Configuration Manual

MSc Research Project
MSc. in Cloud Computing

Anshul Bharadwaj
Student ID: 21197911

School of Computing
National College of Ireland

Supervisor: Dr. Punit Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Anshul Bharadwaj
Student ID:	21197911
Programme:	MSc in Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Punit Gupta
Submission Due Date:	14/08/2023
Project Title:	Configuration manual
Word Count:	1272
Page Count:	11

I hereby certify that the information contained in this (my submission) is informationpertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at therear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator office must be placedinto the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anshul Bharadwaj
Student ID: 21197911

1 Introduction

The configuration guide outlines the procedure for applying Virtual Diagnosis by utilizing heart sounds captured via a digital stethoscope, aimed at aiding medically underserved regions through machine learning. Furthermore, it encompasses the overall arrangement necessary for project tool installation. This instructional document is tailored to benefit students in academia and other researchers, offering enhanced insights into the methodology deployed for executing this research endeavor.

2 Pre-requisites

2.1 AWS EC2

An Amazon EC2 instance is a virtual server in the cloud offered by Amazon Web Services (AWS). The instance type T2.2xlarge belongs to the T2 family and offers a balance of compute, memory, and network resources, catering to various workloads. It features a substantial 8 vCPUs, 32 GB of memory, and the ability to burst CPU performance based on workload demands. This instance type is particularly suitable for applications that require moderate compute power and memory, making it an ideal choice for multitier web applications, medium-sized databases, and development environments. With its versatility and scalability, the T2.2xlarge instance provides a cost-effective solution for businesses seeking to optimize performance while managing expenses in the AWS cloud environment.

2.2 Python

Python plays a pivotal role in the field of Machine Learning (ML) due to its rich ecosystem of libraries and frameworks tailored for data manipulation, analysis, and modeling. Its simplicity and readability make it an ideal language for researchers, developers, and data scientists to implement intricate ML algorithms and techniques. Python's libraries like NumPy, pandas, and scikit-learn provide essential tools for data preprocessing, feature engineering, and model evaluation. Additionally, popular ML frameworks like TensorFlow and PyTorch enable the creation of complex neural networks and deep learning models, driving advancements in image recognition, natural language processing, and more. Python's versatility, combined with its extensive community support, underscores its significance in enabling innovation and advancements within the realm of Machine Learning.

2.3 Flask

Flask is a lightweight and versatile web framework for Python that simplifies the process of building web applications. Known for its minimalistic design and flexibility, Flask allows developers to create web applications quickly by providing essential tools for routing, templating, and handling HTTP requests and responses. With its modular structure, developers can choose the components they need, making it well-suited for both simple and complex projects. Flask's extensive ecosystem of extensions further extends its functionality, enabling tasks such as authentication, database integration, and API development. Its user-friendly nature and scalability have made Flask a popular choice among developers seeking to create efficient and customized web applications.

2.4 Google Colab

Google Colab, a cloud-based platform, has emerged as a valuable tool for Machine Learning (ML) practitioners, offering a collaborative environment for creating, sharing, and executing ML projects seamlessly. It provides free access to GPUs and TPUs, accelerating model training and experimentation. With its integration of Jupyter notebooks, Colab facilitates code development, visualization, and documentation in a single interface. Users can easily import datasets, leverage popular libraries like TensorFlow and PyTorch, and share their work with others. This democratization of resources and collaborative capabilities make Google Colab a preferred choice for individuals and teams to efficiently explore, develop, and deploy Machine Learning solutions.

3 Packages and library used.

3.1 Scikit-learn

Scikit-learn, a widely utilized Python open-source machine learning library, equips developers and data scientists with an extensive toolkit spanning various facets of machine learning. Through its intuitive and coherent interface, scikit-learn simplifies processes such as data preprocessing, feature selection, model training, and evaluation. Its extensive range of algorithms encompasses classification, regression, clustering, and beyond, offering users an efficient means to explore and implement models. The library's seamless integration with other Python libraries, coupled with its emphasis on documentation and best practices, establishes it as an invaluable asset catering to both novices and seasoned experts in the machine learning domain.

3.2 Numpy

NumPy, a fundamental library for numerical computations in Python, provides essential data structures and functions to manipulate large arrays and matrices efficiently. Its array-oriented programming capabilities enable mathematical operations, broadcasting, and element-wise computations, making it an essential foundation for various scientific and data analysis tasks.

3.3 Pandas

Pandas, a versatile data manipulation library, simplifies data analysis in Python by introducing data structures like DataFrames that organize and manipulate structured data. With powerful tools for data cleaning, transformation, and exploration, Pandas streamlines tasks such as indexing, aggregation, and merging, enabling users to effectively handle and analyze tabular and time-series data for insightful decision-making.

3.4 Librosa

Librosa is a Python package specifically designed for analyzing and working with audio and music data. It offers tools to extract various audio features, visualize audio data, and perform tasks like spectrogram computation, beat tracking, and tempo estimation, making it an invaluable resource for researchers and practitioners in the field of audio signal processing and music analysis.

3.5 Node

Node.js is a runtime environment that allows developers to execute JavaScript code on the server-side, enabling the creation of scalable and efficient network applications. It utilizes an event-driven, non-blocking architecture, which makes it particularly well-suited for handling asynchronous operations and building real-time web applications, APIs, and backend services.

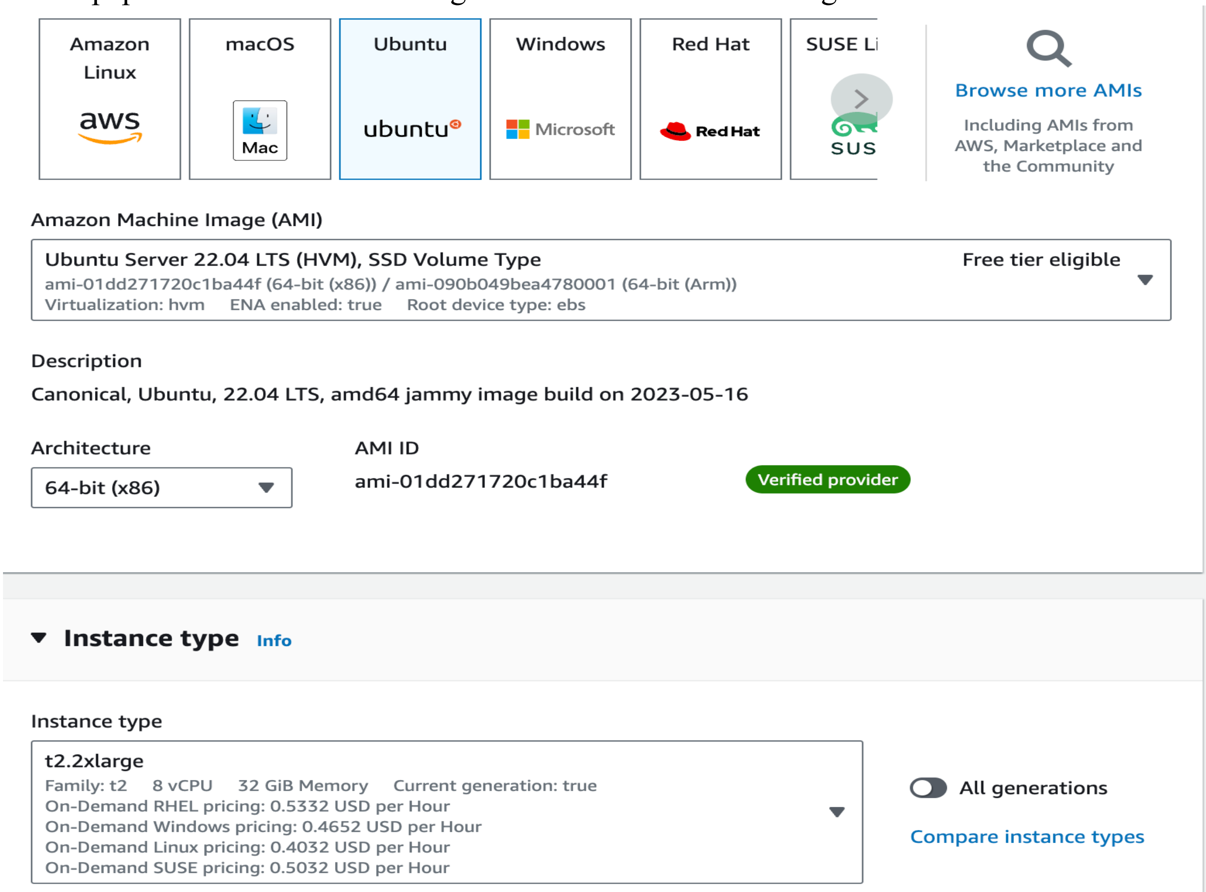
3.6 NPM

npm (Node Package Manager) is the default package manager for Node.js, providing a vast repository of open-source libraries and tools that developers can easily integrate into their projects. It simplifies dependency management, version control, and package distribution, streamlining the process of building and maintaining Node.js applications by enabling developers to access and share modular code solutions.

4 Configurations

4.1 EC2

In this paper we have used t2.2xlarge instance with 32 GB storage.



4.2 Installations on EC2

These are the packages need to install on EC2.

```
[ubuntu@ip-172-31-47-76:~$ cd
[ubuntu@ip-172-31-47-76:~$ cd ai-doc/
[ubuntu@ip-172-31-47-76:~/ai-doc$ ls
 Heart_Sound.npy  README.md  dataset_heart.csv  heart_monitoring_flask.py  std.log
[ubuntu@ip-172-31-47-76:~/ai-doc$ history
    1  ls
    2  git clone https://github.com/jps1001/ai-doc.git
    3  cd ai-doc/
    4  ls
    5  pip install Flask
    6  sudo apt install python3-pip
    7  pip install Flask
    8  pip install scikit-learn
    9  pip install numpy
   10  pip install pandas
   11  pip install librosa
   12  sudo apt install nodejs
   13  sudo apt install npm
   14  sudo npm install pm2 -g
   15  python3 heart_monitoring_flask.py
   16  pm2 start "python3 heart_monitoring_flask.py" --name heart
   17  ls
   18  cd ai-doc/
   19  git pull
   20  pm2 restart heart
   21  git pull
   22  cat std.log
   23  pm2 restart heart
   24  cat std.log
   25  ls
   26  cd ai-doc/
   27  ls
   28  cat heart_monitoring_flask.py
   29  exit
   30  ls
   31  cd ai-doc/
   32  ls
   33  cat Heart_Sound.npy
   34  1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c1;2c
   35  ls
   36  cat
   37  ls
   38  cd ai-doc/
```

5 Machine Learning Model

5.1 Dataset retrieval

```
import warnings                         # To ignore any warnings
warnings.filterwarnings("ignore")
%matplotlib inline
%pylab inline
import os
import pandas as pd
import librosa
import librosa.display
import glob
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
```

```
Populating the interactive namespace from numpy and matplotlib
```

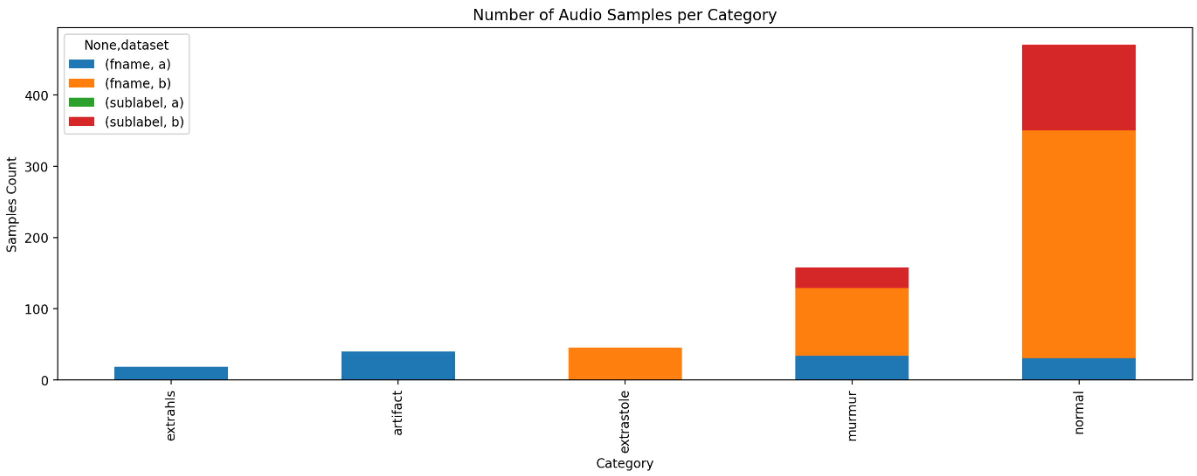
```
# parent folder of sound files
INPUT_DIR="My Drive/"
# 16 KHz
SAMPLE_RATE = 16000
# seconds
MAX_SOUND_CLIP_DURATION=12
```

```
set_a=pd.read_csv(INPUT_DIR+"set_a.csv")
set_a.head()
```

	dataset	fname	label	sublabel
0	a	set_a/artifact__201012172012.wav	artifact	NaN
1	a	set_a/artifact__201105040918.wav	artifact	NaN
2	a	set_a/artifact__201105041959.wav	artifact	NaN
3	a	set_a/artifact__201105051017.wav	artifact	NaN
4	a	set_a/artifact__201105060108.wav	artifact	NaN

```
set_b=pd.read_csv(INPUT_DIR+"set_b.csv")
set_b.head()
```

	dataset	fname	label	sublabel
0	b	set_b/Btraining_extrastole_127_1306764300147_C...	extrastole	NaN
1	b	set_b/Btraining_extrastole_128_1306344005749_A...	extrastole	NaN
2	b	set_b/Btraining_extrastole_130_1306347376079_D...	extrastole	NaN
3	b	set_b/Btraining_extrastole_134_1306428161797_C...	extrastole	NaN
4	b	set_b/Btraining_extrastole_138_1306762146980_B...	extrastole	NaN



5.2 Reading Audio File

```
print('Minimum samples per category = ', min(train_ab.label.value_counts()))
print('Maximum samples per category = ', max(train_ab.label.value_counts()))
```

```
Minimum samples per category =  19
Maximum samples per category =  351
```

```
normal_file=INPUT_DIR+"set_a/normal__201106111136.wav"
```

```
# heart it
import IPython.display as ipd
ipd.Audio(normal_file)
```

0:00 / 0:04

6 Feature Extraction

Feature Extraction is a crucial step in training a ML model. In this paper we have waveplot the audio file.

```
# Load use wave
import wave
wav = wave.open(normal_file)
print("Sampling (frame) rate = ", wav.getframerate())
print("Total samples (frames) = ", wav.getnframes())
print("Duration = ", wav.getnframes()/wav.getframerate())
```

```
Sampling (frame) rate =  44100
Total samples (frames) =  218903
Duration =  4.963786848072562
```

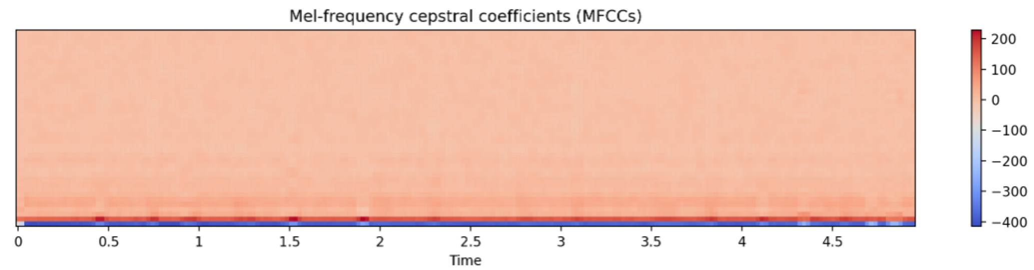
```
# Load use scipy
from scipy.io import wavfile
rate, data = wavfile.read(normal_file)
print("Sampling (frame) rate = ", rate)
print("Total samples (frames) = ", data.shape)
print(data)
```

```
Sampling (frame) rate =  44100
Total samples (frames) =  (218903,)
[-22835 -22726 -22595 ...   -474   -450   -439]
```

7 MFCC Extraction – Mel Frequency Cepstral Coefficients

Mel Frequency Cepstral Coefficients (MFCCs) constitute a widely employed feature set within audio and speech processing, serving purposes like speech recognition, speaker identification, and music genre classification. These coefficients are formulated based on the Mel scale, a perceptual pitch scale designed to mimic the human auditory system's sensitivity to varying frequencies.

```
# Visualize the MFCC series
# Mel-frequency cepstral coefficients (MFCCs)
plt.figure(figsize=(12, 3))
librosa.display.specshow(mfccs, x_axis='time')
plt.colorbar()
plt.title('Mel-frequency cepstral coefficients (MFCCs)')
plt.tight_layout()
```




```
oenv = librosa.onset.onset_strength(y=y, sr=sr)
# Detect events without backtracking
onset_raw = librosa.onset.onset_detect(onset_envelope=oenv, backtrack=False)
# Backtrack the events using the onset envelope
onset_bt = librosa.onset.onset_backtrack(onset_raw, oenv)
# Backtrack the events using the RMS values
rms = librosa.feature.rms(S=np.abs(librosa.stft(y=y)))
onset_bt_rms = librosa.onset.onset_backtrack(onset_raw, rms[0])
```

```
# Plot the results
plt.figure(figsize=(16, 6))
plt.subplot(2,1,1)
plt.plot(oenv, label='Onset strength')
plt.vlines(onset_raw, 0, oenv.max(), label='Raw onsets')
plt.vlines(onset_bt, 0, oenv.max(), label='Backtracked', color='r')
plt.legend(frameon=True, framealpha=0.75)
plt.subplot(2,1,2)
plt.plot(rms[0], label='RMS')
plt.vlines(onset_bt_rms, 0, rms.max(), label='Backtracked (RMS)', color='r')
plt.legend(frameon=True, framealpha=0.75)
```

8 Installing packages and Feature Selection

```
from sklearn.model_selection import StratifiedKFold
import os
import time
import xgboost as xgb
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_score
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import RidgeClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
```

```
##################################### Feature Selection ###########################################
start_time = time.time()
mod = RandomForestClassifier()
param = {"n_estimators": [100],
   "criterion": ["gini","entropy"],
   "max_features": ["auto","sqrt","log2",None],
   "oob_score": [True, False]}

grid = GridSearchCV(mod, param, n_jobs=1)
grid.fit(Xtrain,Ytrain)

clf5 = RandomForestClassifier(n_estimators=grid.best_params_["n_estimators"],criterion=grid.best_params_["criterion"],max_features=grid.best_params_["max_fe

clf5.fit(Xtrain,Ytrain)
clf5.feature_importances_
modi = SelectFromModel(clf5, prefit=True)
Xtrain = modi.transform(Xtrain)
Xtest = modi.transform(Xtest)
print("--- %s seconds ---" % (time.time() - start_time))
print("Feature Selection Code......")
```

9 Machine Learning Models code snippets

This model is trained by Random Forest, KNN, SVM RBF, Logistic Regression etc. The best performance algorithm is SVM RBF.

```
#####################################  Classification Models #####################################

##################################################################################################### SVM
start_time = time.time()
mod = SVC()

#################################### SVR-Sigmoid

g = [pow(2,-15),pow(2,-14),pow(2,-13),pow(2,-12),pow(2,-11),pow(2,-10),pow(2,-9),pow(2,-8),pow(2,-7),pow(2,-6),pow(2,-5),pow(2,-4),pow(2,-3),pow(2,-2),pow(2

C = [pow(2,-5),pow(2,-4),pow(2,-3),pow(2,-2),pow(2,-1),pow(1,0),pow(2,1),pow(2,2),pow(2,3),pow(2,4),pow(2,5),pow(2,6),pow(2,7),pow(2,8),pow(2,9),pow(2,10),p

param = {"kernel": ["sigmoid"],
     "gamma": g,
     "C":C}
random_search = RandomizedSearchCV(mod,param,n_jobs=1,n_iter=100)
random_search.fit(Xtrain,Ytrain)
clf0 = SVC(kernel=random_search.best_params_["kernel"],gamma=random_search.best_params_["gamma"],C=random_search.best_params_["C"])
print ("Check 1")
print("--- %s seconds ---" % (time.time() - start_time))
print("SVR-Sig------")
#################################### SVR-RBF
```

SVM RBF

```
#################################### SVR-RBF
start_time = time.time()
param= {'gamma': g,
    'kernel': ['rbf'],
    'C': C}
grid_search = RandomizedSearchCV(mod,param,n_jobs=1,n_iter=100)
grid_search.fit(Xtrain,Ytrain)
clf1 = SVC(gamma = grid_search.best_params_["gamma"],kernel=grid_search.best_params_["kernel"],C=grid_search.best_params_["C"])


clf0.fit(Xtrain,Ytrain)
z0=clf0.predict(Xtest)
print (z0,Ytest)

pred = pd.DataFrame(z0)
pred_df_svms = pd.concat([pred_df_svms,pred])

clf1.fit(Xtrain,Ytrain)
z1=clf1.predict(Xtest)

pred = pd.DataFrame(z1)
pred_df_svmr = pd.concat([pred_df_svmr,pred])
```

Logistic Regression

```
######################################################################################################## Logistic Regression
start_time = time.time()
g = [pow(2,-15),pow(2,-14),pow(2,-13),pow(2,-12),pow(2,-11),pow(2,-10),pow(2,-9),pow(2,-8),pow(2,-7),pow(2,-6),pow(2,-5),pow(2,-4),pow(2,-3),pow(2,-2),pow(2

C = [pow(2,-5),pow(2,-4),pow(2,-3),pow(2,-2),pow(2,-1),pow(1,0),pow(2,1),pow(2,2),pow(2,3),pow(2,4),pow(2,5),pow(2,6),pow(2,7),pow(2,8),pow(2,9),pow(2,10),p

mod = LogisticRegression()
param = {"penalty":['l1'],
   "dual": [False],
   "C":C,
   "fit_intercept": [True, False],
   "solver": ["liblinear"]}

grid = GridSearchCV(mod,param,n_jobs=1)
grid.fit(Xtrain,Ytrain)

clf2 = LogisticRegression(penalty=grid.best_params_["penalty"],dual=grid.best_params_["dual"],C=grid.best_params_["C"],fit_intercept=grid.best_params_["fit_
print("--- %s seconds ---" % (time.time() - start_time))
print("LR-L1------")
```

Random Forest

```
######################################################################################################## Random Forest
start_time = time.time()
mod = RandomForestClassifier()
param = {"n_estimators": [100,500],
   "criterion": ["gini","entropy"],
   "max_features": ["auto","sqrt","log2",None],
   "oob_score": [True, False]}

grid = GridSearchCV(mod, param, n_jobs=1)
grid.fit(Xtrain,Ytrain)

clf5 = RandomForestClassifier(n_estimators=grid.best_params_["n_estimators"],criterion=grid.best_params_["criterion"],max_features=grid.best_params_["max_fe

clf5.fit(Xtrain,Ytrain)
z5 = clf5.predict(Xtest)

pred = pd.DataFrame(z5)
pred_df_rfc = pd.concat([pred_df_rfc,pred])

print("Random Forest: ",accuracy_score(z5,Ytest))

print("--- %s seconds ---" % (time.time() - start_time))
print("Random Forest ------")
print ("check 5")
```

KNN

```
######################################################################################################## KNN


start_time = time.time()
mod = KNeighborsClassifier()
param = {"n_neighbors": range(1,100,1),
   "weights": ["uniform", "distance"],
   "algorithm": ["auto","ball_tree","kd_tree","brute"],
   "p":[1,2]}
grid = RandomizedSearchCV(mod,param,n_jobs=1,n_iter=100)
grid.fit(Xtrain,Ytrain)
clf11 = KNeighborsClassifier(n_neighbors=grid.best_params_["n_neighbors"],weights=grid.best_params_["weights"],algorithm=grid.best_params_["algorithm"],p=gr
print("--- %s seconds ---" % (time.time() - start_time))
clf11.fit(Xtrain, Ytrain)
z11 = clf11.predict(Xtest)

pred = pd.DataFrame(z11)
pred_df_knnc = pd.concat([pred_df_knnc,pred])

print("KNN: ",accuracy_score(z11,Ytest))
print ("check 7")
```

10 Deployment

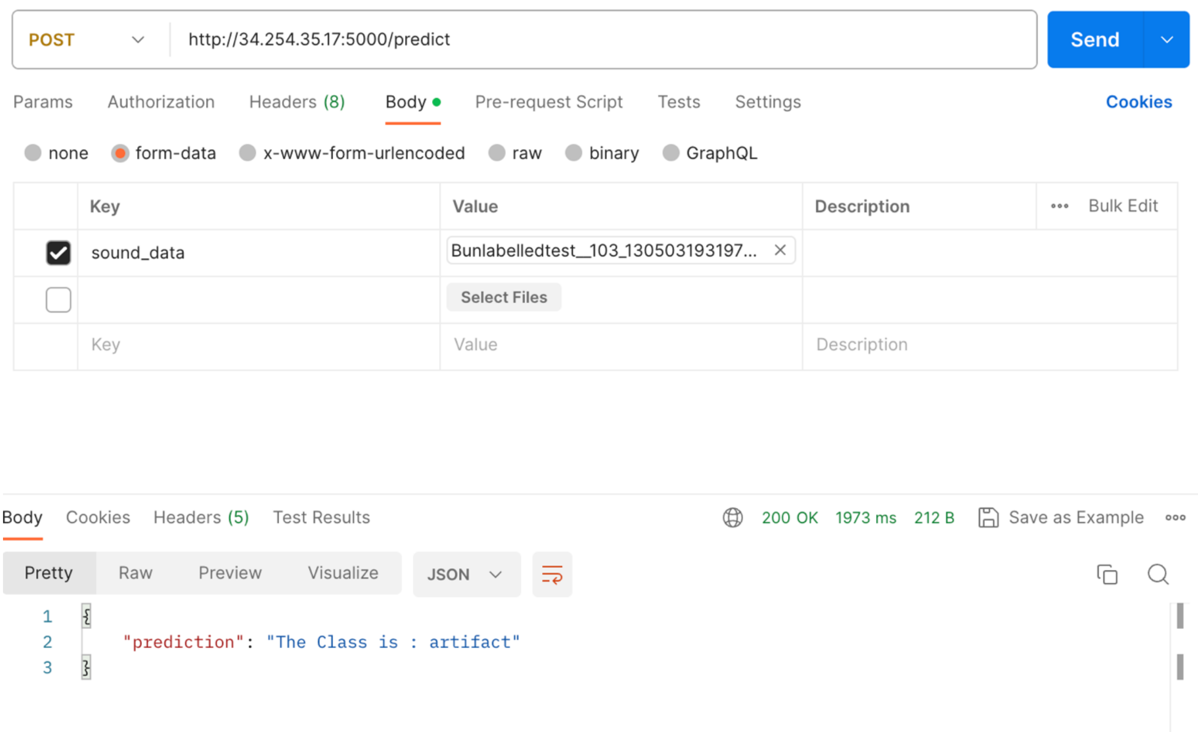
The deployment process involves deploying a machine learning model that utilizes Flask for handling HTTP requests. The application encompasses three key functions: feature extraction, prediction, and API calls. For feature extraction, the librosa library is employed to extract relevant features from audio data, enabling effective analysis. The extracted features are then fed into the prediction function, where a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel is employed. This SVM-RBF model, achieving an accuracy of 74.34%, efficiently predicts the characteristics of sound data. Lastly, an API call function is implemented to enable external interaction with the deployed model. This integrated approach, utilizing Flask, librosa, SVM-RBF, and API calls, ensures a seamless deployment pipeline that empowers users to analyze sound data, predict outcomes accurately, and access the model's capabilities through a user-friendly interface.

```
import pandas as pd
import numpy as np
import librosa
import os
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from flask import Flask, request, jsonify

def extract_features(audio_path):
    y, sr = librosa.load(audio_path, duration=4)
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
    return mfccs
```

```
def predict(sound_data):
    n = data["label"].value_counts()[0]
    target = data.pop("label").values
    training = train_dataset
    sound_data = sound_data.reshape((1, 40 * 173))
    le = LabelEncoder()
    target = le.fit_transform(target.astype(str))
    g = [pow(2,-15),pow(2,-14),pow(2,-13),pow(2,-12),pow(2,-11),pow(2,-10),pow(2,-9),pow(2,-8),pow(2,-7),pow(2,-6),pow(2,-5),po

    C = [pow(2,-5),pow(2,-4),pow(2,-3),pow(2,-2),pow(2,-1),pow(1,0),pow(2,1),pow(2,2),pow(2,3),pow(2,4),pow(2,5),pow(2,6),pow(2
    param= {'gamma': g,
        'kernel': ['rbf'],
        'C': C}
    mod = SVC()
    grid_search = GridSearchCV(mod,param)
    grid_search.fit(training, target)
    clf1 = SVC(gamma = grid_search.best_params_["gamma"],kernel=grid_search.best_params_["kernel"],C=grid_search.best_params_["
    clf1.fit(training, target)
    preds = clf1.predict(sound_data)
    # extracting most confident predictions
    heart_class = le.inverse_transform(preds)
    t = "The Class is : " + str(heart_class)
    return t
```

```
# Define the Flask app
app = Flask(__name__)

# API endpoint for heart sound data prediction
@app.route('/predict', methods=['POST'])
def predict_heart_sound_data():
    try:
        sound_file = request.files['sound_data']
        if sound_file:
            # Save the uploaded sound file temporarily
            temp_file_path = 'temp.wav'
            sound_file.save(temp_file_path)

            # Load and process the sound data
            features = extract_features(temp_file_path)
            t = predict(features)

            # Clean up the temporary file
            os.remove(temp_file_path)

            return jsonify({"prediction": t})
        else:
            return jsonify({"error": "No sound data received."}), 400
    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

11 Testing HTTP through Postman



References

K. Vanisree and J. Singaraju, 2011. “Decision support system for congenital heart disease diagnosis based on signs and symptoms using neural networks,” International Journal of Computer Applications, vol. 19, no. 6, pp. 6–12.

Liang, H., Lukkarinen, S. and Hartimo, I., 1997, September. Heart sound segmentation algorithm based on heart sound envelogram. In Computers in Cardiology 1997 (pp. 105-108). IEEE.

Numpy (2009). *NumPy*. [online] Numpy.org. Available at: https://numpy.org/.