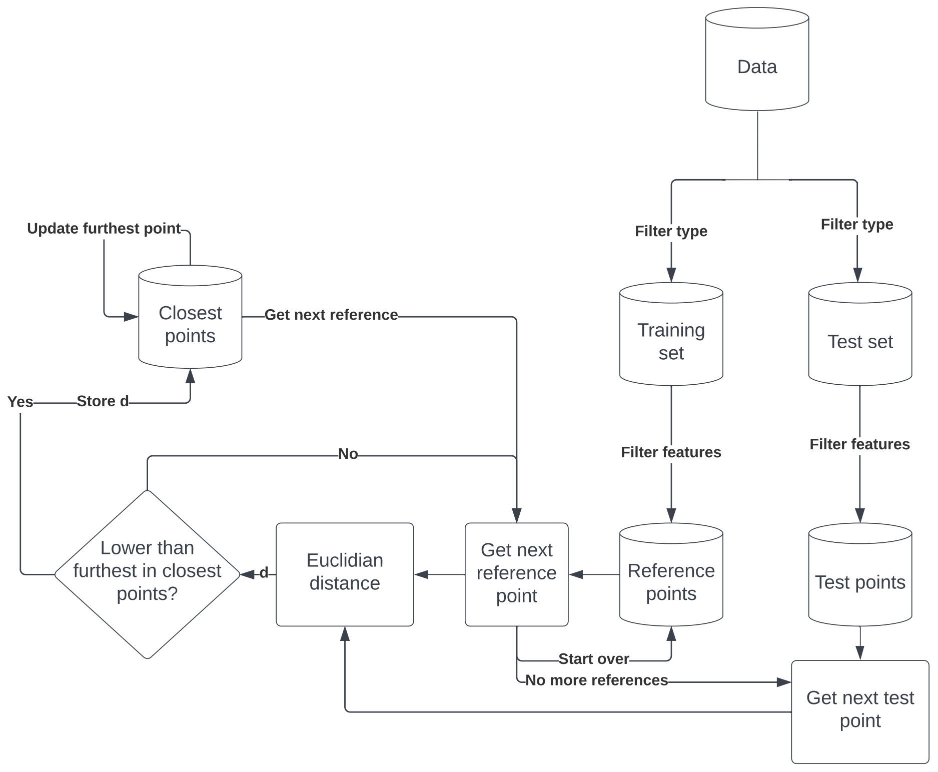


Figure 1: Flow diagram of the filter implementation

4 Implementation and results

In this section the implementations and results to all of the tasks presented previously will be presented. This includes the relevant[1] python implementation, figures and tables presenting the results, and discussions of the results. For easier digestion, the section has four subsections corresponding to the parts previously presented.

4.1 Design and implementation

The designed k-NN classifier uses all of the provided training data as reference, and uses the Euclidian distance to measure similarity. The filter can be visualized with the flow diagram provided by figure 1. Documented code can be found in *Part1.py* in the appendix.

[1]Relevant in this context means only code snippets. All documented code is found in the appendix.

Classified: $\rightarrow$	pop	disco	metal	classical	rock	blues	reggae	hiphop	country	jazz
pop	11	3	1	0	0	1	0	2	1	1
disco	3	5	1	1	4	1	1	2	1	1
metal	0	3	11	0	3	0	0	3	0	0
classical	1	0	0	13	0	0	0	0	2	3
rock	2	6	2	0	2	4	0	1	2	1
blues	1	1	1	1	2	6	0	3	3	2
reggae	1	0	1	1	2	0	7	1	3	4
hip hop	1	3	3	0	1	3	2	5	1	1
country	1	2	2	0	1	2	1	2	5	3
jazz	2	2	0	3	1	2	0	1	0	9

Figure 2: Confusion Matrix for 5-NN classifier. Correct classifications are on the diagonal

This implementation has a quite poor performance with an estimated error rate with respect to the test set, $EER_T = 0.6263$. The confusion matrix is presented in figure 2.

