An Analysis of K-Nearest-Neighbour and Logistic Regression in Heart Disease Classification

Jake Grogan[1], Connor Mulready[1]

[1]School of Computing, Dublin City University, Ireland

jake.grogan8@mail.dcu.ie, connor.mulready2@mail.dcu.ie

Abstract. Heart Disease has been labeled the world's biggest killer, with 1 out of 4 people dying as a result of it [1]. In Ireland alone, heart disease was responsible for 40% of the deaths in 2005, beating cancer which had a mortality rate of 28% [2]. In the US over 370,000 people die annually as a result of the disease, 90,000 of those deaths are avoidable with early diagnosis [1]. This report aims to discuss the various methods required to correctly classify whether or not an individual has heart disease. In order to achieve this we will cover the process we underwent to understand our data as a whole and prepare it by ensuring it was clean and consistent. The report will also cover the strengths and weaknesses of both K-Nearest Neighbour and Logistic Regression classification algorithms in regards to our dataset.

Keywords: Heart Disease, Classification, K-Nearest-Neighbour, Logistic Regression, Analysis

1. Introduction

Heart Disease has been labeled the world's biggest killer, with 1 out of 4 people dying as a result of it [1]. In Ireland alone, heart disease was responsible for 40% of the deaths in 2005, beating cancer which had a mortality rate of 28% [2]. In the US over 370,000 people die annually as a result of the disease, 90,000 of those deaths are avoidable with early diagnosis [1].

This report aims to discuss the various methods required to correctly classify whether or not an individual has heart disease. In order to achieve this we will cover the process we underwent to understand our data as a whole and prepare it by ensuring it was clean and consistent. The report will also cover the strengths and weaknesses of both K-Nearest Neighbour and Logistic Regression classification algorithms in regards to our dataset.

Tool	**Use**
Pandas	Data structure and data analysis library for Python. We used this for reading our dataset into various Python scripts as data frames.
Collections	Python library for creating containers of data types. We used this to aid in extracting the predicted value from K nearest neighbours
SciKit-Learn	Python library for machine learning. We used this to help us shuffle our data set, build a logistic regression model, provide metrics on our models and help us with preprocessing data.
Numpy	Numpy is a library for Python, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level math functions to operate on these arrays.

Table 1. Tools Used

1.1. Dataset Description

For our dataset we opted to use ‘Heart Disease UCI’ dataset from kaggle. The original dataset consisted of 76 attributes and aggregated patient data from many hospitals and institutes however, the version we opted to use only used attributes directly related to the diagnosis of heart disease which reduced it down to 14 attributes. In regards to the number of patients in the data set, there were 303 entries making it a relatively small yet viable dataset.

2. Data Preparation

2.1. Data Summarization

2.1.1 Overview

Before we could begin building our classification model, we first needed to understand our data as a whole. This meant that we would have to understand our data objects and the types of features that it is comprised of. The first step required us to look at each feature and in turn label it with the correct attribute type. This can be easily derived by looking at the observations for each attribute.

Discrete Attributes	**Binary Attributes**	**Continuous Attributes**	**Nominal Attributes**
Age	Sex	Trestbps	Chest Pain
Ca	Fbs	Chol	Restecg
	Exang	Thalach	Slop
	Target	Oldpeak	Thal

By categorising each attribute by its type, we've clearly laid out what we can and can't do in terms of statistical descriptions. For example with the nominal attributes, introducing a mean makes no sense as our attributes are clearly categorical but since they're categorical we can make use of the mode to get a better understanding of our data set.

We also found out that some of our binary attributes are asymmetric, emphasizing males in the sex attribute as they are more likely to be diagnosed with heart disease and no disease in target. This would explain the target attributes choice of using 1 for no disease as it would normally suggest the patient had been diagnosed in a true and false sense.

For the data preprocessing phase to be a success we must thoroughly understand our data, and with our newly found knowledge of our datasets attributes we can begin to develop better statistical descriptions of our data using the following approaches.

