

CHAPTER 1: INTRODUCTION

Music genre is a conventional category that identifies pieces of music as belonging to a shared tradition or set of conventions.” The term “genre” is a subject to interpretation and it is often the case that genres may very fuzzy in their definition. Further, genres do not always have sound music theoretic foundations, e.g. - Indian genres are geographically defined, Baroque is classical music genre based on time period. Despite the lack of a standard criteria for defining genres, the classification of music based on genres is one of the broadest and most widely used. Genre usually assumes high weight in music recommender systems. Genre classification, till now, had been done manually by appending it to metadata of audio files or including it in album info. This project however aims at content-based classification, focusing on information within the audio rather than extraneously appended information. The traditional machine learning approach for classification is used - find suitable features of data, train classifier on feature data, make predictions. The novel thing that we have tried is the use of ensemble classifier on fundamentally different classifiers to achieve our end goal.

1.1 NEED OF THE STUDY:

Many companies nowadays use music classification, either to be able to place recommendations to their customers (such as Spotify, Soundcloud), or simply as a product (for example Shazam). Determining specific music genres is a first step towards this goal.

Companies and Software Industry are currently going through a Transition Phase, there is a need to handle the change by taking a step towards automation. So, the motive is to create a simple user friendly, smart music genre classifier system.

1.2 SCOPE OF THE STUDY:

It will help people in identifying and efficiently searching the type of music they like. If someone has a huge music library, the task identifying the genre of music can be done using this tool without even listening to the song. Companies like spotify, saavn etc. are constantly looking for the best approach for the classification of songs. This technique, although is being used by them, can help us develop or think of a more efficient way to perform this task.

1.3 OBJECTIVE OF THE STUDY:

We aim to create a program using machine learning which can classify genres of different audio files using audio processing. This model can be extended to different applications in the music information processing field which are used in various applications including music streaming applications.

CHAPTER 2: LITERATURE REVIEW

2.1 Machine learning:

Machine learning (ML) is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) from data, without being explicitly programmed.

The name machine learning was coined in 1959 by Arthur Samuel. Machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders, and computer vision.

Machine learning is closely related to (and often overlaps with) computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining, where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning.

Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data.

A core objective of a learner is to generalize from its experience. Generalization in this context is the ability of a learning machine to perform accurately on new, unseen examples/tasks after having experienced a learning data set. The training examples come from some generally unknown probability distribution (considered representative of the space of occurrences) and the learner has to build a general model about this space that enables it to produce sufficiently accurate predictions in new cases.

The computational analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory. Because training sets are finite, and the future is uncertain, learning theory usually does not yield guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common. The bias–variance decomposition is one way to quantify generalization error.

For the best performance in the context of generalization, the complexity of the hypothesis should match the complexity of the function underlying the data. If the hypothesis is less complex than the function, then the model has underfit the data. If the complexity of the model is increased in response, then the training error decreases. But if the hypothesis is too complex, then the model is subject to overfitting and generalization will be poorer.

In addition to performance bounds, computational learning theorists study the time complexity and feasibility of learning. In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results.

Positive results show that a certain class of functions can be learned in polynomial time.

Negative results show that certain classes cannot be learned in polynomial time.

2.2 Approaches

2.2.1 Supervised Learning

This algorithm consists of a target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data.

Examples of Supervised Learning: Regression, Decision Tree, Random Forest, KNN, Logistic Regression etc.

2.2.2 Unsupervised Learning

In this algorithm, we do not have any target or outcome variable to predict / estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples of Unsupervised Learning: Apriori algorithm, K-means.

2.2.3 Reinforcement Learning

Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it trains itself continually using trial and error.

This machine learns from past experience and tries to capture the best possible knowledge to make accurate business decisions. Example of Reinforcement Learning: Markov Decision Process.

2.3 Regression

2.3.1. Linear Regression

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation $Y = a * X + b$.

2.3.2 Logistic Regression

It is a classification not a regression algorithm. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variables(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as logit regression. Since, it predicts the probability, its output values lie between 0 and 1 (as expected).

2.4 Support vector machines

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other. Also

It is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

2.5 Decision Tree

It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this

algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible.

2.6 K-Means

It is a type of unsupervised algorithm which solves the clustering problem. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). Data points inside a cluster are homogeneous and heterogeneous to peer groups.

2.7 KNN

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.

These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If $K = 1$, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing kNN modeling.

2.8 Naive Bayes

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

2.9 LIMITATIONS OF MACHINE LEARNING

Machine learning approaches in particular can suffer from different data biases. A machine learning system trained on your current customers only may not be able to predict the needs of new customer groups that are not represented in the training data. When trained on manmade data, machine learning is likely to pick up the same constitutional and unconscious biases already present in society. Language models learned from data have been shown to contain human-like biases. Machine learning systems used for criminal risk assessment have been found to be biased against black people. In 2015, Google photos would often tag black people as gorillas, and in 2018 this still was not well resolved, but Google reportedly was still using the workaround to remove all gorilla from the training data, and thus was not able to recognize real gorillas at all. Similar issues with recognizing non-white people have been found in many other systems.

2.10 Deep Learning:

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised.

Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design and board game programs, where they have produced results comparable to and in some cases superior to human experts.

Deep learning models are vaguely inspired by information processing and communication patterns in biological nervous systems yet have various differences from the structural and functional properties of biological brains (especially human brain), which make them incompatible with neuroscience evidences.

2.11 Neural networks

2.11.1 Artificial neural networks

Artificial neural networks (ANNs) or connectionist systems are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve their ability) to do tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the analytic results to identify cats in other images. They have found most use in applications difficult to express with a traditional computer algorithm using rule-based programming.

An ANN is based on a collection of connected units called artificial neurons, (analogous to biological neurons in a biological brain). Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream.

Typically, neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times.

The original goal of the neural network approach was to solve problems in the same way that a human brain would. Over time, attention focused on matching specific mental abilities, leading to deviations from biology such as backpropagation, or passing information in the reverse direction and adjusting the network to reflect that information.

Neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

As of 2017, neural networks typically have a few thousand to a few million units and millions of connections. Despite this number being several order of magnitude less than the number of neurons on a human brain, these networks can perform many tasks at a level beyond that of humans (e.g., recognizing faces, playing "Go").

2.12 WEB TECHNOLOGY

2.12.1 Html

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by *tags*, written using angle brackets. Tags such as `` and `<input />` directly introduce content into the page. Other tags such as `<p>` surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

2.12.2 CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

Separation of formatting and content also makes it possible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via

speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.

The name *cascading* comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.

2.12.3 Sass

Sass (Syntactically awesome style sheets) is a style sheet language initially designed by Hampton Catlin and developed by Natalie Weizenbaum. After its initial versions, Weizenbaum and Chris Eppstein have continued to extend Sass with SassScript, a simple scripting language used in Sass files.

Sass is a preprocessor scripting language that is interpreted or compiled into Cascading Style Sheets (CSS). SassScript is the scripting language itself. Sass consists of two syntaxes. The original syntax, called "the indented syntax", uses a syntax similar to Haml. It uses indentation to separate code blocks and newline characters to separate rules. The newer syntax, "SCSS" (Sassy CSS), uses block formatting like that of CSS. It uses braces to denote code blocks and semicolons to separate lines within a block. The indented syntax and SCSS files are traditionally given the extensions `.sass` and `.scss`, respectively.

CSS3 consists of a series of selectors and pseudo-selectors that group rules that apply to them. Sass (in the larger context of both syntaxes) extends CSS by providing several mechanisms available in more traditional programming languages, particularly object-oriented languages, but that are not available to CSS3 itself. When SassScript is interpreted, it creates blocks of CSS rules for various selectors as defined by the Sass file. The Sass interpreter translates SassScript into CSS. Alternatively, Sass can monitor the `.sass` or `.scss` file and translate it to an output `.css` file whenever the `.sass` or `.scss` file is saved.

2.12.4 Flask (Web Framework)

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication

technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program.

Example

The following code shows a simple web application that prints "[Hello World!](#)":

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Fig 1. Flask Example

CHAPTER 3: IMPLEMENTATION OF PROPOSED METHOD

3.1 SOFTWARE REQUIREMENTS:

1. Python 3.0 and above
2. IDE to run Python script (Anaconda etc.)
3. IDE to run and compile Web Languages (Vs Code etc.)
4. Package Manager like NPM
5. Package to be installed in the VsCode IDE
 - a. Npm
 - b. Node-sass
6. Libraries to be installed in the Python IDE:
 - a. Numpy
 - b. Pandas
 - c. LibRosa
 - d. Sci-Kit Learn

3.2 HARDWARE REQUIREMENTS:

1. Processor: Intel Core i3 or above, or similar capability AMD processor.
2. 2Gbs of RAM.
3. 3Gbs of Disk Space.

3.3 INTRODUCTION TO THE LIBRARIES:

1. Numpy Library:

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions

- tools for integrating C/C++ and Fortran
- code useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

2. Pandas Library:

Pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

3. Librosa Library:

LibROSA is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.

• Installation instructions:

pypi

The simplest way to install *librosa* is through the Python Package Index (PyPI). This will ensure that all required dependencies are fulfilled. This can be achieved by executing the following command:

```
pip install librosa
```

or:

```
sudo pip install librosa
```

to install system-wide, or:

```
pip install -u librosa
```

to install just for your own user.

Fig 2. Installation of libROSA

conda

If you use conda/Anaconda environments, librosa can be installed from the *conda-forge* channel:

```
conda install -c conda-forge librosa
```

Source

If you've downloaded the archive manually from the [releases](#) page, you can install using the

`setuptools` script:

```
tar xzf librosa-VERSION.tar.gz
cd librosa-VERSION/
python setup.py install
```

If you intend to develop librosa or make changes to the source code, you can install with *pip install -e* to link to your actively developed source tree:

```
tar xzf librosa-VERSION.tar.gz
cd librosa-VERSION/
pip install -e .
```

fig 3. Installation of libROSA

The librosa package is structured as collection of submodules:

1. librosa.beat

Functions for estimating tempo and detecting beat events

2. librosa.core

Core functionality includes functions to load audio from disk, compute various spectrogram representations, and a variety of commonly used tools for music analysis. For convenience, all functionality in this submodule is directly accessible from the top-level librosa.* namespace.

3. librosa.decompose

Functions for harmonic-percussive source separation (HPSS) and generic spectrogram decomposition using matrix decomposition methods implemented in *scikit-learn*.

4. librosa.display:

Visualization and display routines using matplotlib.

5. librosa.effects:

Time-domain audio processing, such as pitch shifting and time stretching. This submodule also provides time-domain wrappers for the decompose submodule.

6. librosa.feature

Feature extraction and manipulation. This includes low-level feature extraction, such as chromagrams, pseudo-constant-Q (log-frequency) transforms, Mel spectrogram, MFCC, and tuning estimation. Also provided are feature manipulation methods, such as delta features, memory embedding, and event-synchronous feature alignment.

7. librosa.filters

Filter-bank generation (chroma, pseudo-CQT, CQT, etc.). These are primarily internal functions used by other parts of librosa.

8. librosa.onset

Onset detection and onset strength computation.

9. librosa.output

Text- and wav-file output.

10. librosa.segment

Functions useful for structural segmentation, such as recurrence matrix construction, time lag representation, and sequentially constrained clustering.

11. librosa.sequence

Functions for sequential modeling. Various forms of Viterbi decoding, and helper functions for constructing transition matrices.

12. librosa.util

Helper utilities (normalization, padding, centering, etc.)

librosa.feature.mfcc:

```
librosa.feature.mfcc(y=None, sr=22050, S=None, n_mfcc=20, dct_type=2, norm='ortho', **kwargs)  
\[source\]
```

Mel-frequency cepstral coefficients (MFCCs)

Parameters:

- `y`: np.ndarray [shape=(n,)] or None
audio time series
- `sr`: number > 0 [scalar]
sampling rate of `y`
- `S`: np.ndarray [shape=(d, t)] or None
log-power Mel spectrogram
- `n_mfcc`: int > 0 [scalar]
number of MFCCs to return
- `dct_type`: None, or {1, 2, 3}
Discrete cosine transform (DCT) type. By default, DCT type-2 is used.
- `norm`: None or 'ortho'
If `dct_type` is 2 or 3, setting `norm='ortho'` uses an ortho-normal DCT basis.
Normalization is not supported for `dct_type=1`.
- `kwargs`: additional keyword arguments
Arguments to `melspectrogram`, if operating on time series input

Fig 4. MFCC Parameters

Examples

Generate mfccs from a time series

```
>>> y, sr = librosa.load(librosa.util.example_audio_file(), offset=30, duration=5)
>>> librosa.feature.mfcc(y=y, sr=sr)
array([[ -5.229e+02,  -4.944e+02, ...,  -5.229e+02,  -5.229e+02],
       [  7.105e-15,   3.787e+01, ...,  -7.105e-15,  -7.105e-15],
       ...,
       [  1.066e-14,  -7.500e+00, ...,   1.421e-14,   1.421e-14],
       [  3.109e-14,  -5.058e+00, ...,   2.931e-14,   2.931e-14]])
```

Use a pre-computed log-power Mel spectrogram

```
>>> S = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128,
...                                   fmax=8000)
>>> librosa.feature.mfcc(S=librosa.power_to_db(S))
array([[ -5.207e+02,  -4.898e+02, ...,  -5.207e+02,  -5.207e+02],
       [ -2.576e-14,   4.054e+01, ...,  -3.997e-14,  -3.997e-14],
       ...,
       [  7.105e-15,  -3.534e+00, ...,   0.000e+00,   0.000e+00],
       [  3.020e-14,  -2.613e+00, ...,   3.553e-14,   3.553e-14]])
```

Get more components

```
>>> mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
```

Fig 5. MFCC Example

4. Sci-Kit Learn Library:

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

- **NumPy**: Base n-dimensional array package
- **SciPy**: Fundamental library for scientific computing
- **Matplotlib**: Comprehensive 2D/3D plotting
- **IPython**: Enhanced interactive console
- **Sympy**: Symbolic mathematics
- **Pandas**: Data structures and analysis

5. NodeJs

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. JavaScript is used primarily

for client-side scripting, in which scripts written in JavaScript are embedded in a webpage's HTML and run client-side by a JavaScript engine in the user's web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content *before* the page is sent to the user's web browser.