As one can see from the confusion matrix, the classifier is having a hard time with just about every genre, being incorrect the majority of the time. This indicates that there is much overlap between the genres in the features selected. In order to improve the accuracy of the classifier it is important to choose better features, which will be discussed in the next section.

4.2 Seperability

An important step to improving the accuracy of the classifier would be to choose features that don't overlap with other classes. If the classes overlap significantly then the classifier won't get any differentiating information from the feature, so ignoring it would yield approximatly the same result as including it. One other thing that is important to be mindful of in this instance is the relative size of the features. The implemented classifier is using the Euclidian distance (1) to evaluate the similarity, and since there is no weighting implemented on the feature, the size of the feature can be more important than the relative difference. Figure 3 is the histograms corresponding to the feature and genre chosen in this part of the project. The histograms gives a visual insight in how much the features overlap among the classes and will help in choosing features that are more seperable which will improve the accuracy of the classifier.

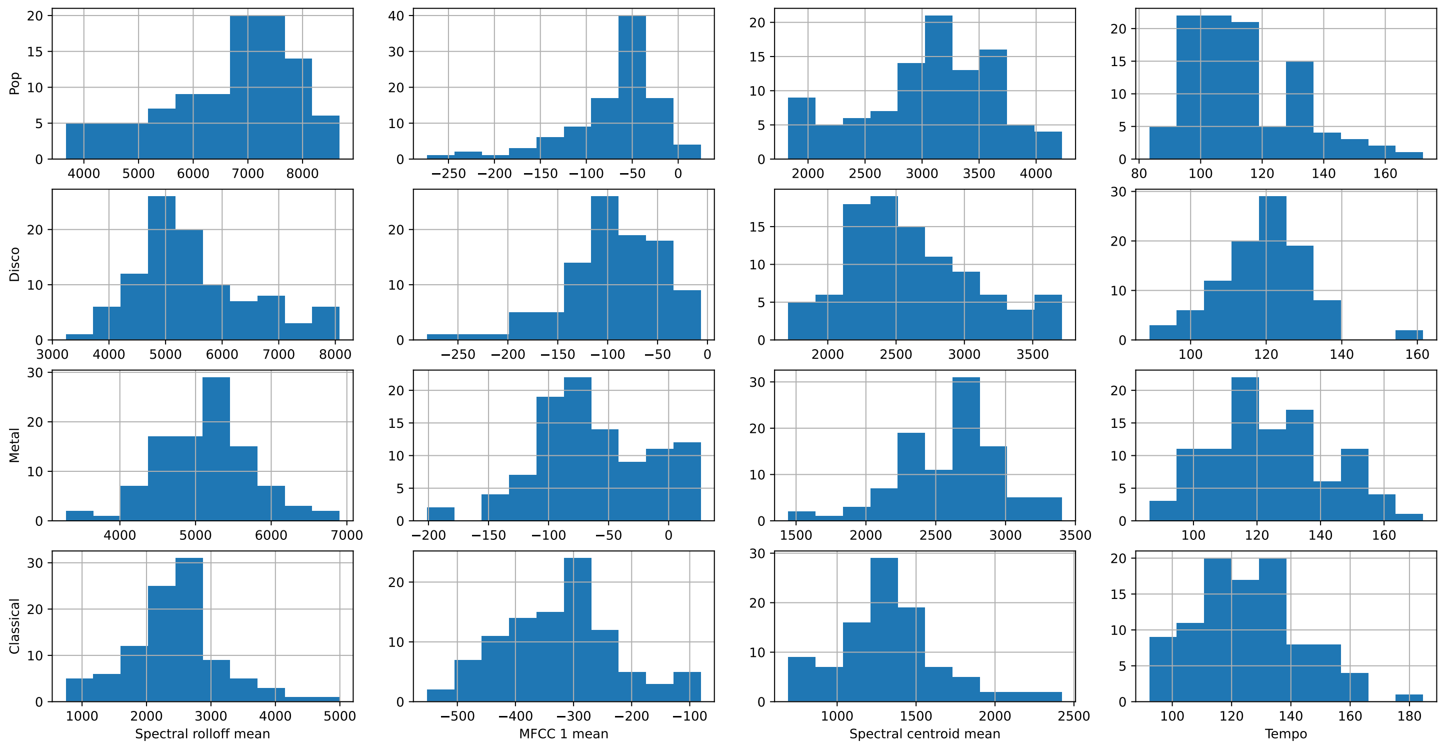


Figure 3: Histogram of the four features and genres

In order to evaluate how much each feature contribute to how an input is classified it is important to consider the extent of overlap between the classes. From the histograms one can see that classical is most separated from the other classes, and there is much overlap between disco and pop, and metal and disco. In this case the tempo is the feature with the lowest values, so it will have the least impact on the similarity, which would be dominated by the bigger features such as spectral rolloff mean, and spectral centroid mean. This is clear in the results, as the results without considering tempo is identical to the results that do consider it.

Figure 4 is a table of the resulting estimated error rates, and confusion matrices when removing one of the features. It is clear from the confusion matrix, that the only genre that might be linearly seperable out of the four is classical, which has a consitently low estimated error rate. This tracks with whats observed from the histograms.
There is also much confusion between the classes disco and pop, and disco and metal, which was also expected. The error rate and confusion matrix when omitting tempo was identical to when it was included. This implies that there was alot of overlap between the classes, and that the size of its values were too small to have a significant effect.

