Time Series Classification for ECG Data

Nathan Andrew Deguara

BSc (Hons) Computer Science

Birmingham City University

Supervisor: Dr Zahraa S. Abdallah

May 2020

Final Year Honours Project

CMP6200

Abstract

Time series classification (TSC) is a field that has the potential to dramatically affect people's lives. One of the major areas that TSC can make difference in is the medical industry, particularly when it comes to the analysis of electrocardiography (ECG) data. This project investigated how time series classification could be used on the ECG data in order to provide a high accuracy without needing a large amount of computational resources. The project particularly focused on the analysis of ECG data and the techniques of data transformation and the use of ensemble classification. A new system was built from evaluating existing methods used in TSC. This system brought several high performing algorithms together in an ensemble and combined it with data transformation techniques such as Symbolic Aggregate Approximation (SAX) and Symbolic Fourier Approximation (SFA). Once the results were gathered, detailed analysis was performed to determine how much of an impact the investigated techniques had on the accuracy and runtime of the system built. The results gathered showed that the highest accuracy was achieved when using an ensemble with no data transformation and the fastest runtime was obtained when using SAX.

Acknowledgements

I would like to thank all my family and friends for their continued support throughout my studies.

I would also like to thank my supervisor Dr Zahraa S. Abdallah for all her time, dedication and support throughout the project.

Contents

Table of Figures

Table of Tables

1.0 Introduction

This project looked at time series classification for electrocardiography (ECG) data and how to improve upon existing methods for analysing ECG data using machine learning techniques. This section will state the problem and the scope of the project before going on to discuss the rational, aims and objectives.

1.1 Problem Definition

Artificial intelligence and machine learning are fields that have the potential to revolutionise the way society functions. Within this domain lies time series classification. While the accuracy of machine learning models continues to increase, one of the major factors holding it back from mainstream use is the computational resources required to run these algorithms while maintaining a high level of accuracy. Time series models generally deal with large amounts of data so are subject to this problem. This project aimed to address the issues of how a time series model could be both accurate and efficient to make it more accessible to a widespread audience.

1.2 Scope

This project specifically focused on the classification of ECG data, although the same techniques could be applied across the domain of time series classification. The key areas focused on were parameter tuning, symbolic representation and the algorithms used.

1.3 Rational

This project focused on the field of machine learning, specifically time series classification and used some of the latest methods and techniques available in order to improve existing ways of working. In the future, a similar application could be applied to hospitals to help doctors quickly identify and diagnose heart conditions which would lead to faster treatment times and provide a better chance of bringing the condition under control. Using such methods could also help hospitals save money which could then be reinvested in other areas such as training more medical staff. While it is unlikely the system built in this project would be used directly (at least not without some modification to the way the data is handled) it can be a basis to be built upon and developed further.

1.4 Aims

The aim of the project was to utilise machine learning methods to classify hearts as being healthy or unhealthy by using time series classification on electrocardiography (ECG) data and where possible diagnose the condition by using multi-variate solutions.

Time series classification has been used to analyse ECG data previously, one example is using recurrent neural networks (Singh et al., 2018). This project began by taking existing methods and then building upon them in order to increase the accuracy where possible. Where existing methods already proved to be extremely accurate the aim was to analyse the methods to find other areas of improvement. For example, if one method had a high accuracy but was only a binary classification algorithm, additional classes could be used to identify what the specific heart condition is (if there is one). This would take the algorithm from being binary classification to multi-variate (Esmael et al., 2012).

1.5 Objectives

There were five major objectives that were achieved in the project.

- Examine existing time series classification for ECG data
- Explore the different types of ECG data and how to apply them
- Understand how to improve, respond and change existing techniques for ECG classification
- Improve upon existing techniques for ECG classification
- Find the best representation of ECG data for time series classification

2.0 Literature Review

The first step was to undertake extensive background research into the key topics of the project. The core topics of this project were time series classification, symbolic representation and the algorithms used for classification. It was also important to analyse existing methods to see what had already been done in the field of time series classification (objective 1).

2.1 Time Series Classification

The core theme of the project was Time Series Classification (TSC). Time series classification is a subfield of machine learning and is defined as "a series of ordered observations made sequentially through time" (Esmael et al., 2012, page 3). This definition has been expanded upon to state that it restores "…a functional dependence between the set of possible time series and the finite set of classes…" (Smirnov & Nguifo, 2018, page 1). To summarise these statements, time series classification uses time stamp data and any other selected features to perform analysis on a given dataset and provide a classification of said data. This classification can either be a binary classification or a multi-variate one, depending on how the models were trained.

Time series classification has many possible uses in different fields. In addition to the uses in the medical industry this project focused on, one area that can benefit is robotics. Using time series classification can help to identify the type of surface that a robot is on. This is done by analysing the sensor data and helps the robot to successfully traverse a room (Lomio et al., 2019). There are also examples of using time series data to improve product quality in a steel factory (Mehdiyev et al., 2017).

Time series classification algorithms can be grouped by using the discriminatory features. It is suggested by Bagnall et al. (2017) that they should be grouped as: whole series, intervals, shapelets, dictionary based, combinations, and model based. Whole series models compare the entire time series using a similarity measure between each series. These models are usually used with nearest neighbour algorithms. Intervals compare different parts within the same time series. This is especially useful when the time series is quite long and has several different subseries and regions. Shapelets are short patterns within a time series. This is useful when the location of the pattern is not important such as when analysing ECG data. Dictionary based algorithms have a repetition of subseries. They represent data as words or letters to allow for improved pattern matching within large time series. The most common form of this symbolic aggregate approximation (see section 2.2.1 Symbolic Aggregate Approximation). Combination algorithms, also known as ensembles, are a mix of two of the previous techniques. Model based algorithms fit a generative model based on similarity. These are not often used for classification tasks (Bagnall et al., 2017).

2.2 Symbolic Representation

2.2.1 Symbolic Aggregate Approximation

The next theme for the project was symbolic representation and data transformation. This focuses on how the data will be read by the models used and the patterns they interpret and is built specifically for time series classification. The core part of this is Symbolic Aggregate Approximation (SAX). Using SAX, a time series of length x can be reduced to a string of length y where $y < x$ (Lin et al., 2007). The reduced string is a representation of the original time series and should allow machine learning models to identify patterns in the data more efficiently, thus increasing the accuracy and reducing the runtime of the algorithm.

For SAX to be used the data needs to be transformed into Piecewise Aggregate Approximation (PAA) and then to symbolise that into a discrete string (Lin et al., 2007). PAA is where the time series is

divided into segments of equal length and then the mean value of all the data points within that segment is calculated (Chakrabarti et al., 2002). This value can then be assigned a string value to represent it. Once the transformation of the time series is complete a distance measure is defined. The standard measure of this distance is the Euclidean distance, however others may be used (Lin et al., 2007). SAX uses a lower bound approximation of the Euclidean distance which is calculated by the PAA representations. If the data is transformed into a symbolic repetition, a MINDIST (minimum distance) function can be defined. This is used to help analyse the patterns in the time series data (Lin et al., 2007).

2.2.2 Symbolic Fourier Approximation

Another form of symbolic representation is Symbolic Fourier Approximation (SFA). Like SAX, SFA transforms time series data into a string representation of it. Where SFA differs is in the process of performing the approximation. SFA uses Fourier coefficients in the form of Discreate Fourier Transform (DFT) for the approximation (Schäfer, 2015). SFA also has a technique called Multiple Coefficient Binning (MCB) which defines the breakpoints within the time series (Lima et al., 2018). Lima goes on to say that normalisation of the data is optional when using SFA whereas it is required for SAX to work correctly.

2.2.3 Use of Symbolic Representation

An example of symbolic representation being used to improve a machine learning model is Co-eye by Abdallah and Medhat. This is a technique which utilises random forests to act as lenses. The data is transformed into a symbolic form and classified separately by each lens. Dynamic voting then occurs to choose the most confident classification of the data (Abdallah & Medhat, n.d.).

Prior research has proven the importance of symbolic representation when working with time series data. On average symbolic representation algorithms are 84.81% faster and use 94.48% less memory in the feature extraction phase than traditional methods (Lima et al., 2018). This has the potential to make a significant impact on the overall runtime of the system which is a major issue when working with large datasets. The reduced amount of resources required when using symbolic representation could help to make the use of time series more accessible and allow for widespread usage within industry. Furthermore, the improvements to runtime and memory usage come without any significant impact to the accuracy compared to conventional algorithms (Lima et al., 2018).

2.3 Algorithms

The final theme of the project was the algorithms that were used. To do this some existing methods were analysed. One of the primary algorithms used was a random forest which is a machine learning algorithm that works by forming an ensemble (collection) of decision trees (Shalev-Shwartz & Ben-David, 2013). Each individual tree makes a prediction of the class then the models vote for the most popular class to determine the final prediction that the model will make. This helps to avoid overfitting and increases the accuracy of the predictions (Pavlov, 2000). Furthermore, random forests are seen as one of the best algorithms to handle large volumes of information while retaining their statistical efficacy (Biau & Scornet, 2016). Random forests have had previous uses within time series classification. One example is as part of Co-eye (Abdallah & Medhat, n.d.). Another algorithm looked at was SVC. SVC is a version of SVM which is a generalised classification algorithm that in often provides reliable data. Unlike random forests, SVC has not seen widespread use for this type of classification however SVR which is another type of SVM has been used. SVR is usually used for regression rather than classification however it can be used for both types of problems (Foresti et al., 2010). Unlike SVR, the use of SVC is specifically for classification.

Some of the existing methods were specialised, custom built algorithms and others were more generic, optimised for time series classification. Some of the best existing methods are HIVE-COTE and Flat-COTE. However since these methods used super computers they were not feasible for study as part of this project so were not used (Lines et al., 2018). Another high performing algorithm is shapelet transform (ST). Shapelets are a subseries of the time series and ST works by using these subseries to detect phase independent features and measuring the similarity between shapelets (Bostrom & Bagnall, 2017). This method constantly provides high accuracy results of over 90%. Another method that has been used is a time series forest. This is a tree ensemble specialised for time series classification (Deng et al., 2013). This method provides a high accuracy rating and is one of the best for binary classification problems. BOSSVS is also a specialised time series algorithm that provides good results (Schäfer, 2015).

Deep learning algorithms have also been used for time series classification and bring a unique set of features (Ismail Fawaz et al., 2019). While the use of these methods is more exclusive than most, they can still be run on most mid and high-end computers, unlike HIVE-COTE, so will still be analysed. The most commonly used deep learning method for time series classification is long short-term memory recurrent neural networks (LSTM RNN) and it constantly outperforms most other algorithms (Bostrom & Bagnall, 2017)(Smirnov & Nguifo, 2018).

Where this project differed to the existing methods is how it went about making the predictions. It used an ensemble classifier to combine the predictions of several different algorithms before coming to a final prediction. This was then combined with data transformation to see what effect that had on the results produced.

3.0 Theory and Design

The design phase of the project focused on designing the system that was built and the experimentation that would be performed. This required forming a hypothesis, gathering the datasets, creating a methodology and setting out a timeline of the project. Another part of this was building test cases and using the findings from them to plan the build of the final system. This would cover exploring types of ECG data and understanding how to improve existing methods (objectives 2 and 3). It would also start to actually improve existing methods with the test cases (objective 4).

3.1 Hypothesis

A hypothesis was formed on what methods could be used to improve the accuracy of predictions made when analysing ECG data for possible heart conditions. The hypothesis decided upon was that the use of an ensemble classifier would improve the accuracy. It was also theorised that the use of symbolic transformation could help to improve the accuracy and runtime further.

3.2 Datasets

Seven datasets were used for this project. They can be found on the UEA and UCR time series classification repository produced by Bagnall et al. The datasets used and their respective number of classes can be seen in *table 1*.

Dataset	Number of Classes
CinCECGTorso	4
ECG200	2
ECG5000	5
ECGFiveDays	2
NonInvasiveFetalECGThorax1	42
NonInvasiveFetalECGThorax2	42
TwoLeadECG	2

Table 1: The datasets used

Each dataset was split into training and test files and pre-processing had already been performed. Each file was originally in a TSV format but were converted into CSV files for this project. The binary classification datasets have two values that are either a healthy heart or unhealthy one. The multi-variate datasets have more than two values and aim to classify a range of specific heart conditions as well as a healthy heart. Due to the variety within the datasets the system built had to be adaptable to work with all the datasets listed without needing major reconfiguration between tests.

3.3 Methodology

To ensure that the project ran smoothly it was important to have a clearly defined methodology for each stage of the project. There are several software development methodologies that could have been used however an agile approach was the most appropriate for this project. This is because new iterations of the system needed to be continuously produced with updated parameters and methods to improve the accuracy and runtime. Due to the need for a continuous test cycle, a more rigorous software development methodology such as waterfall would not be possible. The three stages of the project were design, development and testing.

3.3.1 Design Overview

The design phase of the project focused on deciding how the system would be built. The first step was to find the datasets that would be used to train the machine learning models. Using more than one dataset was ideal as this would help to evaluate the adaptability of the system. Furthermore, a

dataset with several classes would be required to train the system for multi-variate solutions required to provide a possible diagnosis. When selecting the datasets, it was important to consider how reliable the data was, the quality of it, the features gathered in the data and whether the data was binary or multi-class. All these factors determined how the system would be designed and built. Then existing systems were examined. By examining existing methods for ECG classification, tried and tested methods could be built upon to improve them further and anything that did not work so well would not be repeated.

Once the existing methods were examined the machine learning models that would be used were selected. The models chosen initially were random forest, bag of SFA symbols in vector space (BOSSVS) and support vector machine (SVM). The specific SVM used was SVC. Logistic regression was also added during the development phase. Once the algorithms were chosen several test cases were built to find the optimal parameters and what methods could be used to provide the best results. The final part of the design phase was to use the findings of the test cases to produce a UML class diagram for the final build of the system.

3.3.2 Development Overview

The development phase was about building the system itself. This phase would usually start with pre-processing the data. This would involve removing unnecessary data, handling any missing values, and ensuring there is no categorical data. It may then be appropriate to standardise or normalise that data. In this instance the datasets used were already processed so these steps were not required. The next part of the development phase was to start building the system itself. The system built upon ideas that were experimented with in the test cases built as part of the design phase. The models and methods that produced the best results from the test cases were incorporated into the final system build. The system was built in a way that allowed it to classify solutions that had both binary and multi-variate data. While the system built was optimised for ECG data, theoretically it should work for most time series classification applications with some variance in the accuracy.

3.3.3 Testing Overview

The final phase of the project was the testing and evaluation phase. The first step was to test that the system worked. Testing took the form of unit testing, integration testing and regression testing. Unit testing was carried out while the system was being built to ensure that each part of the system worked individually. Integration testing was performed after the initial version of the system had been built and before any refinement or further iterations were carried out. Integration testing checks that the individual parts work together correctly and that the entire system works. Finally, regression testing was used at the end of each iteration cycle to ensure that any changes or fixes made to the system had not broken any other parts of the system.

At the end of each iteration the accuracy and the runtime of the system was calculated. For this project the accuracy was defined as the percentage of how often the system correctly classified the data. A correct classification of the data would mean the prediction was either a true positive or a true negative. From measuring the accuracy and the runtime it could easily be seen whether the changes made improved the system or if the system should be reversed to its previous state. The aim was to have a system that provided reliable results in a short period of time (no longer than a few minutes). Once it was decided that the system was performing sufficiently well the final results were taken. The readings were each taken three times to ensure that an average could be calculated. The runtimes in particular varied as it was system dependent and affected by other system processes that were running. The accuracy was determined by using cross-validation. This is where the training and test datasets are split into several smaller datasets to train/test the

algorithm. The accuracy is then worked out by dividing the correct number of classifications by the number of instances (Kohavi, 1995).

3.4 Project Timeline

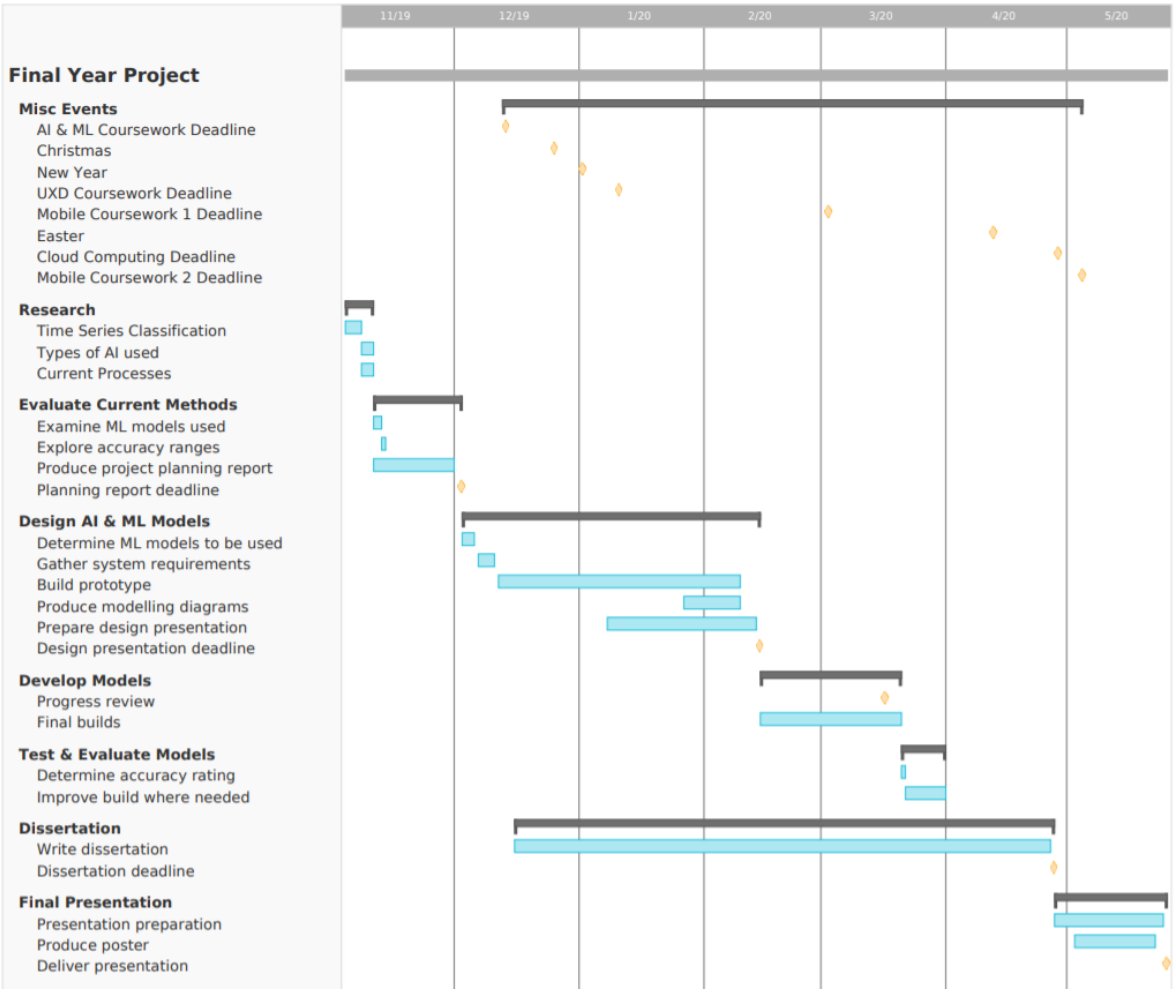

Figure 1: The Gantt chart

Gantt Chart Legend
Black bar – Parent task duration Blue bar – Individual task duration Yellow Diamonds – Milestones/Key dates

3.5 Test Cases

The main stage of the design process was to build prototypes to discover what methods and techniques worked well and could be used as part of the final system build. Since some of these prototypes only had minor changes to previous versions it is more appropriate to refer to them as test cases. As part of this stage, twenty test cases were built. Each of these focused on a particular part of the system whether that be the hyper-parameters, the use of data transformation or having an ensemble classifier. For this project the Python programming language was used due to its strong support for machine learning.

The first two test cases looked at getting a random forest to classify both binary and multi-variate data. This was done using the ECG200 and ECG5000 datasets and both test cases proved successful. The models used had the same parameters and were not optimised. They provided a proof of concept and built the foundations to expand upon. Test case 3 took the basis of the first two test cases and optimised it. It used the ECG5000 dataset with a random forest. To find the best parameters GridSearchCV was used. GridSearchCV is a method that performs a search of specified parameters in order to find the best set of parameters to use for the model. It is available as part of scikit-learn by Pedregosa et al. (2011).

A genetic algorithm is one alternative that could have been used instead of GridSearchCV. Genetic algorithms are based on the principles of evolution and natural selection. They follow the approach of survival of the fittest by only selecting the best algorithm to pass each stage (Scrucca, 2013). It works by comparing multiple algorithms by giving them a fitness score. The higher the score the more likely that algorithm will be selected. Two of the algorithms then come together and pass their features to a new algorithm. This is continued until a specified criteria is met, usually a certain number of generations has been created or a specified fitness level has been met (Scrucca, 2013). This was not used for the project as it was deemed to be too resource intensive and a similar result could be achieved more efficiently through the use of GridSearchCV.

Test case 4 focused on the use of a new algorithm. Instead of using a random forest to classify the data it used a specialised time series classification algorithm called bag of SFA symbols in vector space (BOSSVS). BOSSVS uses SFA to transform the data as part of the algorithm. Then it “describes each time series as an unordered set of substructures using SFA words” (Schäfer, 2015, page 5). The use of this algorithm was chosen as it was built specifically for time series data whereas random forest is used in many different machine learning scenarios. This gave a unique perspective on the data and provided a relatively good accuracy of 79.69% when optimised for the ECG5000 dataset. This optimisation was done in test case 5 by using GridSearchCV.

Test cases 6 and 7 utilised a new algorithm. They used SVC which is a type of support vector machine (SVM). Both used the ECG5000 dataset and test case 7 used optimised parameters for SVC which were found by using GridSearchCV. SVC was chosen as one of the algorithms used as it provides good results especially if the target classes are clearly defined and scales relatively well.

Test case 8 went back to using an unoptimised random forest algorithm but this time it used data transformation. This test case specifically focused on using SAX which is provided as part of the pyts library by Johann Faouzi (2017). It should be noted that SAX is also available through the tslearn library albeit with different parameters which was not used for this project. SAX was implemented by creating a variable which stored the SAX parameters and then using that variable to transform the input variables (X) for the training and testing data.

The transformed values were stored in new variables although the previous variables could have been overwritten. This was done so the values before and after transformation could be easily seen and compared to ensure that everything was working correctly. These transformed variables were then used in replace of the original input variables when fitting the model with the training data and when testing the data to make predictions and calculate the accuracy of the model.

Test case 9 continued with the theme of data transformation but used SFA instead of SAX. It used an unoptimised random forest with the ECG5000 dataset. SFA is also provided by the pyts library. While SFA was used as part of test cases 4 and 5 within BOSSVS, it was not directly implemented as it is carried out by the algorithm. The process for implementing SFA is similar to SAX where a variable is defined to store SFA and then that is used to transform the input variables. The transformed data is then used when fitting the model with the training data and for making the predictions with the test data.

Test case 10 introduced a new method called an ensemble classifier. Ensemble methods combine several different algorithms and take the predictions of each one. Then these individual predictions are compared to provide a final answer. There are several different ways in which these predictions can be processed (Pedregosa et al., 2011). For this project the voting classifier was used. The voting classifier has two ways of providing an answer. The first is through hard voting. This is where the answer is the class with the most individual predictions (votes). For example, if there are three models trying to predict if an apple will be red or green and two of them predict the answer is red and the other predicts it is green, the result from the voting classifier will be a prediction of a red apple.

The second way of providing an answer using a voting classifier is through soft voting. This is where each of the algorithms has a weight associated with it and as well as providing an individual prediction it provides the predicted probability of its answer. Using the example of three models trying to predict the colour of an apple, if they all had an equal weight and 1 of the algorithms said it was 100% sure the apple would be red but the other two algorithms said they were each 40% sure it would be green, the voting classifier would provide a result of red as this has the most certainty associated with it. However, this can be manipulated by associating different weights to the models. If the two algorithms that predicted green in the example had a weighting of 2 and the algorithm that predicted red had a weighting of 1 then the model would provide a result of green despite them only being 40% sure of their answer. This can be very powerful when used correctly but could also negatively affect the outcome of the results so should be used carefully.

For test case 10 a voting classifier with hard voting was used. In this test case the three algorithms experimented with so far (random forest, BOSSVS and SVC) were used. They were each used in their unoptimised forms for simplicity. Once the models were defined a vote variable which stored the voting classifier was defined and then fitted the same as other models.

The next step was to make the predictions and to calculate the accuracy scores. This was done by calling the models and the voting classifier in the cross-validation method. This was stored in a variable called scores and then displayed on the screen. The use of the voting classifier provided a high accuracy score of 94.89% for the ECG5000 dataset. This was one of the best accuracies produced so far and showed just how powerful the use of ensemble methods could be when used correctly. The ensemble methods are provided as part of scikit-learn.

Test case 11 aimed to bring data transformation and the use of ensemble methods together. It continued to use random forest, BOSSVS and SVC. It also continued to use hard voting for the voting classifier. Initially SAX was used for the model as this had shown more promising results thus far. However, when the system was tested it encountered an error when calculating the accuracy and returned a null value. Upon closer inspection the cause was determined to be the fact that BOSSVS performs SFA internally. Due to this it was attempting to perform SFA on data that had been transformed by SAX and the data had become unusable. Due to this the data transformation was changed from SAX to SFA. When the new version of the system was tested the same error was encountered. This was because BOSSVS was now attempting to perform SFA on data that had already been transformed by SFA and was again unusable. While this test case failed to produce a working model, valuable lessons were learnt. BOSSVS was not compatible with data transformation when in an ensemble method. It is possible that there is a work around however due to the time constraints of the project, a solution was not able to be pursued in depth. When it came to design the final system build, serious considerations needed to be made on what was more important to include in the system.

Test case 12 reverted the changes made in test case 11 and focused on optimising the parameters within the ensemble method. This still used hard voting and was optimised with GridSearchCV in the same manner as before. Each individual model was optimised for the ECG5000 dataset. Then the optimised models were put into the ensemble. The models used in this test case were random forest, BOSSVS and SVC. Despite the issues encountered with BOSSVS in test case 11, it was not yet decided what was going to be done about it for the final build of the system, so it was used in this test case. Using optimisation within the ensemble proved a great success and provided a good accuracy score.

Test case 13 shifted away from the ensemble method and instead solely focused on the random forest algorithm again. The aim here was to find a way to make this algorithm adaptable so that it had optimised parameters for all the datasets not just the ECG5000 dataset. This was done with GridSearchCV and split into two parts. The first part was a parameter finder. This had a wider range of parameters available to choose from with a larger range of values for each one. Once the parameter finder had found the optimal parameters for each dataset it displayed them on the screen. These parameters were then put into a table to be analysed and can be seen in *table 2*.

Dataset	Bootstrap	Max depth	Minimum Samples (leaf)	Minimum Samples (split)	Number of estimators	Random state
CinCECGTorso	True	10	1	6	300	10
ECG200	True	10	1	6	100	10
ECG5000	True	10	1	6	100	10
ECGFiveDays	True	10	1	7	300	10
NonInvasiveFetalECGThorax1	True	20	1	6	200	10
NonInvasiveFetalECGThorax2	True	15	1	6	100	10
TwoLeadECG	True	10	1	6	100	10

Table 2: Random forest optimal parameters

The second part of this build was having all the optimal parameters for each dataset available in the parameter list that GridSearchCV choses from. In order to reduce the runtime of the system only the parameters that were optimal a significant amount of the time would be used. This allowed the model to provide the best possible accuracy for most of the individual datasets without having too much of an impact on the runtime of the system. This process was later carried out for each of the algorithms when used with no data transformation, with SAX and with SFA.

Test case 14 looked at the impact that cross-validation had on the models. It removed the ensemble and focused on using an unoptimised random forest for the ECG5000 dataset similar to test case 1. Where it differed was how it was trained and how the accuracy was calculated. Whereas test case 1 calculated the accuracy by using the accuracy scores method from the metrics library, test case 14 used five-fold cross-validation which is available as part of the scikit-learn library. Cross-validation is where the datasets are split further into subsets. In five-fold cross-validation the dataset is split into five subsets. The data is then trained and tested on these subsets to provide multiple accuracy scores. The average of these accuracies can then be calculated to provide a more accurate accuracy rating for the model (Kohavi, 1995). Cross-validation was found to increase the accuracy slightly.

Test case 15 implemented an optimised ensemble method using SAX. Similar to test case 11 it used random forest, BOSSVS and SVC and encountered the same error of not all the accuracies being worked out. At the time of building this test case the cause of the issue was not known. This test case did not provide any new learnings but helped to provide more insight into the underlying issue of using BOSSVS in an ensemble method with data transformation.

Test case 16 provided what was possibly the biggest breakthrough of all the tests to date. It was an ensemble method but this time the voting classifier used soft voting instead of hard voting. To recap this is where each algorithm has a weight and the combination of the algorithms weight and the probability of its answer affect how much it is considered by the voting classifier. This test case used the ECG5000 dataset and utilised a random forest and SVC. The random forest had a weight of two and SVC had a weight of one. This was decided as random forest had a higher accuracy rating than SVC so generally provided better answers. Several other weightings were used including setting them both to one. Having a weighing of two and one provided the best results and provided an accuracy of 95.04% which was one of the highest accuracies recorded so far. Most of the system was the same as when using hard voting, the major difference was in the voting classifier. It should also be noted that a new parameter had to be set in SVC. This was the probability classifier and had to be set to true in order for soft voting to work. This parameter simply made the algorithm return a probability with each of its predictions.

Test case 17 looked at using a new algorithm. It used a Naïve Bayes classifier to make its predictions. There are several types of Naïve Bayes that can be used depending on the situation. This build looked at using multinomial, gaussian and complement Naïve Bayes. At first there was an issue as the dataset contained some negative numbers which Naïve Bayes cannot handle. This was overcome by using data transformation. For this test case SAX was used and since that transformed the negative values Naïve Bayes was able to make its predictions. Complement Naïve Bayes was found to produce the best accuracy of 90.60%. This was then optimised to provide an accuracy of 91.40%. This produced good results however it had a significant drawback that it needed to be used with some form of data transformation. Both SAX and SFA worked for this however it would not be able to be used without transformation which was required for a full comparison. On reflection, it is possible this issue could have been overcome with further pre-processing such as standardisation or normalisation of the data. Alternatively, simply mapping the negative values to a value not present

in the dataset could have been a solution. However, all of these methods would have altered the original data so may not have ensured it was a fair and consistent test.

Test case 18 combined a soft voting ensemble with data transformation in the form of SAX. It used random forest, SVC and Naïve Bayes with a weighing of two, one, one respectively. Complement Naïve Bayes was used for this test case. The build produced an accuracy of 93.80% which is slightly worse than the accuracy produced by test case 16. This was not of significant concern as the accuracy could be improved with optimisation of the individual models and adjustment of the weightings however this could be said for both of the test cases.

Test case 19 continued to experiment with new combinations of algorithms in the ensemble. This time random forest, SVC and k-nearest neighbour (KNN) were used with weightings of two, one, one respectively for the ECG5000 dataset. This produced a new highest accuracy of 95.11%. This record accuracy came at a significant cost to the runtime of the system and took almost fifteen minutes to run. For most of the test cases the runtime was less than three minutes, with the exception of the parameter finder in test case 13. Since it was an improvement of only 0.07% when compared to test case 16, it was decided that KNN would not be used going forward.

Test case 20 was the final test case produced and also introduced a new combination of algorithms into the soft voting ensemble. It used random forest, SVC and logistic regression (LR) with weightings of three, two, one respectively. It used the ECG5000 dataset and provided and accuracy of 95.04% which matched test case 16. These initial results were promising as this result was achieved without any optimisation.

The test cases produced provided invaluable data. They clearly showed what methods and techniques provided the best accuracy scores and how they each impacted the runtime of the system. The learnings from the prototyping stage were then taken forward to directly influence the final build of the system. It should also be noted that the use of deep learning was briefly explored throughout this stage. The use of a recurrent neural network (RNN) with long short-term memory (LSTM) was looked into but unable to be successfully implemented in the strict timeframe of the project.

3.6 System Design

From the test cases a design for the final build of the system was created. This was done by selecting the best features from the test cases in order to maximise the accuracy and where possible provide the quickest runtime. Ultimately it was decided that an ensemble method using a soft voting classifier would be the core component of the final system. The algorithms used were random forest, SVC and logistic regression. They would each use the optimised parameters which were found from the parameter finder built as part of test case 13. The system would be tested with no data transformation, with SAX and then with SFA. Then the accuracy and runtime for each could be compared. The Unified Modelling Language (UML) class diagram can be seen in *figure 2*.

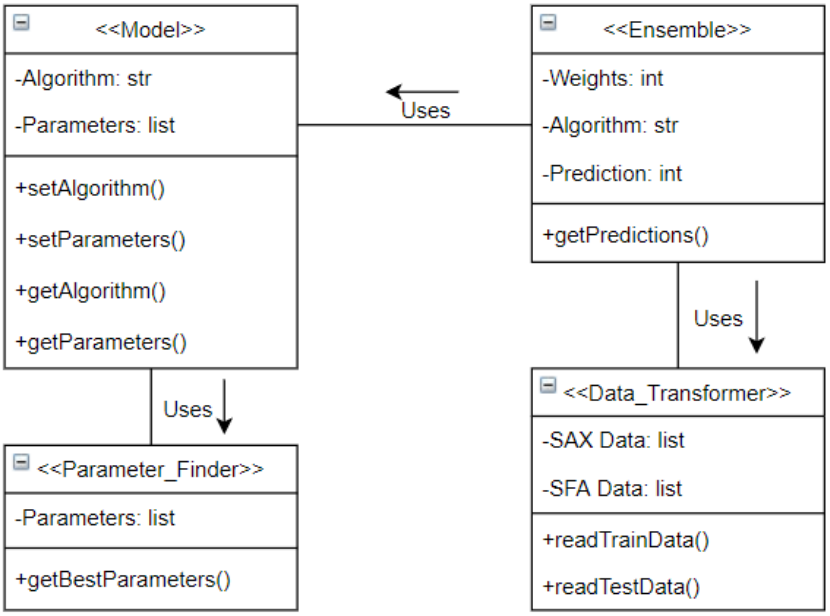

Figure 2: The UML class diagram for the system

The model class would contain each of the algorithms and their parameters. These parameters would be found by the parameter finder class which would contain the list of parameters for GridSearchCV to use. The Ensemble would bring all the models together and contain the voting classifier. The data transformer class would transform the original data into either SAX or SFA if data transformation was being used for that test. All of this would be brought together in a main run section of the script.

3.7 Experiment Design

Due to the project being research focused and about experimentation, it seemed practical to design how the experiments would be carried out in addition to the system design itself. A clearly defined plan is important to allow other researchers to repeat the findings of this project. The steps taken as part of the experiments were to start by building and testing the model. Then the accuracy would be compared to see if it was an improvement compared to previous iterations (if there was one). If it was not an improvement it would be refined and return to the testing step. If it was an improvement on the accuracy the runtime of the model would then be analysed. The runtime did not necessarily have to be an improvement but rather it would be checked to see if it was significantly worse. For

example, if an increase in accuracy of 1% increased the runtime by ten minutes that would not be deemed a suitable trade off and the model would be optimised where possible or the changes reverted. Once this stage was passed the findings were recorded. This would be repeated several times for each dataset. This process can be seen in *figure 3*.

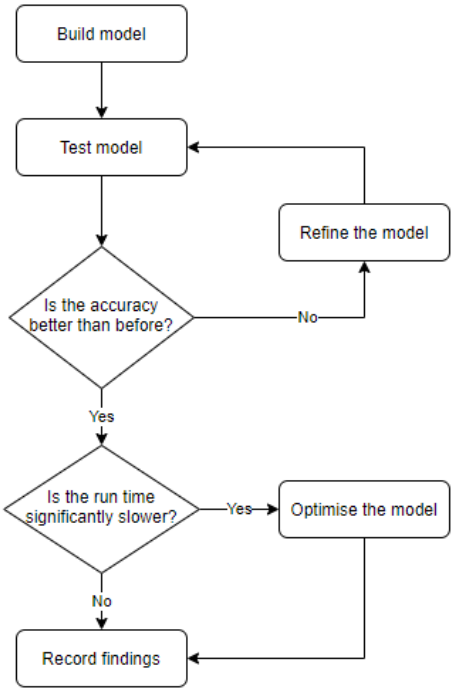

Figure 3: The experimentation flow chart

The repeated readings of the runtime were important as the runtime of the system would be affected by several factors. One factor would be the hardware specifications of the device used. This remained constant throughout the experiment. More information on the specifications of the device used can be found in section 4.5 Evaluation. Another factor that could have affected the results of the runtime was the background processes that were taking place on the device. To keep this to a minimum only the Python IDE used would be open throughout the experiment process. The IDE used was Spyder which is included in Anaconda 3. Taking repeated readings allowed an average runtime to be calculated. The accuracy of the system for each dataset would remain constant for each repeat.

4.0 Development and Experimentation

The development phase of the project involved building, testing and refining the final build of the system. It also involved gathering the results in the form of the accuracy and runtime for each dataset. The aim was to improve upon existing methods (objective 4) and find the best representation of the data (objective 5).

4.1 Building the System

The first part of the development phase was to build the system. Originally it was planned to have four classes but during the refinement phase it was decided to reduce this to three. More information on this is available in section 4.3 System Refinement. The three classes implemented were data transformation, models and the ensemble class. Additionally, there were sections that fell outside of the main classes. This included the library imports, reading the datasets and putting them into a data frame, and the main run section of the code which called the classes and calculated the accuracy and runtime of the system.

The libraries used by the system were pandas, scikit-learn, pyts, warnings and time. The warnings library was used to filter out any minor warnings that were not system critical to ensure the output terminal remained clear for important information. This is particularly useful when using third party libraries in python. The time library was used to calculate the runtime of the system. This was done with the clock method and by storing the system time when the program was initiated in a variable called start_time. At the end of the program this value was subtracted from the current system time to calculate the runtime of the system in seconds.

The next part of the system read the train and test CSV files for the dataset being used. This was done by using the read_csv method that is present in the pandas library. Each of these was then stored in a variable for later use. For each test the file path had to be manually changed to match the dataset being used.

Once the data was stored it needed to be split into input variables and target values. Traditionally input values are assigned to the variable X and target values are y. Since there are two values for each X and y these were called Xtr for the training input variables and Xte for the testing input variables. The same naming convention was used for the y values. The input variables and target values were split by using the iloc method from pandas.

After the data was ready to be used by the system the classes were defined. In depth information on each class can be seen in their dedicated sections (4.1.1 Data Transformation, 4.1.2 Models and 4.1.3 Ensemble). The final part of the program was the main.

The main run section of the code called the classes and brought the system together. It calculated the accuracy of each individual model and the system as a whole for each dataset. It also calculated the overall runtime for the system. The accuracy was calculated using five-fold cross-validation and stored in a variable called scores. This returned five individual accuracies (one for each fold). The scores variable was then printed to be displayed on the screen. This used the mean value of the accuracies. This process was repeated for each algorithm and the ensemble.

For cross-validation the algorithm used is called first. Then the input variables are called. The input variables changed depending on whether SAX, SFA, or no data transformation was taking place for that set of tests. The next variable called was the target value. This remained consistent throughout and did not change depending on the use of data transformation. Then the type of scoring was specified. In this case the accuracy was used however the F1 score, precisions and recall can all be

used for classification problems. The accuracy metric was determined to be the best fit to achieve the aim and objectives stated and is also one of the most commonly used metrics. Finally, the number of folds for cross validation was specified. As stated earlier five-fold cross-validation was used. The last part of the main calculated the overall runtime of the system which was mentioned earlier.

4.1.1 Data Transformation

The data transformer class was responsible for transforming the input variables into either SAX or SFA. The first part of this class was to define two variables to store the SAX and SFA classifiers and their parameters. For SAX, the parameters stated were strategy and alphabet. The strategy affects the width of the bins and was set to uniform which means that all the bins will have an equal width. Alphabet states what form the symbolic representation should take. By default, this is the Latin alphabet however for this project an ordinal alphabet was used. This is where integers are used as the representation, but it achieves the same result. The reason ordinal was used is because some of the algorithms used could only handle numerical values and were not able to perform classification on string input variables.

For SFA only the alphabet parameter was stated. This was set to ordinal for the same reason as SAX. Once the variables were defined, fitting could occur to transform the input variables (Xtr and Xte). This was done for both SAX and SFA and these new values were stored in a new variable. On each test of the system only one of SAX or SFA, and in some cases neither, would be used. They were both defined here to speed up and simply the testing process.

4.1.2 Models

The models class was the combination of the model and parameter finder classes. This class defined the models to be used and gave a list of optimal parameters to choose from by using GridSearchCV. The algorithms used were random forest, SVC and logistic regression. These algorithms were chosen as they provided a high accuracy, had a relatively short runtime and were highly adaptable. While BOSSVS and KNN were both found to provide a high accuracy in the test cases, they did not meet all three of the criteria set. BOSSVS was not adaptable as it could not be used in the ensemble when data transformation was used. KNN had a long runtime compared to other algorithms even when unoptimised. Due to these facts they were not included in the final build however the three algorithms chosen were sufficient for the task at hand. The optimal parameters for each dataset was found using the parameter finder that was built as part of the test cases. The optimal parameters were found to differ slightly depending on if data transformation was used.

The model using the random forest algorithm had consistent values for most of its parameters as they were found to have little variation between datasets. The only field which greatly differed was the number of estimators used by the model. As such, there were three options for the model to choose from. Once the parameters were stored as a list in a variable named parameters, GridSearchCV found the best parameters for the dataset being used. The model was then fitted with the data.

The model using SVC also had several consistent parameters but two of them had significant differences between datasets. These were the C value and the kernel type. The rest of the model followed the same process as random forest where GridSearchCV was used to find the best parameters for the dataset in used and then the model was fitted.

The final part of this class was for the logistic regression model. Logistic regression was found to only have a few parameters and most of them were dependent on the value of other parameters. As such

using optimisation via GridSearchCV was not possible. It was decided that these parameters would not be necessary, and the only parameter defined for logistic regression was its random state.

4.1.3 Ensemble

The ensemble class brought all the models together in a voting classifier. First the models, or estimators, were called from the models class. Then weights were assigned to each of the models. Random forest was given a weighting of two, SVC a weight of two and logistic regression a weight of one. These values were chosen as random forest and SVC generally provided a higher accuracy than logistic regression so their answers should be trusted more. The next step was to define the voting classifier. Soft voting was used for the classifier as it was proven to be significantly better in the test cases. Finally, the voting classifier was fitted with the data.

4.2 Testing

Once the system was built, it needed to undergo testing. Due to the extensive prototyping work done with the test cases, there were no major errors encountered in the final build of the system and everything worked as planned. Once it was determined there were no syntax or runtime errors, it was checked for logical errors. Some of the key areas checked was the data transformation and the accuracy functions. These areas appeared to be working as intended so the system was deemed working and ready for refinement.

4.3 System Refinement

After the system was built and tested, refinements were made to improve the system where possible. While most of the tuning of parameters and settings was done before the system was built in the test cases, some addition tuning took place after the final build was complete. Additionally, it was decided that the parameter finder and models classes would be combined. This was to optimise the system and reduce the number of classes that needed to be called when running the system. This also made the code simpler to read and was more a code quality change rather than one that would have a dramatic effect on the system. Individually the two classes were only a few lines of code and it was decided it was not worth having separate classes when they were both highly integrated into one another to begin with.

4.4 Results

The results of the system can be seen in *tables 3, 4* and *5*. They show the accuracy of each algorithm individually and combined as part of an ensemble. It also shows the average runtime which was calculated by measuring the runtime of the system three times and calculating the mean value. The runtime is the amount of time taken for the entire system to run which involves the parameters being found and fitting to occur on the dataset in use.

Dataset	Random Forest (%)	SVC (%)	Logistic Regression (%)	Ensemble (%)	Average Runtime (Seconds)
CinCECGTorso	97.97	99.93	60.87	99.93	526.36
ECG200	81.00	84.00	85.00	88.00	10.43
ECG5000	95.02	95.38	94.04	95.47	202.98
ECGFiveDays	98.37	100.00	99.88	100.00	24.18
NonInvasiveFetalECGThorax1	86.56	94.20	90.53	93.44	461.68
NonInvasiveFetalECGThorax2	89.47	95.37	93.18	94.71	397.64
TwoLeadECG	98.07	99.39	98.42	99.21	25.74
Average	**92.35**	**95.47**	**88.85**	**95.82**	**235.57**

Table 3: Results using no data transformation

Dataset	Random Forest (%)	SVC (%)	Logistic Regression (%)	Ensemble (%)	Average Runtime (Seconds)
CinCECGTorso	96.30	96.30	44.13	96.81	440.62
ECG200	81.00	79.00	82.00	84.00	10.32
ECG5000	94.29	94.62	93.02	94.64	111.65
ECGFiveDays	96.87	97.79	95.36	97.45	18.80
NonInvasiveFetalECGThorax1	74.30	77.56	77.76	79.19	285.15
NonInvasiveFetalECGThorax2	81.58	83.10	83.46	84.68	281.12
TwoLeadECG	92.72	93.86	92.89	93.77	19.53
Average	**88.15**	**88.89**	**81.36**	**90.87**	**166.74**

Table 4: Results using SAX

Dataset	Random Forest (%)	SVC (%)	Logistic Regression (%)	Ensemble (%)	Average Runtime (Seconds)
CinCECGTorso	90.43	81.09	39.86	82.03	426.05
ECG200	83.00	80.00	80.00	82.00	10.47
ECG5000	94.11	94.47	92.44	94.31	151.98
ECGFiveDays	99.65	99.42	99.19	99.53	19.23
NonInvasiveFetalECGThorax1	84.27	78.68	75.32	79.95	658.42
NonInvasiveFetalECGThorax2	87.43	82.85	80.92	84.07	609.86
TwoLeadECG	98.33	99.65	93.77	99.30	23.17
Average	**91.03**	**88.02**	**80.21**	**88.74**	**276.45**

Table 5: Results using SFA

4.5 Evaluation

The accuracy ratings are generally quite high. On a few of the datasets logistic regression gets a low accuracy however random forest and SVC constantly provide very high scores. Due to this the ensemble method always provides a good result. The lowest score for the ensemble method is 79.19% which is for NonInvasiveECGFetalThorax1 when using SAX. The highest score for the ensemble is 100% when using ECGFiveDays with no data transformation.

One finding of particular interest is the fact that for some of the datasets the ensemble method provides a higher accuracy than any of the individual algorithms. This can be seen for: CinCECGTorso with SAX, ECG200 with no data transformation and with SAX, ECG5000 with no data transformation and with SAX, NonInvasiveECGFetalThorax1 with SAX and for NonInvasiveECGFetalThorax2 with SAX. In several other instances the ensemble matches the highest rated algorithm for that dataset. In a few cases the ensemble performs worse than the highest rated algorithm for that datasets but this reduction in accuracy is usually less than one percent. When averages of the accuracy for each algorithm across all of the datasets are calculated, the ensemble method always provides the best result for no data transformation and when using SAX. The only time the ensemble is not optimal is when using SFA where using a single random forest is best.

The runtime varies widely between datasets and depending on if data transformation was used. Larger datasets and datasets with more classes have a longer runtime. On average, the runtime is shortest when SAX is used and using SFA provides the longest runtimes. For the majority of the datasets the runtime is just a few minutes however the NonInvasiveECGFetalThorax1 and 2 both have very long runtimes particularly when using SFA at around ten minutes.

As stated earlier the runtime would have been affected by the hardware specifications of the device the system was tested on. The device used was a Lenovo V130. It had an intel i5-7200u processor that ran at 2.5GHz. It had 8GB of DDR4 RAM and ran Windows 10 (64-bit). Any unnecessary background tasks were stopped, and all other applications were closed.

Overall the build of the system was good and highly adaptable. Very few changes had to be made between tests. The only changes required were to the dataset that needed to be used and what data transformation was used. This was controlled by editing the values of X to use either no data transformation, SAX or SFA.

5.0 Discussion

The results gathered from the system provided a significant amount of data to analyse. In addition to this a survey was created to gather people's views on artificial intelligence and machine learning and its use in a medical environment.

5.1 Project Results and Findings

The results gathered from the project clearly show that machine learning can provide excellent results for analysing ECG data and classifying hearts. The accuracy and runtimes of the system built vary between datasets and depending on how the input data is interpreted. The system provides the highest accuracies for binary classification problems or on multi-variate datasets that only have a few classes (four or five). The outlier here is that the worst accuracy is gained from the ECG200 dataset which is a binary classification problem. This is likely to be due to the low amount of records present so there is less data to train the models on. The highest accuracies are often reached when no data transformation is used although this is not the case in some instances, namely using SFA for the TwoLeadECG dataset.

The runtime of the system is always fastest when using SAX. In some cases, this is only marginally however it is often a noticeable improvement. The datasets this is most evident on is for NonInvasiveECGFetalThorax1 and 2 where SAX is over 100 seconds quicker for both of the datasets compared to no data transformation and using SFA. When looking at the average runtime across all the datasets SAX is 68.83 seconds faster than using no data transformation and is 109.71 seconds faster than using SFA.

The final results of the system using no data transformation is an average ensemble accuracy of 95.82% and an average runtime of 235.57 seconds. The results of the system when transforming the input data using SAX is an average ensemble accuracy of 90.87% and an average runtime of 166.74 seconds. The results of using SFA on the data read by the system is an average ensemble accuracy of 88.74 and an average runtime of 276.45 seconds.

Looking at the wider picture that the results show, using SAX provides the fastest runtime with a slight cost in the accuracy. Using no data transformation provides the highest accuracies but at a slightly slower speed than using SAX. It should be noted that in most instances this time difference is not significant. Using SFA generally provides the worst accuracy with the longest runtime. Ultimately, what is more important between the accuracy and the runtime depends on context and what the system is predicting. In this case it is predicting heart conditions where the accuracy of the system is more important. However, the aim of machine learning is to improve and speed up existing processes so it should be able to accomplish its task faster than a human could or at least do it in a similar time. Finding a balance between the metrics of accuracy and time is important and can only be achieved though consultation with professionals in the field, in this case doctors, and the people it will impact, which will be the patients. Due to this a survey was made to gather people's opinions on using AI and machine learning to provide a medical diagnosis.

The survey asked four questions and took an online format. There were 40 responses by anonymous participants. The first question asked them if they had heard of the terms AI and machine learning. 45% of participants were familiar with both terms, 45% had heard of AI and 10% had heard of neither term. They were then asked what their view on AI was. 56% had a positive outlook on AI, 2.5% had a negative view and the rest were neutral on the matter. The next question asked on a scale of 1-10 how likely would they be to trust an AI system to diagnose a medical condition, 1 being not at all, 5 being equal to a doctor and 10 being fully trust. The results here were very split the

largest single rating was 5 with 22.5% of the vote. 1 and 10 each received 5% of the vote. The final question asked what could be done to improve their perception of AI and machine learning. Many of the responses said to train it on real world data, which is already the case for most machine learning. This means it is important to educate people on how AI works. A lot of responses also suggested the media had tainted people's views of AI and positive and informative media could alter public perception on the field. There were also a few responses talking about making the code open source and improving transparency of the models to clearly show how they made their decisions. One theme that was consistent across all the answers was for the system to be reliable which emphasises the need for a high accuracy.

The survey provided interesting results and showed that most of the people who took part were open to the idea that in the future a computer could help to provide medical diagnoses. While this does not provide any technical information, it does show that research into this area is justified and highlights that a key concern of possible patients is the accuracy of the system. From the results of this project the highest accuracy is obtained when using no data transformation with an ensemble method.

The results gathered have a slight contrast to the hypothesis and what was expected to happen. It was expected that the ensemble and data transformation would improve the accuracy of the predictions. In fact, the system built usually achieved better results when no data transformation was used. Furthermore, while the ensemble almost always provided the best results, it only occasionally exceeded the single best performing algorithm. This is possibly due to the parameters selected and further fine tuning of parameters and algorithm weightings is needed to determine if this is the case. Additionally, the use of more algorithms in the ensemble could improve the results further. It should be noted that the overall average ensemble accuracy is significantly greater than the accuracy of any individual algorithm.

5.2 Project Evaluation

Overall the project has been a success and provided some very interesting results. It can successfully analyse ECG data and provide a diagnosis with a relatively high accuracy in a short amount of time. It consistently matches or outperforms existing methods of time series classification for the majority of the datasets used.

One of the major limitations faced was the hardware to run the algorithms on. If a dedicated server was set up to run the algorithms, more complex analysis could be performed to increase the accuracy further and also to reduce the runtime. Another area that could be improved upon is the number of algorithms used in the ensemble. With more time, the ensemble could be expanded upon to provide better results. In particular the shapelet transform (ST) algorithm is of particular interest at it was designed specifically for time series classification and could work well for ECG data. Despite the limitations mentioned the system works well and clearly shows the advantages of different methods such as using an ensemble and data transformation within time series classification.

6.0 Conclusion

This project has found that the use of an ensemble classifier improves the average accuracy of predictions made by a system using machine learning. However, these improvements are only marginally better than the top performing algorithm within the classifier and comes with a cost to the runtime of the system. It has also found that the use of data transformation (specifically SAX) can improve the runtime of the system but that comes at a cost to the accuracy of the predictions made.

Ultimately, the project has been successful and has achieved the aim and objectives set out at the start of the project. Examining existing methods for time series classification (objective 1) was achieved from the research carried out into the topics and also when gathering ideas for the test cases. Exploring different types of ECG data and how to apply them (objective 2) was completed by looking at the different types of datasets and the classes within them. Improving, responding and changing existing methods (objective 3 and 4) was done with both the test cases and the final build of the system. Finally finding the best representation of the data (objective 5) was done by exploring the use of data transformation.

The aim has been accomplished by fulfilling all the objectives and by the fact that the system works as intended. As stated in the system evaluation, there were some limitations to the project. These include the hardware used and the number of algorithms in the ensemble. Further work could resolve these issues and allow the system to be improved further. Ultimately the system achieved everything that it set out to do and the project as a whole has been a success. The aim and objectives have been accomplished and some good results have been gathered.

7.0 Recommendations for Further Work

The work carried out in this project could be expanded upon by further study. One of the key areas that could be explored is how deep learning methods could be used to further improve the results. How could deep learning be used with data transformation and could it be incorporated into an ensemble, possibly with other deep learning methods. A deep learning algorithm that appears to be of particular interest is a recurrent neural network with long short-term memory (RNN LSTM). Another way to build upon this project would be to look at shapelet transform (ST). This is an algorithm specifically for time series classification and would work well for ECG data and has the potential to improve the results even further.

8.0 References

Abdallah, Z. S., & Medhat, M. (n.d.). *Co-eye : A Multi-resolution Symbolic Representation to Time Series Diversified Ensemble Classification*.

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *31*(3), 606–660. https://doi.org/10.1007/s10618-016-0483-9

Bagnall, A., Lines, J., William, V., & Eamonn, K. (n.d.). *UEA & UCR Time Series Repository*. Retrieved March 7, 2020, from http://www.timeseriesclassification.com/

Biau, G., & Scornet, E. (2016). A random forest guided tour. *Test*, *25*(2), 197–227. https://doi.org/10.1007/s11749-016-0481-7

Bostrom, A., & Bagnall, A. (2017). Binary shapelet transform for multiclass time series classification. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10420 LNCS*, 24–46. https://doi.org/10.1007/978-3-662-55608-5_2

Chakrabarti, K., Keogh, E., Mehrotra, S., & Pazzani, M. (2002). Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *ACM Transactions on Database Systems*, *27*(2), 188–228. https://doi.org/10.1145/568518.568520

Deng, H., Runger, G., Tuv, E., & Vladimir, M. (2013). A time series forest for classification and feature extraction. *Information Sciences*, *239*, 142–153. https://doi.org/10.1016/j.ins.2013.02.030

Esmael, B., Arnaout, A., Fruhwirth, R. K., & Thonhauser, G. (2012). Multivariate time series classification by combining trend-based and value-based approximations. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7336 LNCS*(PART 4), 392–403. https://doi.org/10.1007/978-3-642-31128-4_29

Foresti, L., Tuia, D., Timonin, V., & Kanevski, M. (2010). Time series input selection using multiple kernel learning. *Proceedings of the 18th European Symposium on Artificial Neural Networks - Computational Intelligence and Machine Learning, ESANN 2010*, *April*, 123–128.

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, *33*(4), 917–963. https://doi.org/10.1007/s10618-019-00619-1

Johann Faouzi. (2017). *pyts*. https://pypi.org/project/pyts/

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *International Joint Conference of Artificial Intelligence*, *March 2001*.

Lima, W. S., de Souza Bragança, H. L., Montero Quispe, K. G., & Pereira Souto, E. J. (2018). Human activity recognition based on symbolic representation algorithms for inertial sensors. *Sensors (Switzerland)*, *18*(11). https://doi.org/10.3390/s18114045

Lin, J., Keogh, E., Wei, L., & Lonardi, S. (2007). Experiencing SAX: A novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, *15*(2), 107–144. https://doi.org/10.1007/s10618-007-0064-z

Lines, J., Taylor, S., & Bagnall, A. (2018). Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data*, *12*(5). https://doi.org/10.1145/3182382

Lomio, F., Skenderi, E., Mohamadi, D., Collin, J., Ghabcheloo, R., & Huttunen, H. (2019). *Surface Type Classification for Autonomous Robot Indoor Navigation*. http://arxiv.org/abs/1905.00252

Mehdiyev, N., Lahann, J., Emrich, A., Enke, D., Fettke, P., & Loos, P. (2017). Time Series Classification using Deep Learning for Process Planning: A Case from the Process Industry. *Procedia Computer Science*, *114*, 242–249. https://doi.org/10.1016/j.procs.2017.09.066

Pavlov, Y. (2000). *Random Forests*. VSP.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825--2830.

Schäfer, P. (2015). Bag-Of-SFA-Symbols in Vector Space (BOSS VS). *ZIB Report*, *30*(May), 15–30. https://doi.org/10.1515/johh-2017-0044

Scrucca, L. (2013). GA: A package for genetic algorithms in R. *Journal of Statistical Software*, *53*(4), 1–37. https://doi.org/10.18637/jss.v053.i04

Shalev-Shwartz, S., & Ben-David, S. (2013). Understanding machine learning: From theory to algorithms. In *Understanding Machine Learning: From Theory to Algorithms* (Vol. 9781107057). https://doi.org/10.1017/CBO9781107298019

Singh, S., Pandey, S. K., Pawar, U., & Janghel, R. R. (2018). Classification of ECG Arrhythmia using Recurrent Neural Networks. *Procedia Computer Science*, *132*(Iccids), 1290–1297. https://doi.org/10.1016/j.procs.2018.05.045

Smirnov, D., & Nguifo, E. M. (2018). Time Series Classification with Recurrent Neural Networks. *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, 1–8.