INFO411 Assignment 2:
ECG Analysis & Classification
Report

Collaborators:
Thomas Koentges - 4501131
Megan Diputado - 7376199
Rory McRae - 6944984
Riya Alagh - 8878519

1. INTRODUCTION:

Detecting abnormalities in electrocardiogram (ECG) measures using fully automated systems has been a prominent research area due to its ability to prevent life-threatening cardiovascular diseases (Luz et al., 2016), which are the leading cause of death worldwide (Ribeiro et al., 2020). In this report, we explore the use of various machine-learning techniques to implement and optimise classification models for detecting abnormalities in ECG measurements. Based on experiments, we found that Convolutional Neural Networks (CNN), Random forests (RF), and K-Nearest Neighbour (KNN) provide the best performance out of all the models examined. As such, these models will be the main focus of this report. After describing datasets and methods in sections 1 and 2, we describe our data experiments and specific parameters in more detail in section 3 and discuss them and their application in sections 4 and 5.

Description of the dataset:

The data used for this assignment is available and sourced from Kaggle, and is derived from the MIT-BIH Arrhythmia database. The original database contains 48 half-hour excerpts of two-channel ambulatory ECG recordings from 47 different studied subjects and the respective metadata. For Kaggle, the original data was modified and each recording was split into 10-second windows. Signals were detected, normalised, padded, and, most importantly, extracted, reducing the observations to 109,446 samples. The authors of the Kaggle dataset do not detail the selection process any further, but from the number of rows in each set, we observed an 80/20 split between training and testing had been applied to the selected signals. This unfortunately means that signals from the same person may be in both the training and testing data samples, which could potentially result in biases in the evaluation process (Li and Zhou, 2016, p. 4).

The original data was divided into 18 types of heartbeats. The data on Kaggle, however, was further summarised into five different categories following the Advancement of Medical Instrumentation (AAMI) recommendations: ‘N’, Normal heartbeat; ‘S’, Supraventricular premature beat; ‘V’, Premature ventricular contraction; ‘F’, Fusion of the ventricular and normal beat; and ‘Q’, Unclassifiable heartbeat (Li and Zhou, 2016, p. 4). The last value in every row corresponds to a class: ['N': 0, 'S': 1, 'V': 2, 'F': 3, 'Q': 4].

Exploratory data analysis:

We first performed some basic exploratory data analysis (EDA), including visualising the mean values of each class of the training dataset to get a first impression (see figure 1), principal component analysis (PCA) (see figure 2), and tSNE. While basic clustering would have been possible to a degree, simply relying on dimensional scaling had major limitations as the translation to the lower dimensional space was not lossless. Since the same person might be in training and testing, we also question whether tSNE can guess people, which would make the task to train a classifier an ill-posed problem (see figure 3).

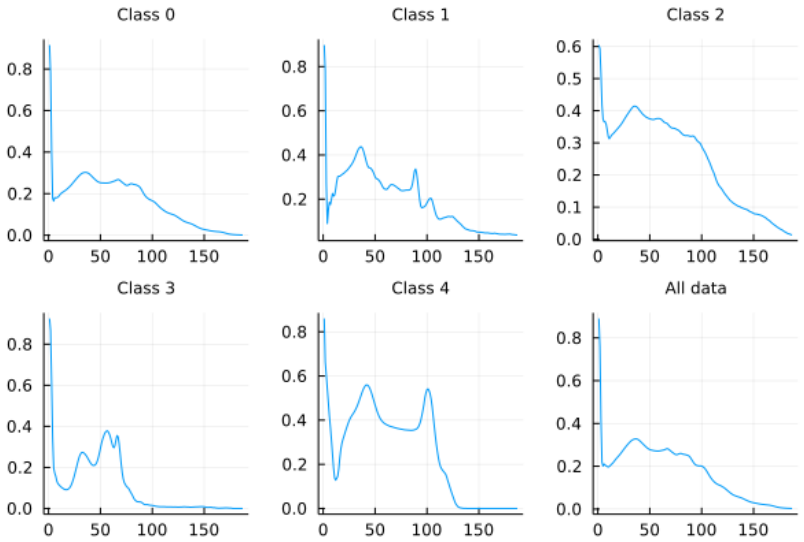

Figure 1: Plots of the mean values of each subsequent class.