3.4 DATASET

Marsyas(MUSIC ANALYSIS,RETRIEVAL AND SYNTHESIS FOR AUDIO SIGNALS)

It is a software framework for rapid prototyping and experimentation with audio analysis and synthesis with specific emphasis to music signals and Music Information Retrieval.

The basic goal is to provide a general, extensible and flexible architecture that allows easy experimentation with algorithms and provides fast performance that is useful in developing real time audio analysis and synthesis tools.

A variety of existing building blocks that form the basis of most published algorithms in Computer Audition are already available as part of the framework and extending the framework with new components/building blocks is straightforward.

GTZAN Genre Collection

This dataset was used for the well-known paper in genre classification " Musical genre classification of audio signals " by G. Tzanetakis and P. Cook in IEEE Transactions on Audio and Speech Processing 2002.

Unfortunately, the database was collected gradually and very early on in our research, so we have no titles (and obviously no copyright permission etc.). The files were collected in 2000-2001 from a variety of sources including personal CDs, radio, microphone recordings, in order to represent a variety of recording conditions.

The dataset consists of 1000 audio tracks each 30 seconds long. It contains 10 genres, each represented by 100 tracks. The tracks are all 22050Hz Mono 16-bit audio files in .wav format.

3.5 MEL-FREQUENCY CEPSTRUM COEFFICIENTS

The first step in any automatic speech recognition system is to extract features i.e. identify the components of the audio signal that are good for identifying the linguistic content and discarding all the other stuff which carries information like background noise, emotion etc.

The main point to understand about speech is that the sounds generated by a human are filtered by the shape of the vocal tract including tongue, teeth etc. This shape determines what sound comes out. If we can determine the shape accurately, this should give us an accurate representation of the phoneme being produced. The shape of the vocal tract manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope.

Mel Frequency Cepstral Coefficients (MFCCs) are a feature widely used in automatic speech and speaker recognition. They were introduced by Davis and Mermelstein in the 1980's and have been state-of-the-art ever since. Prior to the introduction of MFCCs, Linear Prediction Coefficients (LPCs) and Linear Prediction Cepstral Coefficients (LPCCs) and were the main feature type for automatic speech recognition (ASR), especially with HMM classifiers.

Steps at a Glance:

1. Frame the signal into short frames.
2. For each frame calculate the periodogram estimate of the power spectrum.
3. Apply the melfilterbank to the power spectra, sum the energy in each filter.
4. Take the logarithm of all filterbank energies.
5. Take the DCT of the log filterbank energies.
6. Keep DCT coefficients 2-13, discard the rest.

There are a few more things commonly done, sometimes the frame energy is appended to each feature vector. Delta and Delta-Delta features are usually also appended. Liftering is also commonly applied to the final features.

An audio signal is constantly changing, so to simplify things we assume that on short time scales the audio signal doesn't change much (when we say it doesn't change, we mean statistically i.e. statistically stationary, obviously the samples are constantly changing on even short time scales). Therefore, we frame the signal into 20-40ms frames. If the frame is much shorter, we don't have enough samples to get a reliable spectral estimate, if it is longer the signal changes too much throughout the frame.

The next step is to calculate the power spectrum of each frame. This is motivated by the human cochlea (an organ in the ear) which vibrates at different spots depending on the frequency of the incoming sounds. Depending on the location in the cochlea that vibrates (which wobbles small hairs), different nerves fire informing the brain that certain frequencies

are present. Our periodogram estimate performs a similar job for us, identifying which frequencies are present in the frame.

The periodogram spectral estimate still contains a lot of information not required for Automatic Speech Recognition (ASR). In particular the cochlea cannot discern the difference between two closely spaced frequencies. This effect becomes more pronounced as the frequencies increase. For this reason, we take clumps of periodogram bins and sum them up to get an idea of how much energy exists in various frequency regions. This is performed by our Mel filterbank: the first filter is very narrow and gives an indication of how much energy exists near 0 Hertz. As the frequencies get higher our filters get wider as we become less concerned about variations. We are only interested in roughly how much energy occurs at each spot. The Mel scale tells us exactly how to space our filterbanks and how wide to make them.

Once we have the filterbank energies, we take the logarithm of them. This is also motivated by human hearing: we don't hear loudness on a linear scale. Generally, to double the perceived volume of a sound we need to put 8 times as much energy into it. This means that large variations in energy may not sound all that different if the sound is loud to begin with. This compression operation makes our features match more closely what humans actually hear. Why the logarithm and not a cube root? The logarithm allows us to use cepstral mean subtraction, which is a channel normalisation technique.

The final step is to compute the DCT of the log filterbank energies. There are 2 main reasons this is performed. Because our filterbanks are all overlapping, the filterbank energies are quite correlated with each other. The DCT decorrelates the energies which means diagonal covariance matrices can be used to model the features in e.g. a HMM classifier. But notice that only 12 of the 26 DCT coefficients are kept. This is because the higher DCT coefficients represent fast changes in the filterbank energies and it turns out that these fast changes actually degrade ASR performance, so we get a small improvement by dropping them.