Feature omitted	Estimated error rate EER_T	Confusion matrix				
No feature omitted	0.0.3038		pop	disco	metal	classical
		pop	15	3	2	0
		disco	7	8	4	1
		metal	1	4	15	0
		classical	2	0	0	17
Spectral rolloff mean	0.4557		pop	disco	metal	classical
		pop	14	2	4	0
		disco	8	9	2	1
		metal	3	12	5	0
		classical	3	1	0	15
mfcc 1 mean	0.2911		pop	disco	metal	classical
		pop	15	3	2	0
		disco	7	9	3	1
		metal	1	4	15	0
		classical	1	0	1	17
Spectral centroid mean	0.3038		pop	disco	metal	classical
		pop	14	4	2	0
		disco	5	10	4	1
		metal	1	5	14	0
		classical	2	0	0	17
Tempo	0.3038		pop	disco	metal	classical
		pop	15	3	2	0
		disco	7	8	4	1
		metal	1	4	15	0
		classical	2	0	0	17

Figure 4: Table of feature analysis

4.3 Feature selection

To improve the classifier one of the four selected features is to be replaced. The decision of which is to choose the feature with the least amount of seperability with the feature with the most amount of seperability. When analyzing the remaining features provided it is evidendt that the only features that were big enough to actually have the capacity to affect the classifier are spectral centroid var, spectral bandwidth mean, spectral bandwith var and *spectral rolloff var*. The histogram for the candidate features is provided in figure 5. There is good seperation in most of the features, so it is not clear which will be the best feature to include, in the pursuit of certainty the function *tryAll* in the appendix has been implemented to calculate the feature with the highest accuracy. The function was tested on only the four genres, because of time considerations, so it is an assumption that it will also be the most accurate when considering all genres. Swapping *tempo* with *spectral rolloff var* gave the best results, with an estimated error rate of 0.1392 and confusion matrix given by table 1. This is a significant increase and a reasonably good classifier.

Classified genre →	pop	disco	metal	classical
pop	16	4	0	0
disco	3	14	2	1
metal	0	1	19	0
classical	0	0	0	19

Table 1: Confusion matrix of classifier with spectral rolloff var replacing tempo

When apply the swap to all genres the resulting estimated error rate was 0.5859 and confusion matrix given by figure 6. This is only a marginal gain on the estimated error of 0.6262 from the first iteration, and a long way from 0.1392. From the confusion matrix it seems that we have gained significant accuracy in the classes that we have analyzed, but lost most of that gain to increased errors among the other genres. This indicates that among the features selected it is there is not much seperation among all genres, even though we can see good seperation among subsets of all classes.