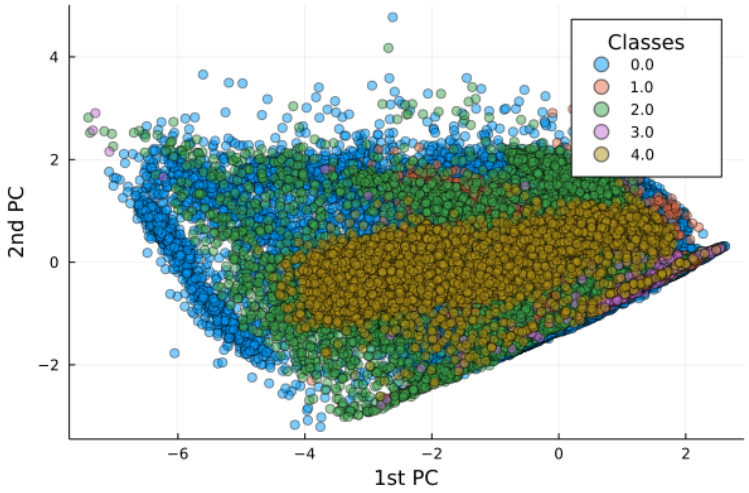

Figure 2: Principal component analysis for classification of classes.

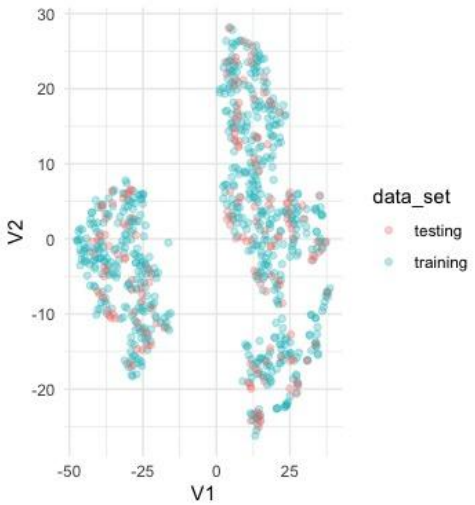

Figure 3: t-SNE representation for class 3 in training and testing. Clusters could indicate people.

The dataset was ordered and while the training and testing data are split equally, the dataset overall is unbalanced: 82.77% of the data is class 0; 2.54% of the data is class 1; 6.61% of the data is class 2; 0.73% of the data is class 3; and 7.35% of the data is class 4. From a layperson's perspective, classes 1 and 2 seem to be the clinically most relevant. Classes 4 and 5 are, to some degree, a fusion of normal and not-normal beats. As the original data was taken from 23 randomly selected recordings and 25 clinically significant recordings (Moody and Mark., 2001), this distribution seemed surprising. This may mean that subjects with clinically significant findings also have a high proportion of 'normal' heartbeats. Given this high proportion, overall accuracy might not be the best measure for the test data. Thus, in addition to accuracy, we also looked at sensitivity for each class, with a special emphasis on the sensitivities for classes 1 and 2 to evaluate our results. This follows AAMI recommendations and previous research (see Li and Zhou, 2016).

2. METHODS:

Data preparation:

We observed through EDA and references in the Kaggle repository that all the data had already been scaled between 0 to 1 (Kachuee et al., 2018, p. 444). Yet, we still had to address the imbalance in the dataset. An imbalanced dataset poses challenges for predictive modelling as it can result in poor accuracy performance for the minority class, which is often the reason for the creation of a model in the first place (Shelke et al., 2017). Our initial experiments confirmed this when a multilayer perceptron trained on unbalanced data mainly resorted to guessing the majority class (see appendix table 9). Since most machine-learning algorithms are designed with the assumption that there are equal samples in each class (Shelke et al., 2017), we aimed to resolve this imbalance prior to modelling our data.

One way to overcome the imbalance in the data was through oversampling. This means replicating existing samples from the minority class/es to increase their size and even out the number of samples in each class (Shelke et al., 2017). Undersampling is another technique, which works in the opposite way of oversampling. For our KNN and RF models, we used a total sample size of 10,000 per class, resulting in 50,000 samples in our training dataset. For our neural networks, we applied an oversampling approach using 35,000 per minority class and all of the majority class. Another approach was combining the upsampling with the downsampling of the majority class using 35,000 observations per class in our training set.

Selection of classification methods:

To determine the best classification methods for the dataset, our strategy was to explore as many methods as possible using a combination of Julia and R. The original plan was to implement everything in Julia, however, we soon encountered issues when trying to train the neural network models on an M1 computer, and instead resorted to R. That being said, all calculated weights are included in our GitHub repository (https://github.com/ThomasK81/INFO_411_Assignment_2) and the models are described below.

Figure 4 shows the initial accuracy ratings obtained from exploring various classification methods using Julia, specifically Decision Trees, KNN, Random Forests, Linear Discriminant Analysis (LDA), and SVM. Based on this, and relevant literature around the classification of ECG data, we decided to look further into RF and KNN, and drop the other models.