2.1.2. Central Tendency

The first step we took to better understanding our data was to measure the central tendency of each attribute using mode, median and depending on the attribute type the mean. We could also measure the center using the midrange however, it isn't as accurate since it only relies on two values. By measuring the central tendency we could easily find out whether our data was symmetric or asymmetric depending on the difference of the values. In all cases, the data was skewed to some degree. However the numerical results on their own didn't provide an overall summarization for us, therefore we made use of histograms for our continuous attributes and bar charts for our categorical features.

We initially began measuring the categorical data (we omitted the use of the mean as it makes no sense to have part of a category) and as expected the data was skewed for each attribute. When reviewing the bar charts we generated, we found a couple of interesting cases where outliers existed, primarily restecg and ca.

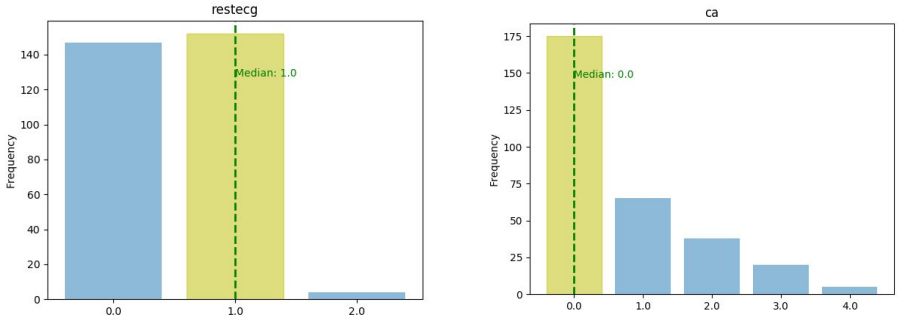

Figure 1. Central tendency of restecg and ca attributes

We decided to investigate these values further as it was unusual to have an entire category dedicated to so little values. From this we found that the ***ca*** attribute had an additional value not mentioned in the original dataset description which actually represented NaN values. This was the only case of NaNs found in our entire dataset and would have to be addressed later during the preprocessing phase.

Moving onto our continuous data we made use of all three mode, median and mean to measure our central tendency and what we found was that all of our attributes were skewed however the majority of them were only skewed by a small margin. We assumed that this was contributed to outliers in our data, as shown in both thalach and age.

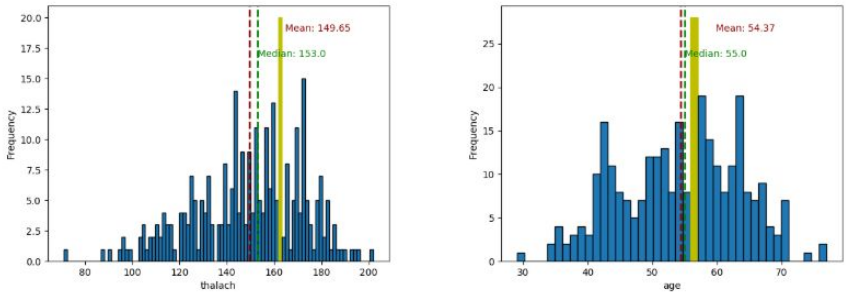

Figure 2. Central tendency measures of thalach and age attributes

In order to ensure that this was the case, we trimmed our data by 2% on the upper and lower boundaries and in turn found this was not the case, the outliers didn't have as big of an impact on the data like we assumed. Trimming the data caused the mean to move towards the median however, the amount that it changed was so insignificant that removing the outliers would have no benefit, especially since our dataset is small, every bit of data matters.