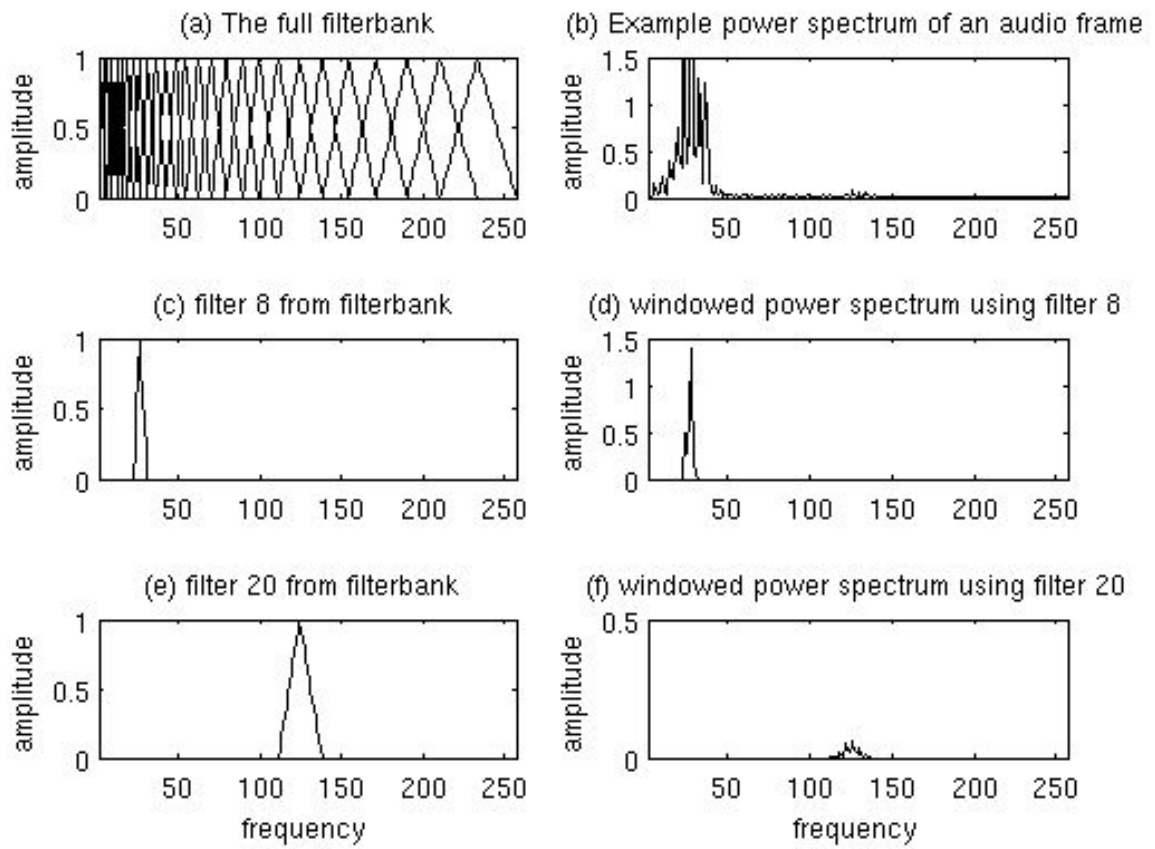


Fig 6. Filterbanks in MFCC

3.6 FLOWCHARTS

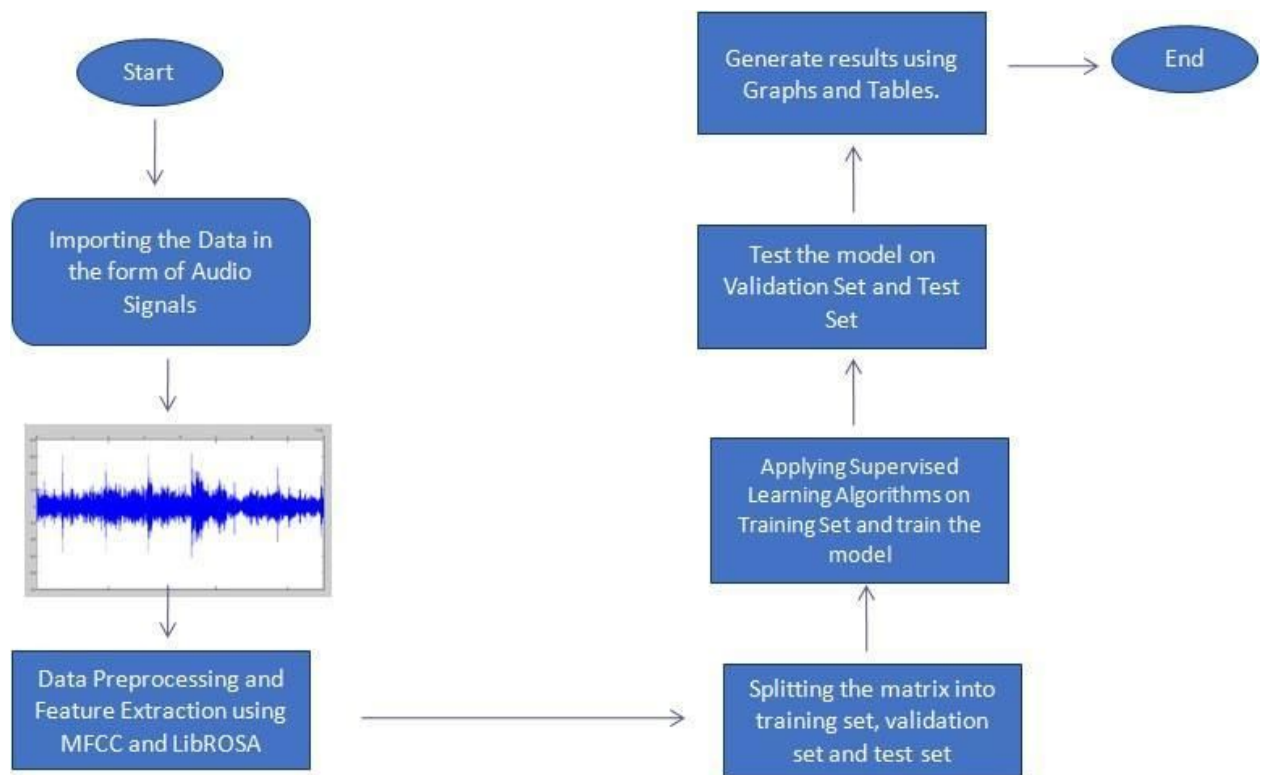


Fig 7. Flowchart 1

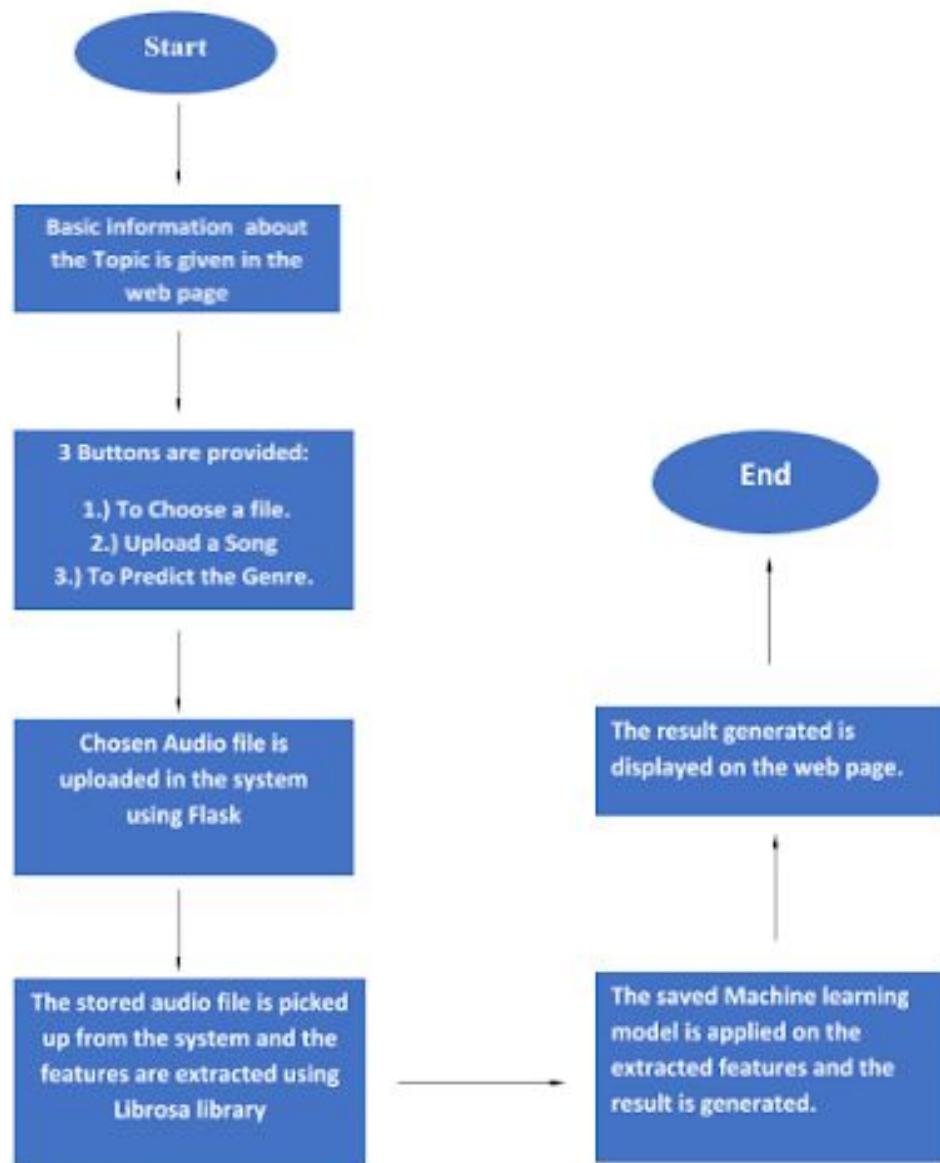


Fig 8. Flowchart 2

3.7 ALGORITHMS

- **K-Nearest Neighbor**

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.

In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally, and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

Both for classification and regression, a useful technique can be used to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of $1/d$, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

We can implement a KNN model by following the below steps:

- Load the data
- Initialize the value of k
- For getting the predicted class, iterate from 1 to total number of training data points
- Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the

most popular method. The other metrics that can be used are Chebyshev, cosine, etc.

- Sort the calculated distances in ascending order based on distance values
- Get top k rows from the sorted array
- Get the most frequent class of these rows • Return the predicted class

K-NN Example

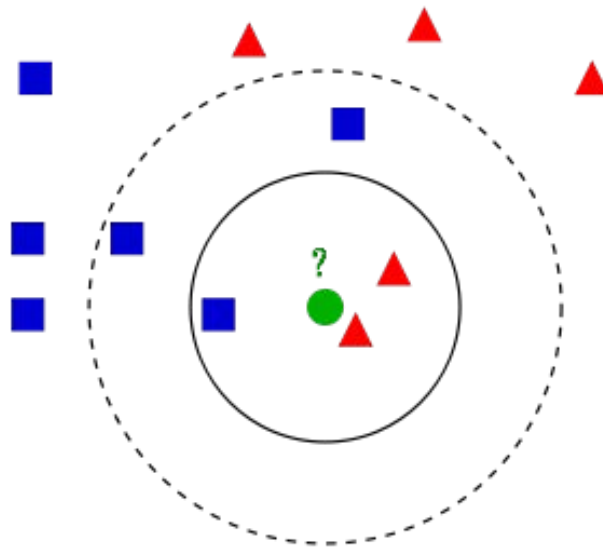


Fig 9. k-NN

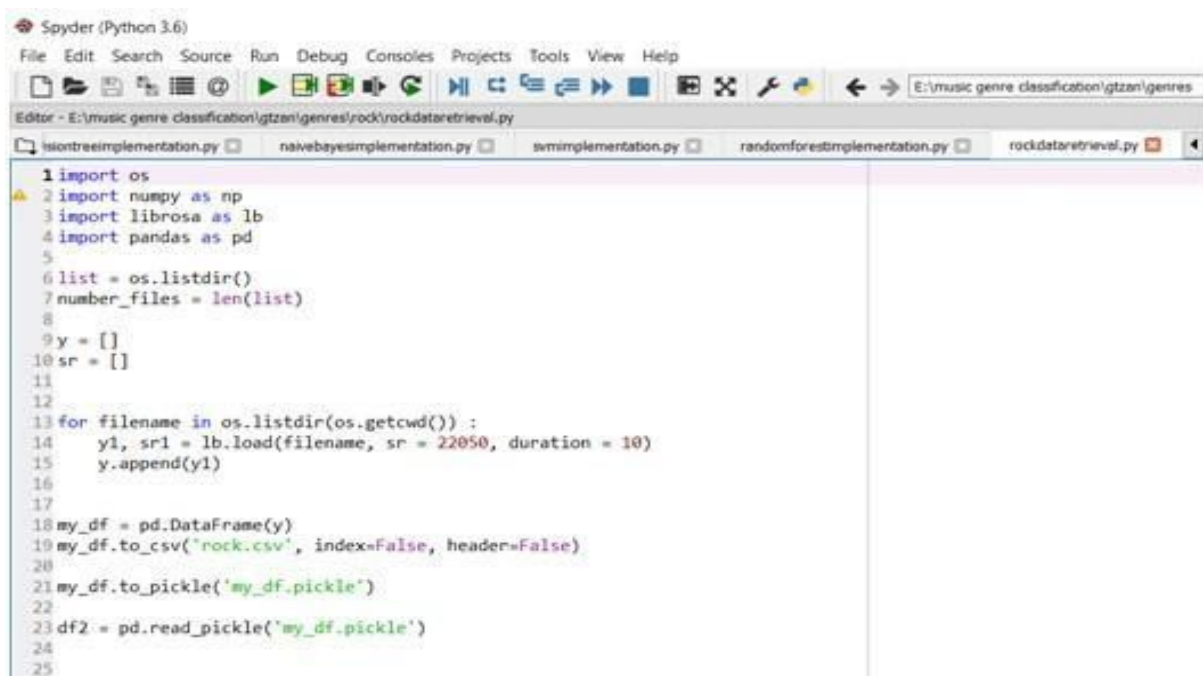
Example of k-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If $k = 3$ (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If $k = 5$ (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

3.8 Implementations and Screenshots:

1. Using librosa library we load the songs which are .au files onto a dataframe using the above code. Then this dataframe is stored as a pickle in our computer systems so that we do not have to run this code every time we want to use this data.

In this dataset there are 100 songs for each genre. So, there are 100 songs each of Rock, Disco, Blues and Jazz genres.

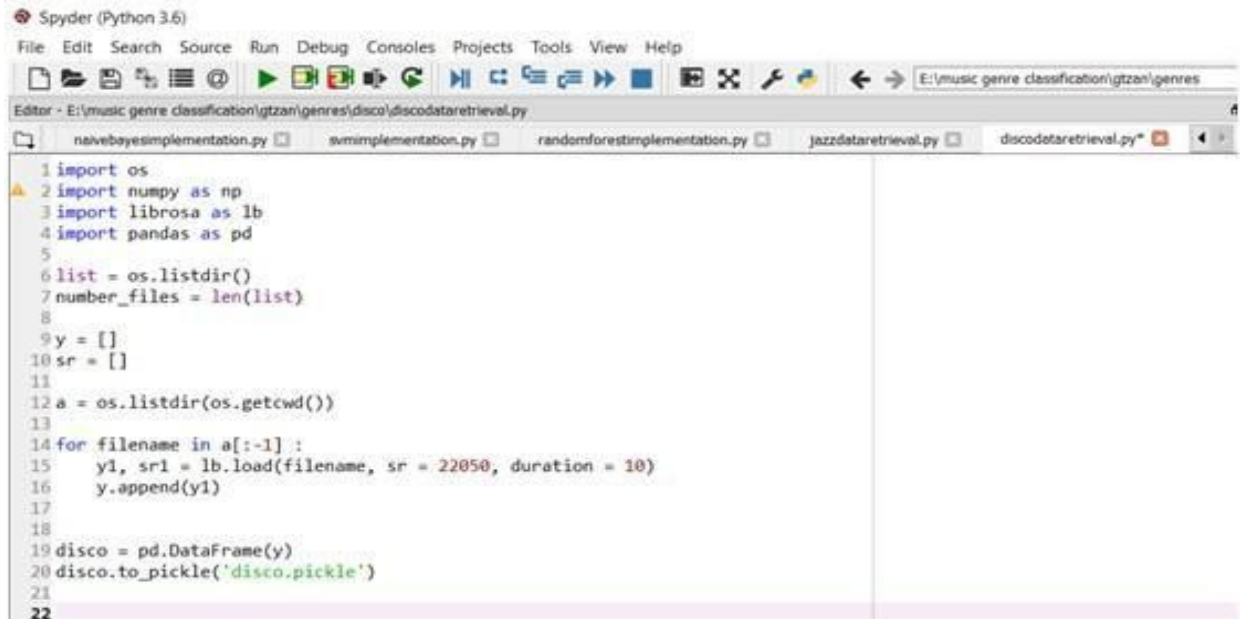
We load the audio files with duration of 10 seconds and the sampling rate is kept at default which is 22050. Sampling rate of 22050 means that for each second there will be 22050 data points. So, for 10 seconds in total there will be 220500 data points. And we load 100 songs of each genre. The screenshots given are codes for retrieval of data for each genre.

The image is a screenshot of the Spyder Python IDE interface. The title bar at the top reads 'Spyder (Python 3.6)'. Below it is a menu bar with options: File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. A toolbar with various icons is positioned below the menu bar. The main editor window displays a Python script with the following code:

```
1 import os
2 import numpy as np
3 import librosa as lb
4 import pandas as pd
5
6 list = os.listdir()
7 number_files = len(list)
8
9 y = []
10 sr = []
11
12
13 for filename in os.listdir(os.getcwd()) :
14     y1, sr1 = lb.load(filename, sr = 22050, duration = 10)
15     y.append(y1)
16
17
18 my_df = pd.DataFrame(y)
19 my_df.to_csv('rock.csv', index=False, header=False)
20
21 my_df.to_pickle('my_df.pickle')
22
23 df2 = pd.read_pickle('my_df.pickle')
24
25
```

The file explorer on the left shows several files: 'isiontreeimplementation.py', 'naivebayesimplementation.py', 'svmimplementation.py', 'randomforestimplementation.py', and 'rockdataretrieval.py', which is currently selected.

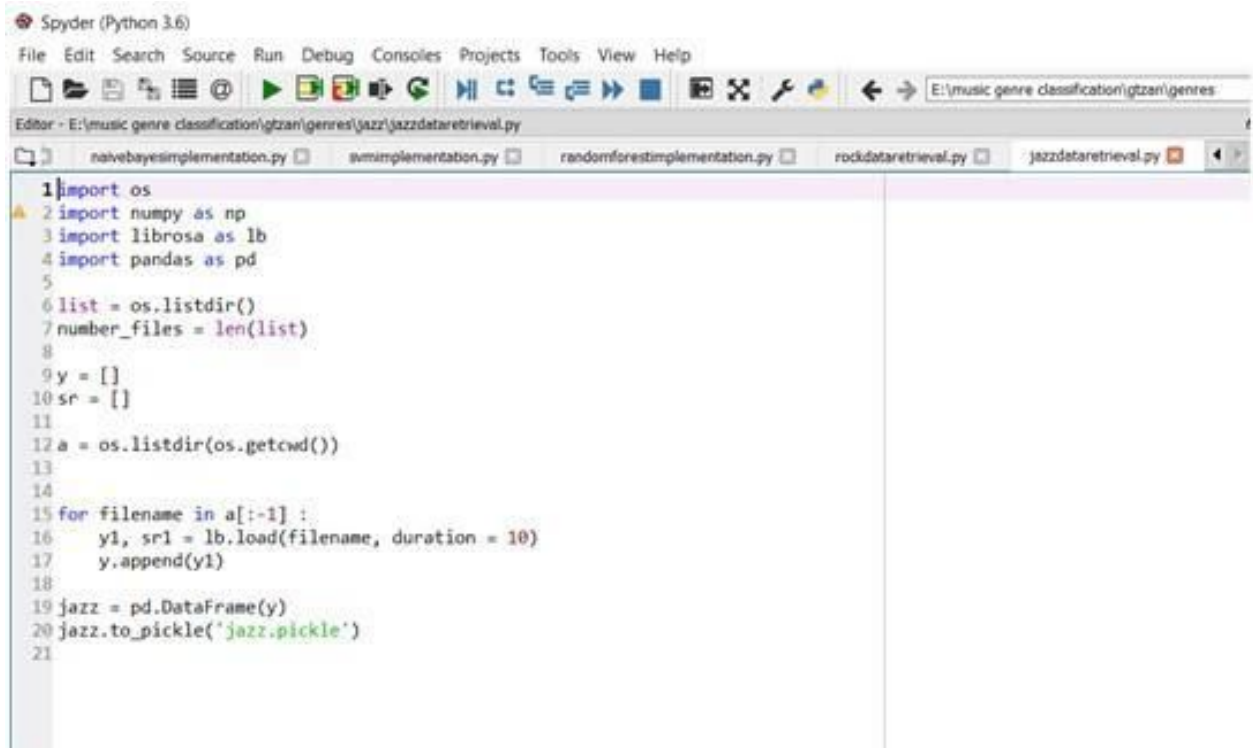
Fig 10. Data Retrieval 1



The screenshot shows the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains icons for file operations, running, and debugging. The editor window displays the file `E:\music genre classification\gtzan\genres\disco\disco\dataretrieval.py`. The code in the editor is as follows:

```
1 import os
2 import numpy as np
3 import librosa as lb
4 import pandas as pd
5
6 list = os.listdir()
7 number_files = len(list)
8
9 y = []
10 sr = []
11
12 a = os.listdir(os.getcwd())
13
14 for filename in a[:-1]:
15     y1, sr1 = lb.load(filename, sr = 22050, duration = 10)
16     y.append(y1)
17
18 disco = pd.DataFrame(y)
19 disco.to_pickle('disco.pickle')
20
21
22
```

