Data Mining and Decision Systems 600092 Assigned Coursework Report

Student ID: 201700039
Date: 3rd November 2020

Due Date: 11th January 2021 by 2pm

Report must be within 10 page maximum. Strict page limits will be enforced. Any extra pages will be ignored and no marks awarded for any work on these. Exclusions to this limit are the front page, the references section, and any appendices. Please keep to the given section headings and format; subsections are permitted.

Methodology

The data mining project which I will be undertaking will follow industry standard methodology and tooling on a legacy data set. The goal is to produce a description and analysis of some data within the cardio-vascular medicine domain, this data will then be used to produce classifiers to predict a categorical target attribute. The methodology I will use during the development of this project is a modified version of the Cross-industry standard process for data mining (CRISP-DM) (Posted by William Vorhies on July 26 and Blog, no date), this six phase iterative model outlines stages for approaching a data mining project. The model is not one way, and it is possible to move backwards and forwards at each stage, this allows changes to be made if with new knowledge it is found the previous stage was not completed successfully.

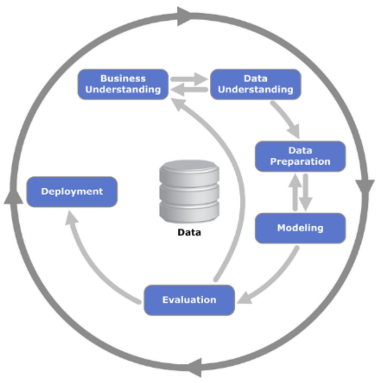


Figure 1 - CRISP-DM - (Posted by William Vorhies on July 26 and Blog, no date)

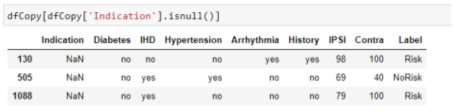
The first stage in the CRISP-DM process is Business Understanding, this involves taking the objectives and requirements of the task from a business perspective. These objectives then need to be translated into a data mining problem definition and an initial plan. This problem definition comes from a question or need, usually a business (*Problem Definition*, no date). In this case the business understanding section of the model was already completed for me as this is a university project, so I was already supplied with a well formulated problem definition. The problem I wish to address is that of predicting if a patient is at risk or not from a cardio-vascular event, using data from within the cardio-vascular medicine domain I will produce classifiers which using given data points predict a Label of Risk or NoRisk.

The next stage in the CRISP-DM cycle is Data Understanding, this involves collecting the data needed to address the previously formulated problem definition, as well as getting familiar with the data by performing some preliminary evaluation. This evaluation looks for issues with the data such as missing or Null values, as well finding interesting subsets of the data which can be used to form interesting hypotheses. A data description/ dictionary can also be produced at this stage which uses domain knowledge to give insight into the attributes and columns within the data set. It also usually outlines the type of data which is stored, such as categorical and numerical.

As this was a university project the data and data description to go along with it was supplied to us, so the first task I needed to complete is ensuring the data we had been supplied was of the correct types. In section 4 of the code, I read the data into a Python Jupyter notebook using pandas, defining the categorical columns correctly, In section 5 I used the info and describe functions to get a better idea of how pandas saw the numerical data. The 'Random' and 'Id' columns were recognized correctly but 'IPSI' was a float and not an integer and 'Contra' was an object which is completely the wrong type. I then attempted to convert the data types of the necessary attributes, but this was not possible as there were null or missing values. Because of this I had to move onto the next stage of CRISP-DM before I could complete this stage.

The third stage of the CRISP-DM model is Data Preparation, this stage involves taking the raw data and transforming and cleaning so that it conforms to the data description and can be used to solve the previously defined problem description. This includes fixing the issues found during the data understanding phase of the process. These issues might include, missing values, duplicate or contradicting data, erroneous values, spelling and casing issues and null values.

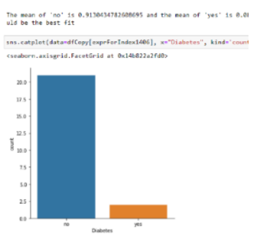
The first issue I looked to fix with the raw data was null values in section 7.6 as this was the immediate problem I discovered when trying to set the data types. This stage can be completed in lots of different ways and lots of research has been completed into the various methods to undergo this fix. Depending on the data different methods may be more suitable, I decided to try a few different options and compare the resulting cleaned data of each during the later stages of the model. The first method I designed myself and can be found in section 7.6.1, this was quite a manual process, it involved first identifying rows from each individual column which contained null values. The next step involved producing a custom query for each row which would search the whole data set. I would then calculate the mode from the returned rows from the missing value column and replace the null value.



```
dfCopy[dfCopy['Indication'].isnull()]
```

	Indication	Diabetes	IHD	Hypertension	Arrhythmia	History	IPSI	Contra	Label
130	NaN	no	no	no	yes	yes	98	100	Risk
505	NaN	no	yes	yes	no	no	69	40	NoRisk
1088	NaN	no	yes	no	no	no	79	100	Risk

Sometimes I found that a query may return contradicting values, in this case I would perform some basic statistical functions on the returned data and use the most common value. There were also times when a query would not return enough information to make an educated prediction on the missing value so the whole row was removed.

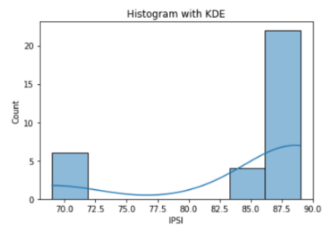


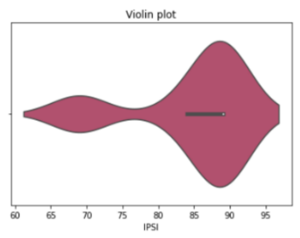
```
The mean of 'no' is 0.9130434782608695 and the mean of 'yes' is 0.0
uld be the best fit
```

```
sns.catplot(data=dfCopy[exprForIndex1406], x="Diabetes", kind='count
```

```
<seaborn.axisgrid.FacetGrid at 0x14b822a2fd0>
```

This method worked very well for the categorical missing data but when it came to continuous numerical attributes it was more difficult to identify a suitable record as the spread of values was usually higher. I first calculated the standard deviation of the query results to identify how spread out the values were, this also told me whether calculating the mean was a viable option, for all the missing numerical values it was not. I then used a histogram and a violin plot to identify the best average value which could be taken from the data.



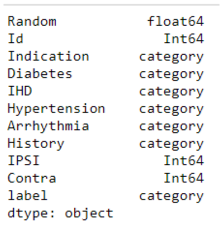


These graphs confirmed that the mean would be a poor indication of the average as the values were so spread out for each missing value. The violin plot showed that the median would be a great substitute for the missing data and gave me confidence that this was the correct decision.

The second method I used for fixing the null values was called KNN (k-Nearest Neighbour) Impute and can be found in section 7.7. This algorithm looks at the closest training examples within the feature space, it does this using the Euclidean distance, K is the number of nearest neighbours which are selected, and the mean is produced from this pool. I chose to set K to 1 as I was using categorical data and using the mean would give me fractions which does not work for categorical values. I used a library from Sklearn to complete this method of fixing but came across the issue that my data was not correctly formatted for this imputer. Because of this I had to move back into the data understanding phase to identify the issues which were stopping me using this technique.

The CRISP-DM model does not allow for movement from the data preparing back to the data understanding phase, so I used a modified version to allow me to move between the two. Back into the data understanding phase I identified issues with spelling and casing in section 6.1, within the ‘Indication’ column which was causing an extra duplicate category to be produced. I also identified an issue within the ‘Label’ column, where there was an additional category of ‘Unknown’. Furthermore, I discovered duplicate values in section 6.3 within the ‘Random’ column, this attribute was supposed to be unique along with the ‘Id’ column.

After identifying the further issues with data, I went back into the data preparation phase. First, I fixed the whitespace problems in section 7.1, I used regular expression to identify offending whitespace values and replaced them with null values to be addressed later. In section 7.2 the data types could now be fixed, I set the relevant attributes to integers.



```
Random            float64
Id                  Int64
Indication       category
Diabetes         category
IHD              category
Hypertension     category
Arrhythmia       category
History          category
IPSI                Int64
Contra              Int64
label            category
dtype: object
```

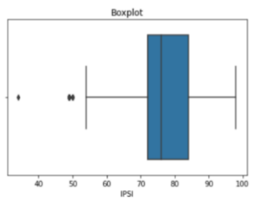
Next, I addressed the problem of spelling and casing in section 7.3, I first used a string function to set all the values within the offending column to upper case and then converted the data type back to that of categorical. I then used the replace function built into pandas to find and replace the ‘Unknown’ value within the Label column, I replaced it with a null identifier. I also fixed the casing of the column name ‘Label’ as the L was lower case which did not match the data description or the theme of the other columns.

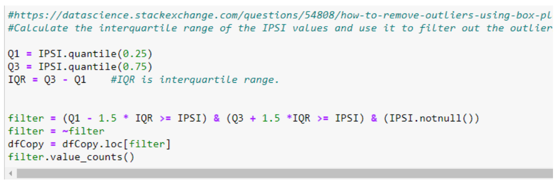
When it came to fixing the duplicated values in section 7.4, I had a few options, so I decided to compare 3 different methods and the effect they would have on the data. First, I tried removing rows which repeated this caused large amounts of data to be lost. After using my domain knowledge and the data description I concluded that the offending row ‘Random’ had no bearing on whether the row was unique, this came from the ‘Id’ column which had no duplicates. Using this information I could confidently remove these two rows as it also helps to address the issue of the curse of dimensionality (Taylor, 2019), this is the problem of the more attributes we have the size of the problem space increases and thus our data set represents a smaller percentage of the available unique rows.

After addressing the previously mentioned problems I could now move back to fixing the null values using the KNN Imputer in section 7.7). I encoded the data replacing categorical attributes with numbers and then used a MinMax scaler to normalize the values. This is done so that none of the attributes create bias, as this can sway the imputers predictions. I then fed the encoded and normalized data into the imputer which successfully replaced all the null values using 1 nearest neighbour.

In section 7.8 I used a third method for replacing the missing null values I simply removed the whole row which contained the offending null value. The final method I used for fixing the null values within the raw data was simply taking the mode of each row and replacing all the missing values of that row with this data, this was in section 7.9. Before I could do this, I needed to encode the data so that it was in numerical form and use the same MinMax scaler to normalize it. I then used the SimpleImputer function built into Sklearn to find the MODE of each column and fix the null values.

In section 7.5 I could fix the final thing as part of the data preparation stage of the process is the erroneous values within the numerical attributes, these erroneous values can be identified by using some basic statistical methods and graphs. First, I calculated the standard deviation to discover how spread out the values within the IPSI and Contra columns. Then I calculated MEAN and Median of each attribute and compared the results, this confirmed what the standard deviation told me. Then I plotted the attributes using a histogram and boxplot, the boxplot allowed me to identify outliers within the IPSI column. I then used a filter to remove all the outlier values within the IPSI column.



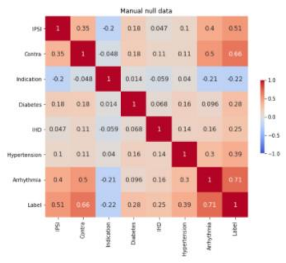


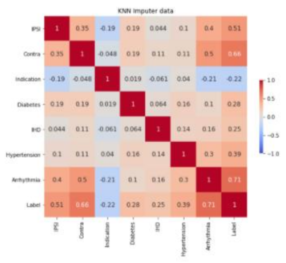
```
#https://datascience.stackexchange.com/questions/54808/how-to-remove-outliers-using-box-pl
#Calculate the interquartile range of the IPSI values and use it to filter out the outlier

Q1 = IPSI.quantile(0.25)
Q3 = IPSI.quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.


filter = (Q1 - 1.5 * IQR >= IPSI) & (Q3 + 1.5 *IQR >= IPSI) & (IPSI.notnull())
filter = ~filter
dfCopy = dfCopy.loc[filter]
filter.value_counts()
```

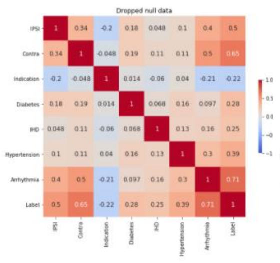
Now that the data was in its final form of cleaning and transformation, I could now move back into the data understanding phase and identify some trends as well as perform feature selection, this happened in section 8.1. To decide which attributes to keep I first calculated all the unique possible values of each attribute. In its current state the data set could have up to 898,560 unique values, if I had not removed the previous Random and Id columns this number would have been much larger as just the Random column could have up to 1000000 possible combinations.

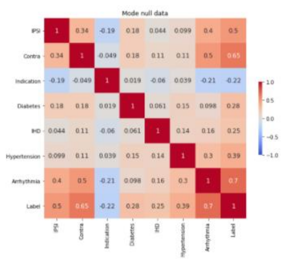
One method for successfully performing feature selection is using wrapper techniques, this involves using combinations of attributes and using statistics, calculating which is the most efficient combination of columns. The issue with this technique is that it is very computationally expensive as you need to train all the subsets of the attributes. I instead used basic statistics and diagrams to show how much effect each attribute had on identifying the label. This included correlation and variance, I identified one column which had little to no effect on whether a person was identified as being at risk, this was the History attribute, so it was removed.

Next, I used visual representations of the data to show trends in section 8.2, this included histograms and count plots. I used these graphs to show strong correlations between different attributes and the Label attribute that I was looking to predict with classifiers later. I also used heatmaps of the correlation data to prove that the alternative ways I used to fix the null values did not change the relationships between the attributes and the Label a significant amount.
The fourth stage of the CRISP-DM cycle is the modelling phase, this occurred in section 9,









this entails selecting and applying models and classifiers, using the cleaned data to produce predictions. I decided to compare two different classifiers, decision trees and a Multi-layer Perceptron.

In section 9.1, I needed to decide which set of cleaned data I wanted to use to do my final modelling and evaluation on. I took each data set, each of them cleaned in slightly different ways and split them using the Sklearn train_test_split function, I chose a random 70/30 split of my data and fed it into the Sklearn DecisionTreeClassifier function, I then calculated the MEAN error score and stored it in a list. I did this 1000 times for each set of data so that I would have a reasonable number of data points for each to evaluate.

The next stage of the CRISP-DM cycle is the Evaluation phase, the CRISP-DM model does not all for movement back and forwards between the modelling and evaluation phase but as I am using my own version of CRISP-DM I allowed myself to do this. The evaluation phase involves evaluating the model and then reviewing the steps taken. It also involves answering the questions of whether the business objectives have been met, as well as the usefulness of the results. I used the evaluation phase to compare the results gained from the previously mentioned stage, for the 4 different data sets in section 9.1.5. I plot the data from MEAN error score values for each data set on box plots and histograms. I also produced confusion matrix for each data set and included a second normalized version for each.

I then went back to the modelling phase of the cycle to properly model the data I chose from the previous evaluation step. In section 9.2.2, I normalized the data to remove any bias towards larger numbers. I then compared splitting the data using two different functions, train_test_split and K fold split. I calculated the metrics using both split techniques and chose a method accordingly.

I then went back to the Evaluation phase in section 9.2.6 and produced statistical metrics on the models, this included mean absolute error and mean squared error values for both the training data and the test data. I also produced a confusion matrix for each model from the predicted data and the test data, using this I could produce more metric statistics such as the Accuracy, Sensitivity, Specificity, and precision of the models.

The Final stage of the CRISP-DM process is Deployment, this stage depends on the original business requirements. Usually, it would involve creating a report or a piece of software. The software produced would be repeatable and allow a business or entity to repeat the stages which were completed by the previous steps, including data prep and transformation, and all the way to modelling and evaluation. Models also need constant supervision and monitoring as the data being used can change, this would call for more testing to ensure newer iterations of the model perform on par with older versions.

As this is a university project the artifact created will not actually be deployed in industry, but if the system were to be deployed, I would need to make certain considerations. The system is designed to be used in a medical setting and contains sensitive patient information; this mean it should not be accessible over the internet. As there is a possibility that updates may need to be applied in the future, physical access to the computer must be available to engineers. The system must not have full autonomy, or the final say on any patient decisions such as prescription or diagnoses', because of this the system should also have a high level of explainability.

Results

Boxplots representing a comparison between 4 different sets of cleaned data

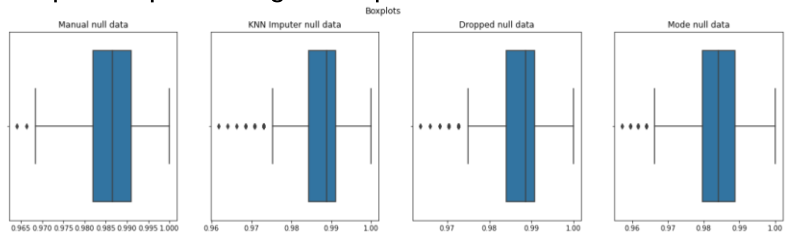


Figure 2

Histograms representing a comparison between 4 different sets of cleaned data

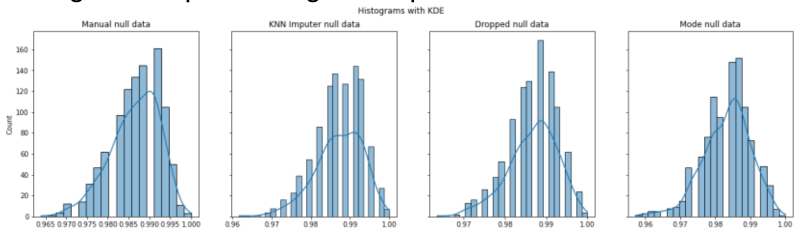


Figure 3

Table data containing the standard deviation of 4 different set of cleaned data

Data set	Mean	Standard deviation
Manually fixed	0.9864662162162162	0.005976290556782635
KNN Imputer	0.9871824324324325	0.006161266052935117
Removed rows	0.9867448747152621	0.006080869336863736
MODE	0.9833310810810811	0.006800981022609673

Table 1

Two sets of comparison confusion matrix, produced by 4 different sets of cleaned data

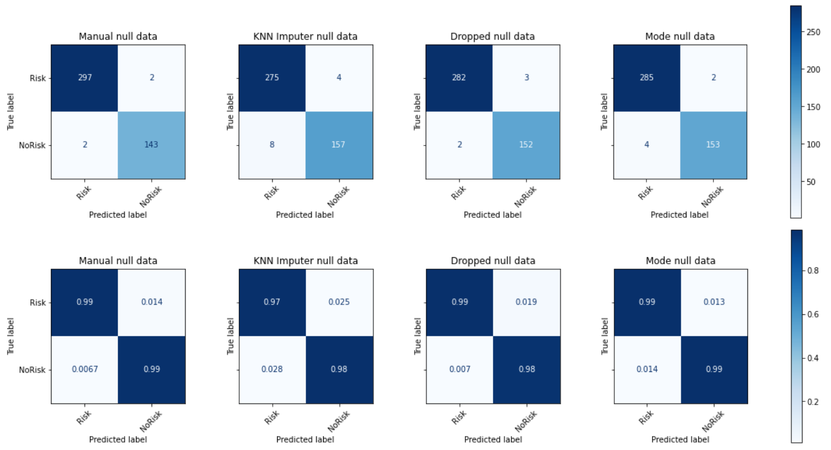


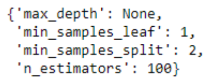
Figure 4

Table data containing the confusion matrix metrics of 4 different data sets

Data set	Accuracy	Sensitivity	Specificity	Precision
Manual Null	0.990991	0.986207	0.993311	0.986207
KNN Imputer	0.972973	0.951515	0.985663	0.975155
Removed Null	0.988610	0.987013	0.989474	0.980645
Mode Null	0.986486	0.974522	0.993031	0.987097

Table 2

Table data containing the Random Forest optimal parameters



```
{'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 100}
```

Table 3

Table data containing the time taken for models to train and fit the data set

Model	Time (Seconds)
Decision Tree	0.0028
MLP Classifier	91.0065
Random Forest	157.5764

Table 4

Table data coming the accuracy score of Random Forest and Decision tree classifiers

Classifier	Accuracy score
Decision Tree	0.9864864864864865
Random Forest	0.990990990990991

Table 5

Table data containing the Mean accuracy score of 2 different data split methods

Type of split	Mean accuracy score
test train split	0.9887387387387387
Kfold split	0.9912070233498804

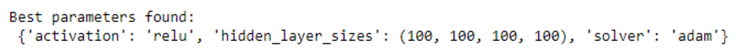
Table 6

Table data containing the statistical metrics of 2 different classifiers

Model	Data	Mean absolute error	Mean squared error
Random Forest	Training	0.000968054211035818	0.000968054211035818
	Test	0.013513513513513514	0.013513513513513514
MLP	Training	0.01452081316553727	0.01452081316553727
	Test	0.018018018018018018	0.018018018018018018

Table 7

Table data containing Mean accuracy score of hyper-parameter optimization



```
Best parameters found:
 {'activation': 'relu', 'hidden_layer_sizes': (100, 100, 100, 100), 'solver': 'adam'}
```

Table 8

Two sets of confusion matrix, one normalized, produced from the predicted train and test data of the manually fixed data from two classifiers

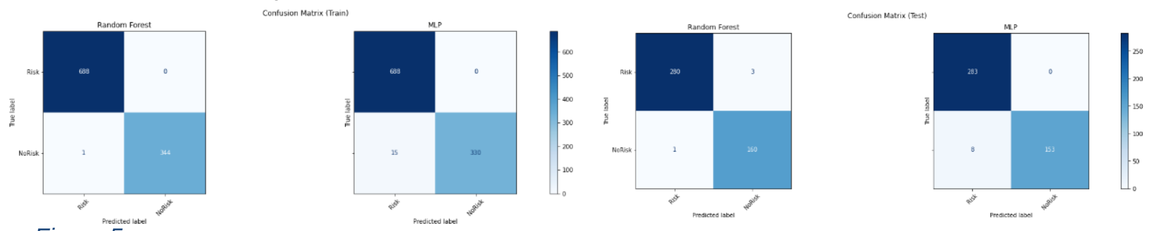


Figure 5

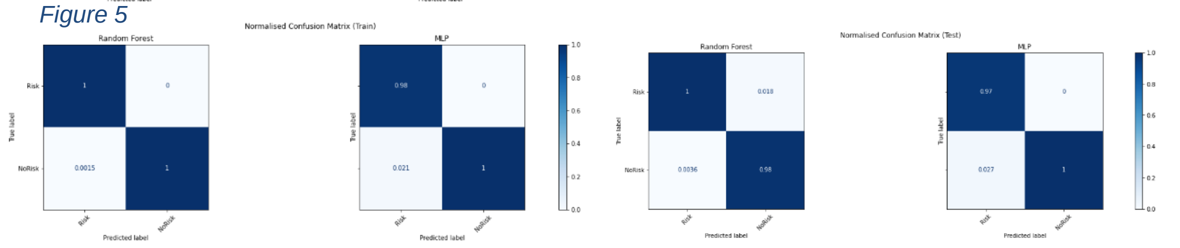


Figure 6

Statistical metrics produced from the confusion matrix from two different classifiers

Model	Data	Accuracy	Sensitivity	Specificity	Precision
Random Forest	Train	0.999032	0.997101	1.000000	1.000000
	Test	0.990991	0.993789	0.989399	0.981595
MLP	Train	0.985479	0.956522	1.000000	1.000000
	Test	0.981982	0.950311	1.000000	1.000000

Table 9

Evaluation & Discussion

The first step to generating the results from the previous section was to first decide which data set I would be using for the classification. I compared 4 different methods of cleaning the data, figure 2 shows 4 boxplots each one representing a different data set. The graphs were generated by recording the MEAN accuracy score 1000 times for each set of data. We can see that each data set produced very similar values, with a tight distribution with a small spread. Figure 3 supports this observation; we see the histograms have a left skewed distribution with one large peak, suggesting the score was more likely to occur under the mean of the graph than above. Using table 1 we can see the mean for all the data sets is within a few decimal places of 98%, showing that each data set performs extremely similarly. Table 1 also shows the standard deviation of each data set and we can see that they are all low showing a small spread, we can also observe that they are all very similar showing that no one set of data performed better or worse consistently.

This is most likely due to the fact there was such a low number of null values compared with the raw data set size. Figure 4 reiterates the observations from the other metrics showing that each data set got a similar number of predictions correct. Table 2 shows that all the data sets accuracy metric is very high, showing they predict labels at a very similar rate. Along with the rest of the confusion matrix metrics we can see that all the data sets predict whether a patient is at risk or not at about the same rate. I have chosen to go with the values that I fixed by hand but any of the data sets would give extremely similar results and would be fine to use to answer the business aims.

Next, I needed to decide how to normalize the data so that no bias was introduced due to number being larger and then being given more weighting. There are two main choice when it comes to normalization, MinMax scaling and Z-Score scaling. MinMax scaling is usually used for classification problems such as the problem I was solving, and Z-score usually is used for regression problems (Brownlee, 2019). It is also mentioned in the Sklearn documentation that “When doing classification in scikit-learn, y is a vector of integers or strings” (*Supervised learning: predicting an output variable from high-dimensional observations — scikit-learn 0.23.2 documentation*, no date), the problem with this is that when I experimented with Z score scaling my a values became floats so they could no longer be used with the classifiers.

The next step to producing the results involved deciding on a model to use. I decided to compare two different types of models, a neural network, and a decision tree-based classifier. First, I compared a standard decision tree against a random forest classifier, this is an ensemble of decision trees. First, I created a decision tree classifier and fit the training data, I then did the same with a Random forest classifier.

I then compared the mean accuracy score of each classifier, this score is calculated from the prediction of Y with respect to actual Y. This metric tells us how consistently correct the estimator is at predicting whether the patient is at risk or not, comparing this value for each model gives us a rough indication of which model is superior. I found that the Decision Tree scored higher with a value of 0.986 compared with the Random Forest’s score of 0.990, these values can be seen in table 5. I also compared the time taken to complete the train and fit in table 4, and we can see that Random Forest took significantly longer to finish but did so with a higher accuracy, so I went with this model.

The neural network model I decided to use is named Multi-layer Perceptron classifier (MLP), this would be compared against Random Forest. Both classifiers take multiple inputs which can be categorical or numerical to predict a label. Random forest uses decision trees which use the CART algorithm to produce rules which break down a large data set into smaller subsets, the tree produced can then be used to predict classifications. Random forest runs the decision tree algorithm multiple times with different subsets of the data, it then takes then takes the average of all the runs. Decision trees inherently have the issue of overfitting and by using Random Forest this helps to combat this. The Multi-layer Perceptron classifier (MLP) optimizes the log-loss function using stochastic gradient descent. Both classifiers can be validated using statistical methods.

Both the Random forest and MLP Classifier can take multiple parameters which depending on the data set effect the accuracy of the model. I used hyperparameter optimization in conjunction with both models to find the most efficient combination of parameters. The GridSearchCV function is used to calculate the hyperparameters for each estimator. GridSearchCV implements a fit and score method in a cross-validation grid search, the function tests combinations of parameters to find the optimal set using the mean test score.

Table 3 and table 8 show the optimal parameters calculated for both the Random Forest and MLP classifier. For the MLP classifier I tested the parameters of hidden layer size, solver and activation, the tool then iterates over the combinations of these parameters and finds the most optimal solution. We can see that the ‘relu’ activation along with the ‘adam’ solver, and a hidden layer size of 100, 100, 100 gave the best results. For Random forest I tested the parameters n estimators, max depth, min samples split and min samples leaf. The optimal parameters found were a max depth of none, 100 estimators, 1 min samples leaf and 2 min samples split.

The next decision I had to make was regarding the splitting of the data, I decided to compare two methods for this task. The first method was a simple random 70:30 train test split, the second option was K-fold split. Table 6 shows two mean error scores produced by splitting and training the data separately and the using Sklearn score functions. If we compare the two values, we can see that both are extremely similar. It is important to say that K-fold is sometimes regarded as a better method for splitting depending on the data set. It solves the issue of overfitting which can happen with some models and some data sets if they are not large enough. The fact the scores are so similar between these splitting methods tells us that the simple train test split would give similar results as the K-fold split but offers better performance. Using this information, I decided to use train test split.

The next step was to compare the two models and decide which one is the best to solve the business aims. I used a variety of metrics including, and mean error, along with confusion matrix and their metrics like Accuracy, Sensitivity, Specificity, and Precision.

When it comes to looking at error metrics there are two main options, Mean Absolute Error (MAE) and Mean Squared Error (MSE). Both metrics are used to draw similar conclusions which is to indicate the quality of a model, but they are both suited for different tasks. MSE penalises large errors more, so is better suited for situations where being wrong by a larger amount is worse such as regression problems. In table 7 we see that the MSE and MAE for each set of data is the same, this most likely due to the classification were trying to predict is either Risk or NoRisk, this means the prediction can only ever be out by 1 and thus there is no weighting for large errors. If we compare the test and train data for each model, we see that the test data has a higher Mean Error value, this suggests that some amount of overfitting has occurred, we will be able to confirm this using confusion matrix later. If we compare the mean error metrics of each model against each other, we find the random forest classifier scores slightly lower, implying it may perform better than the MLP classifier.

Next, we can use confusion matrix to evaluate the accuracy of the classifiers, confusion matrix show a summary of the prediction results from both models. Figure 5 shows two sets of confusion matrix, one set was produced using the training data and the other using the test data. If we also look at the normalised versions of these confusion matrix in Figure 6, we can see that the training data for Random Forest is more accurate suggesting there has been a very minute level of overfitting. The MLP classifier on the other hand has a better score on the test data compared to the training data. This must be an anomalous result and if the test was to be repeated these results to change to reflect the Random Forest classifiers. This anomalous result could suggest an issue with the method used for splitting the data.

If we compare the different models' matrix, we see that for the training data for Random forest scores much higher for True Negative with much fewer False Positives. This mean when Random forest predicted NoRisk it was correct more of the time compared with the MLP classifier. If we compare the test data for both models, we find again that Random Forest has much fewer False Positives but has more False Negative predictions, meaning when Random Forest predicted NoRisk it was Risk more of the time than the MLP Classifier.

If we look at table 9, we can compare the metrics produced against these confusion matrixes. Accuracy is used to calculate the number of correct classifications. When we look at accuracy, we see the training data scores higher for Random Forest whereas the MLP classifiers training data scored lower. This tells us Random Forest predicts a patient's label correctly more frequently using the training data than with the test data, whereas the MLP classifier is the opposite. Comparing the testing accuracy between the models we see that random forest scores higher, this tells us it predicts a patient’s label correctly more frequently overall than the MLP classifier.

Sensitivity is the ratio of correct positive estimations in comparison to all the positive examples, this tells us how many patients we correctly identified as at risk in proportion to the number of possible patients who could be at risk. Looking at the sensitivity we see that both models score higher using the training data, this tells us that when they predict a patient is at risk, they do so more accurately using the training data set. The MLP classifier scores lower using the test data on sensitivity compared to the random forest, this tells us the random forest classifier accurately predicts that a patient is at risk more often.

Specificity is the ratio of correct negative predictions in comparison to all of the possible negative, this tells us how many patients we correctly identified as not at risk in proportion to all of the possible not at risk patients. Specificity is again higher for the training set for Random Forest, this indicates that the model has overfit using the training data. The specificity score for the MLP classifier is 1 for both training and test data meaning it predicts a patient is not at risk correctly 100% of the time correctly. The specificity is also 1 for the Random Forest training data, but the test data scored lower meaning the MLP classifier is better at predicting a not risk estimation.

Precision is the measurement of all the correct risk predictions in proportion to all of the predictions that a patient is at risk. The precision like the other metrics is higher for the training data when using the Random Forest model. This tells us that out of all the predictions of risk the test data gets it wrong more of the time. Again, the MLP classifier scores 100% using this metric for the training and test data, meaning it scores better than the random forest model. The MLP classifier has a higher degree of precision when it comes to the test data, meaning it predicts that a patient is at risk correctly more of the time.

These results are not as expected, the MLP accuracy is higher for the test data suggesting something has gone wrong with the generation of the results, if we were to run the test again this most likely would be different. This suggests a problem with the method I used to split the data and in hindsight a K-fold split may have given more consistent results. When we are talking about the test data, the random forest classifier scores better in Accuracy and Sensitivity whereas the MLP classifier has a higher Specificity and Precision. This tells us random forest gets more predictions correct overall as well as more positive predictions out of all the possible patients which could be at risk. The MLP classifier gets more of the not at risk predictions correct out of all of the possible negative estimations, as well as a higher level of precision.

One benefit the MLP classifier has over random forest is that it has a higher level of performance. Using a timer, we can accurately calculate the performance of each model, this data is in table 4. We can see that Random Forest took 157 seconds to complete the task compared to ML which took 91 seconds.

Random Forest is a whitebox model, this means its inner workings are easier to understand and thus has a higher degree of explainability. The MLP classifier on the other hand is a blackbox model, this means that the way it generates its predictions is much more complex and thus harder to understand. Black box models usually offer a higher predictive capability compared to whitebox but the explainability is much lower.

Using the metrics, I have collected and the comparisons I have made, both models produce equal results. Both models require tuning using hyper-parameter optimisation which has a negative effect on performance, although the Random Forest model estimates with a higher level of accuracy compared to the MLP classifier. Using this information, I feel the Random Forest model is the better model to meet the business aims of this task.

References

Brownlee, J. (2019) 'How to use Data Scaling Improve Deep Learning Model Stability and Performance', *Machine Learning Mastery*, 3 February. Available at: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/ (Accessed: 25 November 2020).

Posted by William Vorhies on July 26, 2016 at 9:15am and Blog, V. (no date) *CRISP-DM – a Standard Methodology to Ensure a Good Outcome*. Available at: https://www.datasciencecentral.com/profiles/blogs/crisp-dm-a-standard-methodology-to-ensure-a-good-outcome (Accessed: 25 November 2020).

Problem Definition (no date). Available at: https://www.saedsayad.com/problem_definition.htm (Accessed: 25 November 2020).

Supervised learning: predicting an output variable from high-dimensional observations — scikit-learn 0.23.2 documentation (no date). Available at: https://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html (Accessed: 25 November 2020).

Taylor, C. R. (2019) *Applications Of Dynamic Programming To Agricultural Decision Problems*. CRC Press.