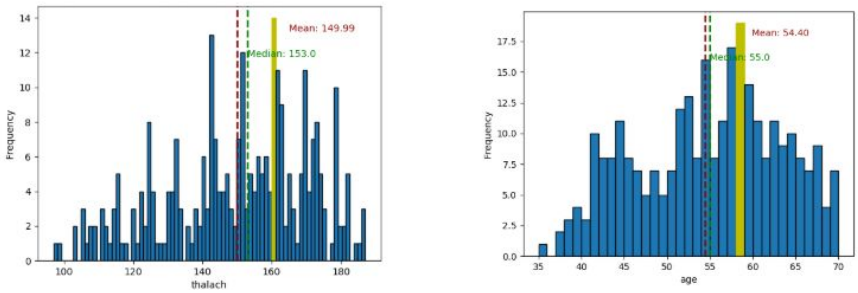

Figure 3. Central tendency measures of thalach and age after trimming.

As a final observation we noticed that our data spiked in multiple occasions. This told us that our data was noisy. An important observation as it should be dealt with during the preprocessing phase.

2.1.3. Data Dispersion

Another approach we used to summarize our data was to measure the variance of our data. In order to measure the dispersion we would generally use the interquartile range and standard deviation however numbers on their own can only give part of the picture. What we wanted to understand from this approach was where exactly our main data was for each attribute. Once we identified that we could more easily identify outliers in our data.

The first step we took was to plot our data into boxplots. This required the five number summary, this involved finding the minimum, maximum values for a given attribute, the upper and lower quartiles and the median. With these values we were able to generate boxplots for our attributes and find out the following data.

One of the first things we noticed was in regards to age. Even though the patients ages ranged from their late twenties to their seventies, the majority of the people were in the range of late fifties to early sixties. Upon review this makes sense as this is the prime age group who are diagnosed with heart disease.

Secondly we assumed outliers existed on both ends of our data, however the boxplots proved that theory wrong. The trestbps attribute had quite a large range of values however we found that the main part of the data oriented towards the lower half of the values which in turn resulted in quite a lot of outliers towards the top.

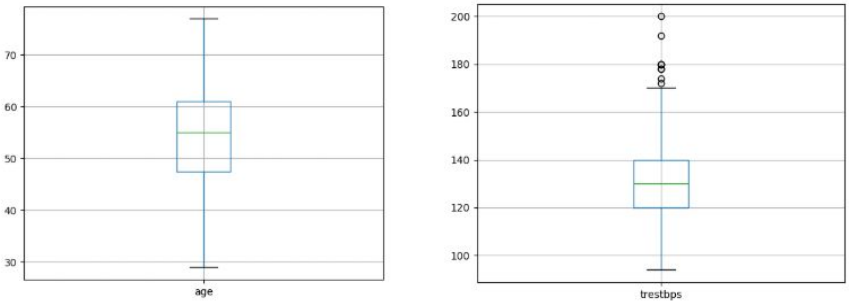

Figure 4. Boxplot of age and trestbps

2.2. Pre-processing

From the new understanding gained by the data summarization stage we now know the steps required to clean and fix. Doing so we aim to tailor our data to be more suitable, complete and consistent for our classification models with the overall goal of increasing the accuracy of the model whilst not overfitting the data. This is especially important in our case since our data set has only 303 observations, we need to ensure that we can get as much information from each observation as possible if we aim to have an accurate classifier.

2.3. Data Cleaning & Transformation

2.3.1 Missing Values

As previously discovered during our measure of central tendency one of our features had NaN values. Initially we contemplated simply removing the rows but as previously stated the number of observations we have are small and therefore valuable. Instead we opted to use the attribute mean for all samples belonging to the same class as a given object. We had four NaN values in total. Three of the four NaNs were classified as people not diagnosed with the disease which after finding the mean equated to 0. In regards to the final NaN value, after calculating the mean it equated to 1. By getting the mean of the classes rather than the collective mean ensured that our data stayed consistent, a key goal of pre-processing.

2.3.2. Noisy Data

Noisy data leads to inconsistency. One of the key goals of pre-processing our data is to amend that. The approach we ended up taking was to use binning by means. By sorting each attribute and placing the observations in bins of equal length we can get the mean of that bin and replace the pre-existing values with our newly computed mean. Doing so reduced the spikes in our data and attempts to spread large values over the size of a bin. However if an outlier is great, it still remains an outlier so that our consistent data isn't over generalised. In our case we aimed to keep our buckets relatively small with a size of 3, as to not over generalize our data. Another approach to binning would be to bin by boundaries however we could see issues in the future with outliers spoiling out data. As previously shown when discussing central tendency out data had spikes. After applying binning by means with bins of size 3 we got the following smoothed and more consistent data.