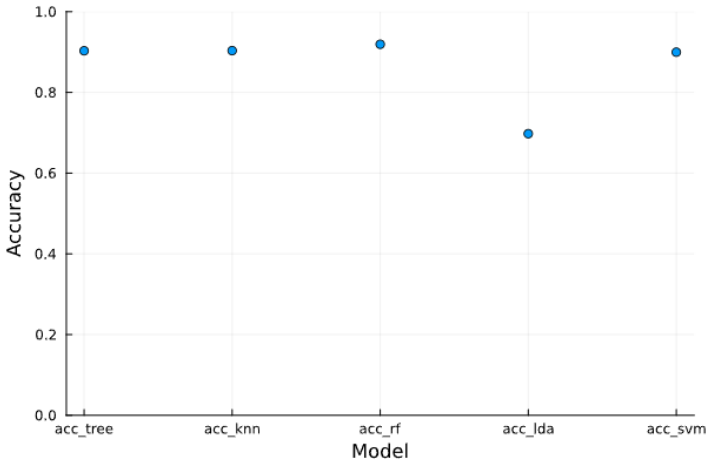

Figure 4: Test accuracy rating of various classification models from our initial run (from L-R: Decision Tree, KNN, Random Forest, Linear Discriminant Analysis, Support Vector Machine).

Additionally, we also looked into Convolutional Neural Networks (CNN) since this was the model employed in the article referenced in Kaggle.

3. EXPERIMENT RESULTS:

a. KNN

Our initial KNN models (with k = 5) yielded low accuracy (ranging from 81 to 83%). Since KNN is more sensitive to noise, we then reduced the features to see whether that would improve accuracy. We revisited our PCA and Decision Tree, determining features of more importance and features that were potentially adding noise. We decided to only use Columns 1 to 135 in our KNN model. After feature reduction, the accuracy improved significantly and jumped to almost 90%. We also tried to apply a distance-weighted voting scheme to the model. This slightly improved the accuracy to 90.3%. Table 1 shows the final confusion matrix for the KNN model after feature reduction and applying a distance-weighting function.

	GROUND TRUTH				
PREDICTED	Class 0	Class 1	Class 2	Class 3	Class 4
	16264	59	29	10	28
	1095	**475**	28	1	11
	387	13	**1345**	15	16
	272	8	40	**136**	1
	100	1	6	0	**1552**
CLASS SENSITIVITY	89.77%	85.43%	92.89%	83.95%	96.52%

Table 1: Confusion Matrix for KNN after feature reduction and adding a distance-weighting function.

While the model did not classify samples in Class 0 well, it outperformed other models in the classification of samples under Classes 1 and 2, which arguably are the more important classes to get right.

b. Random Forest

As we initially expected, RF's overall performance was superior compared to the other models that we explored, neural networks aside. After running the RF model using the default parameters achieved an initial accuracy rating of 91.6%. To further improve the accuracy of the model, we tuned various hyperparameters; for example, max depth, number of sub-features, and number of trees. Figure 5 shows a plot of the accuracy run chart when the max depth is varied from 1 to 25. Based on our results, we picked 16 as our preferred max depth and moved on to experimenting with the number of sub-features to use in our trees.

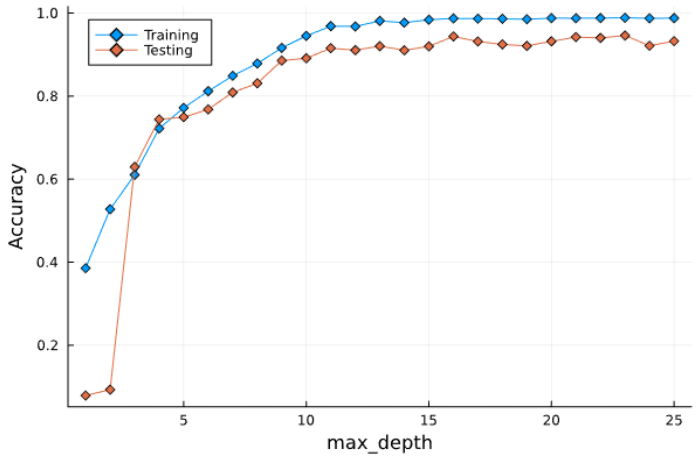

Figure 5: RF accuracy run chart with max depth varied from 1 to 25.

The accuracy run chart with the number of sub-features varying from 1 to 20 is shown in figure 6. Based on this, we selected 8 as the number of sub-features to use.