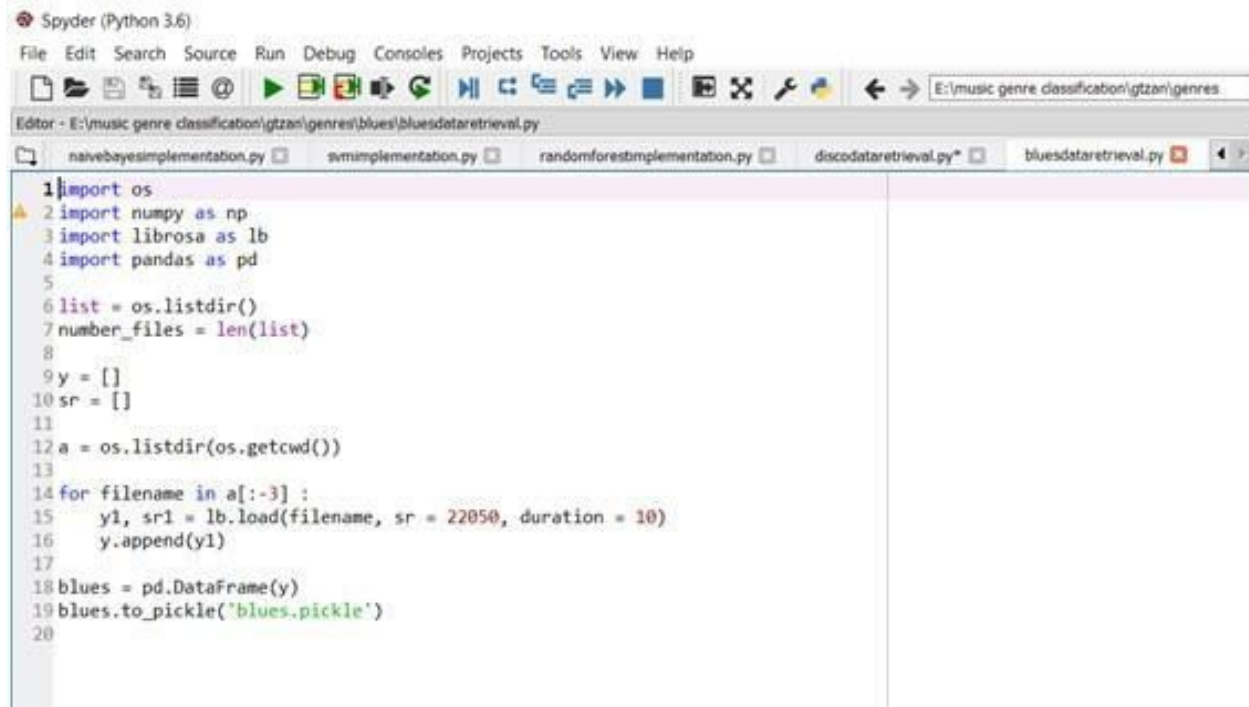
Fig 11. Data Retrieval 2



The screenshot shows the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains icons for file operations, running, and debugging. The editor window displays the file `E:\music genre classification\gtzan\genres\jazz\jazz\dataretrieval.py`. The code in the editor is as follows:

```
1 import os
2 import numpy as np
3 import librosa as lb
4 import pandas as pd
5
6 list = os.listdir()
7 number_files = len(list)
8
9 y = []
10 sr = []
11
12 a = os.listdir(os.getcwd())
13
14
15 for filename in a[:-1]:
16     y1, sr1 = lb.load(filename, duration = 10)
17     y.append(y1)
18
19 jazz = pd.DataFrame(y)
20 jazz.to_pickle('jazz.pickle')
21
```

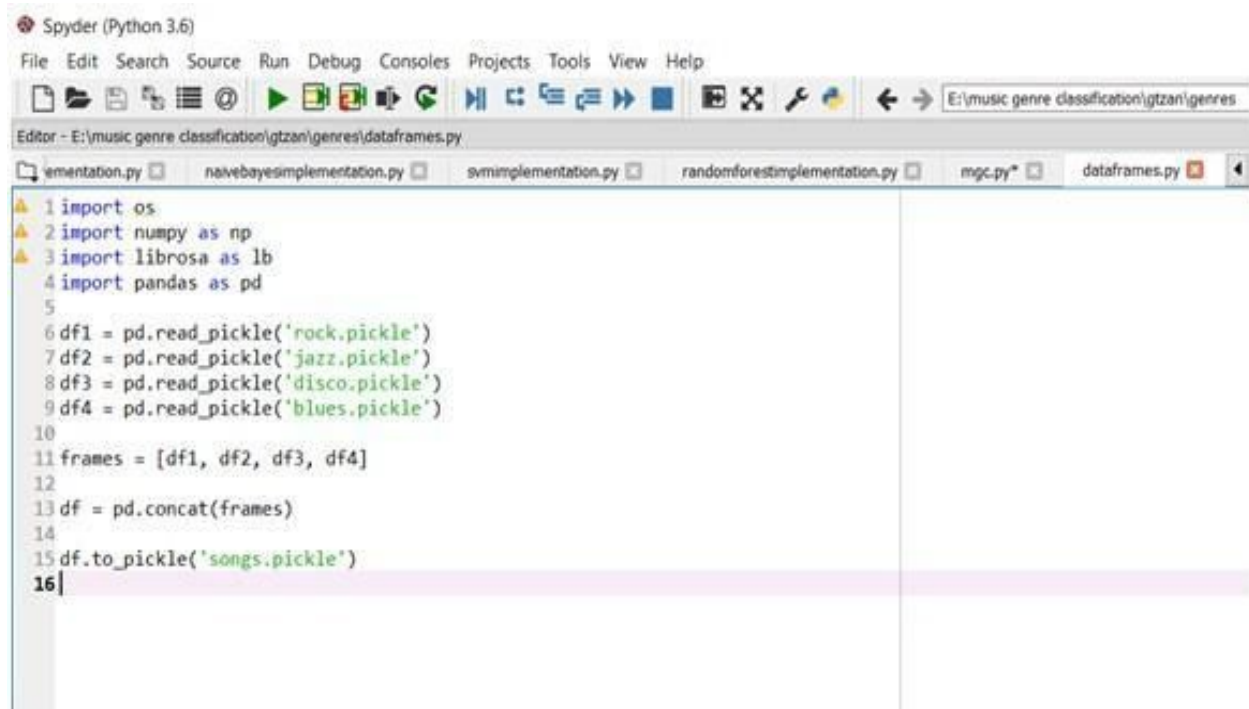
Fig 12. Data Retrieval 3

The image shows the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains icons for file operations, running code, and debugging. The editor window displays the file bluesdataretrieval.py with the following Python code:

```
1 import os
2 import numpy as np
3 import librosa as lb
4 import pandas as pd
5
6 list = os.listdir()
7 number_files = len(list)
8
9 y = []
10 sr = []
11
12 a = os.listdir(os.getcwd())
13
14 for filename in a[:-3] :
15     y1, sr1 = lb.load(filename, sr = 22050, duration = 10)
16     y.append(y1)
17
18 blues = pd.DataFrame(y)
19 blues.to_pickle('blues.pickle')
20
```

Fig 13. Data Retrieval 4

2. After collecting all the data of each genre, the next task was to combine all the data into a single dataframe and further store it into the system as a pickle. The above task was done through the following code.

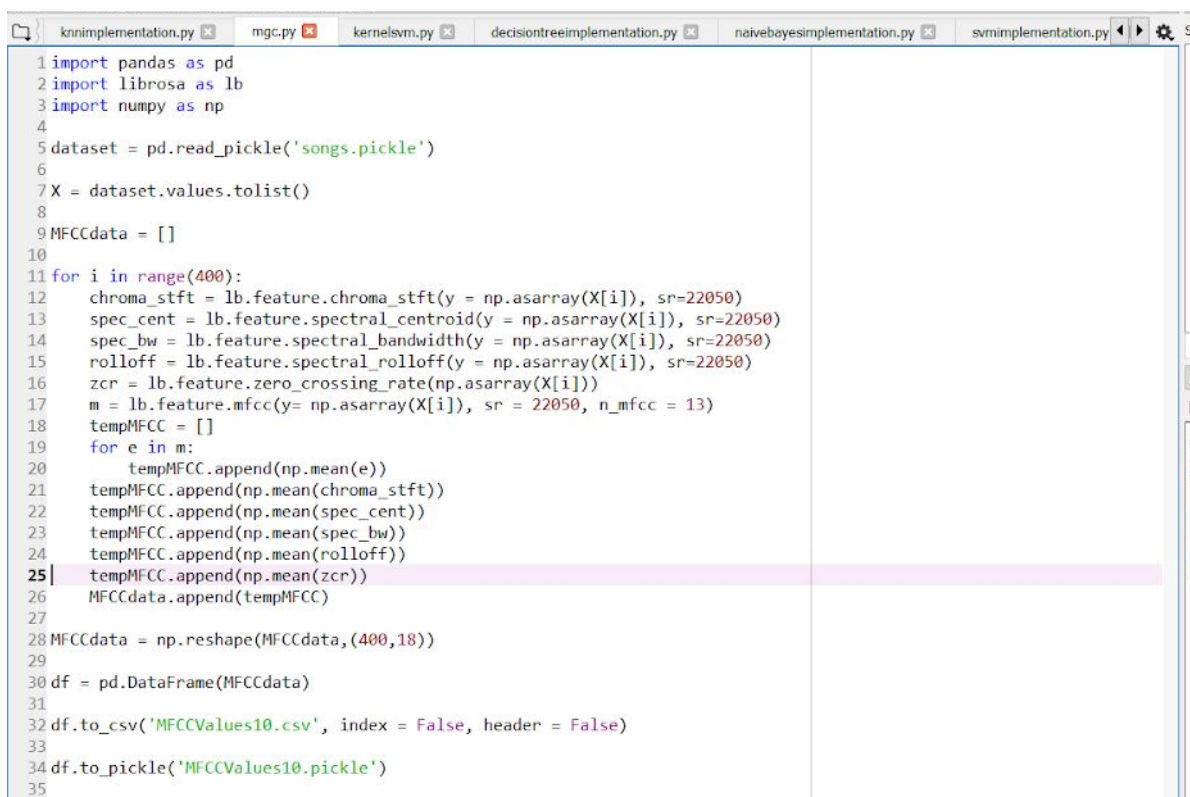
The image shows the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains icons for file operations, running code, and debugging. The editor window displays the file dataframes.py with the following Python code:

```
1 import os
2 import numpy as np
3 import librosa as lb
4 import pandas as pd
5
6 df1 = pd.read_pickle('rock.pickle')
7 df2 = pd.read_pickle('jazz.pickle')
8 df3 = pd.read_pickle('disco.pickle')
9 df4 = pd.read_pickle('blues.pickle')
10
11 frames = [df1, df2, df3, df4]
12
13 df = pd.concat(frames)
14
15 df.to_pickle('songs.pickle')
16
```

Fig 14. Data Frame

3. The next task is to collect the features from the songs. The features we use are MFCC features, Zero-Crossing Rate, Chroma stft, Spectral Bandwidth, Spectral Centroid and Spectral Rolloff. These features were extracted using the LibRosa Library's function of finding the features. The parameters of the function for MFCC extraction are
 1. The array on which the features must be found.
 2. Sampling Rate which is kept at 22050.
 3. Number of MFCC Features to be found. We have kept this number at 13.

Mean values of each feature are found out and are appended into the Feature array. The dimensions of the feature array is 400x18 where there are 13 MFCC features and 5 other features. This file was then stored as a dataframe in a .pkl file for further use.



```

1 import pandas as pd
2 import librosa as lb
3 import numpy as np
4
5 dataset = pd.read_pickle('songs.pickle')
6
7 X = dataset.values.tolist()
8
9 MFCCdata = []
10
11 for i in range(400):
12     chroma_stft = lb.feature.chroma_stft(y = np.asarray(X[i]), sr=22050)
13     spec_cent = lb.feature.spectral_centroid(y = np.asarray(X[i]), sr=22050)
14     spec_bw = lb.feature.spectral_bandwidth(y = np.asarray(X[i]), sr=22050)
15     rolloff = lb.feature.spectral_rolloff(y = np.asarray(X[i]), sr=22050)
16     zcr = lb.feature.zero_crossing_rate(np.asarray(X[i]))
17     m = lb.feature.mfcc(y= np.asarray(X[i]), sr = 22050, n_mfcc = 13)
18     tempMFCC = []
19     for e in m:
20         tempMFCC.append(np.mean(e))
21     tempMFCC.append(np.mean(chroma_stft))
22     tempMFCC.append(np.mean(spec_cent))
23     tempMFCC.append(np.mean(spec_bw))
24     tempMFCC.append(np.mean(rolloff))
25     tempMFCC.append(np.mean(zcr))
26     MFCCdata.append(tempMFCC)
27
28 MFCCdata = np.reshape(MFCCdata,(400,18))
29
30 df = pd.DataFrame(MFCCdata)
31
32 df.to_csv('MFCCValues10.csv', index = False, header = False)
33
34 df.to_pickle('MFCCValues10.pickle')
35

```

Fig 15. Feature Extraction

4. Now we apply the Machine Learning Algorithms. The following steps are followed for each algorithm.

- a. Importing the libraries and the datasets. The dataset is of size 400x19 where the 19th column is of the output or the genre column which can be one of the 4 genres. These genres are in String format.
- b. Since the format of the output is in String format, we have to encode it into numbers which is done by LabelEncoder Class.
- c. Then the dataset is split into Training Set and Test Set. We use `test_train_split` class in the `model_selection` library. We split the training set and the test set in the ratio of 8:2. Also, for balanced splitting we use `stratify`.
- d. We then scale the features for effective application of algorithm.
- e. The next step is to apply the algorithm. We use K-Nearest Neighbours algorithm.

- f. The final step is to save this machine learning model for further use in the website. We use the `joblib` class to dump the model into a `.pkl` file.

We apply is the K-Nearest Neighbors. We use the Sci-Kit library for the application of the algorithm.

The code used is given in the screenshot.

```

1 import pandas as pd
2 import librosa as lb
3 import numpy as np
4 from sklearn.externals import joblib
5
6 dataset = pd.read_csv('MFCCValues10.csv')
7
8 X = dataset.iloc[:, 1:18].values
9 Y = dataset.iloc[:, 18].values
10
11 from sklearn.preprocessing import LabelEncoder, OneHotEncoder
12 # Encoding the Dependent Variable
13 labelencoder_y = LabelEncoder()
14 Y = labelencoder_y.fit_transform(Y)
15
16 from sklearn.model_selection import train_test_split
17 X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42, stratify = Y)
18
19 from sklearn.preprocessing import StandardScaler
20 sc_X = StandardScaler()
21 X_train = sc_X.fit_transform(X_train)
22 X_test = sc_X.transform(X_test)
23
24 from sklearn.neighbors import KNeighborsClassifier
25 classifier = KNeighborsClassifier(n_neighbors = 6, metric = 'minkowski', p = 2)
26 classifier.fit(X_train, y_train)
27
28 # Predicting the Test set results
29 y_pred = classifier.predict(X_test)
30
31 xone = np.reshape(X_test[12], (1,-1))
32 yone = classifier.predict(xone)
33
34 joblib.dump(classifier, 'knnjoblib.pkl')
35
36 # Making the Confusion Matrix
37 from sklearn.metrics import confusion_matrix
38 cm = confusion_matrix(y_test, y_pred)

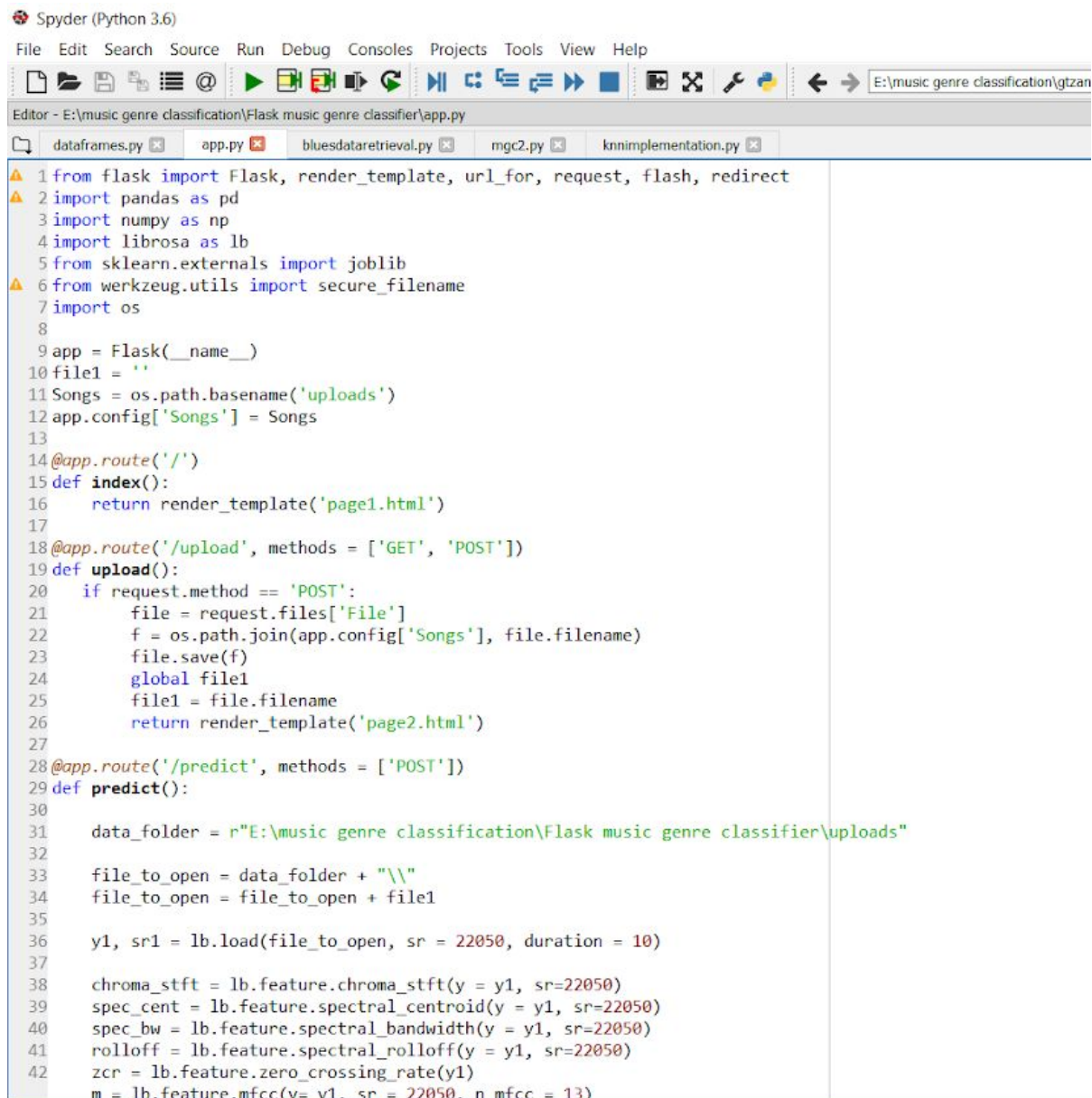
```

Fig 16. k-NN Implementation

5. The code in the given figures below runs the actual website. The code uses the Flask library to help connect the python code with the HTML/CSS code. This code does the actual work behind the website. Three functions have been defined within the Flask application.
 1. The first one is the index() function. It basically renders the first page of the website whenever the app is opened.
 2. The second function is the upload() function. This function performs the task of reading the audio file uploaded in the website and stores it in the system. When the audio file is stored it renders the a page with the acknowledgement that the file has been uploaded and stored in the system successfully.
 3. The third function is the predict() function. This function takes the uploaded audio file and performs the task of data retrieval from the audio file. This is done using the librosa library in python. Various features of the song are extracted that include:

a. MFCC	b. Chromagram of the waveform	c. Spectral Centroid
d. Spectral Bandwidth	e. Rolloff	f. Zero Crossing Rate

These features are used to predict the genre of the song. The trained machine learning model that was saved using Joblib as a pickle file is used now. This file is then loaded in the application and is then used to predict the genre. When the genre is predicted the result is displayed by rendering the result.html page which displays the result accordingly.



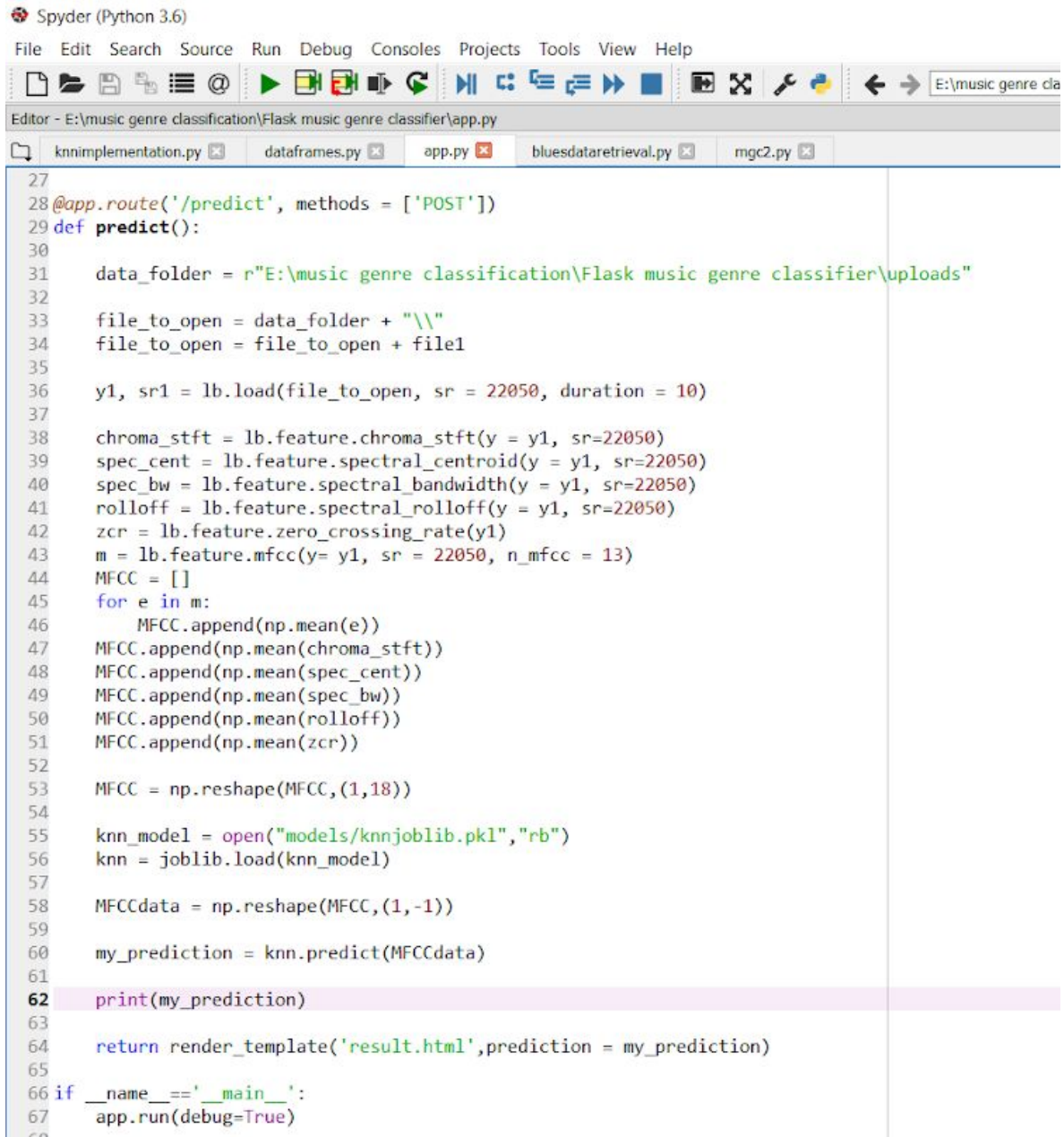
```

Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
E:\music genre classification\gtzan

dataframes.py app.py bluesdataretrieval.py mgc2.py knnimplementation.py
1 from flask import Flask, render_template, url_for, request, flash, redirect
2 import pandas as pd
3 import numpy as np
4 import librosa as lb
5 from sklearn.externals import joblib
6 from werkzeug.utils import secure_filename
7 import os
8
9 app = Flask(__name__)
10 file1 = ''
11 Songs = os.path.basename('uploads')
12 app.config['Songs'] = Songs
13
14 @app.route('/')
15 def index():
16     return render_template('page1.html')
17
18 @app.route('/upload', methods = ['GET', 'POST'])
19 def upload():
20     if request.method == 'POST':
21         file = request.files['File']
22         f = os.path.join(app.config['Songs'], file.filename)
23         file.save(f)
24         global file1
25         file1 = file.filename
26         return render_template('page2.html')
27
28 @app.route('/predict', methods = ['POST'])
29 def predict():
30
31     data_folder = r"E:\music genre classification\Flask music genre classifier\uploads"
32
33     file_to_open = data_folder + "\\\"
34     file_to_open = file_to_open + file1
35
36     y1, sr1 = lb.load(file_to_open, sr = 22050, duration = 10)
37
38     chroma_stft = lb.feature.chroma_stft(y = y1, sr=22050)
39     spec_cent = lb.feature.spectral_centroid(y = y1, sr=22050)
40     spec_bw = lb.feature.spectral_bandwidth(y = y1, sr=22050)
41     rolloff = lb.feature.spectral_rolloff(y = y1, sr=22050)
42     zcr = lb.feature.zero_crossing_rate(y1)
43     m = lb.feature.mfcc(y=y1, sr = 22050, n_mfcc = 13)