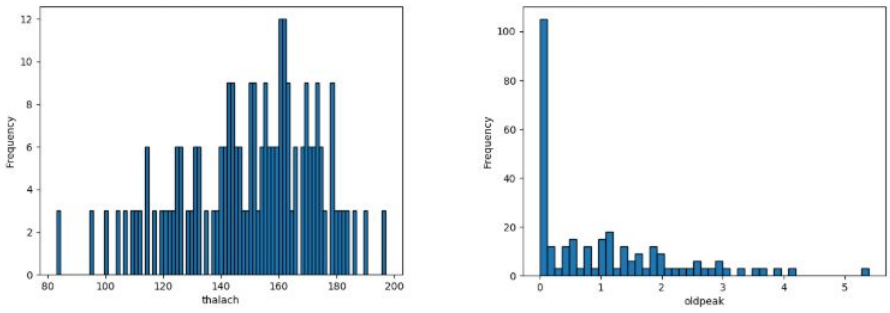

Figure 5. Histograms of thalach and oldpeak after binning

2.4. Data Normalization

As we learned in lectures, continuous data that has attributes with large ranges and other attributes with smaller ranges, normalization can be a good technique to employ in order to reduce one attribute from having a greater influence on the prediction than another. This was the case with our dataset.

As an example, the chol attribute of our dataset ranged between 141 and 321 whereas the oldpeak attribute only ranged between 0.0 and 4.2. This would clearly have an impact on our result as the chol attribute would have a far greater weight than the oldpeak attribute.

To combat this issue, we decided to normalize attributes that have large ranges in order to scale their values so that they fall within a small specified range. Normalization is particularly useful for classification problems especially with distance-based methods such as K nearest neighbours.

After looking into various methods for normalization we decided on min-max normalization in order to map all values in these attributes between a range of 0.0 and 1.0 using the following formula:

$$v' = \frac{v - min_A}{max_A - min_A}$$

Where min_A and max_A are the minimum and maximum values of an attribute and v is the current value. Applying this to each value for an attribute will scale each value between 0.0 and 1.0

2.5. Workshop Findings and Results

Although measuring the central tendency of a dataset is important in gaining a greater insight into understanding a dataset, it probably wasn't the workshop for us. We arrived at this conclusion as all of our continuous data was pretty much normally distributed. There were no large amount of outliers that were impacting our predictions, there was very little variance in the data, and there were no large skews in any of the continuous attributes except for one attribute, oldpeak.

In saying this, through measuring the central tendency of our data we did learn a lot more about our dataset, in particular what each attribute meant. Although this is medical data and the literature around each attribute was not entirely clear, we did gain a high level understanding of what each meant. We also learned that although there were skews in the discrete and nominal attributes, that it actually made sense for them to be there. For example, statistically it is known that men have a higher risk of developing heart disease than women so it makes sense for the gender attribute to be skewed towards men.

In all, although this workshop was not the best suited to us, through preparing for it we learned a lot about our dataset and we did find it beneficial to us as we felt it lended a helping hand when it came to other areas of data summarization and data preprocessing.

3. Algorithm Description

3.1. Classification

In this section we explain our thought process and reasoning with regards to our chosen classification algorithm, why we chose it over other algorithms, and its implementation.

Classification is the process of building a model such that it can predict a class label. The reason for choosing classification is that our aim is to predict a class label from a tuple. Our dataset had two predicted attributes, “Heart disease” and “No heart disease”, both represented as a 0 or a 1 respectively.

To begin building our model, we decided k-nearest-neighbours would be a good fit for our dataset as k-nearest-neighbours can be slow to execute on a large dataset and ours was small enough to run this classifier on. We also felt it would be a good model to go with initially as we believed it would give us a good understanding of whether or not the target attribute could be predicted with a relatively high accuracy.