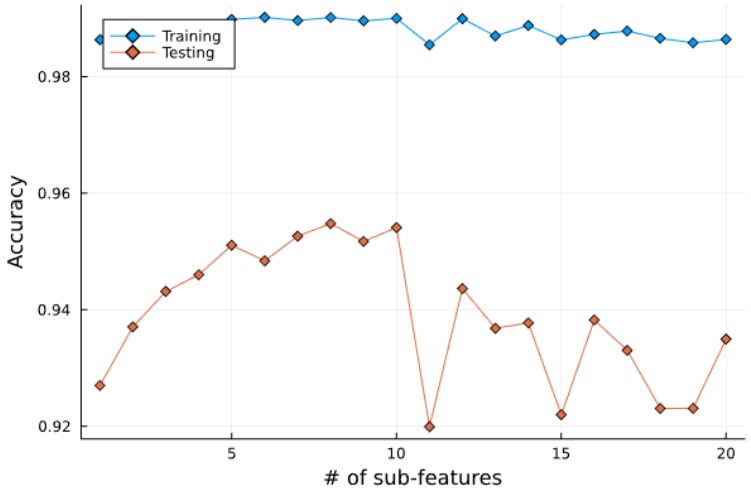

Figure 6: RF accuracy run chart with the number of sub-features varied from 1 to 20

We also looked at increasing the number of trees while incorporating the best max depth and number of sub-features identified above. Interestingly, the accuracy does not improve as the number of trees increases. We ended up selecting 200 trees to add to our final model as this produced the highest accuracy rating.

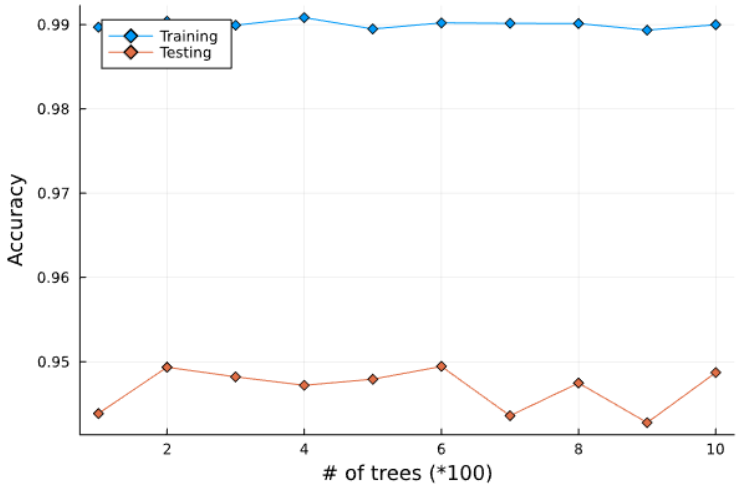

Figure 7: RF accuracy run chart with the number of trees varied from 100 to 1000.

After re-running the RF model using the parameters found above, we were able to obtain an accuracy rating of 94.2%, which is a clear improvement from the initial run.

Finally, we also attempted to do automatic tuning to determine the best settings for our RF model and to see whether they were similar to the settings we found above. We tried including four hyperparameters in our self-tuning model, but unfortunately adding anything more than two significantly increased the time it took to train the model.

In the end, we were only able to include max_depth and n_subfeatures in our self-tuning model, with the best settings per the output was 20 and 9 respectively. This is quite similar to the settings we found in our manual experimentation. After re-running the model using the new settings, we were able to get an accuracy rating of 94.6%, which is a slight improvement from our previous rating. Table 2 shows the final confusion matrix for our RF model:

		GROUND TRUTH				
		Class 0	Class 1	Class 2	Class 3	Class 4
PREDICTED	Class 0	**17276**	122	71	19	44
	Class 1	375	**423**	18	3	3
	Class 2	275	7	**1328**	10	15
	Class 3	115	2	23	**130**	1
	Class 4	77	2	8	0	**1545**
CLASS SENSITIVITY		95.35%	76.01%	91.71%	80.25%	96.08%

Table 2: Confusion Matrix for final Random Forest model. (i.e., using tuned parameters.)

c. Convolutional Neural Network:

After the MLP approach (see appendix table 9) delivered unsatisfactory results and the XGBoost approach (see appendix tables 10 & 11) did not majorly improve on our Random Forests, we turned to convolutional neural networks for the challenge. This was also referenced in the article on Kaggle (Moody and Mark., 2001). While our first network was inspired by the network in the article, we decided to take a more minimal approach. We also combined convolutional layers with dropout and pooling layers, but included only two convolutional layers and increased the dropout rate to 0.25 to address unwanted memorisation. We combined those layers with two fully connected layers. While the last fully connected network had a softmax activation function, all other layers used ReLu.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 183, 32)           192
max_pooling1d_1 (MaxPooling1D) (None, 90, 32)          0
dropout_1 (Dropout)          (None, 90, 32)            0
conv1d (Conv1D)              (None, 86, 32)            5152
max_pooling1d (MaxPooling1D) (None, 41, 32)            0
flatten (Flatten)            (None, 1312)              0
dense_4 (Dense)              (None, 32)                42016
dense_3 (Dense)              (None, 5)                 165
=================================================================
Total params: 47,525
Trainable params: 47,525
Non-trainable params: 0
_________________________________________________________________
```

Table 3: Architecture of the first and second convolutional neural network.

To address the imbalance in the data, we used Synthetic Minority Over-sampling Technique (SMOTE), as KNN already yielded good sensitivity results for classes 1 and 2, and SMOTE essentially grows a group's neighbourhood. We used an over-ratio of 0.483 to bring all minority classes up to around 35,000 observations and left the majority class untouched. With an overall precision of 97.7% and a sensitivity of 81.85% for class 1 and 94.89% of class 2, CNN became the most promising approach right from the start.

	GROUND TRUTH				
PREDICTED	Class 0	Class 1	Class 2	Class 3	Class 4
	17846	85	34	12	25
	154	**454**	9	0	2
	72	12	**1374**	11	5
	31	4	22	**139**	1
	15	1	9	0	**1575**
CLASS SENSITIVITY	98.50%	81.65%	94.89%	85.80%	97.95%

Table 4: Confusion Matrix for CNN without SMOTE sampling.

We next wanted to improve on those results further. Since the network still classified too many observations as class 0, we decided to not only upsample all minority classes with SMOTE but also downsample the majority class. Thus, every class had around 35,000 observations. While the overall accuracy was 96.7% (slightly lower than the first attempt) and the sensitivity for class 2 stayed roughly the same, this approach was successful in raising the sensitivity for class 1 to 83.37%.

	GROUND TRUTH				
PREDICTED	Class 0	Class 1	Class 2	Class 3	Class 4
	17615	73	30	7	18
	273	**463**	13	1	1
	117	13	**1373**	11	8
	81	6	26	**143**	0
	32	1	6	0	**1581**
CLASS SENSITIVITY	97.22%	83.27%	94.82%	88.27%	98.32%

Table 5: Confusion Matrix for CNN with SMOTE sampling.

To see whether a different approach could improve the sensitivity for class 1 even more, we combined classes 0, 3, and 4 into one class. As a first step, we didn't use SMOTE, since we first wanted to see how promising the approach was. As can be seen in table 6, the approach was not promising and we dismissed it.

	GROUND TRUTH		
PREDICTED	Class 0/3/4	Class 1	Class 2
	19814	135	89
	45	**413**	2
	29	8	**1357**
CLASS SENSITIVITY	99.60%	74.30%	93.70%

Table 6: Confusion Matrix for CNN without SMOTE sampling & classes 0/3/4 combined into one.

Finally, we made the network deeper and wider by giving it three convolutional layers with a different number of filters. We also added drop-out layers to each, only adding pooling layers twice (see table 7).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_18 (Conv1D)           (None, 183, 32)           192
dropout_11 (Dropout)         (None, 183, 32)           0
conv1d_17 (Conv1D)           (None, 179, 64)           10304
max_pooling1d_17 (MaxPooling1D) (None, 88, 64)         0
dropout_10 (Dropout)         (None, 88, 64)            0
conv1d_16 (Conv1D)           (None, 84, 32)            10272
max_pooling1d_16 (MaxPooling1D) (None, 40, 32)         0
dropout_9 (Dropout)          (None, 40, 32)            0
flatten_8 (Flatten)          (None, 1280)              0
dense_20 (Dense)             (None, 32)                40992
dense_19 (Dense)             (None, 3)                 99
=================================================================
Total params: 61,859
Trainable params: 61,859
Non-trainable params: 0
_________________________________________________________________
```

Table 7: Architecture of the fourth convolutional neural network.

The network had around 14,000 more trainable parameters. This greatly improved the results; the overall accuracy was 97.4%, comparable to the first network. However, the sensitivity for class 1 was 85.43% and for class 2 was 95.44%. Thus, this approach had the highest sensitivities for the arguably most crucial classes.

	GROUND TRUTH				
PREDICTED	Class 0	Class 1	Class 2	Class 3	Class 4
	17725	69	26	4	12
	217	**475**	5	1	1
	84	7	**1382**	9	5
	73	4	28	**148**	0
	19	1	7	0	**1590**
CLASS SENSITIVITY	97.83%	85.43%	95.44%	91.36%	98.88%

Table 8: Confusion Matrix for a wider and deeper CNN with SMOTE sampling.

4. DISCUSSION:

While our KNN model does not have a particularly exceptional accuracy rating, it is worth noting that this produced the highest sensitivity rating for Class 1, which we think requires the highest classification accuracy. From a medical perspective, we think that false negatives are not as bad as false positives in this context, given that the former may only prompt a patient to seek further examination, whereas the latter may lead to an undiagnosed severe condition. This is why in all our experiments, we emphasised class sensitivity and not just overall accuracy.