4.4 Own design

From analyzing the data it is suspected that the data is not very seperable, so a linear classifier would probably not generate good results. A neural network would probably be the most accurate, but due to implementation complexity it wont be implemented. A Gaussian mixture model (GMM)

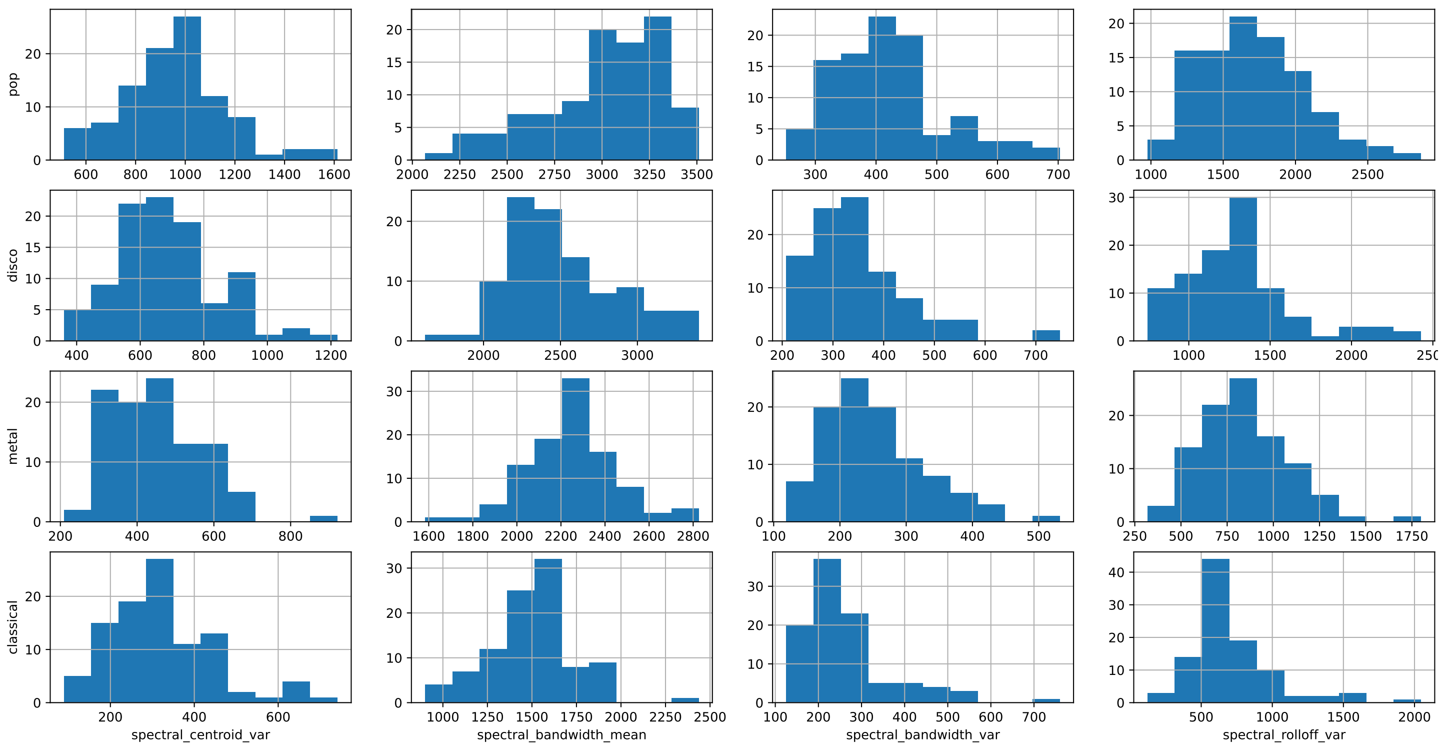


Figure 5: Histogram of the candidate features

Classified: →	pop	disco	metal	classical	rock	blues	reggae	hiphop	country	jazz
pop	14	1	0	0	0	0	1	3	1	0
disco	1	8	1	0	4	0	14	2	0	0
metal	0	0	15	0	3	1	0	0	0	1
classical	0	0	1	14	0	0	0	0	0	4
rock	2	4	3	0	1	3	2	2	2	1
blues	0	0	4	0	0	9	1	1	4	1
reggae	3	1	0	0	2	1	8	3	1	1
hip hop	3	3	4	0	4	0	3	2	0	1
country	2	2	0	0	2	3	2	2	4	2
jazz	1	2	1	2	1	1	2	2	1	7