```

Fig 17. Flask Implementation



The image shows the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains icons for file operations, running, and debugging. The editor window displays the code for `app.py` in the Flask music genre classifier. The code defines a `predict` function that loads an audio file, extracts features using Librosa, calculates MFCCs, and uses a K-Nearest Neighbors (KNN) model for prediction. The file path is set to `E:\music genre classification\Flask music genre classifier\uploads`. The code is as follows:

```
27
28 @app.route('/predict', methods = ['POST'])
29 def predict():
30
31     data_folder = r"E:\music genre classification\Flask music genre classifier\uploads"
32
33     file_to_open = data_folder + "\\"
34     file_to_open = file_to_open + file1
35
36     y1, sr1 = lb.load(file_to_open, sr = 22050, duration = 10)
37
38     chroma_stft = lb.feature.chroma_stft(y = y1, sr=22050)
39     spec_cent = lb.feature.spectral_centroid(y = y1, sr=22050)
40     spec_bw = lb.feature.spectral_bandwidth(y = y1, sr=22050)
41     rolloff = lb.feature.spectral_rolloff(y = y1, sr=22050)
42     zcr = lb.feature.zero_crossing_rate(y1)
43     m = lb.feature.mfcc(y= y1, sr = 22050, n_mfcc = 13)
44     MFCC = []
45     for e in m:
46         MFCC.append(np.mean(e))
47     MFCC.append(np.mean(chroma_stft))
48     MFCC.append(np.mean(spec_cent))
49     MFCC.append(np.mean(spec_bw))
50     MFCC.append(np.mean(rolloff))
51     MFCC.append(np.mean(zcr))
52
53     MFCC = np.reshape(MFCC,(1,18))
54
55     knn_model = open("models/knnjoblib.pkl","rb")
56     knn = joblib.load(knn_model)
57
58     MFCCdata = np.reshape(MFCC,(1,-1))
59
60     my_prediction = knn.predict(MFCCdata)
61
62     print(my_prediction)
63
64     return render_template('result.html',prediction = my_prediction)
65
66 if __name__=='__main__':
67     app.run(debug=True)
68
```

Fig 18. Flask Implementation

HTML IMPLEMENTATION

```
<!DOCTYPE html>
<html lang="en">

<head>
  <link rel="stylesheet" href="C:\Users\abhay\Desktop\Genre\css\styled.css">
</head>

<body>
  <header class="Header">
    <div class="text-box">
      <h1 class="heading-primary">
        <span class="heading-primary-main">Genre<br>Classifier</span>
      </h1>
      <a href="#" class="btn btn-white btn-animated">Dive In</a>
    </div>
  </header>
  <main>
    <section class="section-about">
      <div class="u-center-text u-margin-bottom-big">
        <h2 class="heading-secondary">
          Explore Music
        </h2>
      </div>

      <div class="row">
        <div class="col-1-of-2">
          <h3 class="heading-tertiary u-margin-bottom-small">A Way of Expression</h3>
          <p class="paragraph">Music gives people a way to express who they are inside through many different
forms.Many people today hide who they are on the inside to try to fit in with everyone else because some people are afraid of
being rejected by those around them.Music is important because it gives people a way to express themselves and be who they are
on the inside.</p>
          <h3 class="heading-tertiary u-margin-bottom-small">Music is everything</h3>
          <p class="paragraph">Music is everywhere in our world and music relates to everything as well. Music is in all
of our histories starting from the beginning. Without music we would not have anything, life would be boring and dull.</p>

          <a href="#" class="btn-text">Learn More &arr;</a>
        </div>
        <div class="col-1-of-2">
          <div class="composition">
            
            
            
          </div>
        </div>
      </div>
    </section>

    <section class="section-book">
      <div class="row">
        <div class="book">
          <div class="book__form">
            <form method="POST" action="{{url_for('predict')}}">
              <div class="u-margin-bottom-small">
                <h2 class="heading-secondary">
                  Predict Genre
                </h2>
              </div>
              <div>
                <input type="file" name="NameQuery" class="form__input" id="name" required>
              </div>
              <div>
                <button class="Buttonna-1">Upload</button>
                <button class="Buttonna-2" type="submit" >Predict</button>
              </div>
            </form>
          </div>
        </div>
      </div>
    </section>
  </main>
</body>
```

```

</html>

<html>
<head>
<link rel="stylesheet" href="C:\Users\abhay\Desktop\Genre\css\styled.css">
</head>

<div class="container">
<header class="Header">
<div class="text-box">
<h1 class="heading-primary-1">
<span class="heading-primary-main-1">Results</span>
</h1>
</div>
</header>
</div>
<div class="u-center-text u-margin-bottom-big">
<h2 class="heading-secondary">
Using Machine Learning to Classify Genres of Music
</h2>
</div>
<section>

<div class="card text-center">
<div class="card-header">
Results
</div>
<div class="card-body">
<h5 class="card-title">Genre Classifier</h5>
<p class="card-text"><div class="alert alert-danger" role="alert">
{{ name }}
</div></p>

<!-- Main ML Prediction Display Here -->
.
{% if prediction == 0 %}
<p class="card-text"><div class="alert alert-danger" role="alert">
Blues
</div></p>
{% elif prediction == 1 %}
<p class="card-text"><div class="alert alert-danger" role="alert">
Disco
</div></p>

{% elif prediction == 2 %}
<p class="card-text"><div class="alert alert-danger" role="alert">
Jazz
</div></p>

{% elif prediction == 3 %}
<p class="card-text"><div class="alert alert-danger" role="alert">
Rock
</div></p>

{% endif%}
</div>

<!-- Main ML Prediction Display Ends Here -->
<div class="card-footer text-muted">

```

```

</div>
</div>
</section>
</html>

```

CSS IMPLEMENTATION

```

*,
*::after,
*::before {
  margin: 0;
  padding: 0;
  box-sizing: inherit; }

html {
  font-size: 62.5%; }

body {
  box-sizing: border-box; }

@keyframes moveInLeft {
  0% {
    opacity: 0;
    transform: translateX(-100px); }
  80% {
    transform: translateX(20px); }
  100% {
    opacity: 1;
    transform: translate(0); } }

@keyframes moveInRight {
  0% {
    opacity: 0;
    transform: translateX(100px); }
  80% {
    transform: translateX(-20px); }
  100% {
    opacity: 1;
    transform: translate(0); } }

@keyframes moveInBottom {
  0% {
    opacity: 0;
    transform: translateY(30px); }
  100% {
    opacity: 1;
    transform: translate(0); } }

body {
  font-family: 'Courier New', Courier, monospace;
  font-size: 20px;
  font-weight: 550;
  line-height: 1.8;
  color: grey;
  padding: 30px; }

.heading-primary {
  color: whitesmoke;
  text-transform: uppercase; }

.heading-primary-main {
  display: block;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  font-size: 40px;
  letter-spacing: 30px;
  font-weight: 400;

```

```

animation-name: moveInLeft;
animation-duration: 2s;
animation-delay: 0.5s; }

.heading-secondary {
  display: inline-block;
  text-transform: uppercase;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  font-size: 4.5rem;
  font-weight: 700;
  margin-top: 4rem;
  letter-spacing: 0.3rem;
  margin-bottom: 2rem;
  background-image: linear-gradient(to right, rgba(111, 34, 50, 0.9), rgba(29, 112, 80, 0.9));
  -webkit-background-clip: text;
  color: transparent;
  transition: all .3s; }
.heading-secondary:hover {
  transform: skewY(2deg) skewX(15deg) scale(1.1);
  text-shadow: 0.5rem 1 rem 2rem black; }

.text-box {
  position: absolute;
  top: 25%;
  left: 30%;
  transform: -50%, -50%;
  text-align: center; }

.heading-tertiary {
  font-size: 1.8rem;
  font-weight: 600;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  color: grey;
  text-transform: uppercase; }

.paragraph {
  color: grey;
  font-size: 1.3rem;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; }
.paragraph:not(:last-child) {
  margin-bottom: 2.5rem; }

/*-----*/
-----*/

.heading-primary-1 {
  color: whitesmoke;
  text-transform: uppercase; }

.heading-primary-main-1 {
  display: block;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  font-size: 80px;
  letter-spacing: 30px;
  font-weight: 600;
  animation-name: moveInLeft;
  animation-duration: 2s;
  animation-delay: 0.5s; }

.card-header {
  font-size: 40px;
  text-align: center;
  color: #374A3E;
  text-transform: uppercase;
  text-shadow: 2px; }

.card-title {
  font-size: 30px;
  text-align: center;
  color: #374A3E;
  text-transform: uppercase;
  text-shadow: 2px; }

.card-text {
  text-align: center; }

```

```

.alert-danger {
  text-align: center; }

.u-center-text {
  text-align: center; }

.u-margin-bottom-small {
  margin-bottom: 1.5rem; }

.u-margin-bottom-medium {
  margin-bottom: 4rem; }

.u-margin-bottom-big {
  margin-bottom: 8rem; }

.btn:link, .btn:visited {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  font-size: 85%;
  text-transform: uppercase;
  text-decoration: none;
  padding: 10px 20px;
  display: inline-block;
  border-radius: 100px;
  transition: all 0.2s;
  position: relative;
  margin-top: 30px; }

.btn:hover {
  transform: translateY(-3px);
  box-shadow: 0 10px 20px black; }
  .btn:hover::after {
    transform: scaleX(1.4) scaleY(2);
    opacity: 0; }

.btn:active {
  transform: translateY(-1px);
  box-shadow: 0 5px 10px black; }

.btn-white {
  background-color: whitesmoke;
  color: black; }
  .btn-white::after {
    background-color: whitesmoke; }

.btn::after {
  content: "";
  height: 100%;
  width: 100%;
  display: inline-block;
  border-radius: 100px;
  position: absolute;
  top: 0;
  left: 0;
  z-index: -1;
  transition: all 0.4s; }

.btn-animated {
  animation-name: moveInBottom;
  animation-duration: 1s;
  animation-delay: 0.2s;
  animation-fill-mode: backwards; }

.btn-text {
  font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode', Geneva, Verdana, sans-serif; }
  .btn-text:link, .btn-text:visited {
    font-size: 1.5rem;
    color: rgba(29, 112, 80, 0.9);
    display: inline-block;
    text-decoration: none;
    border-bottom: 1px solid rgba(40, 180, 131, 0.8);
    padding: 3px;
    margin-bottom: 5rem; }
  .btn-text:hover {
    background-color: rgba(40, 180, 131, 0.8);
    color: whitesmoke; }

```

```

    box-shadow: 0 1rem 2rem rgba(0, 0, 0, 0.15);
    transform: translateY(-2px); }
.btn-text:active {
    box-shadow: 0 1rem 1rem rgba(0, 0, 0, 0.15);
    transform: translateY(0); }

.Buttonna-1 {
    margin-top: 6rem;
    background-color: #39755D;
    color: whitesmoke;
    padding: 20px;
    text-align: center;
    text-decoration: none;
    font-size: 16px;
    cursor: pointer;
    border-radius: 50%; }
.Buttonna-1:active {
    box-shadow: 0 1rem 1rem rgba(0, 0, 0, 0.4);
    transform: translateY(0); }
.Buttonna-1:hover {
    background-color: rgba(40, 180, 131, 0.8);
    color: whitesmoke;
    box-shadow: 0 1rem 2rem rgba(0, 0, 0, 0.15);
    transform: translateY(-2px); }

.Buttonna-2 {
    background-color: #39755D;
    color: white;
    padding: 20px;
    text-align: center;
    text-decoration: none;
    font-size: 16px;
    margin: 4px 2px;
    cursor: pointer;
    border-radius: 50%; }
.Buttonna-2:active {
    box-shadow: 0 1rem 1rem rgba(0, 0, 0, 0.4);
    transform: translateY(0); }
.Buttonna-2:hover {
    background-color: rgba(40, 180, 131, 0.8);
    color: whitesmoke;
    box-shadow: 0 1rem 2rem rgba(0, 0, 0, 0.15);
    transform: translateY(-2px); }

.composition {
    position: relative; }
.composition__photo {
    width: 50%;
    box-shadow: 0 1.5rem 4rem rgba(0, 0, 0, 0.4);
    border-radius: 2px;
    position: absolute;
    z-index: 10;
    transition: 0.2s;
    outline-offset: 2rem; }
.composition__photo--p1 {
    left: 0;
    top: -2rem; }
.composition__photo--p2 {
    right: 0;
    top: 2rem; }
.composition__photo--p3 {
    left: 20%;
    top: 10rem; }
.composition__photo:hover {
    outline: 1.5rem solid rgba(40, 180, 131, 0.8);
    transform: scale(1.1);
    box-shadow: 0 1.5rem 4rem rgba(0, 0, 0, 0.9);
    z-index: 20; }
.composition:hover .composition__photo:not(:hover) {
    transform: scale(0.88); }

.row {
    max-width: 114rem;
    margin-left: 5rem; }
.row:not(:last-child) {

```

```

    margin-bottom: 8rem; }

.row::after {
  content: "";
  display: table;
  clear: both; }

.row [class^="col-"] {
  float: left; }

.row [class^="col-"]:not(:last-child) {
  margin-right: 2rem; }

.row .col-1-of-2 {
  width: calc((100% - 2rem) / 2); }

.Header {
  position: relative;
  height: 95vh;
  /*height of box of this element is 95% of viewport height*/
  background-image: linear-gradient(to right bottom, rgba(111, 34, 50, 0.9), rgba(29, 112, 80, 0.9)), url("../img/back.jpg");
  background-size: cover;
  background-position: top;
  /*image remains intact on top*/
  clip-path: polygon(0 0, 100% 0, 100% 75vh, 0 100%); }

.Header-1 {
  position: relative;
  height: 95vh;
  /*height of box of this element is 95% of viewport height*/
  background-image: linear-gradient(to right bottom, rgba(111, 34, 50, 0.9), rgba(29, 112, 80, 0.9)), url("../img/crowd.jpg");
  background-size: cover;
  background-position: top;
  /*image remains intact on top*/
  clip-path: polygon(0 0, 100% 0, 100% 75vh, 0 100%); }

.section-book {
  text-align: center;
  padding: 10rem 0;
  background-image: linear-gradient(to right bottom, rgba(111, 34, 50, 0.9), rgba(29, 112, 80, 0.9)); }

.book {
  background-image: linear-gradient(105deg, rgba(245, 245, 245, 0.9) 0%, rgba(245, 245, 245, 0.9) 50%, transparent 50%),
  url("../img/crowd.jpg");
  background-size: 100%;
  border-radius: 10px;
  box-shadow: 0 1.5rem 4rem rgba(0, 0, 0, 0.3);
  height: 50rem; }

.book__form {
  width: 50%;
  padding: 6rem; }

```

Package.json File

```

{
  "name": "genress",
  "version": "1.0.0",
  "description": "Landing Page",
  "main": "index.js",
  "scripts": {
    "compile:sass": "node-sass sass/main.scss css/styled.css -w"
  },
  "author": "Abhay",
  "license": "ISC",
  "devDependencies": {
    "node-sass": "^4.11.0"
  }
}

```


CHAPTER 4: CONCLUSION AND RESULTS

1. On applying the KNN algorithm we find that 54 test cases were correctly predicted by the model. The following screenshot gives the confusion matrix of the KNN Algorithm. The labels of the matrix represent

- a. '0' represents Blues Genre
- b. '1' represents Disco Genre
- c. '2' represents Jazz Genre
- d. '3' represents Rock Genre

We find that K-NN gives a 70% accuracy

	0	1	2	3
0	16	1	2	1
1	0	16	1	3
2	2	0	18	0
3	3	8	3	6

Fig 19. k-NN Results

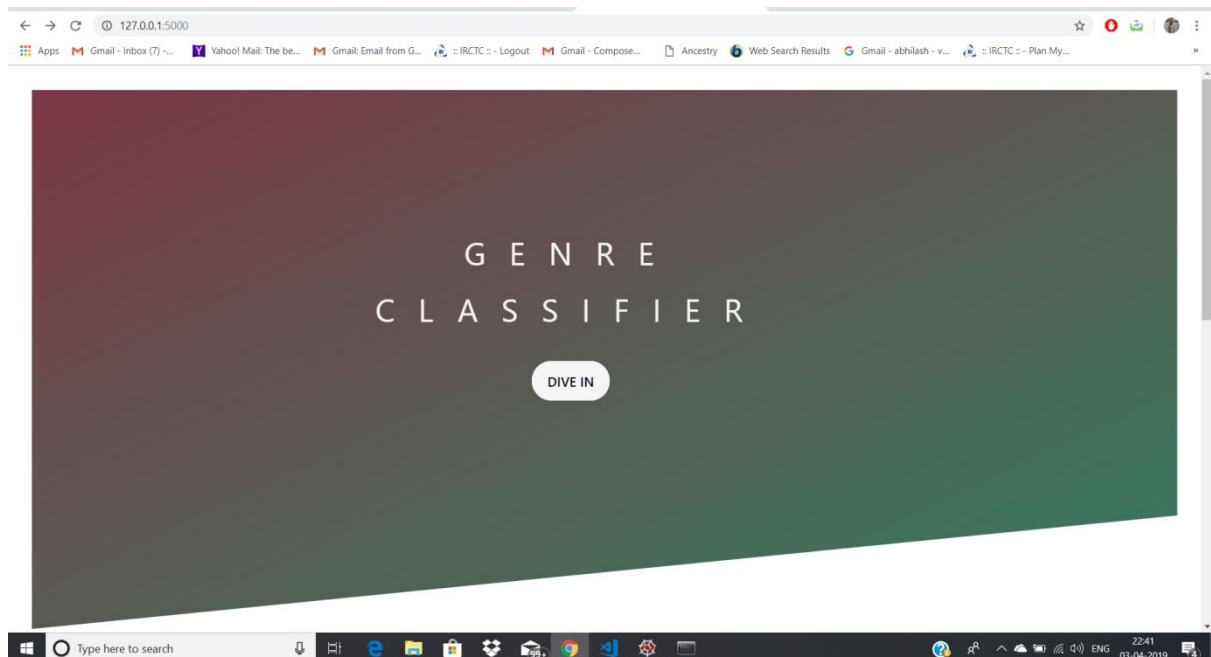


Fig 20. Web Page1

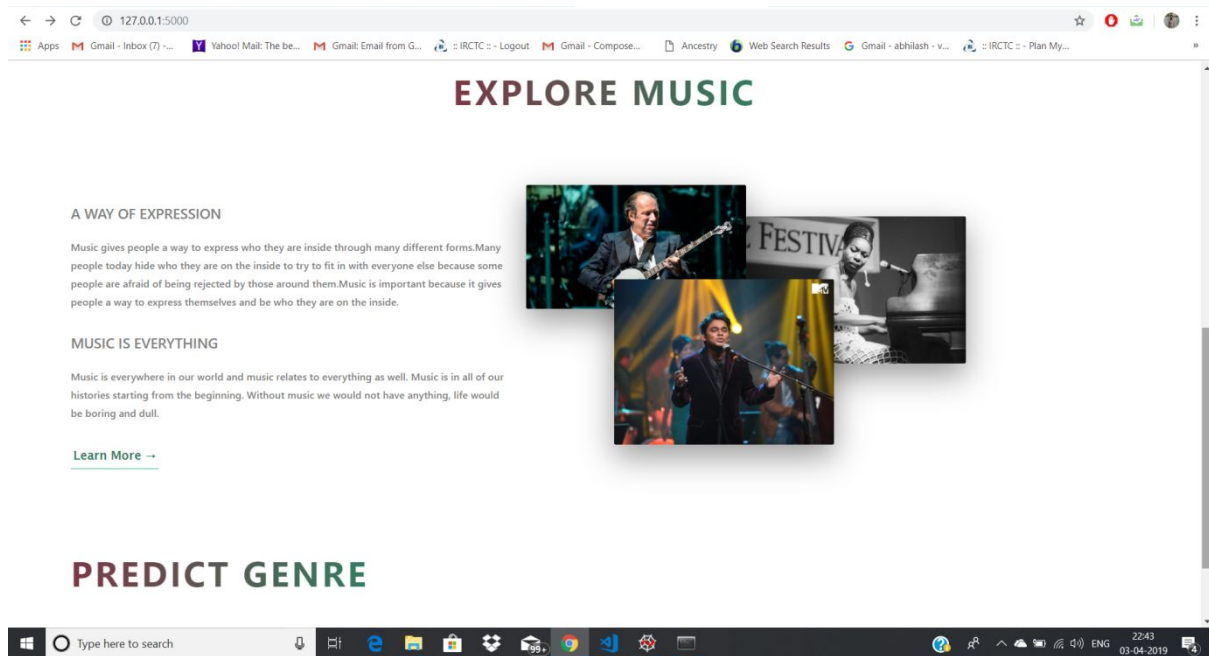


Fig 21. Web Page2

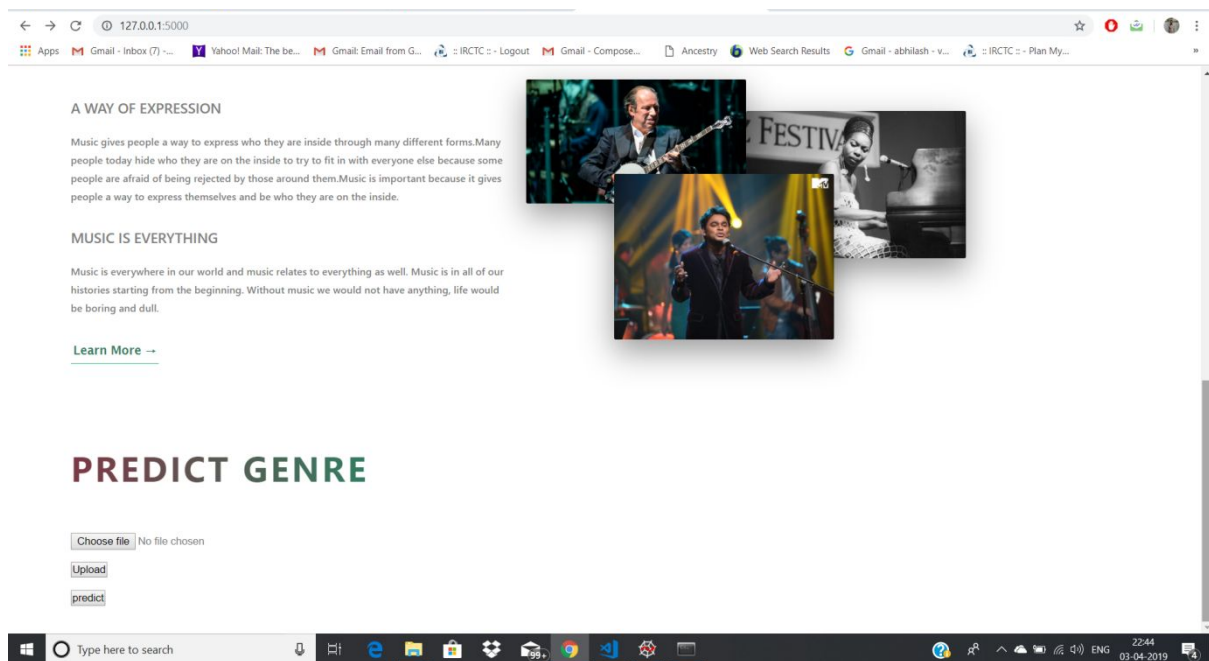


Fig 22. Web Page 3

The above images are of the first page of the website. On the bottom of the first page there are three links that provide the option of choosing the audio file from the system, uploading them on the system and predicting the genre of the song uploaded.

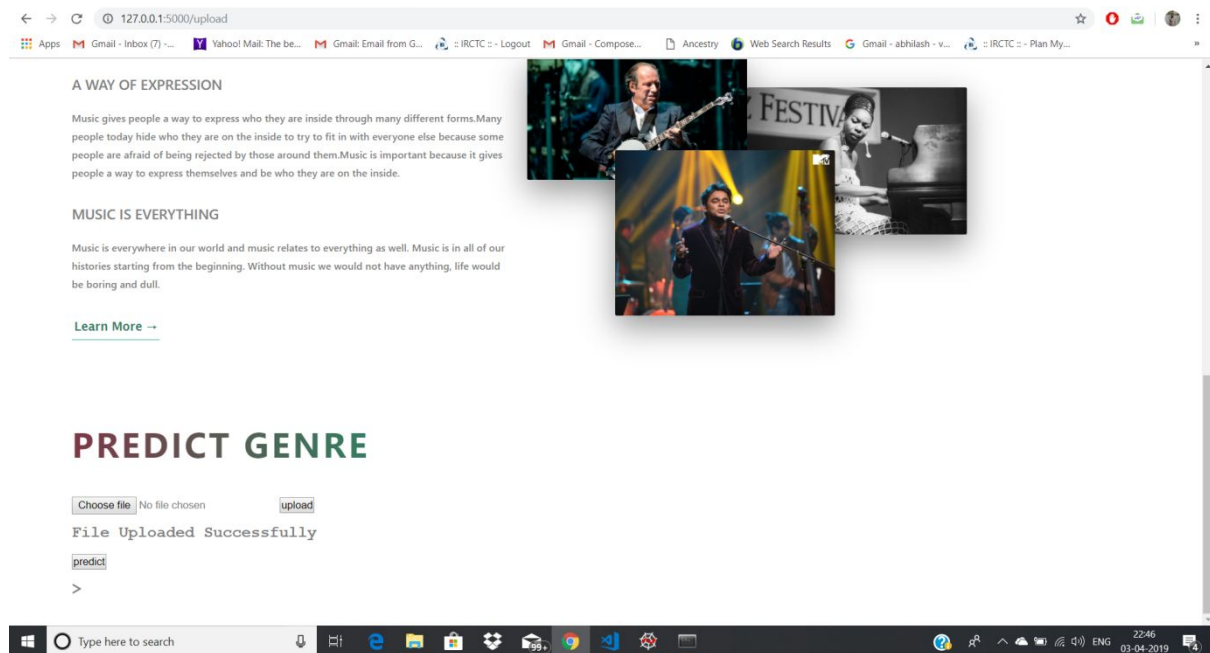


Fig 23. Web Page 4

The above image shows the page with the acknowledgement that the file has been uploaded successfully in the system. The predict button will take the user to the next page that will show the genre of the song that is uploaded.

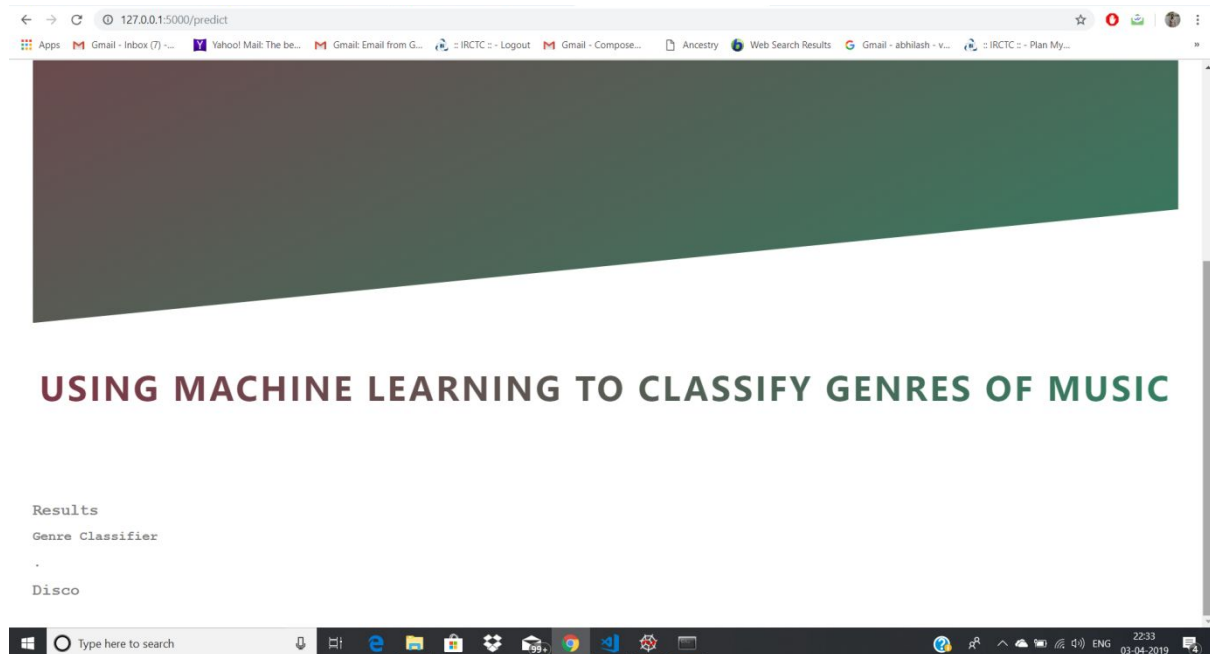


Fig 24. Result 1

1. In the above figure we uploaded a rock song and model predicted that the song is of disco genre.

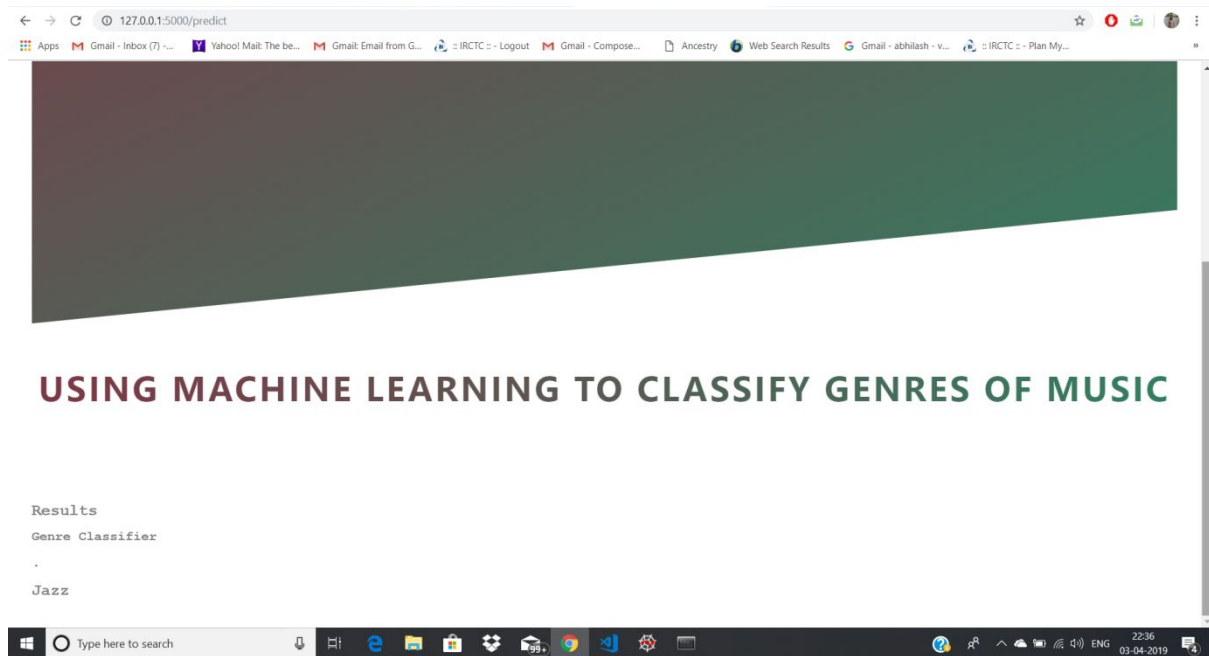


Fig 25. Result 2

2. In the above figure we uploaded a Jazz song and the model correctly predicted that the song is a jazz song.

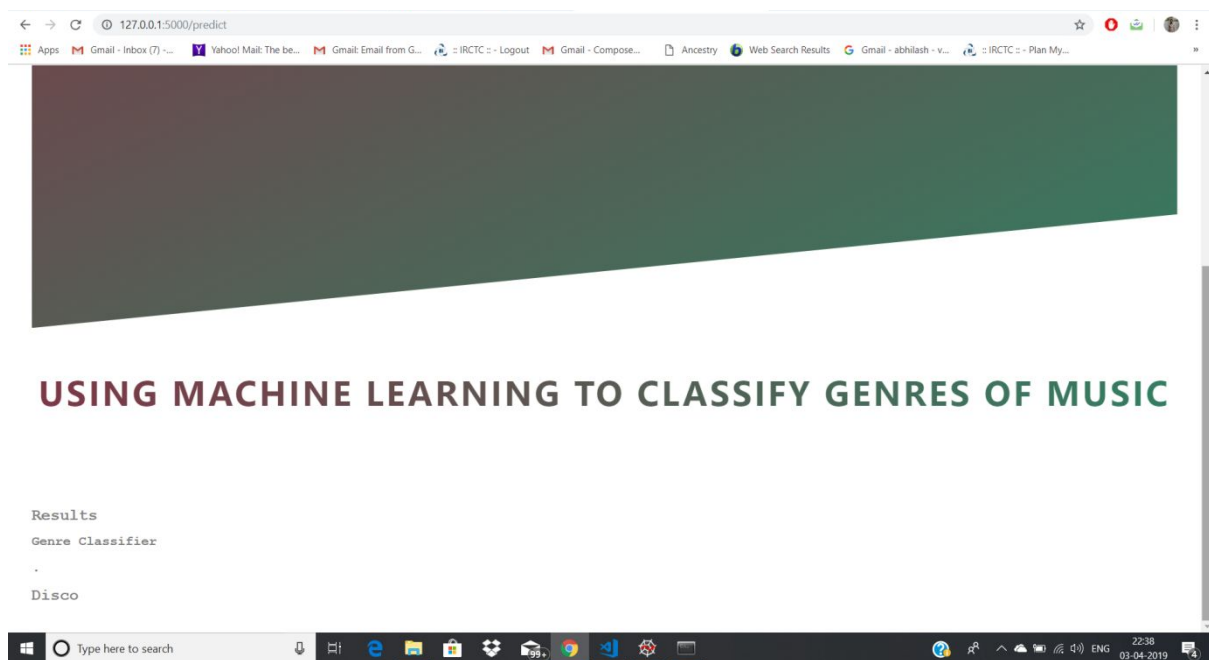


Fig 26. Result 3

3. In the above image we uploaded a song of disco genre and the model correctly predicted that the song is a disco song.

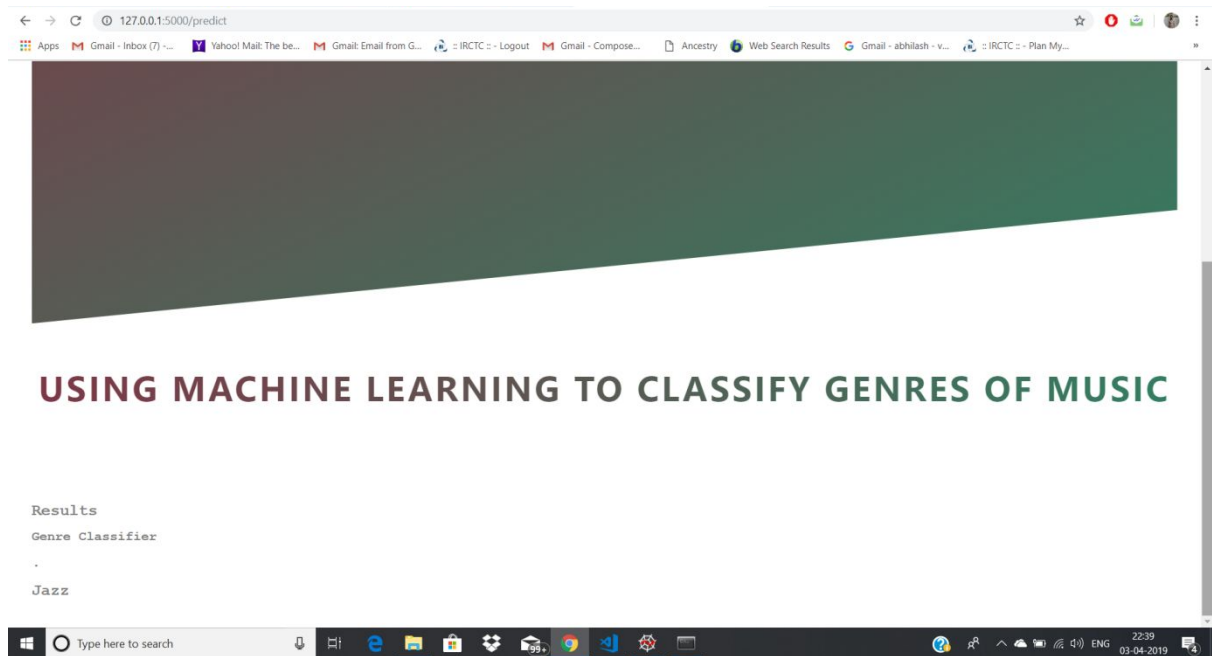


Fig 27. Result 4

4. In the above image we uploaded a Blues song but the model incorrectly predicted that the song is of Jazz genre.

CHAPTER 5: DISCUSSION AND FUTURE WORK

DISCUSSION:

As per our implementations, we find that k-NN works best for this data and gives an accuracy of 70%.

Our algorithms performed as per expectations giving accuracy 70%. It might seem that 70% is a small number, but the reason for accuracy being 70% is that the dataset we chose had only 400 songs in it.

One reason for this number could also be the usage of direct and simple implementations of the machine learning algorithms instead of using much more advanced techniques like Ensemble Learning and Deep Learning. Maybe we could have gotten better results with them.

We developed a product that can be used by a user for different purposes that may be for academic reasons or for maintaining music libraries. The site uses latest technologies that are currently being used by the industry.

FUTURE WORK:

These types of multi-class classification models have a major use in many fields not only limited to Music Information Retrieval. They have uses in image classification, text classification etc. These classification techniques form a stepping stone for cutting edge technologies that are being developed currently.

In this case particularly, Music Genre Classification can be used in technologies that implement library management systems for large number of songs, song identification using Natural Language Processing of the lyrics and analyzing the genres, and also music streaming applications. Applications like Saavn, Shazam, Spotify etc. use this technology extensively to classify their own libraries and identifying the songs with audio processing.

Since genre classification between fairly different genres is quite successful, it makes sense to attempt finer classifications. The exact same techniques used in this project could be easily extended to classify music based on any other labelling, such as artist. In addition, including additional metadata text features such as album, song title, or lyrics could allow us to extend this to music mood classification as well.

CHAPTER 6: REFERENCES

- [1] Archit Rathore – 12152 & Margaux Dorido - EXY1420 “Music Genre Classification”.
- [2] Michael Haggblade, Yang Hong & Kenny Kao “Music Genre Classification”.
- [3] Muhammad Asim Ali & Zain Ahmed Siddiqui, Department of Computer Science SZABIST Karachi, Pakistan “Automatic Music Genres Classification using Machine Learning”.
- [4] MARSYAS (Music Analysis, Retrieval and Synthesis for Audio Signals), <http://marsyas.info/>
- [5] Matthew Creme, Charles Burlin, Raphael Lenain “Music Genre Classification”.
- [6] Mel Frequency Cepstral Coefficient (MFCC) tutorial, <http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>
- [7] LibRosa Tutorial, <https://librosa.github.io/librosa/tutorial.html>
- [8] Finding the genre of a song with Deep Learning, <https://hackernoon.com/finding-the-genre-of-a-song-with-deep-learning-da8f59a61194>
- [9] Li Guo(liguo94), Zhiwei Gu(zhiweig) & Tianchi Liu(kitliu5) “Music Genre Classification via Machine Learning”.
- [10] W3Schools “<https://www.w3schools.com/>”