For our RF model, while this produced good overall accuracy ratings, its classification for the Class 1 samples is inferior compared to other models. This seems to be a particular issue for tree-based models as we have noticed the same problem with Decision Trees. Further improvements can potentially be observed if more hyperparameters are automatically tuned in the RF model. Regardless, this is an important limitation in our RF model that needs to be highlighted.

Another reason why accuracy alone is not the best measure is that the data is so unbalanced. If it were, then training a CNN on training data that has slightly but not fully rebalanced would be the best model, as can be seen in our first CNN, which has an accuracy of 97.7%. Unfortunately, it only has a sensitivity of 81.65%. This could be addressed by a) balancing the data further using a combination of SMOTE sampling and undersampling of the majority class and b) making the network a bit wider and deeper. This resulted in our final network having the best sensitivities for almost all classes (e.g., 85.43% for class 1) with only a minor accuracy trade-off (97.4%).

5. CONCLUSION:

All of the classifications and analyses were completed using the MIT-BIH database while following recommendations from the AAMI and other articles. From our experiments, we conclude that in general CNN was the best approach to create a model to classify heartbeats for the given data. Their softmax activation layer also results in an additional instrument to improve sensitivities for specific classes. An added benefit of using a softmax classification layer is that it enables us to do secondary matching. For instance, one could define a probability threshold for class 1 when observations that were classified as 0, should be corrected to class 1. This feature may have some real-world applications since a public health system has to combine considerations of costs and benefits (health and economic) for misclassifying specific conditions.

If we had information on which heartbeat belongs to which person, we could have also used this to address false negatives, since the classification wouldn't depend on a single heartbeat, but a group of heartbeats. Unfortunately, our results are somewhat limited due to the partially undisclosed methods used by the dataset's original authors, and their decision regarding the creation of the test and training data. Both training and the missing link of observation to subjects make it impossible to evaluate the model better. Additionally, the data was already normalised, filtered, and padded and did not leave room to apply our strategies in that regard and we were limited to simply addressing sampling issues. If our work were to be applied beyond the project work submitted for examination, we would strongly suggest re-doing our experiments with the original data.

REFERENCES:

Kachuee, M., Fazeli, S., Sarrafzadeh, M. (2018). ECG Heartbeat Classification: A Deep Transferable Representation. *IEEE International Conference on Healthcare Informatics (ICHI), 2018,* pp. 443–444, doi: 10.1109/ICHI.2018.00092.

Li, T., & Zhou, M. (2016). ECG Classification Using Wavelet Packet Entropy and Random Forests. *Entropy, 18*: 285.

Luz, E. J., Schwartz, W. R., Cámara-Chávez, G., & Menotti, D. (2016). ECG-based Heartbeat Classification for Arrhythmia Detection: *A Survey. Computer Methods and Programs in Biomedicine, 127:* 144–164, http://www.sciencedirect.com/science/article/pii/S0169260715003314.

Moody, G. B, & Mark, R. G. (2001, May–June). The Impact of the MIT-BIH Arrhythmia Database. *IEEE Eng in Med and Biol 20*(3): 45–50, PMID: 11446209.

Ribeiro, A. H., et al. (2020). Automatic Diagnosis of the 12-Lead ECG Using a Deep Neural Network. *Nat Commun* 11(1760), https://doi.org/10.1038/s41467-020-15432-4.

Shelke, M., Deshmukh, P., & Shandilya, V. (2017, April). A Review on Imbalanced Data Handling Using Undersampling and Oversampling Technique. *International Journal of Recent Trends in Engineering and Research, 3*(4).

APPENDIX:

I. Overview of Classification Methods

a. Primary classification methods:

I. Random Forests

Random forests are an ensemble learning method for classification that operate by constructing a multitude of decision trees at a selected training time. In this case, since we are using the random forest for classification, the output of the random forest is the class selected by the most trees. The benefit of using a random forest is its inherent correction for other models' habit of overfitting to the training set.

II. CNN

CNN's are a class of artificial neural networks. CNNs operate based on the shared weight architecture of the convolution kernels/filters that slide along input features and provide translation-equivariant responses known as feature maps. An advantage of using CNNs is that they are regularised, which means that instead of the addition of weight decay or trimming connectivity, CNN approaches regularisation by using the hierarchical pattern in the data and assembling patterns of increasing complexity using smaller and simpler patterns in the filters.

