

# Single note Musical Instrument Recognition using PCA and Machine Learning

## Group -07

Henil Shah	-	AU1940205
Kathan Joshi	-	AU1940067
Devanshu Magiawala	-	AU1940190
Devarsh Sheth	-	AU1940189

## 2.Introduction:

### 1. Background:-

Musical instrument recognition has been gaining interest in developers for the last many years. Many people have the capability of classifying different musical sounds. Making a program that has a brain like these people are really difficult. This program is to detect single-note sounds of various musical instruments. For example, a twofold bass is known to have a much lower pitch reach to a violin, all in all, instruments are recognizable in any event when played at a similar pitch and uproar. As characterized by their nature of hear-able sensation by which an audience can recognize two hints of equivalent uproar, duration and pitch are known as timbre. The timbre of an instrument that is difficult to classify or measure. We have used the PCA method for classifying the different timbres.

Musical instrument recognition mostly revolves around frequency domain analysis such as sinusoidal analysis and cepstral coefficients and as well as shape analysis for extracting some various sets of features. Instruments are classified using k-NN classifiers, HMM, Kohonen SOM, and Neural Networks. Even the Gaussian Mixture Model (GMM) and the Support Vector Machines (SVM) are also used.

### 2. Motivation and Overview:-

Other approaches to solving this problem are K-NN or SVM:

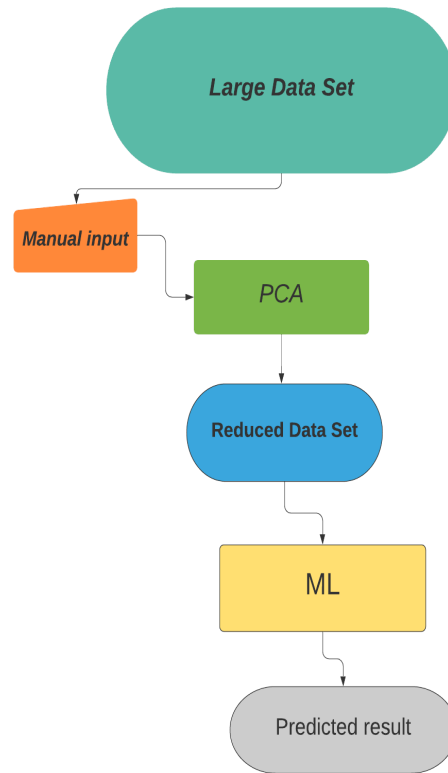
- a) K-NN: K-Nearest Neighbour algorithm, abbreviated as K-NN is an approach that is used to classify a data point that is nearer to be a member of one group or to which group it is near to. This method does not build a model using the training set till a query on the data set is generated.
- b) SVM: Support Vector Machine is an algorithm that analyzes data for regression and classification analysis. It first analyses data and sorts it into one of the two categories. It gives a map of the sorted data with a margin in

between as an output. SVMs are also used in image classification, text categorization, handwriting recognition, etc.

This study aims to create an automatic single note musical instrument classifier by extracting and understanding relevant features. These features are used for representing the timbre of the instrument. This model has been examined on two types of instruments as explained further.

### 3. Approach: -

1. The Project uses a Machine Learning approach based on PCA(Principal Component Analysis). As the project is using Machine Learning approach the size of the data sets matters a lot; with large data sets it may take huge time to train the model or even test it; therefore we have used PCA to reduce the size of the data.



The flowchart represents the basics of our project.

**Input:** an audio file (different sounds of instruments within 4th octave).

**Output:** Prediction of the Musical Instrument.

**Process:** PCA based approach to optimize the time complexity of the algorithm.

1. Firstly, the audio file is converted into a matrix; The Mel Frequency Cepstral Coefficients (MFCCs) of each music piece was extracted using Librosa(A python library). For each audio file, its MFCCs are averaged to produce the final, length-20 feature vector(original data set)
2. After converting the audio file into a 20X20 matrix; PCA based approach is used to reduce the dimensions of the feature vectors.
3. After classification, Predicted results are displayed on the terminal window along with the accuracy of the obtained results.

### **3.1.1 Principal Component Analysis:-**

PCA is a method for dimensionality-reduction i .e.transforming a large set of data into a small one based on the information each field carries in the data set. In general, PCA reduces data redundancy based on the variable that contributes most to the data set.

