AMATH 582 Homework 4

Yiyun Dong

Department of Physics — University of Washington
Email: yiyund@uw.edu — Github: yeewantung

Abstract

This project consists of two parts.

In the first part, we analyze two sets of face data using SVD. The first group of faces are cropped and motion of the subjects are removed, leaving only the change in light intensity. The second group of faces are not cropped or optimized with removing the subject's motion. We get eigenfaces through SVD and then use a few eigenfaces and weight information from right singular vectors to reconstruct a specific face.

In the second part, we use machine learning algorithms to classify music of different genres. We perform classification on three different tasks, specifically, classification on different composers from different genre, classification on different composers from same genre and classification on different genres directly.

1 Introduction and Overview

From the previous project, we learned the use of Singular Value Decomposition (SVD) in Principle Component Analysis (PCA). Through SVD, the principle modes of the object motion can be extracted without a hitch.

Apart from its application in PCA, SVD has many applications in mathematics and fields of data analysis and machine learning. One example is that image compression can be realized by the low rank approximation of SVD without a significant decrease in image quality. Another example is that we can make use of the characteristic of SVD in extracting features of a set of data. This makes features and weights of each data piece clear and can be served as a pre-processing of data for future analysis by machine learning algorithms.

In this project, we explore the power SVD and classification algorithms on two parts.

In the first part, we analyze two sets of face data using SVD. The first group of faces are cropped and motion of the subjects are removed, leaving only the change in light intensity. The second group of faces are not cropped or optimized with removing the subject's motion. We get eigenfaces through SVD and then use a few eigenfaces and weight information from right singular vectors to conduct reconstruction on specific faces.

In the second part, we use machine learning algorithms to classify music of different genres. We perform classification on three different tasks, specifically, classification on different composers from different genre, classification on different composers from same genre and classification on different genres directly. We will discuss the effect of hyper-parameters on the results of classification. The examples of hyper-parameters are the number of features used and the number of neighbors in K-nearest neighbors algorithm.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

As we learned from the lecture, every matrix $A \in \mathbb{C}^{m\times n}$ has a singular value decomposition (SVD),

$$A = U\Sigma V^*, \tag{1}$$

where U and V are two unitary matrix, and $Sigma$ is a diagonal matrix. The diagonal elements of Σ is called the singular values of matrix A. The column vectors of U and V are called left and right singular vectors respectively [5, 18]. Therefore, we can also write matrix A in a sum of rank-1 matrices,

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^*, \tag{2}$$

here r is the rank of matrix A.

In fact, SVD provides best approximation of a matrix A by matrices of lower rank. For $0 \leq \nu \leq r$, define

$$A_\nu = \sum_{j=1}^{\nu} \sigma_j u_j v_j^*, \tag{3}$$

then,

$$\|A - A_\nu\|_2 = \inf_{\substack{B \in \mathbb{C}^{m \times n} \\ rank(B) \leq \nu}} \|A - B\|_2 = \sigma_{\nu+1}. \tag{4}$$

Furthermore, the singular values σ_i are uniquely determined. If A is a square matrix and the singular values σ_i are distinct, the left and right singular vectors u_j and v_j are uniquely determined up to complex signs. This explains the existence and uniqueness of SVD for an arbitrary matrix [5].

2.2 Classification Algorithms

2.2.1 K-Nearest Neighbors (KNN)

The k-nearest neighbors algorithm (KNN) is a non-parametric method used for classification and regression in pattern recognition [15]. It is conducted while we have knowledge of the labels of data. When using KNN to predict the class of a data point, we compute the distance between the given data point and all the labelled points in training set. Then we pick k nearest points to the given data point, and predict the class of the given data point to be the most frequent class among k nearest points [3].

In this algorithm. the number k of nearest neighbors picked is a hyper-parameter we should take care about. An excessively small k may cause overfitting since the boundary will not be smooth, while an excessively large k may cause underfitting. Both overfitting and underfitting will lead to a decrease in classification accuracy.

2.2.2 Gaussian Naive Bayes

As we learned from lecture, the Bayes' theorem describe a relation in probability

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{5}$$

It provides a way that we can calculate the probability of a piece of data belonging to a given class $P(class|data)$, given our prior knowledge [8].

Naive Bayes classifiers are a group of probabilistic classifiers based on applying Bayes' theorem with strong independence assumptions between the features [17].

2.2.3 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a method used in statistics, pattern recognition, and machine learning to find a linear combination of features that characterizes two or more classes of objects [16]. It is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. The idea is to project a dataset onto a lower-dimensional space with good separability of class [6]. In this way, it can avoid overfitting and reduce computational costs at the same time.

2.2.4 Support-vector machine (SVM)

Support-vector machines (SVM) are supervised learning models to analyze data by performing classification and regression analysis [19]. The idea of SVM is as follows. We can understand classification generally as a process that finds a division line passing between the clusters in hyperspace [13]. SVM is an algorithm that optimize the location of the division line by positing the division line as far away as possible from the clusters while getting the training data separation right [13]. In this way, it can guarantee a higher chance of correctly labeling the test data.

3 Algorithm Implementation and Development

3.1 Eigenface analysis and face reconstruction

1. Batch import cropped (or uncropped) images of faces.
2. Reshape each image of face into a column vector to form a matrix of images.
3. Subtract the mean of each column.
4. Compute SVD of the image matrix.
5. Reshape and plot first several eigenfaces.
6. As for face reconstruction, compute low rank approximation of a face.
7. Reshape and plot the reconstructed face with different ranks. Compare reconstructed faces with original face.

Algorithm 1: Eigenface analysis

Initialize a Numpy matrix to store images
for Image j in images **do**
  Import image j in grayscale as a Numpy matrix j
  Reshape matrix j to a column vector j
  Subtract the mean from column j
  Append column vector j to the image matrix
end for
Compute SVD of the image matrix
Reshape and plot first several eigenfaces

Algorithm 2: Face Reconstruction

Compute SVD of the image matrix, $A = U\Sigma V^*$
Compute rank r approximation to face n: $a_{i,r} = \sum_{i=1}^{r} \sigma_{ii} u_i v^*_{i,n_{start}:n_{end}}$
Reshape and plot the reconstructed face with different ranks

3.2 Music classification

1. Batch import music files of different bands or genres.
2. Crop each song to emit the beginning and end where there is no sound.
3. Extract multiple 5-sec-sound-clips from each song to form a music matrix. Each column of the music matrix represents a 5-sec-sound-clip.

4. Compute spectrogram of each 5-sec-sound-clip. Reshape the spectrogram into a column vector to form a spectrogram matrix
5. Subtract the mean of each column and compute SVD of the spectrogram matrix.
6. Use rows of the right singular matrix as dataset. Select a few rows as features of sound clips.
7. Do train-test-split. Choose a classifier and optimize hyperparameter by cross validation.
8. Fit the train set and obtain the classification accuracy by cross validation and predicting test set.

Algorithm 3: SVD of music clips

```
Initialize a Numpy matrix to store spectrograms of 5-sec-music-clips
for Music j in music files do
    Import music j as a Numpy array j
    Crop and pre-process music j
    Extract multiple 5-sec-sound-clips from music j
    Compute spectrogram of each 5-sec-sound-clip extracted
    Reshape each spectrogram to a column vector and subtract the mean
    Append computed spectrogram vectors to the spectrogram matrix
    Save the band or genre of music j in a Numpy array
end for
Compute SVD of the spectrogram matrix
```

Algorithm 4: Music Classification

```
Use rows of the right singular matrix as dataset
Select range R of rows as features of sound clips
Do train-test-split
Choose a classifier and optimize hyperparameter (e.g., range R) by cross validation
Fit the train set
Obtain the classification accuracy of cross validation and test set
```

4 Computational Results

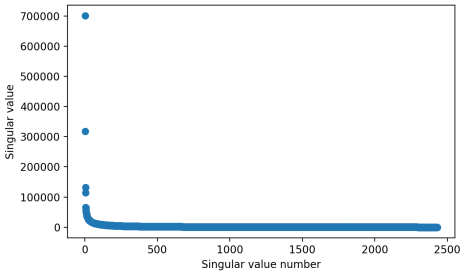
4.1 Eigenface analysis and face reconstruction

4.1.1 Cropped images

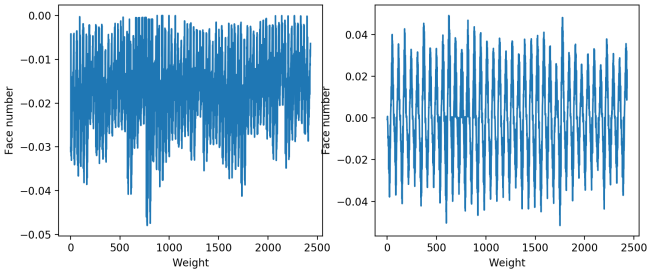
In this part, we compute the SVD of the cropped images in *Yale Faces B* dataset, as shown in Figure 1. It can be seen that each eigenface contains a characteristic of human face, such as eyes, noses, light contrast and shadow, *etc.*. The singular values shown in Figure 1a illustrates how important each eigenface (characteristic) is in describing a human face.

Recall that each right singular vector represents weights of all faces projected into the corresponding eigenface. It can also be seen that there is oscillation appearing in the second right singular vector. Linking this piece of information to the second eigenface, we can see that the oscillation is due to the light contrast and shadow appeared on the human face.

We then use eigenfaces and weight information from right singular vectors to conduct reconstruction on specific faces. This is known as low rank approximation by SVD. A list of ranks $\{10, 20, 50, 100, 200\}$ are tested in low rank approximation of face reconstruction, as shown in Figure 2. It can be seen that the human face can be recognized when rank= 100 and the human face is enough detailed when rank= 200. In conclusion, rank= 200 is a good choice of rank for low rank approximation of face reconstruction, so we will apply this conclusion in face reconstruction of uncropped images.



(a) Singular values of cropped images



(b) First two right singular vectors, i.e., weights of all faces in first two eigenfaces

(c) First three eigenfaces

Figure 1: SVD of cropped images

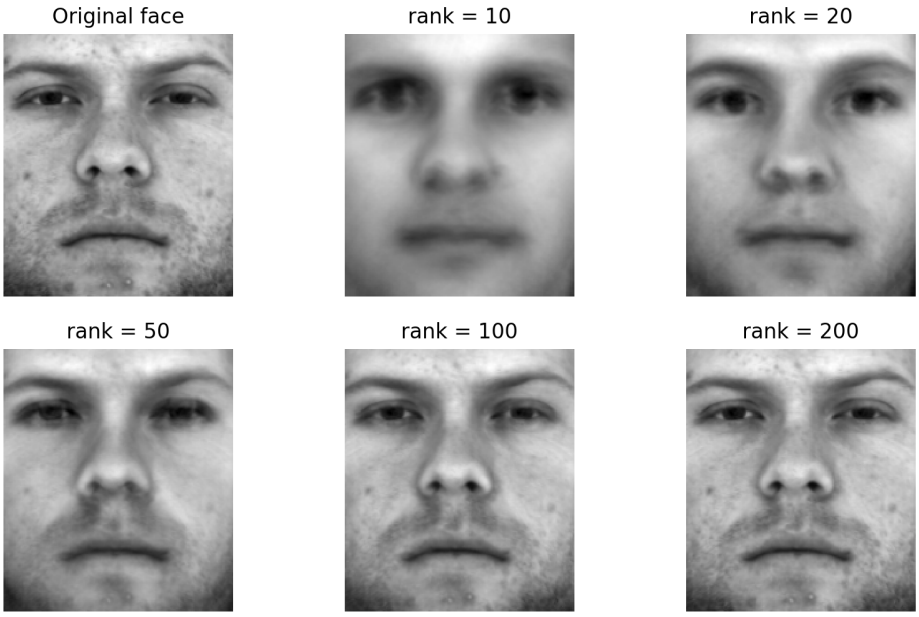


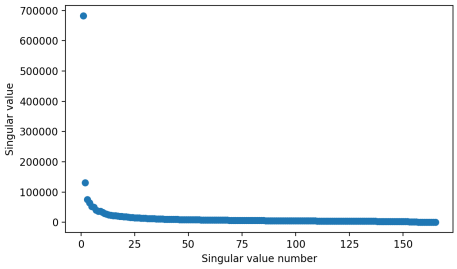
Figure 2: Original and reconstructed faces of cropped images

4.1.2 Uncropped images

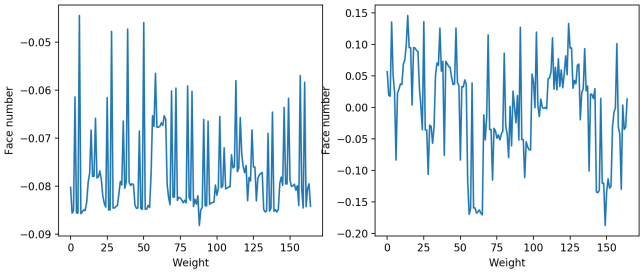
In this part, we compute the SVD of the uncropped images in *Yale Faces B* dataset, as shown in Figure 3. The singular values shown in Figure 3a illustrates how important each eigenface (characteristic) is in describing a human face. In addition to our discovery in the previous part, the eigenfaces of uncropped images contains human motion such as shaking. It can also be seen that there is non-uniform oscillation appearing in the right singular vectors. Since the human motion is not as uniform as light intensity change, we can say that the oscillation is due to the human motion.

Although there is human motion, the face reconstruction process is not significantly affected. We use eigenfaces and weight information from right singular vectors to conduct reconstruction not only on one expression, but also averaging over expressions to see how good the approximation can be, as shown in Figure 4. Rank is chosen to be 200 for low rank approximation.

It can be seen that the human expression can be reproduced when rank= 200. In addition, we can get rid of defects of camera by averaging over expressions.



(a) Singular values of uncropped images

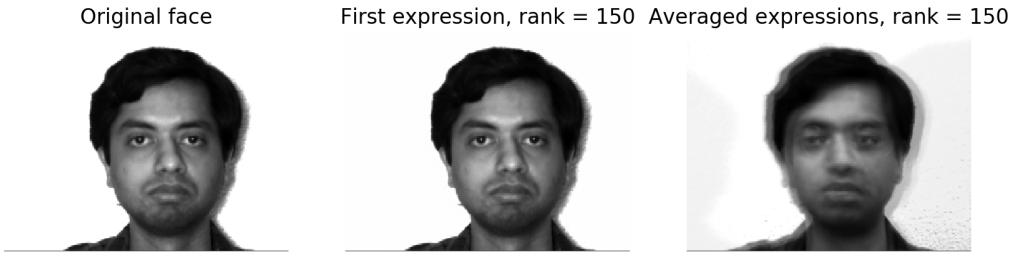


(b) First two right singular vectors, i.e., weights of all faces in first two eigenfaces

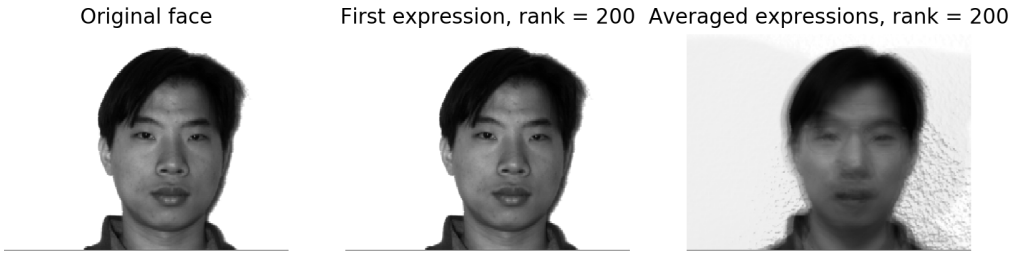
(c) First three eigenfaces

Figure 3: SVD of uncropped images

(a) Reconstructed face 1



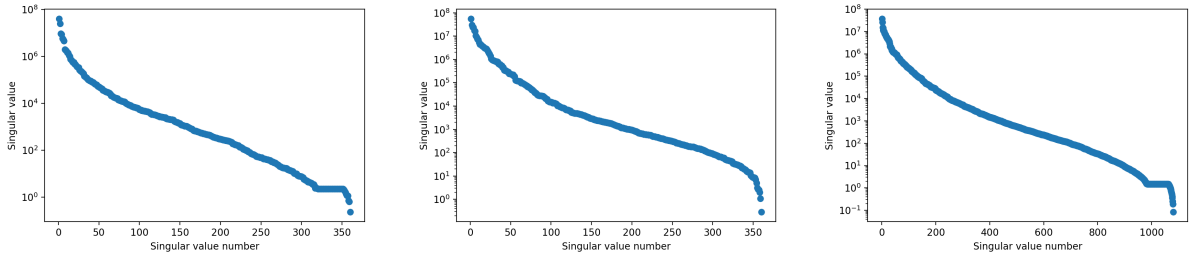
(b) Reconstructed face 2



(c) Reconstructed face 4

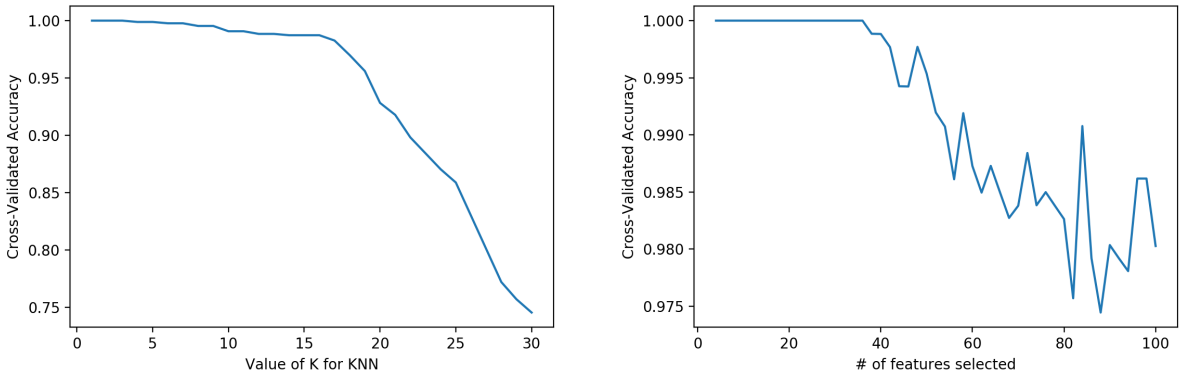
Figure 4: Original and reconstructed faces of uncropped images

4.2 Music classification

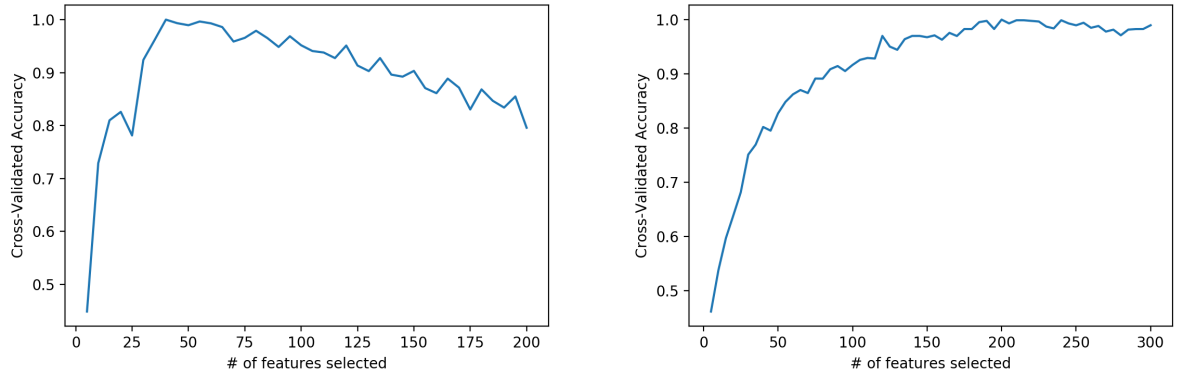


(a) Test 1 (b) Test 2 (c) Test 3

Figure 5: Singular values of 5-sec-music-clips in three tests



(a) Varying number of nearest neighbors in KNN, Test 3 (b) Varying number of features selected in KNN, Test 3



(c) Varying number of features selected in SVM, Test 2 (d) Varying number of features selected in SVM, Test 3

Figure 6: Examples of testing on hyperparameters by cross-validation

In this part, we use machine learning algorithms to classify music of different genres. We perform classification on three different tests. Test 1 is classification on different composers from different genre, more specifically, songs of Beethoven, Michael Jackson and Soundgarden. Test 2 is classification on different composers from same genre, more specifically, songs of Soundgarden, Alice in Chains and Pearl Jam. Test 3 is classification on different genres, more specifically, rock, classic and jazz.

We first compute the SVD of three tests. The singular values of three tests are shown in Figure 5. It

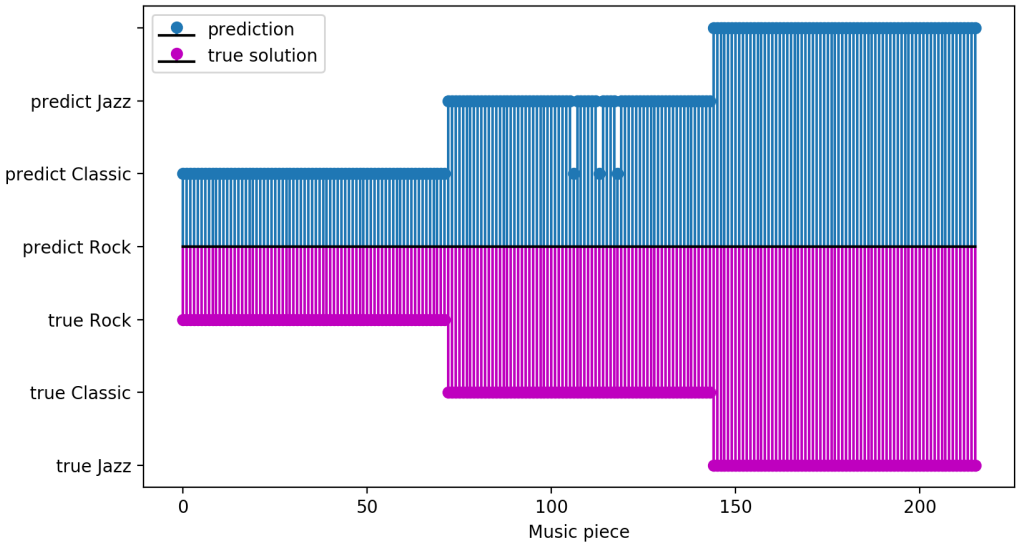


Figure 7: Prediction versus true class of 5-sec-music-clips in Test 3 by Naive Bayes

Classifier	KNN	Naive Bayes	LDA	SVM
Test 1	1.0	1.0	1.0	0.927
Test 2	1.0	1.0	1.0	1.0
Test 3	1.0	0.995	0.996	0.957

Table 1: Classification score of classifiers by cross-validation

Classifier	KNN	Naive Bayes	LDA	SVM
Test 1	1.0	1.0	1.0	0.888
Test 2	1.0	1.0	1.0	1.0
Test 3	1.0	0.986	1.0	0.995

Table 2: Classification score of classifiers by predicting test set

can be seen that first 50 singular values capture most of the variances in the data. Since each row of right singular matrix represents weights of eigenmodes in a music clip, we select a range of weights in each row, especially first several weights, as features to describe and classify each music clip.

Four kinds of classifiers are implemented in classify the music clips. They are KNN, Naive Bayes, LDA and SVM as discussed in Section 2. The classification accuracy of classifiers for cross-validation and test set are shown in Table 1 and 2. It can be seen that good classification for four classifiers can be obtained by selecting suitable hyper-parameters.

Here the hyper-parameters are the number of features used and the number of neighbors in K-nearest neighbors algorithm. Figure 6 illustrates the effect of hyper-parameters on the classification accuracy. Here we use KNN model for Test 3, SVM model for Test 2 and 3 as example.

As shown in Figure 6a, the accuracy of KNN model decreases as number of neighbors increases due to underfitting. It suggests we should choose a small value of k as number of neighbors for KNN model. As shown in Figure 6b, the accuracy of KNN decreases as more number of features are selected, due to overfitting. It suggests we should choose a small range of features for KNN model to avoid overfitting.

As shown in Figure 6c, the accuracy of SVM model first increases then decreases as number of neighbors increases. To avoid overfitting and underfitting, we choose 40 as a moderate number of features selected for SVM model in Test 2. As for Test 3, since we are classifying among genres, there are more possible useful features extracted by SVD, as shown in Figure 6d. Thus, we choose 250 as a moderate number of features selected for SVM model in Test 3.

An example of Prediction result is shown in Figure 7, so that we can see which music pieces are incorrectly classified. It seems that there is a little confusion between rock and classic for Naive Bayes model in genre classification.

In addition, a random piece of 5-sec-music-clip of Grunge music is generated to test out the ability of classifiers. This music clip is not appeared in either test set or train set. Since Grunge music is closest to rock, we expect the prediction to be rock by classifiers. However, only Naive Bayes and LDA yield correct prediction. I would say that there is still overfitting appeared in my KNN and SVM model.

5 Summary and Conclusions

In this project, we explore the power of SVD in eigenface analysis and feature extraction for music classification. We render that SVD is useful in extracting the features of a dataset, where only main features can be kept for data compression. Besides, we conduct music classification with four classifiers and test out different hyper-parameters to see the effect on the classification accuracy. I realize that choosing hyper-parameters by cross validation is extremely important to avoid overfitting and underfitting. If possible, it would be better if we do a grid search of hyper-parameters with cross validation. In this way, doing a high-dimensional plot of accuracy with respect to hyper-parameters woule make it clear which hyper-parameter to choose.

References

[1] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.1.55. 2018. URL: http://github.com/google/jax.

[2] *Imageio Reference*. URL: https://imageio.github.io/.

[3] *Introduction to k-Nearest Neighbors: A powerful Machine Learning Algorithm*. URL: https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/.

[4] *Jax tutorial*. URL: https://towardsdatascience.com/turbocharging-svd-with-jax-749ae12f93af.

[5] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[6] *Linear Discriminant Analysis - Bit by Bit*. URL: https://sebastianraschka.com/Articles/2014_python_lda.html.

[7] *Matplotlib Reference*. URL: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html.

[8] *Naive Bayes Classifier From Scratch in Python*. URL: https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/.

[9] *NumPy Reference*. URL: https://docs.scipy.org/doc/numpy/reference.

[10] *Pydub Reference*. URL: http://pydub.com/.

[11] *Scikit-learn Reference*. URL: https://scikit-learn.org/stable/index.html.

[12] *Scipy Signal Reference*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.spectrogram.html.

[13] *Support Vector Machine (SVM) Tutorial*. URL: https://blog.statsbot.co/support-vector-machines-tutorial-c1618e635e93.

[14] *Wavfile I/O Reference*. URL: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.io.wavfile.read.html.

[15] *Wikipedia - K-nearest neighbors algorithm*. URL: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[16] *Wikipedia - Linear discriminant analysis*. URL: https://en.wikipedia.org/wiki/Linear_discriminant_analysis.

[17] *Wikipedia - Naive Bayes classifier*. URL: https://en.wikipedia.org/wiki/Naive_Bayes_classifier.

[18] *Wikipedia - Singular value decomposition*. URL: https://en.wikipedia.org/wiki/Singular_value_decomposition.

[19] *Wikipedia - Support-vector machine*. URL: https://en.wikipedia.org/wiki/Support-vector_machine.

Appendix A Python Functions used

Below are the Python functions implemented in the code with a brief explanation.

- `imageio.imread(uri, format=None, **kwargs)` reads an image from the specified file and returns a numpy array [2].
- `pydub.AudioSegment.from_mp3(file)` creates a multi-channel audio segment out of multiple mp3 file [10].
- `audio.set_frame_rate(fs)` creates an equivalent version of this Audio Segment with the specified frame rate (in Hz) [10].
- `scipy.io.wavfile.read(filename, mmap=False)` returns the sample rate (in samples/sec) and data from a WAV file [14].
- `scipy.signal.spectrogram(x, fs=1.0, window=('tukey', 0.25))` computes a spectrogram with consecutive Fourier transforms [12].
- `numpy.zeros(shape, dtype=float, order='C')` returns a new array of given shape and type, filled with zeros [9].
- `numpy.copy(a, order='K')` returns an array copy of the given object [9]. When assigning the values of a matrix A to another matrix B and changing the matrix elements of B, this function is used to avoid the change in the same elements of A.
- `numpy.concatenate((a1, a2, ...), axis=0, out=None)` joins a sequence of arrays along an existing axis [9].
- `numpy.linalg.svd(a, full_matrices=True, compute_uv=True, hermitian=False)` computes Singular Value Decomposition [9].
- `jax.scipy.linalg.svd(a, full_matrices=True, compute_uv=True, hermitian=False)` computes Singular Value Decomposition in a faster way [1, 4].
- `matplotlib.pyplot.imshow(X, cmap=None)` displays an image, i.e. data on a 2D regular raster [7].
- `matplotlib.pyplot.scatter(x, y)` displays a scatter plot of y vs x with varying marker size and/or color [7].
- `matplotlib.pyplot.stem(*args, linefmt=None, markerfmt=None, basefmt=None, label=None, use_line_collection=False)` creates a stem plot that plots vertical lines at each x location from the baseline to y, and places a marker there [7].
- `sklearn.model_selection.train_test_split(*arrays, **options)` splits arrays or matrices into random train and test subsets [11].
- `sklearn.model_selection.KFold(n_splits=5, shuffle=False, random_state=None)` provides train/test indices to split data in train/test sets [11].
- `sklearn.model_selection.cross_val_score(estimator, X, y=None, groups=None, scoring=None, cv=None)` evaluates a score by cross-validation [11].
- `sklearn.naive_bayes.GaussianNB(priors=None, var_smoothing=1e-09)` performs Gaussian Naive Bayes classification [11].
- `sklearn.discriminant_analysis.LinearDiscriminantAnalysis(solver='svd', shrinkage=None, priors=None, n_components=None, store_covariance=False, tol=0.0001)` performs Linear Discriminant Analysis classification [11].
- `sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None)` performs C-Support Vector Classification [11].

Appendix B Python Codes

```
#!/usr/bin/env python
# coding: utf-8

# In[4]:


# import packages
import numpy as np
import jax.scipy as jsp # boost computation
import matplotlib.pyplot as plt
import imageio
from os import listdir


# In[5]:


# interactive plotting
get_ipython().run_line_magic('matplotlib', 'notebook')
get_ipython().run_line_magic('matplotlib', 'notebook')


# # Yale Faces B

# ## Function Definition

# ### Batch import images

# In[186]:


def imread_batch(folder, mode = 'Stack', fname_spec = None):
    '''
    Batch import static images, 2 modes available.
    Input:
        folder: (String) directory name
        mode: 'Stack' or 'Avg'
            'Stack': Stack images of each face as a matrix
            'Avg': Take average of images of each face as a column vector
        fname_spec: (String) specify what is contained in filename
    Output:
        return a 2d numpy matrix with reshaped images as column vectors
    '''

    # sanity check on mode
    if (mode != 'Stack') and (mode != 'Avg'):
        print('Error: incorrect mode. Choose between "Stack" or "Avg". ')
        return

    # sanity check on folder name
    if folder.endswith('/') == False:
        folder = folder + '/'
```

```
    # fetch a list of filenames
    if fname_spec == None:
        # assume all files in folder are images
        fnames = sorted(listdir(folder))
    else:
        fnames = list(fname for fname in sorted(listdir(folder)) if fname_spec in fname)
        if len(fnames) == 0:
            print('Error: No file name contains ' + str(fname_spec))
            return

    # determine the dimensions of each image
    temp = imageio.imread(folder + fnames[0])
    idx = temp.shape
    img_num = len(fnames)

    # initialize
    img_temp = np.zeros((idx[0]*idx[1], img_num))
    for i in range(img_num):
        # assume the image is in grayscale
        img = imageio.imread(folder + fnames[i])
        img = img.reshape(idx[0]*idx[1])
        img_temp[:,i] = img

    # Average mode
    if mode == 'Avg':
        img_data = np.zeros((idx[0]*idx[1], 1))
        img_data[:,0] = img_temp.mean(axis = 1)

    # Stack mode
    elif mode == 'Stack':
        img_data = img_temp

    return img_data


# In[187]:


def imread_batch_multidir(folder, mode = 'Stack', fname_spec = None, dirname_spec = None):
    '''
    Batch import static images from multiple directories in folder, 2 modes available.
    Input:
        folder: (String) directory name
        mode: 'Stack' or 'Avg'
            'Stack': Stack images of each face as a matrix
            'Avg': Take average of images of each face as a column vector
        fname_spec: (String) specify what is contained in filename
        dirname_spec: (String) specify what is contained in directory name
    Output:
        return a 2d numpy matrix with reshaped images as column vectors
    '''

    # sanity check on mode
```

```
    if (mode != 'Stack') and (mode != 'Avg'):
        print('Error: incorrect mode. Choose between "Stack" or "Avg". ')
        return

    # sanity check on folder name
    if folder.endswith('/') == False:
        folder = folder + '/'

    # fetch a list of directory names
    if dirname_spec == None:
        # assume all items in folder are directories
        dirnames = sorted(listdir(folder))
    else:
        dirnames = list(dirname for dirname in sorted(listdir(folder)) if dirname_spec in dirname)
        if len(dirnames) == 0:
            print('Error: No directory name contains ' + str(dirname_spec))
            return

    # avoid ambiguity when passing arguments
    str1, str2 = mode, fname_spec

    # Batch import images in each folder
    img_data = imread_batch(folder + dirnames[0], mode = str1, fname_spec = str2)
    for i in range(1, len(dirnames)):
        data = imread_batch(folder + dirnames[i], mode = str1, fname_spec = str2)
        img_data = np.concatenate((img_data, data), axis = 1)

    return img_data


# ### Face reconstruction

# In[255]:


def face_reconstruct(u, s, vh, face_no, faces_num = 64, rank = 10, reshape = False, dims = [192, 168], r
    '''
    Reconstruct a specified face by low rank approximation out of SVD
    Input:
        u, s, vh: svd of original faces
        face_no: specify which face to reconstruct
        faces_num: number of images based on the specified face
        rank: number of modes used to reconstruct
        reshape: whether to reshape the vector to a matrix for image
        dims: (list) [length pixels, width pixels]
        mode: 'Avg' or 'Single'
    Output:
        return a vector/reshaped matrix of face
    '''

    ur = u[:,0:rank]
    sr = np.diag(s[0:rank])
    vhr = vh[0:rank, faces_num*(face_no-1):faces_num*face_no]
    faces_approx = np.linalg.multi_dot([ur, sr, vhr])
```

```
    if mode == 'Single':
        face_approx = faces_approx[:,0]
    else: # average
        face_approx = faces_approx.mean(axis = 1)

    if reshape == True:
        face_approx = face_approx.reshape(dims[0], dims[1])

    return face_approx


# In[223]:


def face_reconstruct_plot(u, s, vh, face_no, dir_origin, faces_amount = 64, rank_list = [10, 20, 50, 100
    '''
    Plot original face & reconstructed faces by low rank approximation
    Input:
        u, s, vh: svd of original faces
        face_no: specify which face to reconstruct
        dir_origin: (String) location of original face
        faces_amount: amount of images for each face
        rank_list: (list) list of rank used
        fname_spec: (String) specify what is contained in filename
        dims: (list) [length pixels, width pixels]
    Output:
        plot images
        return nothing
    '''

    # fetch the data of original face
    if fname_spec:
        str1 = fname_spec # avoid ambiguous
        face_origin = imread_batch(dir_origin, mode = 'Avg', fname_spec = str1)
    else:
        face_origin = imread_batch(dir_origin, mode = 'Avg')
    face_origin = face_origin.reshape(dims[0], dims[1])

    # plot original
    plt.figure(figsize = (10,6))
    plt.subplot(2,3,1)
    plt.imshow(face_origin, cmap = plt.get_cmap("gray"))
    plt.title('Original face')
    plt.axis('off')

    for i in range(len(rank_list)):

        # compute reconstructed face
        face_approx = face_reconstruct(u, s, vh, face_no, faces_num = faces_amount, rank = rank_list[i]
        # plot reconstructed face
        plt.subplot(2,3,i+2)
        plt.imshow(face_approx, cmap = plt.get_cmap("gray"))
        plt.title('rank = %d' % rank_list[i])
        plt.axis('off')
```

```
    plt.show()

    return


# In[271]:


def face_reconstruct_plot_s(u, s, vh, face_no, dir_origin, faces_amount = 64, rank = 200, fname_spec = N
    '''
    Plot original face & reconstructed faces by low rank approximation (simple, only 1 rank)
    Input:
        u, s, vh: svd of original faces
        face_no: specify which face to reconstruct
        dir_origin: (String) location of original face
        faces_amount: amount of images for each face
        fname_spec: (String) specify what is contained in filename
        dims: (list) [length pixels, width pixels]
    Output:
        plot images
        return nothing
    '''

    # fetch the data of original face
    if fname_spec:
        str1 = fname_spec # avoid ambiguous
        face_origin = imread_batch(dir_origin, mode = 'Stack', fname_spec = str1)
        face_origin = face_origin[:,0]
    else:
        face_origin = imread_batch(dir_origin, mode = 'Avg')
    face_origin = face_origin.reshape(dims[0], dims[1])

    # compute reconstructed face for first expression
    face_approx_s = face_reconstruct(u, s, vh, face_no, faces_num = faces_amount, rank = rank, reshape =
    # compute reconstructed face averaged over expressions
    face_approx_a = face_reconstruct(u, s, vh, face_no, faces_num = faces_amount, rank = rank, reshape =

    # plot original
    plt.figure(figsize = (10,3))
    plt.subplot(1,3,1)
    plt.imshow(face_origin, cmap = plt.get_cmap("gray"))
    plt.title('Original face')
    plt.axis('off')

    # plot reconstructed face
    plt.subplot(1,3,2)
    plt.imshow(face_approx_s, cmap = plt.get_cmap("gray"))
    plt.title('First expression, rank = %d' % rank)
    plt.axis('off')

    plt.subplot(1,3,3)
    plt.imshow(face_approx_a, cmap = plt.get_cmap("gray"))
    plt.title('Averaged expressions, rank = %d' % rank)
```

```
    plt.axis('off')

    plt.show()

    return


# ## Cropped images
#
# dataset description: 38 faces * 64 images (192 pixels * 168 pixels)

# ### SVD

# In[15]:


img_crop = imread_batch_multidir('CroppedYale/', mode = 'Stack', dirname_spec = 'yale')


# In[61]:


get_ipython().run_cell_magic('time', '', 'u_crop_fast, s_crop_fast, vh_crop_fast = jsp.linalg.svd(img_c


# In[62]:


get_ipython().run_cell_magic('time', '', 'u_crop = np.asarray(u_crop_fast)\ns_crop = np.asarray(s_crop_


# In[34]:


# plot singular values on eigenfaces
plt.figure(1, figsize = (10,4))
plt.subplot(1,2,1)
plt.scatter(np.arange(1,len(s_crop)+1), s_crop)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.subplot(1,2,2)
plt.semilogy(s_crop)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.show()


# In[51]:


# plot eigenfaces
plt.figure(2, figsize = (10,3))
plt.subplot(1,3,1)
plt.imshow((u_crop[:,0]).reshape(192, 168), cmap = plt.get_cmap("gray"))
```

```
plt.subplot(1,3,2)
plt.imshow((u_crop[:,1]).reshape(192, 168), cmap = plt.get_cmap("gray"))
plt.subplot(1,3,3)
plt.imshow((u_crop[:,2]).reshape(192, 168), cmap = plt.get_cmap("gray"))
plt.show()


# In[35]:


# plot projection of faces onto eigenfaces
plt.figure(3, figsize = (10,3))
plt.subplot(1,3,1)
plt.plot(vh_crop[0])
plt.subplot(1,3,2)
plt.plot(vh_crop[1])
plt.subplot(1,3,3)
plt.plot(vh_crop[2])
plt.show()


# ### Reconstruct faces
#
# SVD:
# $$
# \begin{aligned}
# A &= U\Sigma V \\
# A &= \sum_{i} \sigma_{ii} u_i v^*_i \\
# a_{ij} &= u_{ik}\sigma_{kk}v^*_{kj}.
# \end{aligned}
# $$
#
# A column vector represents an image of face. The $j$th column of $A$ satisfies
# $$
# a_j = U\Sigma (V^*)_j.
# $$
# Here $(V^*)_j$ is the $j$th column of $V^*$, the Hermitian of $V$.
#
# Full rank reconstruction to images regarding the $n$th face is
# $$
# \begin{aligned}
# A_{(64n-63):64n} &= U\Sigma (V^*)_{(64n-63):64n} \\
# A_{(64n-63):64n} &= \sum_{i} \sigma_{ii} u_i v^*_{i,(64n-63):64n}.
# \end{aligned}
# $$
#
# The low rank approximation to images regarding the $n$th face is
# $$
# A_{r,(64n-63):64n} = \sum_{i=1}^r \sigma_{ii} u_i v^*_{i,(64n-63):64n}
# $$
# with rank $r$.
#
# I want to merge those images into one single image for the $n$th face in order to display. Therefore,
```

```
# In[272]:


# plot reconstructed faces versus the original
face_reconstruct_plot(u_crop, s_crop, vh_crop, 1, 'CroppedYale/yaleB01', rank_list = [10, 20, 50, 100, 1


# ## Uncropped images
#
# dataset description: 15 faces * 11 expressions (243 pixels * 320 pixels)

# ### SVD

# In[190]:


img_uncrop = imread_batch('yalefaces_uncropped/yalefaces/')


# In[191]:


get_ipython().run_cell_magic('time', '', 'u_uncrop_fast, s_uncrop_fast, vh_uncrop_fast = jsp.linalg.svd


# In[192]:


get_ipython().run_cell_magic('time', '', 'u_uncrop = np.asarray(u_uncrop_fast)\ns_uncrop = np.asarray(s


# In[197]:


# plot singular values on eigenfaces
plt.figure(11, figsize = (10,4))
plt.subplot(1,2,1)
plt.scatter(np.arange(1,len(s_uncrop)+1), s_uncrop)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.subplot(1,2,2)
plt.semilogy(s_uncrop)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.show()


# In[200]:


# plot eigenfaces
plt.figure(12, figsize = (10,3))
plt.subplot(1,3,1)
plt.imshow((u_uncrop[:,0]).reshape(243, 320), cmap = plt.get_cmap("gray"))
```

```
plt.subplot(1,3,2)
plt.imshow((u_uncrop[:,1]).reshape(243, 320), cmap = plt.get_cmap("gray"))
plt.subplot(1,3,3)
plt.imshow((u_uncrop[:,2]).reshape(243, 320), cmap = plt.get_cmap("gray"))
plt.show()


# In[201]:


# plot projection of faces onto eigenfaces
plt.figure(13, figsize = (10,3))
plt.subplot(1,3,1)
plt.plot(vh_uncrop[0])
plt.subplot(1,3,2)
plt.plot(vh_uncrop[1])
plt.subplot(1,3,3)
plt.plot(vh_uncrop[2])
plt.show()


# ### Reconstruct faces
#
# For this case, full rank reconstruction to images regarding the £n£th face is
# ££
# \begin{aligned}
# A_{(11n-10):11n} &= U\Sigma (V^*)_{(11n-10):11n} \\
# A_{(11n-10):11n} &= \sum_{i} \sigma_{ii} u_i v^*_{i,(11n-10):11n}.
# \end{aligned}
# ££
#
# The low rank approximation to images regarding the £n£th face is
# ££
# A_{r,(11n-10):11n} = \sum_{i=1}^r \sigma_{ii} u_i v^*_{i,(11n-10):11n}
# ££
# with rank £r£.
#
# I want to merge those images into one single image for the £n£th face in order to display. Therefore,

# In[273]:


# plot reconstructed faces versus the original
face_reconstruct_plot_s(u_uncrop, s_uncrop, vh_uncrop, 1, 'yalefaces_uncropped/yalefaces/', faces_amoun


# In[265]:


# plot reconstructed faces versus the original
face_reconstruct_plot_s(u_uncrop, s_uncrop, vh_uncrop, 10, 'yalefaces_uncropped/yalefaces/', faces_amou


# In[274]:
```

```
# plot reconstructed faces versus the original
face_reconstruct_plot_s(u_uncrop, s_uncrop, vh_uncrop, 4, 'yalefaces_uncropped/yalefaces/', faces_amoun


# # Music Classification

# In[207]:


# import packages
from pydub import AudioSegment
from scipy.io import wavfile
from scipy import signal


# ## Batch import & process music piece

# In[194]:


def mp3read(fname, resample_rate = None):
    '''
    Import mp3 music piece
    Input:
        fname: (String) file name
    Output:
        return a numpy array of music
    '''

    song = AudioSegment.from_mp3(fname)
    if resample_rate:
        song = song.set_frame_rate(resample_rate)

    fname_new = fname[:-4] + '.wav'
    song.export(fname_new, format = 'wav')
    fs, data = wavfile.read(fname_new)

    # merge sound channels
    if data.shape[1] == 2:
        data = data.mean(axis = 1)

    return fs, data


# In[169]:


def data_crop(data):
    '''
    Crop out beginning and end when there's no sound
    Input:
        data: (Numpy Array) music piece
```

```
    Output:
        return a numpy array of music
    '''

    thres = abs(data).mean() / 20
    idx = np.where(abs(data) >= thres)[0]
    data_c = data[idx[0]:idx[-1]].copy()

    return data_c


# In[189]:


def data_5sec_init(fs, data, num):
    '''
    Extract multiple 5-sec-sound-pieces from data
    Input:
        fs: (int) sampling rate
        data: (Numpy Array) music piece longer than 5 seconds
        num: (int) number of music pieces extracted
    Output:
        return a 2d numpy array (column = each sound piece)
    '''

    # dimension
    data_len = data.shape[0]

    # sanity check
    if data_len <= fs*5:
        print('Error: Music piece must be longer than 5 seconds. ')
        return
    if isinstance(num, int) != True:
        print('Error: Invalid # of music pieces. Must be integer. ')
        return

    # get max partition number
    num_max = np.floor(data_len / (fs*5))
    if num >= num_max:
        print('Error: Invalid # of music pieces. Must be smaller than {0}. '.format(num_max))
        return

    # randomized index
    idx_all = np.arange(num_max, dtype = np.int16)
    np.random.shuffle(idx_all)
    idx_list = idx_all[:num]

    # initialize the output array, column: music piece
    data5sec = np.zeros((fs*5, num))

    # extract music pieces
    for i, idx in enumerate(idx_list):
        data5sec[:,i] = data[idx:(idx + fs*5)].copy()
```

```
    return data5sec


# In[195]:


def genre_init(folder, fname_spec = '.mp3', num_clips = 20, fs_default = 44100):
    '''
    Batch import mp3 music pieces of 1 genre
    Extract multiple 5-sec-music-clips as dataset of 1 genre
    Input:
        folder: (String) directory name
        fname_spec: (String) specify what is contained in filename
        num_clips: (int) number of music pieces extracted from each song
        fs_default: (int) resampled frame rate
    Output:
        return a 2d numpy matrix with 5-sec-music_clips as column vectors
    '''

    # sanity check on folder name
    if folder.endswith('/') == False:
        folder = folder + '/'

    # fetch a list of filenames
    if fname_spec == None:
        fnames = sorted(listdir(folder))
    else:
        fnames = list(fname for fname in sorted(listdir(folder)) if fname_spec in fname)
        if len(fnames) == 0:
            print('Error: No file name contains ' + str(fname_spec))
            return

    # initialize matrix
    num_songs = len(fnames)
    data_5sec = np.zeros((num_songs, fs_default*5, num_clips))

    for i, fname in enumerate(fnames):

        fs, data_i = mp3read(folder + fname, fs_default) # import
        data_c = data_crop(data_i) # crop
        data_5sec[i] = data_5sec_init(fs, data_c, num_clips) # extract 5 pieces

    genre_data = np.concatenate(data_5sec, axis = 1)

    return genre_data


# In[245]:


def genre_spec(genre_data, fs = 44100):
    '''
    Generate spectrogram of music pieces
    Input:
```

```
        genre_data: (Numpy Array) a 2D matrix with music pieces as column vectors
        fs: sampling rate
    Output:
        return a 2D matrix with spectrograms concatenated as column vectors
    '''

    # extract the dimension
    f, t, Sxx = signal.spectrogram(genre_data[:,0], fs)
    length = Sxx.shape[0] * Sxx.shape[1]
    width = genre_data.shape[1]
    data_spec = np.zeros((length, width))

    for i in range(genre_data.shape[1]):
        f, t, Sxx = signal.spectrogram(genre_data[:,i], fs)
        data_spec[:,i] = np.concatenate(Sxx)

    return data_spec


# ## Test 1

# ### Initialize

# In[220]:


# initialize data
beethoven = genre_init('dataset/Beethoven', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
beethoven_spec = genre_spec(beethoven)


# In[250]:


# initialize data
mj = genre_init('dataset/Michael_Jackson', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
mj_spec = genre_spec(mj)


# In[255]:


# initialize data
soundgard = genre_init('dataset/Soundgarden', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
soundgard_spec = genre_spec(soundgard)


# In[278]:


# concatenate into a big matrix
```

```
test1 = np.concatenate((beethoven_spec, mj_spec, soundgard_spec), axis = 1)
# subtract mean
test1 = test1 - test1.mean(axis = 0)


# ### SVD

# In[279]:


get_ipython().run_cell_magic('time', '', 'u1f, s1f, vh1f = jsp.linalg.svd(test1, full_matrices = False)


# In[280]:


get_ipython().run_cell_magic('time', '', 'u1 = np.asarray(u1f)\ns1 = np.asarray(s1f)\nvh1 = np.asarray(


# In[657]:


# plot singular values of music pieces
# each singular value represents a feature
plt.figure(101, figsize = (10,4))
plt.subplot(121)
plt.scatter(np.arange(1,len(s1)+1), s1)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.subplot(122)
plt.semilogy(np.arange(1,len(s1)+1), s1)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.show()


# ### Clarification
#
# u1: columns = eigenmodes
#
# s1: std dev of features
#
# vh1: columns = weights of features for each music piece
#
# vh1: rows = weights of music pieces for each feature
#
# v1: vh1.transpose

# ### Cross Validation

# In[347]:


# import packages
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


# In[559]:


def train_test_gen(data, range_feat, num_clip):
    '''
    Generate train and test set randomly
    Input:
        data: (Numpy array) data with features (each data = each row)
        range_feat: (Numpy array) range of features selected
        num_clip: (int) number of clips for each genre
    Output:
        return train and test set as numpy arrays
    '''

    # dimension
    shape = data.shape
    # sanity check
    if float.is_integer(shape[0]/num_clip) == False:
        print('Error: Invalid num_clip. ')
        return

    num_genre = int(shape[0]/num_clip)
    num_feat = len(range_feat)

    test_size = 0.2
    xtrain = np.zeros((num_genre, num_clip - int(num_clip*test_size), num_feat))
    xtest = np.zeros((num_genre, int(num_clip*test_size), num_feat))
    ytrain = np.zeros((num_genre, num_clip - int(num_clip*test_size)))
    ytest = np.zeros((num_genre, int(num_clip*test_size)))

    for i in range(num_genre):

        xtemp = data[i*num_clip:(i+1)*num_clip, range_feat]
        ytemp = np.ones(num_clip)*(i+1) # class
        # train test split
        xtrain[i], xtest[i], ytrain[i], ytest[i] = train_test_split(xtemp, ytemp, test_size = 0.2)

    xtrain = np.concatenate(xtrain)
    xtest = np.concatenate(xtest)
    ytrain = np.concatenate(ytrain)
    ytest = np.concatenate(ytest)

    # shuffle train set
    idx = np.arange(len(ytrain))
    np.random.shuffle(idx)
    xtrain = xtrain[idx]
    ytrain = ytrain[idx]

    return xtrain, xtest, ytrain, ytest
```

```
# ### Classification - KNN

# In[522]:


# import packages
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn import svm


# In[595]:


# params
range_feature = np.arange(1,10)
num_clip = 120
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh1.transpose(), range_feature, 120)


# In[661]:


# choose knn parameter
k_scores = []
#k_errors = []
k_range = np.arange(1,31)
for k in k_range:
    knn = KNN(n_neighbors = k)
    scores = cross_val_score(knn, xtrain, ytrain, cv = 6, scoring='accuracy')
    k_scores.append(scores.mean())
    #errors = abs(cross_val_score(knn, xtrain, ytrain, cv = 6, scoring='neg_mean_squared_error'))
    #k_errors.append(errors.mean())

plt.figure(102, figsize = (6, 4))
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[597]:


# knn
knn_model = KNN(n_neighbors = 2)
knn_model.fit(xtrain, ytrain)
ytest_pred = knn_model.predict(xtest)
```

```
# In[598]:


test_score = knn_model.score(xtest, ytest)
cv_score = cross_val_score(knn_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(103, figsize = (8, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true SG', 'true MJ', 'true BE', 'predict BE', 'predict MJ', 'predict SG'])
plt.legend()
plt.show()

print('The accuracy of KNN with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by KNN is {0}. '.format(test_score))


### Classification - Naive Bayes

# In[601]:


# params
range_feature = np.arange(200)
num_clip = 120
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh1.transpose(), range_feature, 120)


# In[602]:


gnb_model = GaussianNB()
cross_val_score(gnb_model, xtrain, ytrain, cv = 6, scoring = 'accuracy')


# In[603]:


gnb_model = GaussianNB()
gnb_model.fit(xtrain, ytrain)
ytest_pred = gnb_model.predict(xtest)


# In[604]:


test_score = gnb_model.score(xtest, ytest)
cv_score = cross_val_score(gnb_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
```

```
plt.figure(104, figsize = (8, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true SG', 'true MJ', 'true BE', 'predict BE', 'predict MJ', 'predict SG']
plt.legend()
plt.show()

print('The accuracy of Naive Bayes with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by Naive Bayes is {0}. '.format(test_score))


# ### Classification - LDA

# In[646]:


# params
range_feature = np.arange(50)
num_clip = 120
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh1.transpose(), range_feature, 120)


# In[647]:


lda_model = LDA()
cross_val_score(lda_model, xtrain, ytrain, cv = 5, scoring = 'accuracy')


# In[648]:


lda_model = LDA()
lda_model.fit(xtrain, ytrain)
ytest_pred = lda_model.predict(xtest)


# In[649]:


test_score = lda_model.score(xtest, ytest)
cv_score = cross_val_score(lda_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(105, figsize = (8, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true SG', 'true MJ', 'true BE', 'predict BE', 'predict MJ', 'predict SG']
plt.legend()
```

```
plt.show()

print('The accuracy of LDA with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by LDA is {0}. '.format(test_score))


# ### Classification - SVM

# In[670]:


# determine num of features selected
num_list = np.arange(5, 205, 5)
cv_scores = []
for num in num_list:
    range_feature = np.arange(num)
    num_clip = 120
    # train test split
    xtrain, xtest, ytrain, ytest = train_test_gen(vh1.transpose(), range_feature, 120)
    svm_model = svm.SVC(gamma = 'auto')
    score = cross_val_score(svm_model, xtrain, ytrain, cv = 5, scoring = 'accuracy')
    cv_scores.append(score.mean())

# plot
plt.figure(116, figsize = (6, 4))
plt.plot(num_list, cv_scores)
plt.xlabel('# of features selected')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[641]:


# params
range_feature = np.arange(90)
num_clip = 120
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh1.transpose(), range_feature, 120)


# In[642]:


svm_model = svm.SVC(gamma = 'auto')
cross_val_score(svm_model, xtrain, ytrain, cv = 6, scoring = 'accuracy')


# In[643]:


svm_model = svm.SVC(gamma = 'auto')
svm_model.fit(xtrain, ytrain)
ytest_pred = svm_model.predict(xtest)
```

```
# In[645]:


test_score = svm_model.score(xtest, ytest)
cv_score = cross_val_score(svm_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(106, figsize = (8, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true SG', 'true MJ', 'true BE', 'predict BE', 'predict MJ', 'predict SG']
plt.legend()
plt.show()

print('The accuracy of LDA with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by LDA is {0}. '.format(test_score))


# ## Test 2

# ### Initialization

# In[650]:


# initialize data
pj = genre_init('dataset/Pearl_Jam', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
pj_spec = genre_spec(pj)


# In[651]:


# initialize data
aic = genre_init('dataset/Alice_in_chains', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
aic_spec = genre_spec(aic)


# In[652]:


# concatenate into a big matrix
test2 = np.concatenate((soundgard_spec, aic_spec, pj_spec), axis = 1)
# subtract mean
test2 = test2 - test2.mean(axis = 0)


# ### SVD
```

```
# In[653]:


get_ipython().run_cell_magic('time', '', 'u2f, s2f, vh2f = jsp.linalg.svd(test2, full_matrices = False)


# In[654]:


get_ipython().run_cell_magic('time', '', 'u2 = np.asarray(u2f)\ns2 = np.asarray(s2f)\nvh2 = np.asarray(


# In[656]:


# plot singular values of music pieces
# each singular value represents a feature
plt.figure(121, figsize = (10,4))
plt.subplot(121)
plt.scatter(np.arange(1,len(s2)+1), s2)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.subplot(122)
plt.semilogy(np.arange(1,len(s2)+1), s2)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.show()


# ### Classification - KNN

# In[686]:


# determine num of features selected
num_list = np.arange(4, 52, 2)
cv_scores = []
for num in num_list:
    range_feature = np.arange(num)
    num_clip = 120
    # train test split
    xtrain, xtest, ytrain, ytest = train_test_gen(vh2.transpose(), range_feature, 120)
    knn_model = KNN(n_neighbors = 5)
    score = cross_val_score(knn_model, xtrain, ytrain, cv = 5, scoring = 'accuracy')
    cv_scores.append(score.mean())

# plot
plt.figure(126, figsize = (6, 4))
plt.plot(num_list, cv_scores)
plt.xlabel('# of features selected')
plt.ylabel('Cross-Validated Accuracy')
plt.show()
```

```
# In[677]:


# choose knn parameter
k_scores = []
#k_errors = []
k_range = np.arange(1,31)
for k in k_range:
    knn = KNN(n_neighbors = k)
    scores = cross_val_score(knn, xtrain, ytrain, cv = 6, scoring='accuracy')
    k_scores.append(scores.mean())
    #errors = abs(cross_val_score(knn, xtrain, ytrain, cv = 6, scoring='neg_mean_squared_error'))
    #k_errors.append(errors.mean())

plt.figure(122, figsize = (6, 4))
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[738]:


# params
range_feature = np.arange(1,20)
num_clip = 120
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh2.transpose(), range_feature, 120)


# In[739]:


# knn
knn_model = KNN(n_neighbors = 5)
knn_model.fit(xtrain, ytrain)
ytest_pred = knn_model.predict(xtest)


# In[740]:


test_score = knn_model.score(xtest, ytest)
cv_score = cross_val_score(knn_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(123, figsize = (8, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true PJ', 'true AiC', 'true SG', 'predict SG', 'predict AiC', 'predict P
```

```
plt.legend()
plt.show()

print('The accuracy of KNN with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by KNN is {0}. '.format(test_score))


# ### Classification - Naive Bayes

# In[688]:


# determine num of features selected
num_list = np.arange(5, 105, 5)
cv_scores = []
for num in num_list:
    range_feature = np.arange(num)
    num_clip = 120
    # train test split
    xtrain, xtest, ytrain, ytest = train_test_gen(vh2.transpose(), range_feature, 120)
    gnb_model = GaussianNB()
    score = cross_val_score(gnb_model, xtrain, ytrain, cv = 5, scoring = 'accuracy')
    cv_scores.append(score.mean())

# plot
plt.figure(131, figsize = (6, 4))
plt.plot(num_list, cv_scores)
plt.xlabel('# of features selected')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[741]:


# params
range_feature = np.arange(80)
num_clip = 120
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh2.transpose(), range_feature, 120)


# In[742]:


gnb_model = GaussianNB()
cross_val_score(gnb_model, xtrain, ytrain, cv = 6, scoring = 'accuracy')


# In[743]:


gnb_model = GaussianNB()
gnb_model.fit(xtrain, ytrain)
```

```
ytest_pred = gnb_model.predict(xtest)


# In[744]:


test_score = gnb_model.score(xtest, ytest)
cv_score = cross_val_score(gnb_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(134, figsize = (8, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true PJ', 'true AiC', 'true SG', 'predict SG', 'predict AiC', 'predict P.
plt.legend()
plt.show()

print('The accuracy of Naive Bayes with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by Naive Bayes is {0}. '.format(test_score))


# ### Classification - LDA

# In[694]:


# determine num of features selected
num_list = np.arange(5, 105, 5)
cv_scores = []
for num in num_list:
    range_feature = np.arange(num)
    num_clip = 120
    # train test split
    xtrain, xtest, ytrain, ytest = train_test_gen(vh2.transpose(), range_feature, 120)
    lda_model = LDA()
    score = cross_val_score(lda_model, xtrain, ytrain, cv = 5, scoring = 'accuracy')
    cv_scores.append(score.mean())

# plot
plt.figure(141, figsize = (6, 4))
plt.plot(num_list, cv_scores)
plt.xlabel('# of features selected')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[745]:


# params
range_feature = np.arange(40)
num_clip = 120
```

```
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh2.transpose(), range_feature, 120)


# In[746]:


lda_model = LDA()
cross_val_score(lda_model, xtrain, ytrain, cv = 6, scoring = 'accuracy')


# In[747]:


lda_model = LDA()
lda_model.fit(xtrain, ytrain)
ytest_pred = lda_model.predict(xtest)


# In[748]:


test_score = lda_model.score(xtest, ytest)
cv_score = cross_val_score(lda_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(145, figsize = (8, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true PJ', 'true AiC', 'true SG', 'predict SG', 'predict AiC', 'predict P
plt.legend()
plt.show()

print('The accuracy of LDA with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by LDA is {0}. '.format(test_score))


# ### Classification - SVM

# In[717]:


# determine num of features selected
num_list = np.arange(5, 205, 5)
cv_scores = []
for num in num_list:
    range_feature = np.arange(num)
    num_clip = 120
    # train test split
    xtrain, xtest, ytrain, ytest = train_test_gen(vh2.transpose(), range_feature, 120)
    svm_model = svm.SVC(gamma = 'auto')
    score = cross_val_score(svm_model, xtrain, ytrain, cv = 5, scoring = 'accuracy')
```

```
    cv_scores.append(score.mean())

# plot
plt.figure(156, figsize = (6, 4))
plt.plot(num_list, cv_scores)
plt.xlabel('# of features selected')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[749]:


# params
range_feature = np.arange(40)
num_clip = 120
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh2.transpose(), range_feature, 120)


# In[750]:


svm_model = svm.SVC(gamma = 'auto')
cross_val_score(svm_model, xtrain, ytrain, cv = 6, scoring = 'accuracy')


# In[751]:


svm_model = svm.SVC(gamma = 'auto')
svm_model.fit(xtrain, ytrain)
ytest_pred = svm_model.predict(xtest)


# In[752]:


test_score = svm_model.score(xtest, ytest)
cv_score = cross_val_score(svm_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(166, figsize = (8, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true PJ', 'true AiC', 'true SG', 'predict SG', 'predict AiC', 'predict P
plt.legend()
plt.show()

print('The accuracy of LDA with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by LDA is {0}. '.format(test_score))
```

```
# ## Test 3

# ### Initialization

# In[729]:


# initialize data
beatles = genre_init('dataset/The_Beatles', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
bea_spec = genre_spec(beatles)


# In[730]:


# initialize data
guns = genre_init('dataset/Guns_N\'_Roses', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
guns_spec = genre_spec(guns)


# In[731]:


# initialize data
oasis = genre_init('dataset/Oasis', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
oasis_spec = genre_spec(oasis)


# In[732]:


# initialize data
tchai = genre_init('dataset/Tchaikovsky', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
tchai_spec = genre_spec(tchai)


# In[733]:


# initialize data
moz = genre_init('dataset/Mozart', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
moz_spec = genre_spec(moz)


# In[734]:


# initialize data
```

```
jazz = genre_init('dataset/Jazz', fname_spec = '.mp3', fs_default = 44100)
# calculate spectrogram
jazz_spec = genre_spec(jazz)


# In[753]:


# concatenate into a big matrix
test3 = np.concatenate((bea_spec, guns_spec, oasis_spec, beethoven_spec, tchai_spec, moz_spec, jazz_spe
# subtract mean
test3 = test3 - test3.mean(axis = 0)


# ### SVD

# In[755]:


get_ipython().run_cell_magic('time', '', 'u3f, s3f, vh3f = jsp.linalg.svd(test3, full_matrices = False)


# In[756]:


get_ipython().run_cell_magic('time', '', 'u3 = np.asarray(u3f)\ns3 = np.asarray(s3f)\nvh3 = np.asarray(


# In[757]:


# plot singular values of music pieces
# each singular value represents a feature
plt.figure(201, figsize = (10,4))
plt.subplot(121)
plt.scatter(np.arange(1,len(s3)+1), s3)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.subplot(122)
plt.semilogy(np.arange(1,len(s3)+1), s3)
plt.xlabel('Singular value number')
plt.ylabel('Singular value')
plt.show()


# ### Classification - KNN

# In[764]:


# determine num of features selected
num_list = np.arange(4, 52, 2)
cv_scores = []
for num in num_list:
```

```
    range_feature = np.arange(num)
    num_clip = 360
    # train test split
    xtrain, xtest, ytrain, ytest = train_test_gen(vh3.transpose(), range_feature, num_clip)
    knn_model = KNN(n_neighbors = 4)
    score = cross_val_score(knn_model, xtrain, ytrain, cv = 5, scoring = 'accuracy')
    cv_scores.append(score.mean())

# plot
plt.figure(226, figsize = (6, 4))
plt.plot(num_list, cv_scores)
plt.xlabel('# of features selected')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[760]:


# choose knn parameter
k_scores = []
#k_errors = []
k_range = np.arange(1,31)
for k in k_range:
    knn = KNN(n_neighbors = k)
    scores = cross_val_score(knn, xtrain, ytrain, cv = 6, scoring='accuracy')
    k_scores.append(scores.mean())
    #errors = abs(cross_val_score(knn, xtrain, ytrain, cv = 6, scoring='neg_mean_squared_error'))
    #k_errors.append(errors.mean())

plt.figure(222, figsize = (6, 4))
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[819]:


# params
range_feature = np.arange(1,20)
num_clip = 360
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh3.transpose(), range_feature, num_clip)


# In[820]:


# knn
knn_model = KNN(n_neighbors = 4)
knn_model.fit(xtrain, ytrain)
ytest_pred = knn_model.predict(xtest)
```

```
# In[768]:


test_score = knn_model.score(xtest, ytest)
cv_score = cross_val_score(knn_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(223, figsize = (9, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true Jazz', 'true Classic', 'true Rock', 'predict Rock', 'predict Classi
plt.legend()
plt.show()

print('The accuracy of KNN with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by KNN is {0}. '.format(test_score))


### Classification - Naive Bayes

# In[771]:


# determine num of features selected
num_list = np.arange(5, 305, 5)
cv_scores = []
for num in num_list:
    range_feature = np.arange(num)
    num_clip = 360
    # train test split
    xtrain, xtest, ytrain, ytest = train_test_gen(vh3.transpose(), range_feature, 360)
    gnb_model = GaussianNB()
    score = cross_val_score(gnb_model, xtrain, ytrain, cv = 5, scoring = 'accuracy')
    cv_scores.append(score.mean())

# plot
plt.figure(231, figsize = (6, 4))
plt.plot(num_list, cv_scores)
plt.xlabel('# of features selected')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[772]:


# params
range_feature = np.arange(200)
num_clip = 360
# train test split
```

```
xtrain, xtest, ytrain, ytest = train_test_gen(vh3.transpose(), range_feature, num_clip)


# In[773]:


gnb_model = GaussianNB()
cross_val_score(gnb_model, xtrain, ytrain, cv = 6, scoring = 'accuracy')


# In[774]:


gnb_model = GaussianNB()
gnb_model.fit(xtrain, ytrain)
ytest_pred = gnb_model.predict(xtest)


# In[775]:


test_score = gnb_model.score(xtest, ytest)
cv_score = cross_val_score(gnb_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(234, figsize = (9, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true Jazz', 'true Classic', 'true Rock', 'predict Rock', 'predict Classi
plt.legend()
plt.show()

print('The accuracy of Naive Bayes with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by Naive Bayes is {0}. '.format(test_score))


# ### Classification - LDA

# In[779]:


# determine num of features selected
num_list = np.arange(5, 205, 5)
cv_scores = []
for num in num_list:
    range_feature = np.arange(num)
    num_clip = 360
    # train test split
    xtrain, xtest, ytrain, ytest = train_test_gen(vh3.transpose(), range_feature, num_clip)
    lda_model = LDA()
    score = cross_val_score(lda_model, xtrain, ytrain, cv = 6, scoring = 'accuracy')
    cv_scores.append(score.mean())
```

```
# plot
plt.figure(241, figsize = (6, 4))
plt.plot(num_list, cv_scores)
plt.xlabel('# of features selected')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[780]:


# params
range_feature = np.arange(125)
num_clip = 360
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh3.transpose(), range_feature, num_clip)


# In[781]:


lda_model = LDA()
cross_val_score(lda_model, xtrain, ytrain, cv = 6, scoring = 'accuracy')


# In[782]:


lda_model = LDA()
lda_model.fit(xtrain, ytrain)
ytest_pred = lda_model.predict(xtest)


# In[784]:


test_score = lda_model.score(xtest, ytest)
cv_score = cross_val_score(lda_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(245, figsize = (9, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true Jazz', 'true Classic', 'true Rock', 'predict Rock', 'predict Classi
plt.legend()
plt.show()

print('The accuracy of LDA with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by LDA is {0}. '.format(test_score))
```

```
# ### Classification - SVM

# In[787]:


# determine num of features selected
num_list = np.arange(5, 305, 10)
cv_scores = []
for num in num_list:
    range_feature = np.arange(num)
    num_clip = 360
    # train test split
    xtrain, xtest, ytrain, ytest = train_test_gen(vh3.transpose(), range_feature, num_clip)
    svm_model = svm.SVC(gamma = 'auto')
    score = cross_val_score(svm_model, xtrain, ytrain, cv = 6, scoring = 'accuracy')
    cv_scores.append(score.mean())

# plot
plt.figure(256, figsize = (6, 4))
plt.plot(num_list, cv_scores)
plt.xlabel('# of features selected')
plt.ylabel('Cross-Validated Accuracy')
plt.show()


# In[790]:


# params
range_feature = np.arange(250)
num_clip = 360
# train test split
xtrain, xtest, ytrain, ytest = train_test_gen(vh3.transpose(), range_feature, num_clip)


# In[791]:


svm_model = svm.SVC(gamma = 'auto')
cross_val_score(svm_model, xtrain, ytrain, cv = 6, scoring = 'accuracy')


# In[792]:


svm_model = svm.SVC(gamma = 'auto')
svm_model.fit(xtrain, ytrain)
ytest_pred = svm_model.predict(xtest)


# In[793]:


test_score = svm_model.score(xtest, ytest)
```

```
cv_score = cross_val_score(svm_model, xtrain, ytrain, cv = 6, scoring='accuracy')

# plot prediction
plt.figure(266, figsize = (9, 5))
plt.stem(ytest_pred, basefmt = 'black', label = 'prediction')
plt.stem(-ytest, linefmt = 'm', markerfmt = 'mo', basefmt = 'black', label = 'true solution')
plt.xlabel('Music piece')
plt.ylabel('class')
plt.yticks(np.arange(-3, 4), ['true Jazz', 'true Classic', 'true Rock', 'predict Rock', 'predict Classi
plt.legend()
plt.show()

print('The accuracy of LDA with cross validation is {0}. '.format(cv_score.mean()))
print('The accuracy of predicting test set by LDA is {0}. '.format(test_score))


# ### Test on an arbitrary 5-sec music clip

# In[841]:


# generate a 5-sec music clip
#testfs, testdata = wavfile.read('dataset/Oasis/Supersonic - Oasis.wav')
testfs, testdata = wavfile.read('dataset/Pearl_Jam/Yellow LedBetter - Pearl Jam.wav')
testdata = testdata.mean(axis = 1)
testdata = data_crop(testdata)
test5sec = data_5sec_init(testfs, testdata, 1)
spec5sec = genre_spec(test5sec)
# this is in genre 'rock'


# ££A = U\Sigma V^*††
# ††\Sigma^{-1}U^*A = V^*††

# In[842]:


# project onto feature space
vh5spec = np.linalg.multi_dot((np.diag(1/s3), u3.transpose(), spec5sec))


# In[843]:


# knn prediction
range_feat = np.arange(1,20)
knn_model.predict(vh5spec[range_feat].transpose())
# we can see that the prediction is not rock (class 1)


# In[844]:


# gnb prediction
```

```
range_feat = np.arange(200)
gnb_model.predict(vh5spec[range_feat].transpose())
# we can see that the prediction is rock (class 1)


# In[845]:


# lda prediction
range_feat = np.arange(125)
lda_model.predict(vh5spec[range_feat].transpose())
# we can see that the prediction is rock (class 1)


# In[846]:


# svm prediction
range_feat = np.arange(250)
svm_model.predict(vh5spec[range_feat].transpose())
# we can see that the prediction is not rock (class 1)


# In[ ]:
```