III. KNN

KNN is a non-parametric supervised learning model whose inputs consist of the 'k' closest training examples in the applied dataset. Using KNN as a classifier, the output consists of class membership by assigning data points to classes based upon its nearest 'k' neighbours. Despite its simplicity, we decided to use KNN as one of our models as it showed great potential in the classification of the 'difficult' classes.

b. Secondary classification methods explored:

I. eXtreme Gradient Boosting (XGBoost):

XGBoost works as a root-finding algorithm producing consecutive improved approximations to the roots. However, unlike gradient boosting, a second-order Taylor expansion is used as the loss function. While the advantage of using XGBoost is the model often achieves better accuracy than a standard decision tree, it lacks the intrinsic interpretability of decision trees. Following the path of a general decision tree is trivial and self-explanatory but following the magnitude of trees produced via XGBoost is more complex. This can be overcome via the utilisation of model compression techniques transforming XGBoost into a single decision tree that approximates the original complex function.

II. LDA

LDA is a dimensionality reduction technique built upon a generalised application of Fisher's linear discriminant. Fisher's linear discriminant is a supervised learning classifier that attempts to find the vector that maximises the separation between classes of data. LDA uses Fisher's linear discriminant to reduce the dimensionality of the data while maximising the separation between classes. It achieves this by maximising the distance between means and minimising within-class variance. Notably, LDA is closely related to principal component analysis (PCA). LDA attempts to model the difference between the classes of data while PCA does not factor in any difference in classes.

III. Decision Tree:

Decision tree is a supervised model where the data is continuously split according to defined parameters. In our use case, a classification tree is used as the predictive model to draw conclusions from the ECG dataset. The tree is built by splitting the source dataset into subsets. The splitting of data into subsets is determined by the rules outlining the classification features. This process is repeated until splitting no longer adds values to outlined classification metrics. The tree can be further explored and visualised via decision nodes and the corresponding leaves.

IV. Multilayer Perceptron (MLP):

MLP is a fully connected feed-forward artificial neural network (FFANN). MLPs consist of a minimum of three layers of nodes, comprised of the initial layer taking the inputs, and the last layer that produces the outputs based upon the hidden third layer. With each node being connected to every node of the series of layers the information is constantly fed forward between layers, hence the classification FFANN. The process that MLP uses for learning is a supervised learning technique called backpropagation. This model distinguishes itself from linear perceptrons via the use of multiple layers in the design, meaning it can distinguish data that is not linearly separable. Learning occurs via changing connection weights after each piece of data is processed and applying weight decay based on the number of errors generated in the output compared to the expected result.

III. SVM

SVM is a supervised learning model that analyses data for classification and regression analysis. Again for our use case, we are using the model for the classification of the ECG dataset. SVM's operate via a training algorithm building a model assigning data points to categories. The resulting map shows the trained examples allocated to maximise the space between the categories.

II. Results of Alternative Methods:

a. MLP:

As a first simple neural network approach, we designed a multilayer perceptron (MLP) with one hidden layer (60 units) and a dropout of 0.2 for 20 epochs. The 0.2 dropout rate addresses the memorisation effect, while the 60 units should give the network enough capability to learn. To see how such a simple network performs out-of-the-box we didn't address the imbalance of the data for this approach. While the MLP achieved an overall accuracy of 95.5% on the test set, it had an extremely low sensitivity for class 1. We abandoned this approach for a more promising convolutional neural network approach (CNN).

	GROUND TRUTH				
PREDICTED	Class 0	Class 1	Class 2	Class 3	Class 4
	18032	277	214	152	91
	10	**171**	20	1	0
	50	107	**1202**	9	16
	0	0	0	**0**	0
	26	1	12	0	**1501**
CLASS SENSITIVITY	99.53%	30.76%	83.01%	0.00%	93.35%

Table 9: Confusion Matrix for Multilayer Perceptron.

b. XGBoost:

To see whether we can improve on the Random Forest approach, we employed the XGBoost algorithm with and without sampling to address the class imbalance. For the unbalanced approach, we used 20 trees and a depth of 8, attempting to straddle lessons learned from the Random Forest approach and processing time concerns. While the unbalanced approach reached 97.5%, even higher accuracy on the test set than MLP, it still had relatively poor sensitivity for class 1.

	GROUND TRUTH				
	Class 0	Class 1	Class 2	Class 3	Class 4
PREDICTED	**18071**	200	150	40	75
	14	**354**	1	0	0
	21	1	**1276**	16	4
	2	0	14	**106**	0
	10	1	7	0	**1529**
CLASS SENSITIVITY	99.74%	63.67%	88.12%	65.43%	95.09%

Table 10: Confusion Matrix for XGBoost without SMOTE sampling.