It is a linear model in mapping m-dimensional feature into an n-dimensional feature(where,  $m > n$ ). Thus, creating a small dimensional feature space than the original data. This process maximizes the variance of the data in the new feature space. Thus, it removes the components with less significance and preserves the principal directions with the highest variance. It transforms the data linearly into new orthogonal (uncorrelated) variables that carry most variability. And these variables are referred to as principal components.

Although it affects the accuracy of our ML-based approach it makes it much faster which can be considered as a valuable Trade-off, it also makes the data sets easier to analyze and process further. So basically, the Project uses PCA to get a new reduced data set, while preserving as many variations as possible from the original data set, used for recognizing the instrument to optimize the time taken by the machine learning algorithm.

It works mainly on 4 parts:

1. Identifying the relationship between features using the Covariance matrix.
2. Calculating the eigenvalues and eigenvectors using EVD(eigenvalue decomposition).
3. Transforming eigenvectors into principal components.
4. Keeping significant components using eigenvalues.

## **2. Linear Algebra Concepts Used: -**

## Principal Component Analysis(PCA):

The algorithm for PCA mainly involves the following mathematical concepts:

1. Variance and covariance.
2. Eigenvalues and Eigenvectors.

PCA approach mainly involves the following steps:

1. Standardization of data.
2. Computing the covariance matrix for the data.
3. Computing the eigenvalues and the corresponding eigenvector for the computed covariance matrix.
4. Calculating Principal Components based on the eigenvectors.
5. Computing new feature vectors.
6. Getting Final data set based on the new feature vector.

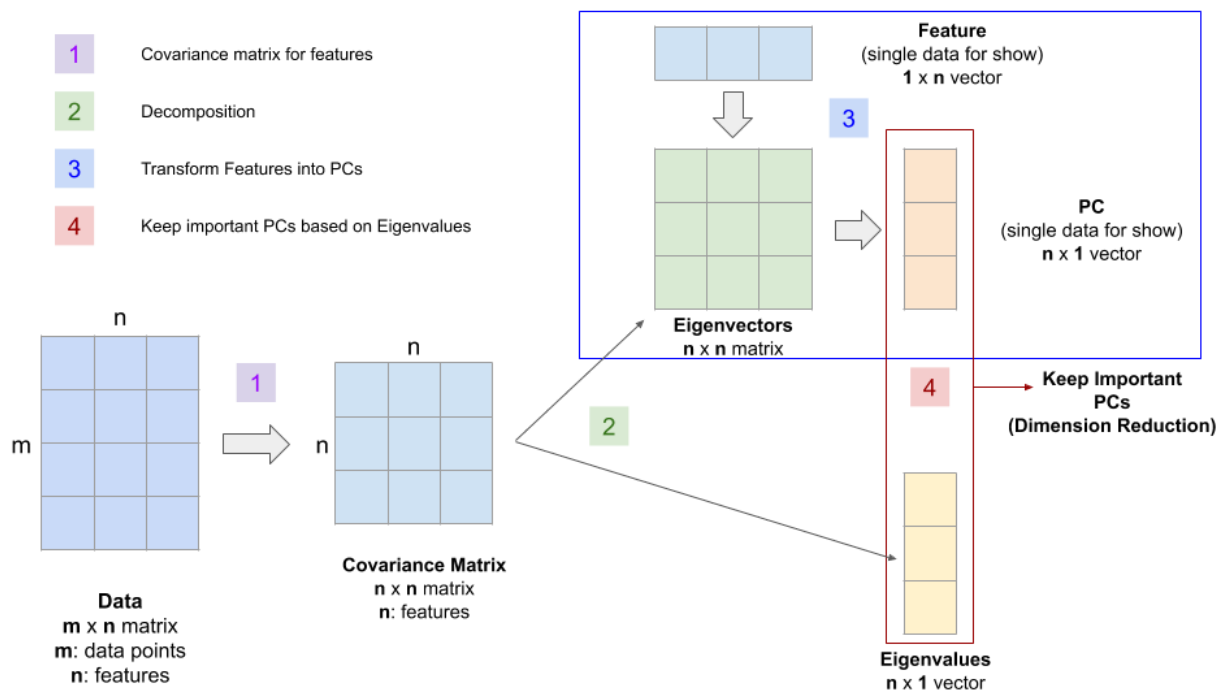


Figure 1(K. Zhao, "Feature Extraction using Principal Component Analysis - A Simplified Visual Demo," *Medium*, 10-Dec-2019. [Online]. Available: <https://towardsdatascience.com/feature-extraction-using-principal-component-analysis-a-simplified-visual-demo-e5592ced100a>. [Accessed: 04-Nov-2020].)

### 2.1(a) Standardizing the data set:

The Standardization of the data set is required to make sure that each entity contributes equally to the analysis. The aim is to make each component of the same range. For example, there might be the case in which some components are in range 1 to 100 while some are in range 0 to 1, in this case, each component contributes differently to the analysis just because each component is of a different range. Thus, standardization is required before PCA.

Mathematically this can be possible by subtracting each value by the mean and dividing by the standard deviation. Thus, after standardization, each component is transformed onto the same scale.

$$z = \frac{\text{value} - \text{mean}}{\text{standard deviation}}$$

### 2.1(b) Computing Covariance Matrix:

The covariance matrix tells us about how the variables of the data are varying about mean concerning each other. So, to sum up it gives the relationship between the variables. Sometimes, variables are highly correlated in such a way that they contain redundant information, thus to identify these correlations, the covariance matrix is computed.

The Covariance matrix would be an  $n \times n$  matrix, with column vectors being the feature vector. For any matrix  $A$   $m \times n$

Covariance matrix( $C$ ) of any matrix( $A$ ):  $C = A * A^T / (n-1)$

### 2.1(c) Computing the eigenvalues and the corresponding eigenvector:

After, we compute the eigenvalues and eigenvectors of the calculated covariance matrix and sort them accordingly in decreasing order of their respective eigenvalues. The project uses the Jacobi algorithm to compute the eigenvalues of the matrix. It is an iterative algorithm with time complexity:  $O(n^3)$ . For, Jacobi algorithm we assume the matrix to be symmetric.

Here, the Givens matrix is computed from the indices of the maximum element in the upper triangular portion of the symmetric matrix. After, the rotation matrix will transform each of the diagonal elements into

eigenvalues. Further, Mathematical explanations for the Jacobi algorithm can be found [here](#).

### 2.1(d) Constructing the Principal Components:

After computing the eigenvalues and eigenvectors, we sort them respectively according to their corresponding eigenvalues. For constructing Principal components eigenvectors play an important role. There are as many principal components as there are variables in data, but the Principle Components are constructed in such a way that the first component accounts for the largest possible variance in the data set. For example, consider the following scatter plot as the data set:-

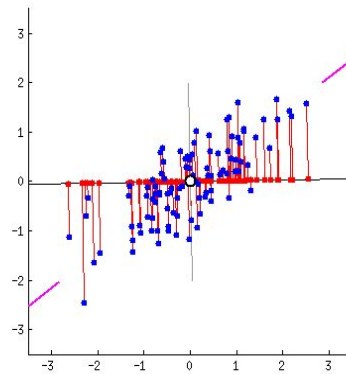


Figure 2(Z. Jaadi, "A Step by Step Explanation of Principal Component Analysis," *Built In*. [Online]. Available: <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>. [Accessed: 04-Nov-2020].)

Geometrically, the first principal component will approximately be the line when it overlapped with the purple marks; Because at this point, firstly it passes through the origin and the projection of the scatter plot(blue dots) on the line(red dots) is most spread out, thus at this point, it covers most of the variation of the scatter plot. Similarly, the second PC can be calculated uncorrelated to the first one thus, the second Principal component should be perpendicular to the first one.

Calculating Principal components: we transform each feature into PC by multiplying the feature with a matrix  $S$  having eigenvectors in decreasing order as its column.

### Transformation: From Features to Principal Components

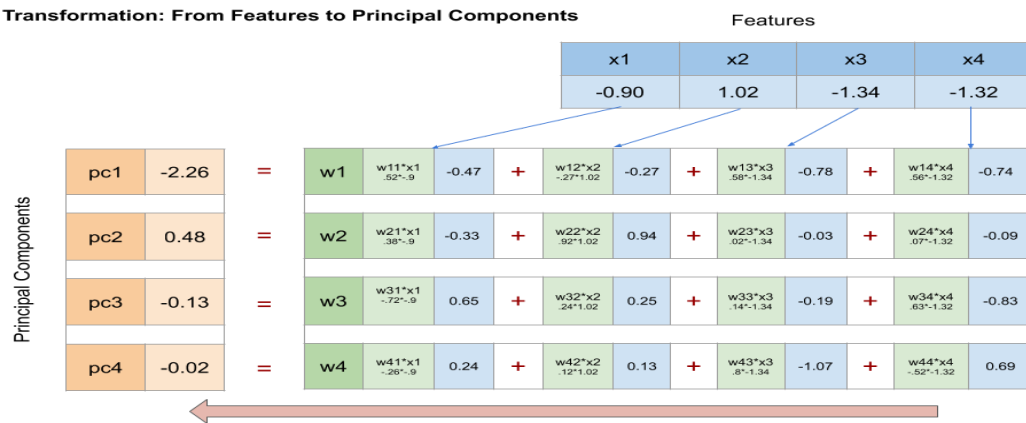


Figure 3(K. Zhao, "Feature Extraction using Principal Component Analysis - A Simplified Visual Demo," *Medium*, 10-Dec-2019. [Online]. Available: <https://towardsdatascience.com/feature-extraction-using-principal-component-analysis-a-simplified-visual-demo-e5592ced100a>. [Accessed: 04-Nov-2020].)

Therefore, the principal components  $P$  matrix(of each column of  $C$  or old feature vector) =  $S * (\text{old feature vector})$

### 2.1(e) Computing new feature vector

For computing the new feature vector, Only Principal components with high variance or to sum with high magnitudes are selected and thus the corresponding eigenvector with the lower principal components can be ignored.

### Feature Extraction



Figure 4(K. Zhao, "Feature Extraction using Principal Component Analysis - A Simplified Visual Demo," *Medium*, 10-Dec-2019. [Online]. Available: <https://towardsdatascience.com/feature-extraction-using-principal-component-analysis-a-simplified-visual-demo-e5592ced100a>. [Accessed: 04-Nov-2020].)

To sum, we choose only  $k$  eigenvectors from  $n$ , and thus the final data set will only have  $k$  dimensions instead of  $n$ . These  $k$  eigenvector components are chosen based on their corresponding eigenvalues,

## **2.1(f) Obtaining Final data set based on the new feature vector**

Final data set = standardized(original data set) \* feature vector space(chosen eigenvector as column)

Suppose, the original data set is of  $200 \times 4$ , and only 1 eigenvector has a maximum variance, then we can reduce the feature vector to  $4 \times 1$ , thus reducing the total dimension to  $200 \times 1$ .

Hence, the size of the final data set has been significantly reduced.

## **4. Coding and Simulation:-**

### **1. Coding Strategy:-**

Project Coding Strategies mainly involves the comparison of three data sets namely guitar, trumpet and mandolin, each of an individual instrument and comparing within these data sets against the user-given filename to predict the instrument(of a given file).

#### **1. The flow of Code:**

The project mainly consists of 4 files:

- a. Preprocessing.py**
- b. Classify.py**
- c. Demo.py**
- d. Scratch\_pca.py**
- e. jacobi.py
- f. Last\_pca.model



#### **4.1.1(a) Preprocessing.py**

This file is used to extract the matrix representation of the whole data set consisting of 3 individual datasets each having a different number of audio files for a specific instrument, after removing the unnecessary silence in a single audio file. It returns the source name, matrix representation of audiofile and the solution set for the file.

#### **4.1.1(b) Classify.py(main file)**

It uses the data returned by the Preprocessing.py and use the feature space to train and test the model at first using the file Scratch\_pca.py(team 7 defined function). It generates and stores the trained model data into the “Last\_pca.model”, and use it whenever called. This is the main file used to predict(under predict() function) the result when the demo.py is run.

#### **4.1.1(c) Demo.py**

This file is used to predict the audio file played based on the pre-trained model. Firstly, it extracts the matrix form of the given audio file and compares it against the pre-trained model obtained values, after applying the PCA to the extracted matrix. So, this is the main to run to predict the instrument of the audio file.

#### **4.1.1(d) Jacobi.py**

This file is team 7 defined, it is used to calculate the eigenvector and eigenvalues for the matrix form of the audio file.

#### **4.1.1(e) Scratch\_pca.py**

This file is the replacement for the original inbuilt PCA libraries, designed by group - 7. It replaces the inbuilt `pca.transform()` and `pca.fit_transform()` functions with our own created functions `scratch_fit_transform()` and

scratch\_transform(). Both these functions are designed based on the mathematical approach described above.

## 2. Modules Used:

Libraries Used	Use of that modules/libraries
sklearn	By using LogisticRegression, the model is trained by fitting the train data and tested using its predict method
librosa,ffmpeg	Both the libraries are used to convert the mp3 into a matrix data
glob	This library is used to load a whole dataset/ files for which the model is trained
numpy	Not a very extensive use, only for generating numpy arrays, identity matrix, diagonal matrix.

## 3. Figures

```

181 # Train and test the model
182 train_and_test(model_params, 'pc')
183
184 # Print the accuracy rate
185 print('Accuracy rate: %.4f' % accuracy_score(y_test, y_pred))
186
187 # Print the confusion matrix
188 print('Confusion matrix: \n', confusion_matrix(y_test, y_pred))
189
190 # Print the classification report
191 print('Classification report: \n', classification_report(y_test, y_pred))
192
193 if __name__ == '__main__':
194     main()

```

```

14 correct out of 15
accuracy rate: 0.9333333333333333
start Fold: 1
start Feature vector dim: (139, 20)
training done, start testing...
13 correct out of 15
accuracy rate: 0.8666666666666667
[0.8666666666666667, 0.9333333333333333, 0.8666666666666667, 0.9333333333333333, 0.9333333333333333, 1.0, 0.9333333333333333, 0.9333333333333333, 0.9333333333333333, 0.8666666666666667]
average: 0.9200000000000001
Process finished with exit code 0

```

**Figure 1: The accuracy rate(92%) using the in-built PCA function**

```

training done, start testing...
46 correct out of 67
accuracy rate: 0.6865671641791845
[0.7761194829858746, 0.7611948298587462, 0.746268656716418, 0.6417918447761194, 0.6567164179184478, 0.8288955223888597, 0.7761194829858746, 0.746268656716418, 0.8059781492537313, 0.7761194829858746]
average: 0.7417918447761195
Process finished with exit code 0

```

**Figure 2: The accuracy rate(74.1%) using grp created PCA function**

```

181 # Train and test the model
182 train_and_test(model_params, 'pc')
183
184 # Print the accuracy rate
185 print('Accuracy rate: %.4f' % accuracy_score(y_test, y_pred))
186
187 # Print the confusion matrix
188 print('Confusion matrix: \n', confusion_matrix(y_test, y_pred))
189
190 # Print the classification report
191 print('Classification report: \n', classification_report(y_test, y_pred))
192
193 if __name__ == '__main__':
194     main()

```

```

14 correct out of 15
accuracy rate: 0.9333333333333333
start Fold: 1
start Feature vector dim: (139, 20)
training done, start testing...
13 correct out of 15
accuracy rate: 0.8666666666666667
[0.8666666666666667, 0.9333333333333333, 0.8666666666666667, 0.9333333333333333, 0.9333333333333333, 1.0, 0.9333333333333333, 0.9333333333333333, 0.9333333333333333, 0.8666666666666667]
average: 0.9200000000000001
Process finished with exit code 0

```

```

181 # Train and test the model
182 train_and_test(model_params, 'pc')
183
184 # Print the accuracy rate
185 print('Accuracy rate: %.4f' % accuracy_score(y_test, y_pred))
186
187 # Print the confusion matrix
188 print('Confusion matrix: \n', confusion_matrix(y_test, y_pred))
189
190 # Print the classification report
191 print('Classification report: \n', classification_report(y_test, y_pred))
192
193 if __name__ == '__main__':
194     main()

```

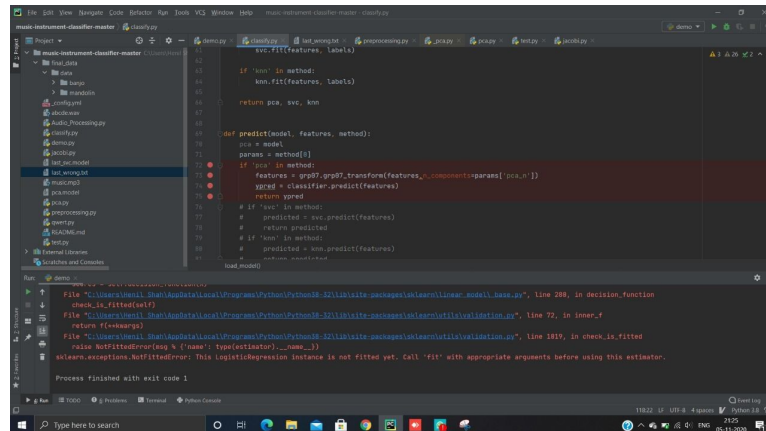
```

14 correct out of 15
accuracy rate: 0.9333333333333333
start Fold: 1
start Feature vector dim: (139, 20)
training done, start testing...
13 correct out of 15
accuracy rate: 0.8666666666666667
[0.8666666666666667, 0.9333333333333333, 0.8666666666666667, 0.9333333333333333, 0.9333333333333333, 1.0, 0.9333333333333333, 0.9333333333333333, 0.9333333333333333, 0.8666666666666667]
average: 0.9200000000000001
Process finished with exit code 0

```

**Figure 3,4,5: The prediction results as desired from the trained and tested model for the recognition of the three instruments**

Following images shows the errors faced during the execution of codes and below it contains some information on how we approached to solve it.



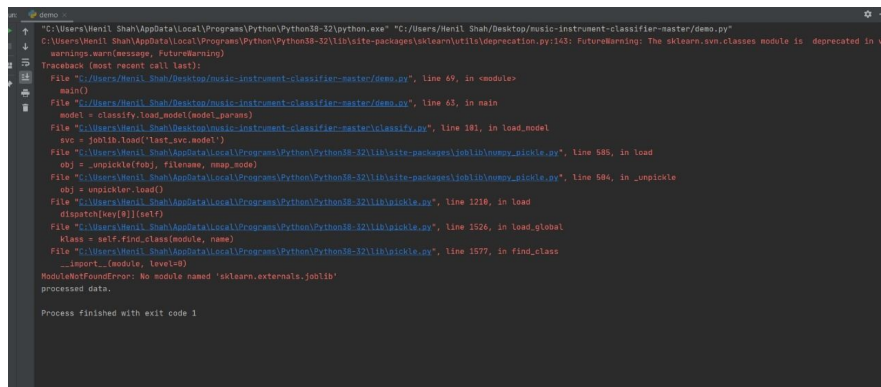
```

11 svr.fit(features, labels)
12
13 if 'svm' in method:
14     svm.fit(features, labels)
15
16 return proc_svm_svm
17
18 def predict(model, features, method):
19     params = method[s]
20     if 'svm' in method:
21         features = gpdt.gpdt.transform(features, components=params[proc_s])
22         ypred = classifier.predict(features)
23         return ypred
24     # if 'svm' in method:
25     #     predicted = svm.predict(features)
26     #     return predicted
27     # if 'svm' in method:
28     #     predicted = svm.predict(features)
29
30 load_model()

```

ValueError: This LogisticRegression instance is not fitted yet. Call 'fit' with appropriate arguments before using this estimator.

**Figure 6: Data fitting Error(Solved by using `scratch_fit_transform` and `scratch_transform` func.)**



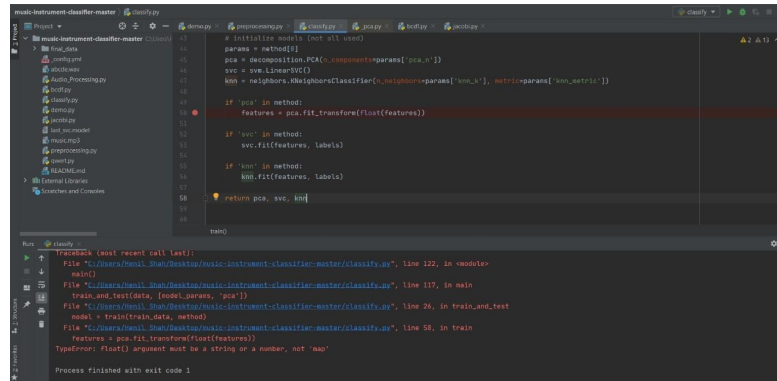
```

11 svr.fit(features, labels)
12
13 if 'svm' in method:
14     svm.fit(features, labels)
15
16 return proc_svm_svm
17
18 def predict(model, features, method):
19     params = method[s]
20     if 'svm' in method:
21         features = gpdt.gpdt.transform(features, components=params[proc_s])
22         ypred = classifier.predict(features)
23         return ypred
24     # if 'svm' in method:
25     #     predicted = svm.predict(features)
26     #     return predicted
27     # if 'svm' in method:
28     #     predicted = svm.predict(features)
29
30 load_model()

```

ModuleNotFoundError: No module named 'sklearn.externals.joblib'

**Figure 7: Joblib Error(Solved by adding [FFmpeg](#) file path to the environment variable)**



**Figure 8: Data type Error(Solved by changing the map object to dtype list)**

## 2. Inferences:-

The basic motto of the project was to classify the musical instrument according to the input file given to provide the predicted musical instrument. If the model is trained on the basis library function of sklearn.decomposition, pca.fit\_transform and pca.transform, the model is getting trained in a better way and while the time of testing, the prediction is sharp enough that it predicts with an accuracy of almost 92%. From figure 1, it can be inferred that the model is trained and tested successfully with 92% accuracy. This model works perfectly fine as it predicts pretty well.

But as to implement linear algebra concepts and ideas, the model was changed and it was trained on the same data but with pca functions applied through scratch i.e. scratch\_pca.fit\_transform and scratch\_pca.transform. Through these functions, the model is trained and while testing the model can predict the musical instrument with an accuracy of 73-80%. Figure 2 depicts the model training and testing with 74.1% accuracy. Figure 3,4, and 5 shows the predictions made by the scratch based pca.This model classifies musical instruments pretty well.

Comparing both the approaches and model training and testing, it can be said that model which are trained on inbuilt functions have better accuracy and better results. Sometimes, some predictions might get wrong as well.

**Reasons:** We might not have perfectly replicated the exact functioning or working of the in-built function pca.transform.

## 5. Conclusion:

The project successfully works with an approximate accuracy of 75% for the prediction of the instrument.

### Individual Contribution:

1. **Code:** Henil Shah, Kathan Joshi, Devanshu Magiawala
2. **Report:** Henil Shah, Kathan Joshi, Devanshu Magiawala, Devarsh Sheth
3. **PPT:** Henil Shah, Kathan Joshi, Devanshu Magiawala, Devarsh Sheth

### 6.References:-

- J. Hui, "Machine Learning - Singular Value Decomposition (SVD) & Principal Component Analysis (PCA)," *Medium*, 08-Feb-2020. [Online]. Available: <https://jonathan-hui.medium.com/machine-learning-singular-value-decomposition-svd-principal-component-analysis-pca-1d45e885e491>. [Accessed: 04-Nov-2020].
- A. Dubey, "The Mathematics Behind Principal Component Analysis," *Medium*, 25-Dec-2018. [Online]. Available: <https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-ff2d7f4b643>. [Accessed: 04-Nov-2020].
- "Principal component analysis," *Wikipedia*, 19-Oct-2020. [Online]. Available: [https://en.wikipedia.org/wiki/Principal\\_component\\_analysis](https://en.wikipedia.org/wiki/Principal_component_analysis). [Accessed: 04-Nov-2020].
- Z. Jaadi, "A Step by Step Explanation of Principal Component Analysis," *Built In*. [Online]. Available: <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>. [Accessed: 04-Nov-2020].
- O. G. Foundation, "Algorithm of Principal Component Analysis (PCA)," *OpenGenus IQ: Learn Computer Science*, 07-Jan-2019. [Online]. Available: <https://iq.opengenus.org/algorithm-principal-component-analysis-pca/>. [Accessed: 04-Nov-2020].
- "Principal Component Analysis," *Dr. Sebastian Raschka*, 27-Jan-2015. [Online]. Available: [https://sebastianraschka.com/Articles/2015\\_pca\\_in\\_3\\_steps.html](https://sebastianraschka.com/Articles/2015_pca_in_3_steps.html). [Accessed: 04-Nov-2020].
- K. Zhao, "Feature Extraction using Principal Component Analysis - A Simplified Visual Demo," *Medium*, 10-Dec-2019. [Online]. Available: <https://towardsdatascience.com/feature-extraction-using-principal-component-analysis-a-simplified-visual-demo-e5592ced100a>. [Accessed: 04-Nov-2020].

- “Jacobi eigenvalue algorithm,” *Wikipedia*, 28-Aug-2020. [Online]. Available: [https://en.wikipedia.org/wiki/Jacobi\\_eigenvalue\\_algorithm](https://en.wikipedia.org/wiki/Jacobi_eigenvalue_algorithm). [Accessed: 04-Nov-2020].
- L. Guignard and G. Kehoe, “Learning Instrument Identification,” *CS229 Machine Learning*, 2015. [Online]. Available: [http://cs229.stanford.edu/proj2015/010\\_report.pdf](http://cs229.stanford.edu/proj2015/010_report.pdf). [Accessed: 04-Nov-2020].