After some experimentation we also took a look at logistic regression as a model to predict whether or not someone had heart disease. We looked into logistic regression as a possible model as historically logistic regression had been used in biological sciences and it would work well with our dataset as it is used as a model when the target attribute is categorical.

3.2. K-Nearest-Neighbour

K-Nearest-Neighbour (KNN) is a simple yet powerful classification algorithm. KNN assumes that similar things exist in close proximity. It is commonly used in predicting class labels, which is what we wanted to achieve for our project.

The general approach is as follows:

1. Plot all tuples in the test set in N-dimensional space (14 dimensions in our case)
2. Plot an unseen tuple from the training set
3. Locate the K nearest training instances to the test tuple.
4. Select the most commonly occurring (mode) classification for these K instances.

There were a few variables for us to take into account when we were starting out with building this model. The first was choosing how big our training and test sets should be. We went with a 70/30 train-test split to begin with.

Secondly, we had to choose a value for K. We chose a small number, 15, as our value for K. As our model was a binary classifier, k should be an odd number as it ensures there is always one prevailing classification when we go to test our model.

As shown, it is quite a simple algorithm and works well for us as our dataset isn’t too large. A large dataset can have a massive impact on the performance of KNN as it is computationally costly to run.

Initially our models accuracy was not the best, however, there are a few optimizations that can be made to boost this. These will be discussed in a later section.

3.3. Logistic Regression

Logistic regression is one of the most commonly used classification algorithms when it comes to two-class classification. Logistic regression estimates the relationship between one dependent binary variable and the independent variables. Although variations of logistic regression exist that can predict multiple classes, they did not apply to us here.

For our implementation we chose to use the Scikit Learn library for Python to build the logistic regression model.

Our main goal was to increase the accuracy we were achieving with k-nearest-neighbours. To achieve this we wanted a classifier that would prevent picking up “peculiarities” or “noise”.

3.3.1. L1 & L2 Regularization

SKLearn’s logistic regression class implements regularized logistic regression by default. For us, this means that regularization is adding bias if our model suffers from high variance. This means we avoid overfitting the training data by applying a penalty to the models cost function.

By default, SKLearn applies L2 regularization to the logistic regression model, however we can override this to use L1 regularization.

L2 regularization adds a squared magnitude to the coefficients as a penalty term to the loss function while L1 regularization adds an absolute value of magnitude to the coefficients as a penalty term to the loss function.

The key difference between both forms of regularization is that L1 regularization shrinks the coefficients of less important features to zero, thereby removing some features altogether. This works well for feature selection in the case where we have a large number of features and since we have 14 features in our dataset, we thought this might be a good idea.

3.3.2. Optimization Problem Solvers

The linear regression class provides many solvers to help solve optimization problems such as stochastic average gradient, Newton’s method, the LMBFGS algorithm and the LibLinear solver. The logistic regression model we are using uses the LibLinear solver to help solve optimization problems. It uses a coordinate descent algorithm that solves optimization problems by successively performing approximate minimization along coordinate directions or coordinate hyperplanes.

The LibLinear solver solves the following optimization problem [4]:

$$min_w \; = \; ||w||_1 \; + \; C \sum_{i=1}^{l} log(1 \; + \; e^{-y_i w^T x_i})$$

Where $log(1 \; + \; e^{-y_i w^T x_i})$ is the models cost function and C is a penalty cost applied to the cost function. The vector w is a weight vector generated by the model and x is an input training vector.

We chose this solver for our model as it works very well with smaller datasets and logistic regression models which apply L1 regularization which is the case with our logistic regression model.

4. Final Results and Analysis

4.1. Analysis of K-Nearest-Neighbours

To build our KNN model, we decided to build it from scratch in order to gain a better understanding of the technique rather than using any libraries to build it for us.

