

Classification of Heart Sounds using Machine learning algorithms

PRUTHVI TEJA ANUMANDLA

pruthvitejaanumandla@my.unt.edu

On my honor as a UNT student, I have neither given nor received unauthorized assistance on this work.

Abstract

Lack of proper medical attention in remote areas where there is no proper medical infrastructure tantamount to severe health consequences. In particular, heart-related issues cause approximately two million deaths per year, according to WHO survey in 2009. Diagnosis of heart disease involves heart auscultations for irregular sounds by medical specialists, followed by a further investigation. These tests are expensive and require specialized technicians to operate. We present, a handy setup for initial screening of the heart sounds that can potentially be used by the users on themselves who can later share the readings with their healthcare providers. An innovative mobile health service platform is created for analyzing and classifying heart sounds using machine learning. The smartphone application facilitates the recording, processing, visualization, listening, and classification of heart sounds. With the help of this equipment, we can make the preliminary diagnosis handy and affordable, reaching even to remote areas with less medical assistance.

Introduction

What is the problem you are addressing?

This project aims to implement a machine learning application that identifies disease faced by the user with the help of recording their heartbeats through a mobile health service platform. It helps to facilitate preliminary diagnosis for people, which could show what kind of disease one is facing without professional's intervention with a cost-effective equipment.

What claim do you hope to make when your work is complete?

With help of these machine algorithms, I am trying to get an F1-measure of over 0.8 on each disease type for new/unseen patients, based on processing 10 seconds of user's heartbeat. This approach is unique as few of the applications which are in market use equipment which requires a computer or professional inference to classify the disease, which is all done here by machine learning on a mobile application.

Who cares? Why do they care?

It is useful both to people who want to get a preliminary diagnosis and for medical professionals to visualize and store data in the patient's electronic health record in compliance with federal patient privacy rules.

What is lacking from previous attempts to solve the problem (in particular, mention weakness that your paper is going to overcome)?

The current available electronic stethoscopes in the market are Welch-Allyn Elite stethoscope, Jabes electronic stethoscope, 3M Littmann electronic stethoscope, Thinklabs One digital stethoscope and Eko Core digital stethoscope. Though they perform with good accuracy, the first three stethoscopes require being connected to a computer to record, view, share and save heart sounds. Eko Core and Thinklabs are the only digital stethoscopes to work with a smartphone application. However, these applications still require a medical practitioner for interpreting sound and classifying disease. Furthermore, these electronic stethoscopes are costly, ranging from \$199-\$499.

What do you know about previous attempts to solve it? List key publications that you are aware of.

These are few papers which discuss different machine learning algorithms and how they are employed for detecting the heart sounds.

- DigiScope — Unobtrusive collection and annotating of auscultations in real hospital environments.
D. Pereira ; F. Hedayioglu; I. Dutra ; F. Almeida ; S. S. Mattos ; M. Coimbra
<http://ieeexplore.ieee.org/abstract/document/6090280/>
- A comparative study of the SVM and KNN machine learning algorithms for the diagnosis of respiratory pathologies using pulmonary acoustic signals.
Rajkumar Palaniappan, Kenneth Sundaraj and Sebastian Sundaraj
<https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-15-223>
- Support Vectors Machine-based identification of heart valve diseases using heart sounds.
Ilias Maglogiannis, Euripidis Loukis, Elias Zafiropoulos, AntonisStasis
<http://www.sciencedirect.com/science/article/pii/S0169260709000339>

What have previous results been?

- The K-NN classifier was found to be better than the SVM classifier for detecting signals from pathological and normal subjects obtained in one of the work.
- Whereas in other works SVM classifiers were found performing more accurately than back-propagation neural networks, k-nearest-neighbor, and naïve Bayes classifiers.
- The classification accuracies of the SVM and KNN classifiers stated in these papers are 92.19% and 98.26%, respectively.

The key contributions of this work are:

- Worked with different machine learning algorithms on available data
- Was able to increase f1-measure from initial 0.13 to more than 0.30 by employing different methods like oversampling, parameter tuning with available data.



The equipment used for collecting data from users.

Data

The data is collected from the equipment, as shown above. Once a person's heart beats are collected, they are stored in CSV format representing the heart sound for 10 seconds, the lub-dubs are extracted manually from these files. Each row of the data represents a Lub-Dub extracted from the users on which we train the model.

A1		feature set						1; 0.459335; 0.458356; 0.456948;	
Label	A	B	C	D	E	F	G		
1	1	0.459335	0.458356	0.456948	0.455184	0.453156	0.450971	0.44874	
2	1	0.459610	0.460571	0.461169	0.461420	0.461376	0.461119	0.46075	
3	1	0.446598	0.445033	0.443304	0.441454	0.439530	0.437577	0.43564	
4	1	0.483927	0.482239	0.479995	0.477319	0.474339	0.471181	0.46795	
5	1	0.460871	0.462368	0.463492	0.464218	0.464542	0.464484	0.46408	
6	1	0.483262	0.484604	0.485721	0.486594	0.487222	0.487615	0.48775	
7	1	0.469578	0.468289	0.466866	0.465369	0.463881	0.462497	0.46135	
8	1	0.479083	0.476570	0.473889	0.471150	0.468479	0.466002	0.46385	
9	1	0.465573	0.462903	0.460484	0.458376	0.456629	0.455276	0.45435	
10	1	0.478386	0.477430	0.476749	0.476338	0.476179	0.476245	0.47650	
11	1	0.466406	0.465823	0.465307	0.464881	0.464561	0.464358	0.46428	
12	1	0.479601	0.481420	0.483376	0.485381	0.487332	0.489126	0.49067	
13	2	0.429637	0.432443	0.435376	0.438054	0.440031	0.441816	0.44395	
14	2	0.482077	0.485361	0.488152	0.490888	0.492585	0.493789	0.49493	
15	2	0.490284	0.487915	0.486256	0.485782	0.485664	0.486848	0.48775	
16	3	0.560909	0.557517	0.554056	0.550594	0.547098	0.543566	0.53996	
17	3	0.516986	0.516871	0.516613	0.516268	0.515722	0.515004	0.51422	
18	4	0.529700	0.531447	0.534068	0.536487	0.538301	0.540384	0.54226	
19	4	0.448407	0.447645	0.446273	0.444597	0.441396	0.439796	0.43758	
20	4	0.512137	0.511097	0.510062	0.509058	0.508123	0.507165	0.50623	
21	4	0.450974	0.452495	0.454096	0.455683	0.457301	0.458959	0.46053	
22	5	0.486324	0.485050	0.484731	0.485004	0.485869	0.485778	0.48477	
23	5	0.448346	0.443594	0.439072	0.434669	0.430528	0.426611	0.42305	
24	5	0.485092	0.483597	0.482050	0.480468	0.478821	0.477186	0.47549	
25	6	0.237613	0.245855	0.254382	0.263193	0.272383	0.281786	0.29156	
26	6	0.498029	0.499199	0.500325	0.501668	0.503011	0.504311	0.50582	

These data are labeled based on the heart beats collected from persons of different conditions, the diseased and normal files are collected from UT Southwestern where the persons whose disease is known are auscultated with the above shown equipment and saved under the respective label and in the same way persons who are known to be normal are auscultated and labeled normal. The above data is annotated with numerical labels which represent different disease as given in below file. The file below also depicts how data is split into the train (60%), test (20%) and validation (20%) for the tuning and evaluation of algorithms.

<C:\Laptop data Jan 18\Desktop\ML Sem\PML\Screenshots of Outputs\statistics.xlsx>

The class distribution of the data is shown in the table below with ninety percent of the people who are auscultated are found to be normal and remaining ten percent are classified as having a disease with below distribution respectively.

Normal (N)	90%
Aortic stenosis Pulmonary stenosis (AS_PS)	3%
Mitral valve prolapses (MVP)	1%
Mitral regurgitation Tricuspid regurgitation (MR_TR)	2%
Mitral stenosis Tricuspid stenosis (MS_TS)	1%
Flow murmurs (FM)	1%
Aortic regurgitation Pulmonary regurgitation (AR_PR)	1%
Patent Ductus Arteriosus (PDA)	1%

Resampling data to represent the distribution stated above.

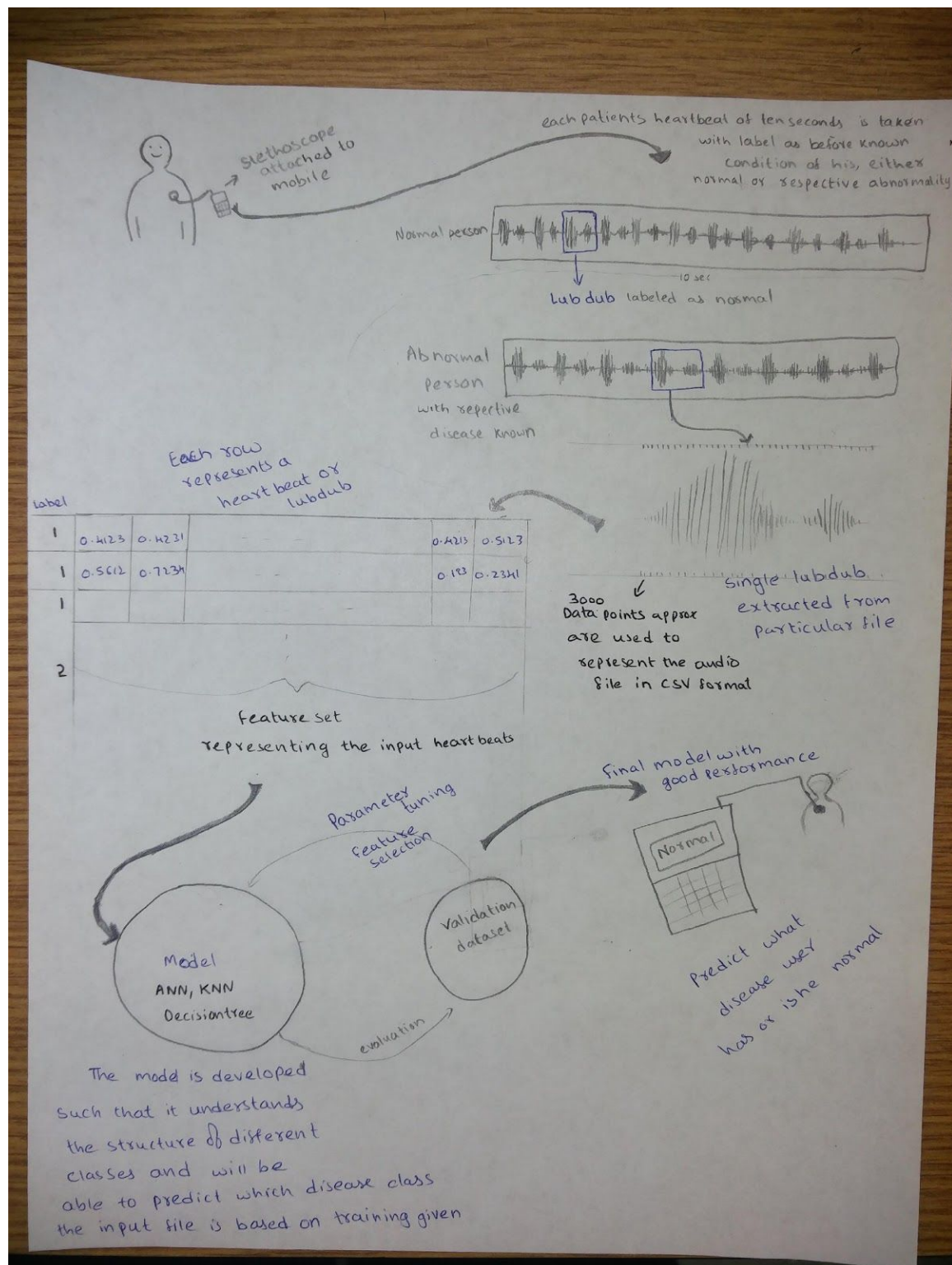
```
Train = pd.read_csv('NewTr.csv', delimiter=';', header=None)
MajorityClassdata = Train.iloc[0:38]
MinorityClassdata = Train.iloc[39:106]
#Resampling data to | address selection bias
ResampledTrain1 =resample(MajorityClassdata , n_samples = 90, random_state = 1)
ResampledTrain2 =resample(MinorityClassdata, n_samples = 10, random_state = 1)

TrainUpSampled = pd.concat([ResampledTrain1, ResampledTrain2])
Test = pd.read_csv('NewVal.csv', delimiter=';')
```

The features in this problem are the data points in CSV file which represent the heartbeat or one lub-dub each row, these files are extracted from the ten-second heartbeat and annotated according to their disease. Each row contains approximately 3000 data points which represent single heartbeat (lub-dub) as a whole. These heartbeats of diseased person would be different from a normal person which helps our machine learning algorithm learn which are normal and which are abnormal to predict their disease.

The link below gives access to different heart sound files used for training the model and does also contain a clear description of those data files used from the beginning. The NewTr, NewTes and NewVal files represent latest files, which contain more number of normal and different disease files.

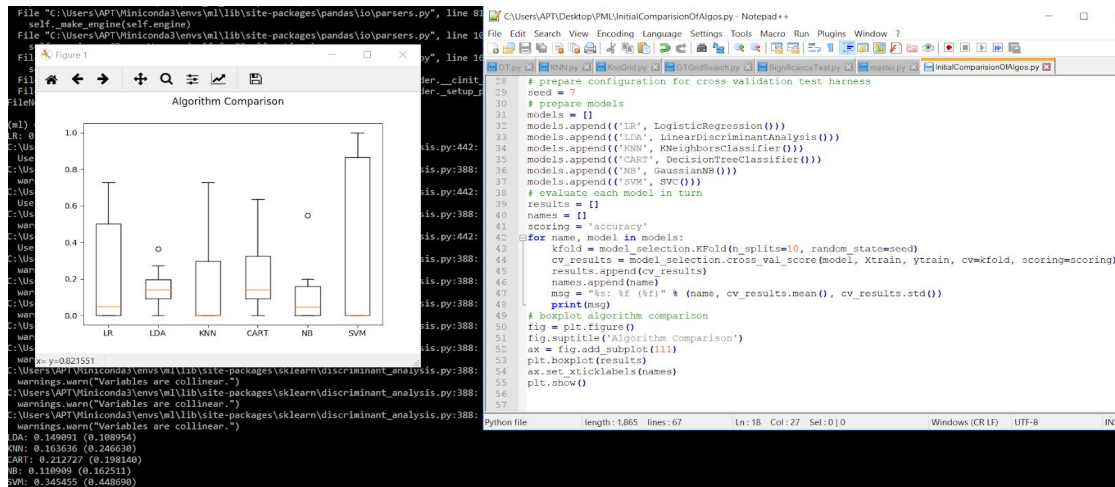
<https://github.com/aptr288/MLProjectCode/tree/master/MLHeartData>



The above picture gives an overview of data creation, feature extraction, and final model preparation.

Approach / Methods

Hypothesis: The machine learning algorithms used in this project are artificial neural networks, k nearest neighbor, and decision tree. I trained them on the heartbeat files with data points in each row representing each lub-dub as features.



The above picture shows the initial search process I have done to see performances of different algorithms on my data. The link for this algorithm comparison code is given below.

<https://github.com/aptr288/MLProjectCode/blob/master/InitialComparisonOfAlgorithms.py>

As the heart beats are to be elicited from human beings, it was hard to collect a sizable number of files, so have started evaluating different algorithm's performance on existing data. I preferred KNN, ANN and Decision tree algorithms because they seemed efficient in dealing with such scarce data.

I tried to train the model such that it can learn from the training data structure to discriminate different classes and variation in different hear sounds so as it can better predict on the unseen data, whether it is a normal heartbeat or abnormal heartbeat if abnormal which kind of abnormality the user is suffering from.

As I continued to work with the above-stated algorithms I used different strategies like parameter tuning and feature engineering to increase the performance and employed oversampling and penalizing majority class with weights to manage the data imbalance.

The link below gives access to all the codebase which I have implemented so far.

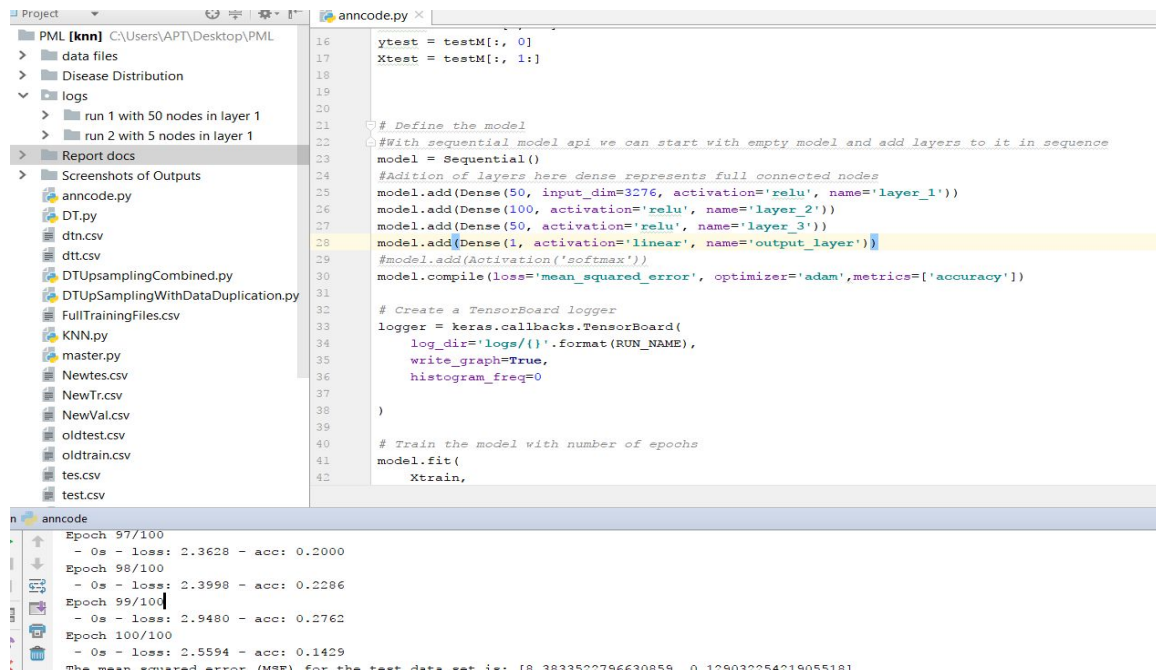
<https://github.com/aptr288/MLProjectCode>

Design / Methods

Artificial neural networks with Keras

I started the project with Artificial neural networks to see how well the algorithm can learn from my dataset. I used Keras to abstract all the complexities to build a neural network on tensor flow.

With Keras I have built a basic neural net with two hidden layers and used different activation functions like Relu, Linear, SoftMax and tried different optimizers like 'adam' and 'sgd'. As there was no significant increase in performance tried changing the number of hidden layers and even number of nodes in each layer.

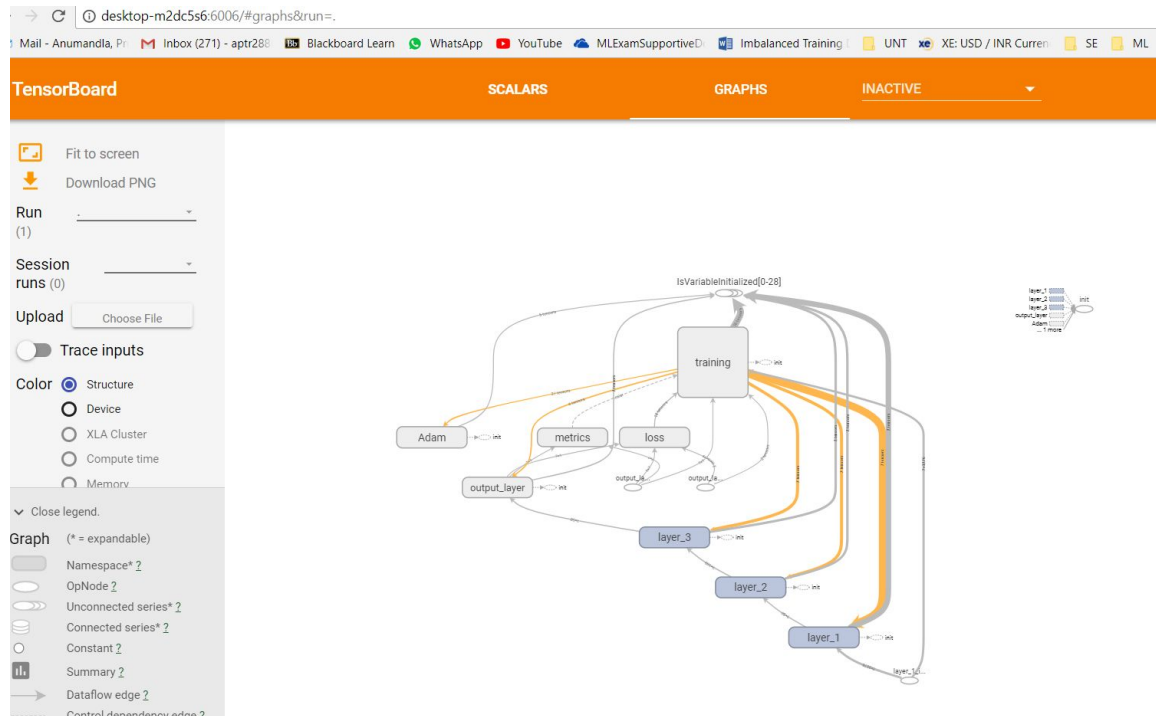


The screenshot shows an IDE with a project named 'PML [knn]' and a file named 'anncode.py'. The code defines a sequential model with three layers: a hidden layer with 50 nodes using 'relu' activation, another hidden layer with 100 nodes using 'relu' activation, and an output layer with 1 node using 'linear' activation. The model is compiled with 'mean_squared_error' loss and 'adam' optimizer. A TensorBoard logger is set up to log training metrics. The model is trained for 100 epochs. The output window shows the training progress, including loss and accuracy for each epoch, and the final mean squared error (MSE) for the test data.

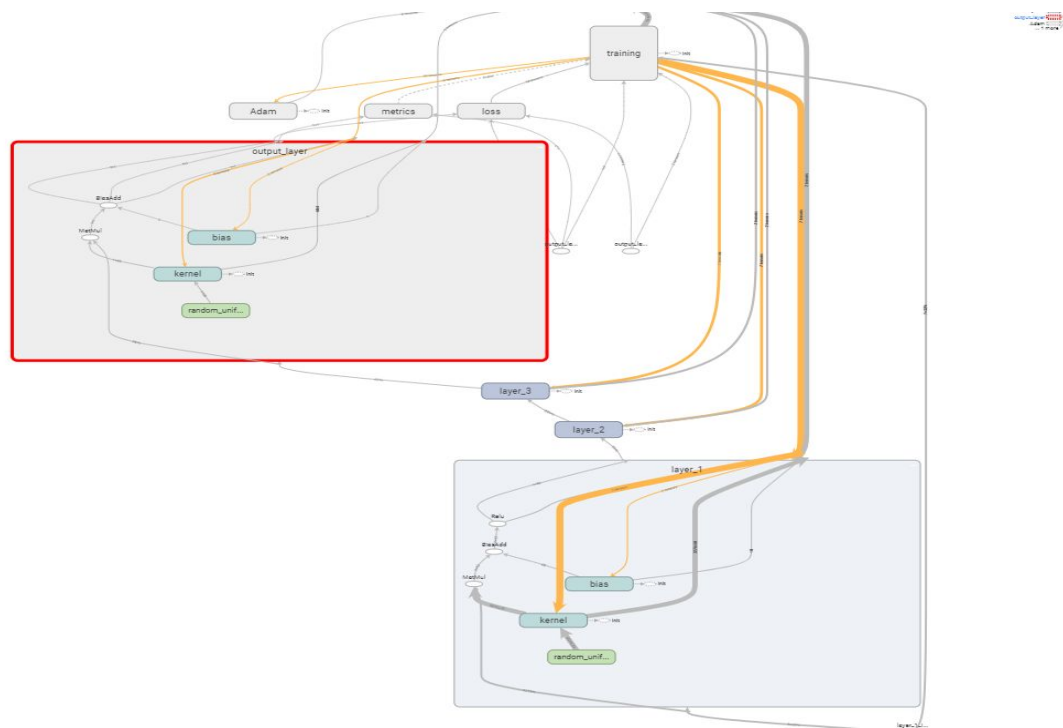
```
16 ytest = testM[:, 0]
17 Xtest = testM[:, 1:]
18
19
20
21 # Define the model
22 #With sequential model api we can start with empty model and add layers to it in sequence
23 model = Sequential()
24 #Addition of layers here dense represents full connected nodes
25 model.add(Dense(50, input_dim=3276, activation='relu', name='layer_1'))
26 model.add(Dense(100, activation='relu', name='layer_2'))
27 model.add(Dense(50, activation='relu', name='layer_3'))
28 model.add(Dense(1, activation='linear', name='output_layer'))
29 #model.add(Activation('softmax'))
30 model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
31
32 # Create a TensorBoard logger
33 logger = keras.callbacks.TensorBoard(
34     log_dir='logs/{}'.format(RUN_NAME),
35     write_graph=True,
36     histogram_freq=0
37 )
38
39
40 # Train the model with number of epochs
41 model.fit(
42     Xtrain,
```

Epoch 97/100
- 0s - loss: 2.3628 - acc: 0.2000
Epoch 98/100
- 0s - loss: 2.3998 - acc: 0.2286
Epoch 99/100
- 0s - loss: 2.9480 - acc: 0.2762
Epoch 100/100
- 0s - loss: 2.5594 - acc: 0.1429
The mean squared error (MSE) for the test data set is: 18.3833500706630858 0.10903025401905181

Finally, after getting more data tried again changing all the parameter with new data but results were not satisfactory. Though the neural networks didn't give superior results, I learned few things like creating and altering the hidden layers and visualizing neural networks using the tensor board.



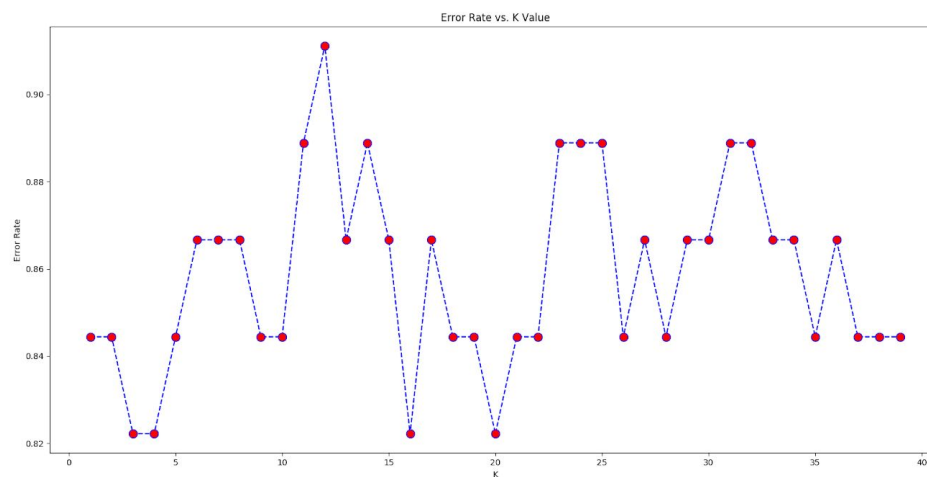
The above picture shows how each layer is connected and what are all functions involved in neural networks I implemented.



The internal structure of input layer and an output layer.

KNN, Decision Tree, and few ensemble methods

After working with ANN, I preferred to work with both KNN and decision tree algorithms simultaneously in sci-kit learn. Firstly, let me discuss different approaches I used in KNN.



With an initial set of data tried to visualize the error rate with different values of k as shown above.

Manually trying all the combinations of parameters seemed time-consuming, so I used sci-kit learn's GridSearchCV function giving different parameters like distance function metrics, number of folds for cross-validation and different type of weights. This inbuilt function gave a best combination of parameters to get better performance as shown below.

```
C:\Users\APTV\Miniconda3\envs\ml\lib\site-packages\sklearn\tree\tree.py:282: DeprecationWarning: The parameter 'max_depth' is deprecated in favor of 'max_depth'.
DeprecationWarning)
C:\Users\APTV\Miniconda3\envs\ml\lib\site-packages\sklearn\tree\tree.py:282: DeprecationWarning: The parameter 'min_samples_split' is deprecated in favor of 'min_samples_split'.
DeprecationWarning)
Best accuracy possible and best parameters to achieve them
0.3584905668377358 ('max_depth': 3, 'min_impurity_split': 1, 'min_samples_leaf': 1, 'min_samples_split': 2)

(al) C:\Users\APTV\Desktop\Python\python knngrid.py
C:\Users\APTV\Miniconda3\envs\ml\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: The classes and functions are moved. Also note that the interface of the new CV iterators are different.
DeprecationWarning)
C:\Users\APTV\Miniconda3\envs\ml\lib\site-packages\sklearn\cross_validation.py:42: DeprecationWarning: The classes and functions are moved. This module will be removed in 0.20.
DeprecationWarning)
Traceback (most recent call last):
  File "knngrid.py", line 30, in <module>
    cv = cross_validation.StratifiedKFold(n_splits=2, random_state=None, shuffle=True)
TypeError: __init__() got an unexpected keyword argument 'n_splits'

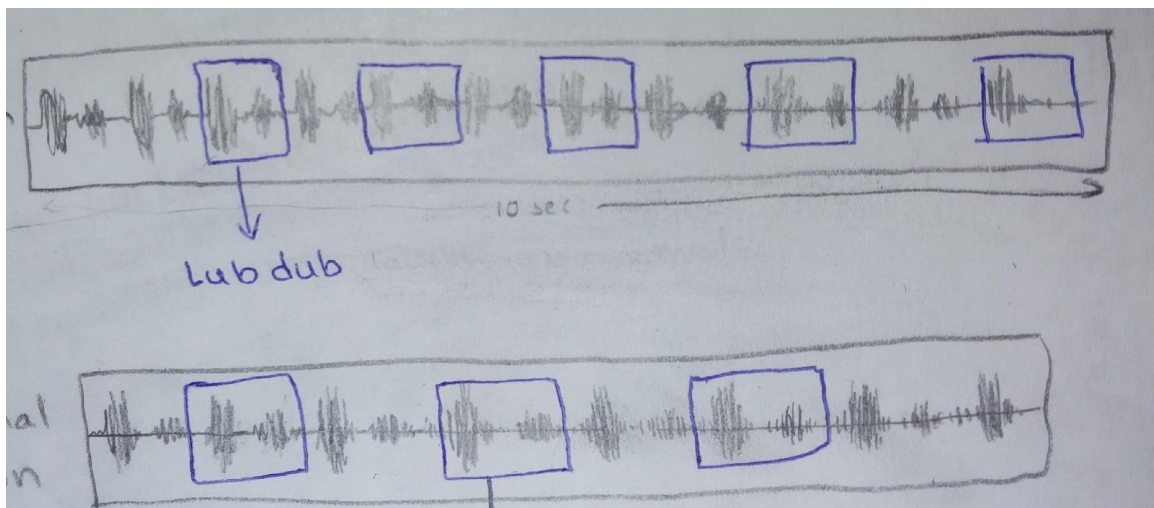
(al) C:\Users\APTV\Desktop\Python\python knngrid.py
C:\Users\APTV\Miniconda3\envs\ml\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: The classes and functions are moved. Also note that the interface of the new CV iterators are different.
DeprecationWarning)
C:\Users\APTV\Miniconda3\envs\ml\lib\site-packages\sklearn\cross_validation.py:42: DeprecationWarning: The classes and functions are moved. This module will be removed in 0.20.
DeprecationWarning)
Best accuracy possible and best parameters to achieve them
0.3113207547109811 ('metric': 'minkowski', 'n_neighbors': 9, 'weights': 'uniform')

12 Train = pd.read_csv('NewTr.csv', delimiter=';', header=None)
13 Test = pd.read_csv('NewVal.csv', delimiter=';')
14
15 trainM = Train.as_matrix()
16 testM = Test.as_matrix()
17 ytrain = trainM[:, 0]
18 Xtrain = trainM[:, 1:]
19
20
21 ytest = testM[:, 0]
22 Xtest = testM[:, 1:]
23 nFolds = 4
24 random_state = 1234
25 metrics = ['minkowski', 'euclidean', 'manhattan']
26 weights = ['uniform', 'distance'] #10.0*np.arange(-5,4)
27 numNeighbors = np.arange(5,10)
28 param_grid = dict(metric=metrics, weights=weights, n_neighbors=numNeighbors)
29 cv = cross_validation.StratifiedKFold(ytrain, n_folds = 3)
30 grid = GridSearchCV(neighbors.KNeighborsClassifier(), param_grid=param_grid, cv=cv)
31 grid.fit(Xtrain, ytrain)
32 knn_preds = grid.predict_proba(Xtest)[:, 1]
33
34
35 print("Best accuracy possible and best parameters to achieve them ")
36
37 print (grid.best_score_, grid.best_params_)
38
39
40
Python file length: 1,238 lines: 40 Ln: 24 Col: 11 Sel: 0/10 Windows (CR LF) UTF-8
```

Link to the grid search code is given below.

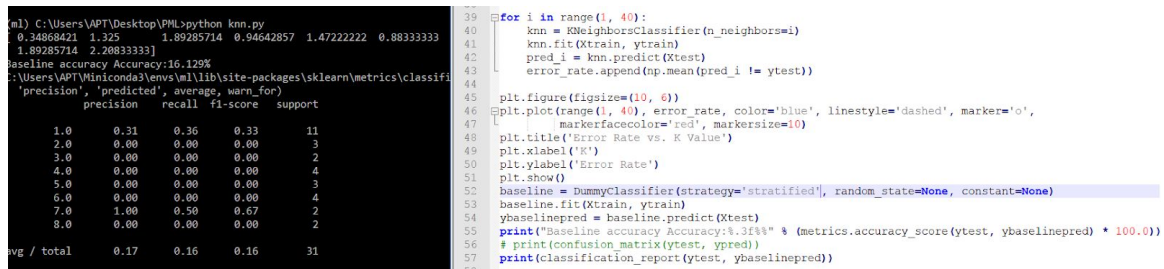
<https://github.com/aptr288/MLProjectCode/blob/master/KnnGrid.py>

Training the model with decision tree also gave insignificant results so tried few ensemble methods like boosting and random forest. Also used grid search for getting a better combination of parameters as shown below.

[illegible]

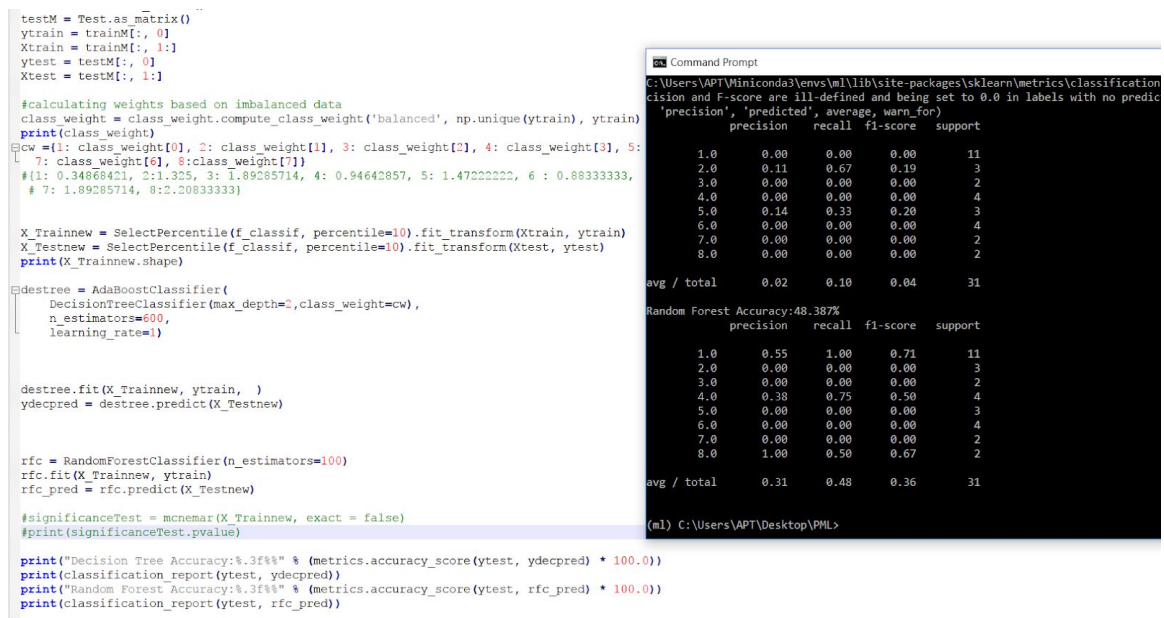
As the number of persons with each disease classification is limited, the data set is obtained by extracting each alternative lub-dub from the 10 seconds file of a heartbeat so as they are independent and some of the lub-dubs of the same person are included in validation file. But for the test file, the lub-dubs are extracted from a different person which are not used by training or validation file so as we can better evaluate the performance of the model on unseen data.

The baseline solution for these datasets has an f1-measure of 0.16 which is obtained by using Sci-kit learns dummy classifier, the heuristic or strategy used is “stratified”: which generates predictions by respecting the training set’s class distribution. The results are much better with increasing data set compared to baseline solution, the state-of-the-art solution for this problem would be results stated in the literature with accuracy more than 90%.



Feature Analysis

I have done feature selection in two ways to check how the model performs with a different set of features. Firstly, I used sci-kit learns inbuilt feature selection functions like Select Percentile, GenericUnivariateSelect and Recursive feature elimination which tried selecting features based on the heuristics they follow like 'SelectPercentile technique removes all but a user-specified highest scoring percentage of features with user-specified percentile as shown below'.



Secondly used gridsearchcv method to manually check whether there is difference in systems performance by selecting few set of features each time and running grid search with the range specified but the variation in performance of the model was not that significant.

Results

Algorithms	F1-Score with Initial dataset	F1-Score with Latest dataset	F1-score with oversampling and other parameter tuning	F1-measure with Train + Validation data On Test
ANN	20%*	25%	27%	27%
KNN	0.13	0.21	0.24	0.51
Decision tree	0.14	0.24	0.30	0.30
Random forest	0.14	0.26	0.35	0.37

*f1-measure metric is removed from Keras, so using accuracy as evaluation metric only for Artificial Neural Networks.

*Initial dataset consists of less number of normal and disease files whereas latest files consist of more number of samples per class split into the train, validation, and test.

ANN Results

Let me start with the performance of the artificial neural networks from where I have started the project. While using ANN I tried not only different parameter like a number of hidden layers or different activation functions, but also a different number of epochs to train the model. The change in accuracy for each epoch can be seen in below pictures.

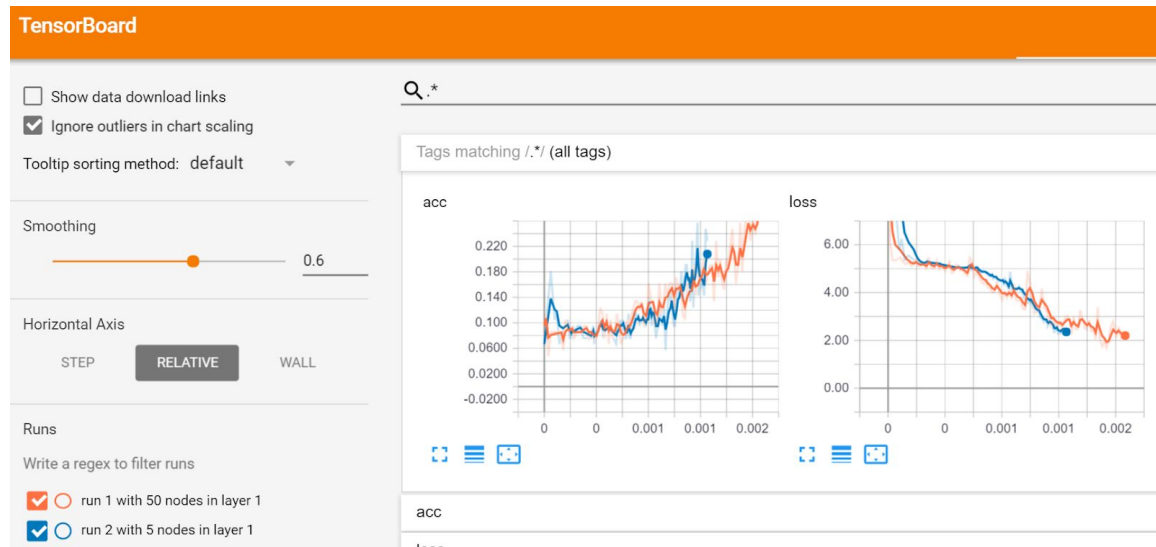

```

anncode
C:\Users\APT\Miniconda3\envs\ml\python.exe C:/Users/APT/Desktop/FML/anncode.py
Using TensorFlow backend.
2017-12-11 20:27:38.667401: I C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\platform\cpu_feature_
Epoch 1/100
- 1s - loss: 10.8094 - acc: 0.0803
Epoch 2/100
- 0s - loss: 7.6311 - acc: 0.1314
Epoch 3/100
- 0s - loss: 7.3178 - acc: 0.0949
Epoch 4/100
- 0s - loss: 5.5998 - acc: 0.1241
Epoch 5/100
- 0s - loss: 6.0321 - acc: 0.1241

Epoch 99/100
- 0s - loss: 2.5705 - acc: 0.2044
Epoch 100/100
- 0s - loss: 2.2845 - acc: 0.2482
The mean squared error (MSE) for the test data set is: [2.0736699104309082, 0.22580644488334656]
The disease to be classified are- $[[ 2.]
[ 2.]
[ 1.]
[ 3.]
[ 1.]
[ 2.]
[ 2.]
[ 3.]
[ 2.]
[ 3.]

```

Tried to visualize the graph of accuracy and loss through the tensor board which is shown below. Variation in the curve with the change in the number of nodes can be observed.



KNN and Decision Tree Results

The initial set of data is very small so have split data into train and test, done the parameter tuning using the k cross-validation. As the data is imbalanced I preferred f1-measure as the evaluation metric and the results of the initial scarce data are as shown below. Which has an f1-score of 0.13 approximately for all the three algorithms.

```

-----Initial evaluation with cross validation -----
KNN Accuracy with 95 percent confidence interval: 0.14 (+/- 0.25)
Decision Tree Accuracy with 95 percent confidence interval: 0.14 (+/- 0.15)
Random Forest Accuracy with 95 percent confidence interval: 0.29 (+/- 0.20)
-----Final evaluation results on Test data -----
KNN Accuracy:17.778%
C:\Users\APT\Miniconda3\envs\ml\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: Precision and F-score
'precision', 'predicted', average, warn_for)
precision recall f1-score support
0.0 0.20 0.29 0.24 7
1.0 0.00 0.00 0.00 7
2.0 1.00 0.14 0.25 7
3.0 0.00 0.00 0.00 6
4.0 0.15 0.80 0.26 5
5.0 0.00 0.00 0.00 3
6.0 0.33 0.17 0.22 6
7.0 0.00 0.00 0.00 4
avg / total 0.25 0.18 0.13 45
Decision Tree Accuracy:11.111%
precision recall f1-score support
0.0 0.00 0.00 0.00 7
1.0 0.00 0.00 0.00 7
2.0 0.00 0.00 0.00 7
3.0 0.40 0.33 0.36 6
4.0 0.10 0.20 0.13 5
5.0 0.00 0.00 0.00 3
6.0 0.18 0.33 0.24 6
7.0 0.00 0.00 0.00 4
avg / total 0.09 0.11 0.09 45
Random Forest Accuracy:24.444%
precision recall f1-score support
0.0 0.00 0.00 0.00 7
1.0 0.20 0.14 0.17 7
2.0 0.40 0.29 0.33 7
3.0 0.29 0.33 0.31 6
4.0 0.20 0.20 0.20 5
5.0 0.00 0.00 0.00 3
6.0 0.40 0.67 0.50 6
7.0 0.25 0.25 0.25 4
avg / total 0.23 0.24 0.23 45

```

Though tried different parameter tunings which did not improve the performance so finally considered increasing the data. Have split the increased data set into train, validation, and test. Then tried different methods like Oversampling, Grid search and feature engineering whose results attached below.

```

Train = pd.read_csv('FinalTrain.csv', delimiter=';', header=None)
Test = pd.read_csv('NewTest.csv', delimiter=';')

trainM = Train.as_matrix()
testM = Test.as_matrix()
ytrain = trainM[:, 0]
Xtrain = trainM[:, 1:]

ytest = testM[:, 0]
Xtest = testM[:, 1:]

class_weight = class_weight.compute_class_weight('balanced', np.unique(ytrain), trainM)
print(class_weight)
cw = {1: class_weight[0], 2: class_weight[1], 3: class_weight[2], 4: class_weight[3], 5: class_weight[4], 6: class_weight[5], 7: class_weight[6], 8: class_weight[7]}
ros = RandomOverSampler()

X_resampled, y_resampled = SMOTE(ratio='minority').fit_sample(Xtrain, ytrain)

error_rate = []

for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(Xtrain, ytrain)
    pred_i = knn.predict(Xtest)
    error_rate.append(np.mean(pred_i != ytest))

plt.figure(figsize=(10, 6))
plt.plot(range(1, 40), error_rate, color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()

baseline = DummyClassifier(strategy='stratified', random_state=None, constant=None)

```

Command Prompt

```

6.0 0.33 0.50 0.40 4
7.0 0.00 0.00 0.00 2
8.0 0.00 0.00 0.00 2

avg / total 0.14 0.13 0.13 31

KNN Accuracy:58.065%
precision recall f1-score support

1.0 0.50 1.00 0.67 11
2.0 0.33 0.33 0.33 3
3.0 1.00 0.50 0.67 2
4.0 1.00 0.50 0.67 4
5.0 0.00 0.00 0.00 3
6.0 0.00 0.00 0.00 4
7.0 1.00 0.50 0.67 2
8.0 1.00 1.00 1.00 2

avg / total 0.53 0.58 0.51 31

```

(ml) C:\Users\APT\Desktop\PML>

Results of KNN with on final evaluation as shown above

```

import pandas as pd
import numpy as np
from sklearn import metrics
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE, ADASYN

Train = pd.read_csv('NewTr.csv', delimiter=';', header=None)
Test = pd.read_csv('NewVal.csv', delimiter=';')

trainM = Train.as_matrix()
testM = Test.as_matrix()
ytrain = trainM[:, 0]
Xtrain = trainM[:, 1:]

ytest = testM[:, 0]
Xtest = testM[:, 1:]
X_resampled, y_resampled = SMOTE(ratio='minority').fit_sample(Xtrain, ytrain)

destree = tree.DecisionTreeClassifier()
destree.fit(X_resampled, y_resampled)
ydecpred = destree.predict(Xtest)

rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_resampled, y_resampled)
rfc_pred = rfc.predict(Xtest)

print("Decision Tree Accuracy:%.3f%%" % (metrics.accuracy_score(ytest, ydecpred)) * 100)
print(classification_report(ytest, ydecpred))
print("Random Forest Accuracy:%.3f%%" % (metrics.accuracy_score(ytest, rfc_pred)) * 100)
print(classification_report(ytest, rfc_pred))

```

```

avg / total 0.08 0.13 0.09 31

Random Forest Accuracy:35.484%
precision recall f1-score support

1.0 0.39 1.00 0.56 11
2.0 0.00 0.00 0.00 3
3.0 0.00 0.00 0.00 2
4.0 0.00 0.00 0.00 4
5.0 0.00 0.00 0.00 3
6.0 0.00 0.00 0.00 4
7.0 0.00 0.00 0.00 2
8.0 0.00 0.00 0.00 2

avg / total 0.14 0.35 0.20 31

(ml) C:\Users\APT\Desktop\PML>python dt.py
Decision Tree Accuracy:29.032%
precision recall f1-score support

1.0 0.47 0.73 0.57 11
2.0 0.00 0.00 0.00 3
3.0 0.00 0.00 0.00 2
4.0 0.00 0.00 0.00 4
5.0 0.00 0.00 0.00 3
6.0 0.00 0.00 0.00 4
7.0 0.33 0.50 0.40 2
8.0 0.00 0.00 0.00 2

avg / total 0.19 0.29 0.23 31

Random Forest Accuracy:41.935%
C:\Users\APT\Miniconda3\envs\ml\lib\site-packages\sklearn\metrics\classification
d and being set to 0.0 in labels with no predicted samples.
'precision', 'predicted', average, warn_for)
precision recall f1-score support

1.0 0.41 1.00 0.58 11
2.0 0.00 0.00 0.00 3
3.0 0.00 0.00 0.00 2
4.0 0.00 0.00 0.00 4
5.0 0.00 0.00 0.00 3
6.0 1.00 0.25 0.40 4
7.0 1.00 0.50 0.67 2
8.0 0.00 0.00 0.00 2

avg / total 0.34 0.42 0.30 31

```

Results for oversampling the minority class with SMOTE method

	precision	recall	f1-score	support
1.0	0.41	1.00	0.58	11
2.0	0.00	0.00	0.00	3
3.0	0.00	0.00	0.00	2
4.0	0.50	0.25	0.33	4
5.0	0.00	0.00	0.00	3
6.0	0.00	0.00	0.00	4
7.0	0.00	0.00	0.00	2
8.0	0.00	0.00	0.00	2
avg / total	0.21	0.39	0.25	31


```
(ml) C:\Users\APT\Desktop\PML>python dt.py
```

Decision Tree Accuracy:32.258%

	precision	recall	f1-score	support
1.0	0.70	0.64	0.67	11
2.0	0.00	0.00	0.00	3
3.0	0.00	0.00	0.00	2
4.0	0.00	0.00	0.00	4
5.0	0.00	0.00	0.00	3
6.0	1.00	0.50	0.67	4
7.0	0.50	0.50	0.50	2
8.0	0.00	0.00	0.00	2
avg / total	0.41	0.32	0.35	31

Random Forest Accuracy:45.161%

C:\Users\APT\Miniconda3\envs\ml\lib\site-packages\sklearn\metrics\classification_report.py:137: UserWarning: Precision is 0.0 in labels with no predicted samples. 'precision', 'predicted', average, warn_for)

	precision	recall	f1-score	support
1.0	0.42	1.00	0.59	11
2.0	0.00	0.00	0.00	3
3.0	0.00	0.00	0.00	2
4.0	0.50	0.25	0.33	4
5.0	0.00	0.00	0.00	3
6.0	1.00	0.25	0.40	4
7.0	1.00	0.50	0.67	2
8.0	0.00	0.00	0.00	2
avg / total	0.41	0.45	0.35	31

Results for oversampling the minority class with ADASYN method

	precision	recall	f1-score	support
5.0	0.00	0.00	0.00	3
6.0	0.00	0.00	0.00	4
7.0	1.00	0.50	0.67	2
8.0	0.00	0.00	0.00	2
avg / total	0.27	0.39	0.28	31


```
(ml) C:\Users\APT\Desktop\PML>python dt.py
```

Decision Tree Accuracy:12.903%

C:\Users\APT\Miniconda3\envs\ml\lib\site-packages\sklearn\metrics\classification_report.py:137: UserWarning: Precision is 0.0 in labels with no predicted samples. 'precision', 'predicted', average, warn_for)

	precision	recall	f1-score	support
1.0	0.33	0.18	0.24	11
2.0	0.11	0.33	0.17	3
3.0	0.00	0.00	0.00	2
4.0	0.25	0.25	0.25	4
5.0	0.00	0.00	0.00	3
6.0	0.00	0.00	0.00	4
7.0	0.00	0.00	0.00	2
8.0	0.00	0.00	0.00	2
avg / total	0.16	0.13	0.13	31

Random Forest Accuracy:38.710%

	precision	recall	f1-score	support
1.0	0.43	0.91	0.59	11
2.0	0.00	0.00	0.00	3
3.0	0.00	0.00	0.00	2
4.0	1.00	0.25	0.40	4
5.0	0.00	0.00	0.00	3
6.0	0.00	0.00	0.00	4
7.0	1.00	0.50	0.67	2
8.0	0.00	0.00	0.00	2
avg / total	0.35	0.39	0.30	31

(ml) C:\Users\APT\Desktop\PML>

Results for manually oversampling the data with Resample method.

Have conducted few significance tests to make sure that the results which are obtained are significant and not just by chance. The results for the decision tree got a p-value less than 0.05 which shows that the results are significant for decision tree. The results for KNN are around 0.7 which can be interpreted as it is near borderline of significance.

```
Xtest = testM[:, 1:]
print("The mean of the dataset", Xtrain.mean())
print("The Variance of the dataset", Xtrain.var())
destree = DecisionTreeClassifier()
destree.fit(Xtrain, ytrain)
ydecpred = destree.predict(Xtest)

knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(Xtrain, ytrain)
ypred = knn.predict(Xtest)

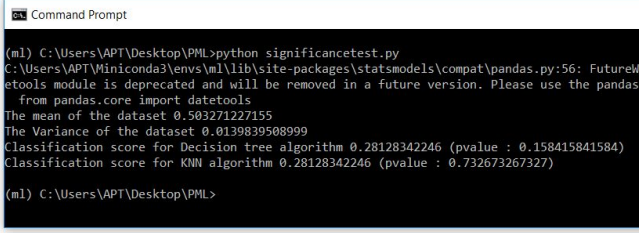
X2 = sm.add_constant(Xtrain)
est = sm.OLS(ytrain, X2)
est2 = est.fit()
#print(est2.summary())
#print(stats.normaltest(ytrain))

#significanceTest = mcnemar(X_Trainnew, exact = false)
#print(significanceTest.pvalue)

cv = StratifiedKFold(2)

score, permutation_scores, pvalueDT = permutation_test_score(
    destree, Xtrain, ytrain, scoring="accuracy", cv=cv, n_permutations=100, n_jobs=1)

score, permutation_scores, pvalueKnn = permutation_test_score(
    knn, Xtrain, ytrain, scoring="accuracy", cv=cv, n_permutations=100, n_jobs=1)
```



The link for the code is given below.

<https://github.com/aptr288/MLProjectCode/blob/master/SignificanceTest.py>

Discussion

As the size of data increases the f1-measure has increased from 0.13 to 0.3 which is so far good with such less data, as it is hard to collect diseased class files I tried upsampling the minor classes so as it would deal with data imbalance and increase the minor class files. I tried with two approaches one is sci-kit learn inbuilt oversampling techniques like SMOTE and ADASYN and then even tried to manually oversample each minor class with resampling method whose increased performance is stated in above results section. Have used class weight to calculate the weights of each class and used them to compensate the influence of majority classes on minority classes as shown below.

```
#calculating weights based on imbalanced data
class_weight = class_weight.compute_class_weight('balanced', np.unique(ytrain), ytrain)
print(class_weight)
cw = {1: class_weight[0], 2: class_weight[1], 3: class_weight[2], 4: class_weight[3], 5: class_weight[4], 6: class_weight[5],
      7: class_weight[6], 8: class_weight[7]}
#{1: 0.34868421, 2: 1.325, 3: 1.89285714, 4: 0.94642857, 5: 1.47222222, 6: 0.88333333,
# 7: 1.89285714, 8: 2.20833333}
```

In case of artificial neural networks tried different tunings like increasing number of epochs, using different activation and optimization functions and altering the hidden layers, but still, the results didn't increase even with increasing data set, I

think it needs a lot more data to properly correct the weights on the hidden layer to predict more accurately.

The term project only deals with machine learning aspects of the project like experimenting with different machine learning algorithms on the data and different approaches to increase the performance.

Conclusion

Have started the project to see how the results are for different machine learning algorithms on the heartbeat data sets and learned a lot on parameter tuning, data sampling, and feature engineering techniques by implementing them in my project. Though the results which are now with 0.3 f1-measure seems less accurate, but there is a significant increase from 0.13 to 0.3 with an increase in the dataset and different techniques employed, in future by getting more data could improve the performance of the algorithms lot more.

References:

1. MOBILE-BASED SMART AUSCULTATION Master thesis, dissertation by Anurag Chitnis
2. DETECTION AND CLASSIFICATION OF HEART SOUNDS USING A HEART-MOBILE INTERFACE doctoral dissertation by Shanti R Thiyagaraja, B.Eng
3. Chih-Chung Chang and Chih-Jen Lin, LIBSVM: A library for support vector machines ACM <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.