Figure 6: Confusion Matrix when replacing tempo with spectral rolloff var

Classified: $\rightarrow$	pop	disco	metal	classical	rock	blues	reggae	hiphop	country	jazz
pop	14	2	0	0	0	0	1	3	0	0
disco	2	6	3	0	1	1	1	6	0	0
metal	0	1	19	0	0	0	0	0	0	0
classical	0	2	1	7	1	0	2	0	2	4
rock	1	5	4	0	4	0	2	4	0	0
blues	0	3	4	0	2	6	5	0	0	0
reggae	1	2	0	0	2	0	11	4	0	0
hip hop	3	2	0	0	0	0	1	14	0	0
country	3	6	0	0	2	0	3	0	5	0
jazz	1	0	0	0	1	3	4	3	0	8

Figure 7: Confusion Matrix with all features included, and relative distance

could be a good attempt, but looking at the data, a gaussian distribution doesnt seem to be accurate, and implementing a completly different classifier would require alot of potentially unnecesary work. A k-NN seems like a good option, but we can definatly improve upon our current implementation. Our main problem with the current implementation is that it uses the Euclidian distance to measure the similarity between points. This means that bigger values will tend to dominate the outcome. To correct for this we can use the relative distance, instead of the absolute distance to measure similarity, descirbed in equation (2).

Since we are not certain how the feature analysis we have done previously will affect the accuracy we decide to include all of the features. The reasoning behind the choice is that more features allow for more possibilities to find features able to separate the genres from each other. It is found that the resulting estimated error rate of 0.5253, and confusion matrix given by figure 7.

The results are not good, but they are an improvement on previous iterations. Further improvements can be obtained from better feature selection, but it is not clear how much it will be able to improve, so a k-NN classifier might not be the right choice for this problem.

5 Conclusion

We have built a k-NN classifier to classify a set of tracks into 10 different genres, taking four features into consideration. The estimated error rate was 0.6263, which is high. Attempts to improve its performance included looking at how the feature selection affects the performance of the classifier by evaluating the subset of classes *Pop, Disco, Metal* and *Classical.* The inital estimated error rate in this subset was 0.3038, but with a better choice of features we were able to bring the estimated error rate down to 0.1392. This did not however result in significant the classifier when considering all classes, as the estimated error rate was only reduced to 0.6263.

The biggest problem for our implementation of the classifier, was using the Euclidian distance to measure similarity. It became apparant that the features with the highest values became too dominant, so even though there might be some features that were seperable, they would become insignificant to largerer features. An attempt to solve this was replacing the Euclidian distance with a relative distance. This did have a some improvement on the classifier, reducing the estimated error rate from 0.6263 to 0.5253, but this is still poor.

Further improvements could be made by further analysis of the features and choose the most seperable, as the final implementation takes all features into account. However it is unclear how much this will be able to improve the classifier, so the choice of tempelate based classifier might not be satisfactory. A major advantage with the k-NN classifier is that it is relativly simple to implement, but it might not be sufficient for this application, so more complex classifiers as neural networks might be necessary.

A Documented Code

File:"Part1.py" - Implementation of the k-NN classifier from the provided data

```
import numpy as np
import pandas as pd

# ALLFEATURES is a list of all the provided features
ALLFEATURES = ['zero_cross_rate_mean', 'zero_cross_rate_std',
        'rmse_mean', 'rmse_var', 'spectral_centroid_mean',
       'spectral_centroid_var', 'spectral_bandwidth_mean',
       'spectral_bandwidth_var', 'spectral_rolloff_mean',
       'spectral_rolloff_var', 'spectral_contrast_mean',
       'spectral_contrast_var', 'spectral_flatness_mean',
       'spectral_flatness_var', 'chroma_stft_1_mean',
       'chroma_stft_2_mean','chroma_stft_3_mean',
       'chroma_stft_4_mean', 'chroma_stft_5_mean',
       'chroma_stft_6_mean', 'chroma_stft_7_mean',
       'chroma_stft_8_mean','chroma_stft_9_mean',
       'chroma_stft_10_mean', 'chroma_stft_11_mean',
       'chroma_stft_12_mean', 'chroma_stft_1_std',
       'chroma_stft_2_std','chroma_stft_3_std',
       'chroma_stft_4_std', 'chroma_stft_5_std',
       'chroma_stft_6_std', 'chroma_stft_7_std',
       'chroma_stft_8_std','chroma_stft_9_std',
       'chroma_stft_10_std', 'chroma_stft_11_std',
       'chroma_stft_12_std', 'tempo', 'mfcc_1_mean',
       'mfcc_2_mean','mfcc_3_mean', 'mfcc_4_mean',
       'mfcc_5_mean', 'mfcc_6_mean','mfcc_7_mean', 'mfcc_8_mean',
       'mfcc_9_mean', 'mfcc_10_mean', 'mfcc_11_mean', '
       mfcc_12_mean', 'mfcc_1_std', 'mfcc_2_std', 'mfcc_3_std',
       'mfcc_4_std', 'mfcc_5_std', 'mfcc_6_std', 'mfcc_7_std',
       'mfcc_8_std', 'mfcc_9_std', 'mfcc_10_std', 'mfcc_11_std',
       'mfcc_12_std',
       ]

# Map that corresponds a genre to the index of the row in
# the generated confusion matrix
GENREMAP = {"pop": 0, "disco": 1, "metal" : 2, "classical" : 3,
           "rock" : 4, "blues" : 5, "reggae" : 6, "hiphop" : 7,
           "country" : 8, "jazz" : 9}

# Store the provided data in a pandas dataframe
ClassData30 = pd.read_table("Classification music(1)\Classification
    music\GenreClassData_30s.txt")

# Task 1
    # (A) Design a k-NN classifier (with k=5) for all ten genres
    # using only: spectral_rolloff_mean, mfcc_1_mean,
    #sepctral_centroid_mean and tempo. # I.e find the 5 closest
    # references and choose the class with the most references

    # Training a NN classifier would be the same as choosing good
    # references. The references in this case will be the entire
    # training set. The similarity measure is the Euclidian
    # dinstance

def euclidianDistance(x, u):
    """Calculates the Euclidian distance between the input x and
        reference u

    Parameters
    ----------
    x : list(int)
        list of input values
    u : list(int)
        list of reference values
    """
    return np.matmul(np.transpose(x-u), (x-u))

# Relaative distance measure for part4
def relativeEuclidian(x, u):
    """Calculates the relative distance between the input x and
        reference u

    Parameters
    ----------
    x : list(int)
        list of input values
    u : list(int)
        list of reference values
    """
    return np.matmul(np.transpose(np.divide((x-u),u)),
        np.divide((x-u),u))


def NNclassifier(data=ClassData30,
        mapIndex=GENREMAP,
        distance=euclidianDistance,
        k=5,
        features=['spectral_rolloff_mean', 'mfcc_1_mean',
            'spectral_centroid_mean', 'tempo']):

    """A k-NN classifier for the data.


       Parameters
        ----------
        data : pd.Dataframe, optional
            A pandas dataframe containing the relevant data.
        mapIndex : list(tuple(str, int)), optional
            List of the tuple (Genre, index). Genre is a string of a
                genre in data.
            index is an integer that corresponds to the row number
                of Genre in the
            generated confusion matrix
        distane : function, optional
            A function for the choice of similarity measure.
                Euclidian distance is
            the default and recommended in this implementation
        k : int
            Determines the k in k-NN classifier. Gives the number of
                most similar points
            to consider
        features : list(string), optional
            A list of the features the classifier should consider
                when deciding similarity.
            Need to correspond to keys in data
    """

    # Create the reference set
    referenceSet = data.query("Type == 'Train'")
    referenceSet =
        referenceSet[referenceSet.Genre.isin(mapIndex.keys())]

    # Create the test set
    testSet = data.query("Type == 'Test'")
    testSet = testSet[testSet.Genre.isin(mapIndex.keys())]


    start = time.time()
    # The confusion matrix generated by the classifier
    confusionMatrix = pd.DataFrame(0,
        index=list(mapIndex.values()), columns=mapIndex.keys())


    for i in range(len(testSet)): # Iterate over all test points
        # A list of the k closest points
        closestPoints = [(np.inf, 'Genre')] * k
        # The furthest of the closest points
        furthestPoint = (np.inf, 'Genre')

        # x is the test point to consider
        x = np.array(testSet.iloc[i,
        [testSet.columns.get_loc(c) for c in features]])
```

```
        # Iterate over all reference points
        for j in range(len(referenceSet)):
            # u is the refernce point to consider
            u = np.array(referenceSet.iloc[j,
                [referenceSet.columns.get_loc(c) for c in features]])
            # d is the distance/similarity between the test point and
            # reference point
            d = distance(x, u)

            # d is closer than the furthest of the k closest points
            if d < furthestPoint[0]:
                # replace the furthest of the k points with d
                closestPoints.remove(furthestPoint)
                closestPoints.append((d, referenceSet.iloc[j,
                    referenceSet.columns.get_loc("Genre")]))
                furthestPoint = max(closestPoints,
                    key = lambda t : t[0]) # update the furthest point

        # Sort the closest k points to correctly resolve tiebreaks
        closestPoints.sort(key=lambda x : x[0])

        # count is a dictionary {genre : int} that counts how many
        # times a genre is in the k closest points
        count = {}
        for z in range(len(closestPoints)):
            if closestPoints[z][1] in count.keys():
                count[closestPoints[z][1]] += 1
            else:
                count[closestPoints[z][1]] = 1

        # Classification is the genre with the k closest points
        classification = max(count, key=count.get)
        # The actual genre for the point
        genre = testSet.iat[i, testSet.columns.get_loc("Genre")]

        # Update the confusion matrix
        confusionMatrix.at[mapIndex[genre], classification] += 1


    # The number of correct classifications is on the diagonal
    correctClassifications = pd.Series(np.diag(confusionMatrix),
        index=[confusionMatrix.index, confusionMatrix.columns])
    num = correctClassifications.sum()
    errorRate = 1-num/confusionMatrix.sum().sum() # eer = 1 -
        correct/total
    end = time.time()
    print(end - start, 'Seconds')
    return errorRate, confusionMatrix
```

```
# Answer to task 1 by running following lines:
# errorRate, confusionMatrix = NNclassifier()
# print('The estimated error rate =', errorRate)
# print('Confusion matrix')
# print(confusionMatrix)

# Get the error rate and confusion matrix to part 4:
# errorRate, confusionMatrix = NNclassifier(features=ALLFEATURES,
    distance=relativeEuclidian)
# print('The estimated error rate =', errorRate)
# print('Confusion matrix')
# print(confusionMatrix)
```

File:"Part2Histogram.py" - Code for producing histograms

```
import matplotlib.pyplot as plt
from Part1 import ClassData30
# Part 2 - Seperability and its effect on the performance

# (a) For each of the four features compare the feature
    distribution for the four classes
    # - pop
    # - disco
    # - metal
    # - classical

# A list of the genres relevant to part2
genres = ['pop', 'disco', 'metal', 'classical']
# The features to consider in this part
features = ['spectral_rolloff_mean', 'mfcc_1_mean',
    'spectral_centroid_mean', 'tempo']

def plotHistogram(data, genres, features, grid=True):
    """The function produces a grid of histograms for corresponding
        genres and features

        Produces a nxm grid where n is the number of genres and m is
            the number of features.

        Parameters
        ----------
        data : pd.Dataframe
            A dataframe of all the relevant data
        genres : [str]
            A list of genres that is to be considered. Strings need
                to match with the keys of data
        features : [str]
            A list of the features that is to be considered. Strings
                need to match with the keys of data
        grid : Boolean, optional
            True sets a grid on each plot
    """
    # Declare a subplot grid
    fig, ax = plt.subplots(len(genres), len(features))

    for g in range(len(genres)):
        for f in range(len(features)):
            genre = genres[g]
            # dg is all the relevant data for the genre
            dg = data.query("Genre == @genre")
            ax[g][f].hist(dg[features[f]])
            if grid:
                ax[g][f].grid()
            if g == len(genres)-1:
                ax[g][f].set_xlabel(features[f])
            if f == 0:
                ax[g][f].set_ylabel(genres[g])

    fig.align_labels()
    plt.show()
    return fig

# Run the following line to produce the relevant histograms
# For Task 2a
# fig = plotHistogram(data=ClassData30, genres=genres,
    features=features)
```

File:"Part3.py" - Finding the best feature for part 3

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from Part1 import NNclassifier, GENREMAP, ClassData30, ALLFEATURES

# FEATURES is a list of the relevant features for part 1-3
FEATURES = ['spectral_rolloff_mean', 'mfcc_1_mean',
    'spectral_centroid_mean', 'tempo']
# mapGenre is a map of genre to index to row in the resulting
# confusion matrix
mapGenre = {"pop": 0, "disco": 1, "metal" : 2, "classical" : 3}

def tryAll(features=ALLFEATURES, base=FEATURES, data=ClassData30,
    Gmap=GENREMAP):
    """tryAll tries the k-NN classifier for all possible
        combinations of features where
    the base list of features base replace one of its element with
        an element from the features list
    Parameters
    ----------
    features : list(str), optional
        A list of the features that is to replace one of the
            elements in base
    base : list(str), optional
        A list of the base features.
    data : pd.Dataframe
        A pandas datafram that contains the relevant data
    Gmap : list(tuple(str, int)), optional
        List of the tuple (Genre, index). Genre is a string of a
            genre in data.
        index is an integer that corresponds to the row number of
            Genre in the
        generated confusion matrix

    """
    error = {}
    for i in range(len(base)):
        for j in range(len(features)):
            f = base.copy()
            f[i] = features[j]
            errorRate, confusionMatrix = NNclassifier(data=data,
                features=f, mapIndex=Gmap)
            error[(base[i], features[j])] = (errorRate,
                confusionMatrix)
    return error

# Run following code to find the feature pair
# (original, replacement) with the best accuracy
# for the k-NN classifier from part 1 where
# original is replaced with replacement in the
# list of features considered. This takes
# quite some time..

# all = tryAll(Gmap=mapGenre)
# best = min(all.values(), key=lambda t:t[0])
# bestPair = list(all.keys())[list(all.values()).index(best)]
# print(bestPair)
# print(best[0], '\n', best[1])
# print('end')
```

References

[1] Music genre classification. `https://learn-eu-central-1-prod-fleet01-xythos.content.blackboardcdn.com/5def77a38a2f7/15700466?X-Blackboard-S3-Bucket=learn-eu-central-1-prod-fleet01-xythos&X-Blackboard-Expiration=1682996400000&X-Blackboard-Signature=5mkp34WR`,, 03 2022.

[2] Geoffroy Peeters. A large set of audio features for sound description (similarity and classification) in the cuidado project. *CUIDADO Ist Project Report*, 54(0):1–25, 2004.

[3] Anssi Klapuri and Manuel Davy. *Signal Processing Methods for Music Transcription*. 01 2006.

[4] Simone Dalla Bella, Isabelle Peretz, Luc Rousseau, and Nathalie Gosselin. A developmental study of the affective value of tempo and mode in music. *Cognition*, 80(3):B1–B10, 2001.