When we initially built the KNN model, our dataset had not gone through any preprocessing except for some cleaning to remove NaN values. Our approach was to run experiments to test the accuracy of the model, then do some additional preprocessing and re-run those experiments to find out what impact preprocessing had on our models accuracy (essentially an iterative process). We found this gave us a much clearer understanding as to how important preprocessing is to data mining.

When we initially ran these experiments, our models accuracy was roughly 56%. This was essentially a coin toss. After some study of our dataset it was clear that some attributes were having a greater influence than others so we normalized the attributes that were causing this. Those attributes were:

- tresbps
- chol
- thalach

We then ran some more experiments after normalization of the above attributes. We expected quite a good increase in model accuracy, however the accuracy only rose by about 5%. This confused us so we decided to investigate a little further.

During investigation we noticed that our dataset was ordered by the attribute we were trying to predict. This meant when we were splitting our dataset into training and test sets, the training set did not have a good mix of "Heart disease" and "No heart disease", it primarily consisted of "Heart disease". We used the shuffle() method from SKLearn to shuffle the data frame and after shuffling our models accuracy increased significantly to just above 70%.

Although our model was performing relatively well, we weren't satisfied yet and believed we could improve the accuracy by taking yet another look at our dataset. We took a more in depth look at certain attributes and plotted them on a bar chart. We discovered that some attributes were quite noisy and there were a lot of jagged spikes in their plots. From our reading of *Data mining: concepts and techniques*, we came across a method to reduce the amount of noise in data, in particular, the binning by mean method. This method was discussed in the Data Cleaning and Transformation section. We applied this method to various attributes which we found contained noise. As an example, the thalach attribute was found to contain a lot of noise. A before and after of its bar chart plot can be found below to show the impact that binning by mean had on reducing the amount of noise in this attribute.

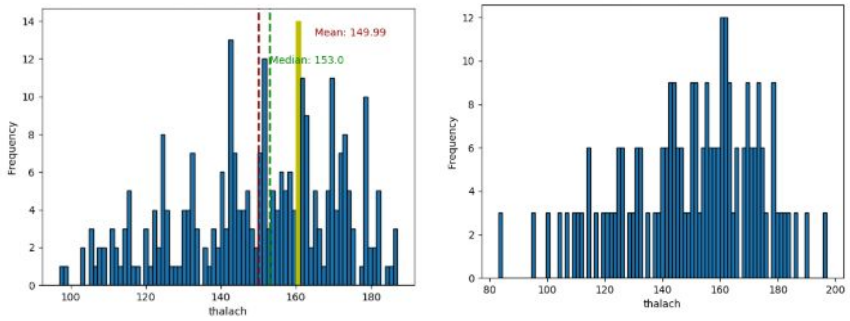

Figure 6. Comparison of thalach before and after binning

After applying this and running more experiments we found our accuracy had gone up to roughly 87%. This was a significant increase, yet we still believed we could do better.

After applying these data transformation and cleaning techniques, we took a look at optimizing our KNN model. Upon examining our code, there was one variable that we thought we should look at, namely ***K***, the number of nearest neighbours to use to make a prediction. We had initially set the value of ***K*** to 15, this was arbitrarily chosen at the time of building the model. We decided to re-run our experiments, varying the value of ***K*** each time. We varied the value of ***K*** between 1 and 50 taking steps of 2 each time in order to ensure that ***K*** would always be an odd number. We plotted a line chart of the value of K vs. the accuracy of the model which can be found below:

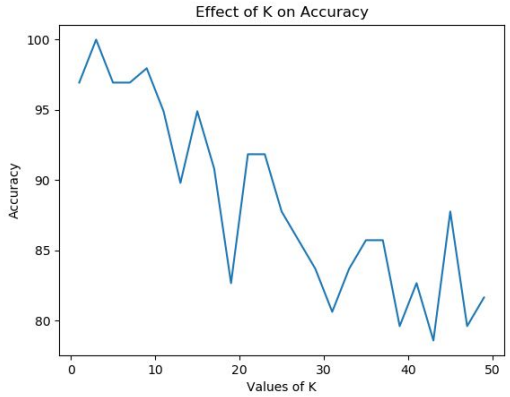

Figure 7. Effect of varying K on the accuracy

From this graph it is clear that the optimal value of ***K*** is a small number, 3 in this case. We changed it's value in our model and re-ran the experiments a final time and to our surprise the accuracy of the model had jumped from 87% to 96% on average.