We then used Synthetic Minority Over-sampling Technique (SMOTE) when we addressed the imbalance since KNN already yielded good sensitivity results for classes 1 and 2, and SMOTE essentially grows a group's neighbourhood. For the balanced approach, we made the fit a bit wider and deeper, increasing the trees to 25 and the depths to 10. The balanced approach yielded a lower accuracy of 96.7%, but with a sensitivity of 80.79% for class 1 and 94.54% for class 2, it achieved a much more useful classification overall and at this stage the highest sensitivity for class 2.

	GROUND TRUTH				
	Class 0	Class 1	Class 2	Class 3	Class 4
PREDICTED	**17636**	95	44	15	27
	249	**450**	8	0	4
	117	6	**1369**	10	5
	85	2	20	**137**	1
	31	3	7	0	**1571**
CLASS SENSITIVITY	99.73%	80.79%	94.54%	84.57%	97.70%

Table 11: Confusion Matrix for deeper and wider XGBoost with SMOTE sampling.

c. SVM:

Our SVM model produced satisfactory results, particularly after experimenting with various kernel degrees and cost functions. Kernel functions return the dot product between two points in a suitable feature space, therefore implying a notion of similarity with minimal computational cost. The kernel trick also addresses the non-linearity of the data, which is the case for our ECG dataset. Finally, the cost function was also useful in training the SVM as it ensures that the model is as accurate as possible.

The initial run resulted in an accuracy rating of 88.7%. However, after adding the kernel and cost function, it increased to 91.4%. Figures 7 and 8 show the experiments we did on varying kernel

degrees and cost functions to determine the best combination that yields the highest accuracy. We found that a kernel degree of 7, and a cost function of 6 were the best settings for our SVM model. The final confusion matrix for the SVM model is shown in table 12.

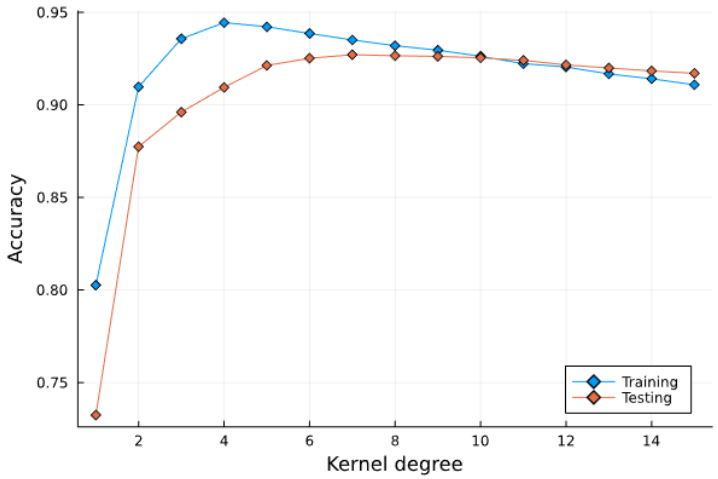

Figure 7: SVM accuracy run chart with kernel degree varied from 1 to 15.

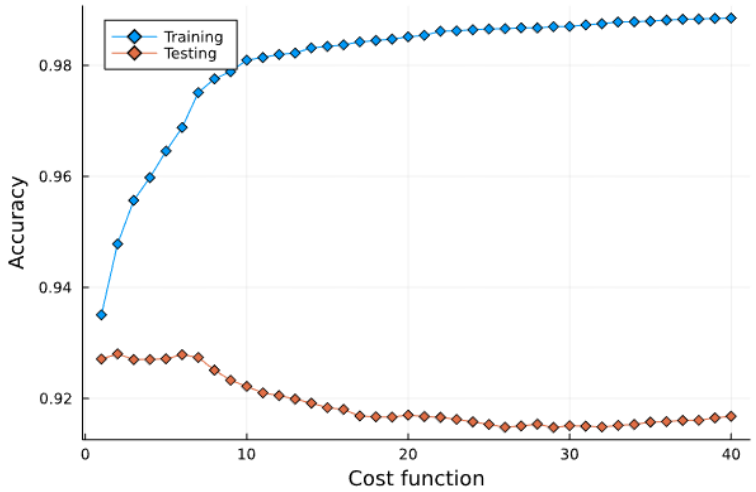

Figure 8: SVM accuracy run chart with kernel degree varied from 1 to 40.

	GROUND TRUTH				
PREDICTED	Class 0	Class 1	Class 2	Class 3	Class 4
	16584	111	41	18	45
	620	**415**	16	1	9
	587	24	**1348**	8	17
	241	4	37	**135**	1
	86	2	6	0	**1536**
CLASS SENSITIVITY	91.53%	74.64%	93.09%	83.33%	95.52%

Table 12: Confusion Matrix for final SVM model with kernel and cost functions added.