We have summed up how our use data engineering techniques improved our models accuracy below:

	Initial Tests	**Normalization**	**Binning by Mean**	**Optimal K value**
Accuracy	56%	71%	87%	96%

Table 1. Accuracy of KNN model after each optimization

4.2. Analysis of Logistic Regression

We introduced logistic regression at a later phase in our experimentation with the dataset. We decided to test with this too as Logistic Regression works in classifying binary attributes. It is also has a more advanced implementation than KNN so we thought it might get better results than what we were achieving already.

We started to experiment with this model just before we removed noise in the data by binning by mean.

In fact, from studying logistic regression and its implementation in the SKLearn library we learned that the LibLinear solver does not perform as well with data that is not smooth. This is where we decided to look into methods of reducing noise in our dataset and applied binning by mean, which in turn had a great effect on our KNN model accuracy.

There was not much more we could do with our data without over-engineering our dataset. Below is a table showing the accuracy of our Logistic Regression classifier model before and after applying noise reduction techniques.

	Before Noise Reduction	**After Noise Reduction**
Accuracy	85%	93%

Table 2. Comparison of accuracy after noise reduction

We also wanted to understand how our logistic regression model was performing on predicting each attribute correctly, to do this we constructed a confusion matrix. The output of the confusion matrix looked like

33	7
2	49

Table 3. Confusion matrix of logistic regression model performance by attribute

Although this confusion matrix is small, larger confusion matrices can be difficult to grasp at an initial glance, for this reason we plotted the confusion matrix as a heatmap.

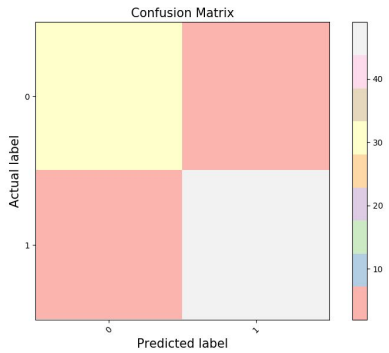

Figure 8. Heatmap of confusion matrix

From looking at the heatmap it was clear our model had predicted very few false positives and false negatives.

5. Discussion

Throughout this report we've studied and applied various data summarization techniques to gain an understanding of our dataset, what different attributes mean and how certain attributes may be having a greater influence on a models output than others.

We've learned about various techniques for cleaning our data along with transforming our dataset and applied these methods in an iterative process, showing the importance of each and how each technique impacts our classification models results.

We've looked at two classification algorithms and discussed how each work and how each apply separate techniques to classify information. We've looked at their individual parameters and what they mean, how they apply to the data and how those parameters impact a models output.

We've contrasted both KNN and logistic regression as applied to our dataset and shown that KNN is clearly the classifier to use for this particular dataset out of the two. Although the accuracy of KNN was only 3% more than that of logistic regression, in trying to predict whether someone has heart disease or not, every percentage matters.

In conclusion, when we started this project, we put emphasis on the algorithms and omitted the importance of data preprocessing. From this project, we've learned this is not the way to approach these types of problems and that data summarization and preprocessing is equally, if not more important in achieving accurate results for your classifiers.

6. References

[1] CDC: Heart Disease Facts - https://www.cdc.gov/heartdisease/facts.htm

[2] Irish Heart Disease Awareness - https://ihda.ie/

[3] J. Han, M. Kamber, J. Pei (2011), *Data Mining: Concepts and Techniques*, Morgan Kaufmann; 3rd edition

[4] R. Fan et al. (2008), *LIBLINEAR: A library for large linear classification*, Journal of Machine Learning 9, 1871-1874 - https://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf