MIRNA ARIVALAGAN - 220142881

MACHINE LEARNING – SIT720

TRIMESTER 2 2021

Table of Contents

Task 1

Read the article and reproduce the results presented in Table-4 using Python modules and packages (including your own script or customised codes). Write a report summarising the dataset, used ML methods, experiment protocol and results including variations, if any. During reproducing the results:

To replicate the results in table 4, I first checked the data set for any missing values and there weren't any missing values, I then proceeded to drop the **time** feature as this feature was not used to produce the results in table 4.

The researchers had 2 slightly different approaches in their model building process, as they had models where the data was split 80%-20% and models that were split 60%-20%-20%.

I replicated the same and below is the list of models where I have applied a 805-20% split:

- Random Forest
- Decision tree
- Naïve Bayes
- XGBoost
- Linear Regression
- One Rule

And here's the list of the models that I have split 60%-20%-20%:

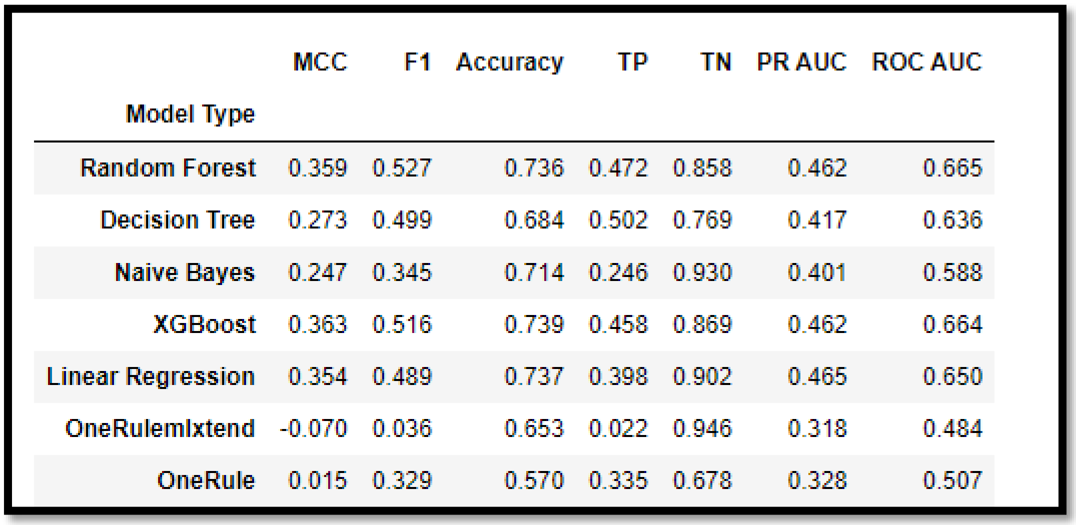
- KNN
- SVC Linear
- SVC Radial
- ANN

I have designed 3 different sets of loops that runs the models 100 times to get the average results for each model. The first set of 100 times loop is a loop for the 80%-20% split models except for linear regression. This loop splits the data randomly within each run making the results more reliable. The second 100 times loop that I had developed is a loop that runs 100 times, and it also has a nested loop within it running to find the best K value for KNN, best C value for both SVC models and the best hidden layer unit for the ANN model. Within this nested loop the validation dataset was used as a method to validate the best parameters for the model. The linear regression loop is similar to the first one, the only difference is that I have transformed the continuous predictions to a value of either 0 or 1 based on the 0.5 threshold that I have set.

I did not do any normalisation or scaling to the datasets as the researchers did not mention anything about this step in the research paper. I had a look at the R script but could not work out what was done in this aspect.

I also noticed that some of the model's R scripts, it appears that some oversampling was done for some models which was not mentioned in the research paper, so I have opted not to do this.

80% - 20% Split Results:



Model Type	MCC	F1	Accuracy	TP	TN	PR AUC	ROC AUC
Random Forest	0.359	0.527	0.736	0.472	0.858	0.462	0.665
Decision Tree	0.273	0.499	0.684	0.502	0.769	0.417	0.636
Naive Bayes	0.247	0.345	0.714	0.246	0.930	0.401	0.588
XGBoost	0.363	0.516	0.739	0.458	0.869	0.462	0.664
Linear Regression	0.354	0.489	0.737	0.398	0.902	0.465	0.650
OneRulemlxtend	-0.070	0.036	0.653	0.022	0.946	0.318	0.484
OneRule	0.015	0.329	0.570	0.335	0.678	0.328	0.507

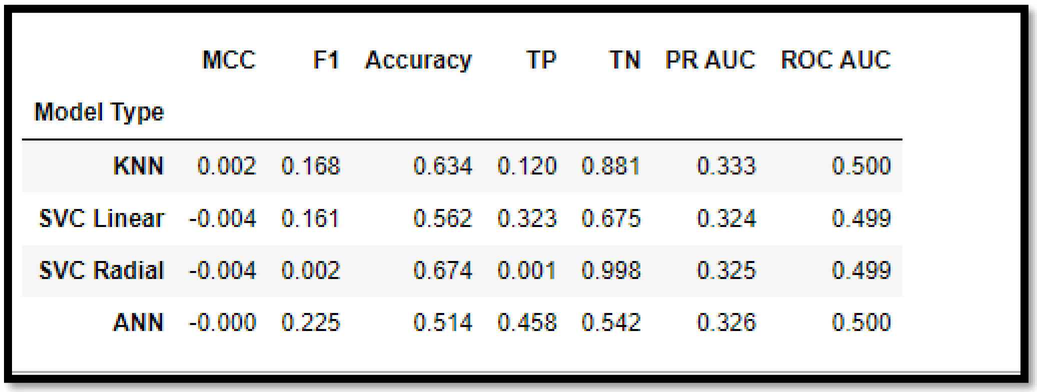
For the **Random Forest** model, my results for MCC, F1, Accuracy, TP & TN are quite close to the results in table 4. I believe the variation in results is due to the randomised nature of the loops and how the data is split. The PR and ROC AUC are quite different compared to table 4. I believe this is due to using a sklearn package to calculate these metrics. I had a look at the R script for this and could not work out how this was calculated. **I can also see within the R script that the researchers have done an oversampling method which was not mentioned in the research paper.**

The **Decision Tree** results are quite different across all metrics. I have done a little bit of digging to understand why this may be and have found that there is a difference between the Decision Tree algorithm in R and Python. **The main difference in the algorithm between both languages is that R implementation has post pruning inbuilt within it which limits the issue of overfitting. And the other difference to note is that the R algorithm handles categorical variables and does not require any data scaling**. I have run my python without any data scaling, so it makes sense that the results are quite different.

The **Naïve Bayes** model produced results that are very similar to the results produced by the researchers. The **XGBoost** and **Linear Regression** model also produced results that are like table 4 across all metrics.

Unfortunately, I was not able to replicate the results in table 4 for the One Rule model. I have tried 2 methods using 2 different algorithms – Dummy Classifier and Mlxtend's OneRClassifier and was not able to get any results close. **By reading the python one rule algorithms documentation, both these algorithms require data scaling prior to being fitted into the model which I have not done, as we are lacking clarity around what data scaling techniques were used by the researchers.**

60% - 20% - 20% Split Results:



Model Type	MCC	F1	Accuracy	TP	TN	PR AUC	ROC AUC
KNN	0.002	0.168	0.634	0.120	0.881	0.333	0.500
SVC Linear	-0.004	0.161	0.562	0.323	0.675	0.324	0.499
SVC Radial	-0.004	0.002	0.674	0.001	0.998	0.325	0.499
ANN	-0.000	0.225	0.514	0.458	0.542	0.326	0.500

The **KNN** model my results were slightly better than the results in table 4. The best K that **I have found on my loop is K = 7**, which is different to the researchers K = 3. Do note that this value changes each time I run my notepad due to the randomised nature of the data splits and the 100 iterations.

Both my **SVC linear** and **Radial** models were no where close to the results that the researchers have reported. For SVC Linear, I had opted to use the SVCLinear algorithm, as when I tried standard SVC with kernel = linear, it was running for more than 12 hours. For SVC Radial, I have used standard SVC with kernel = 'rbf' passed through it. **My best C value for both Linear SVC & Radial SVC is C = 10.** The C Values the researchers reported for Radial is C = 10, and for linear, it is C = 0.1. **I believe the reason why the results are so different is because the R SVC model packages automatically scales the data for you, whereas with python we need to do some scaling prior to fitting the data in the**

model. I did a test and scaled my data and was able to get the results close to table 4, thus proving this is the reason why there is a difference, and again I have reported without scaling as we are being marked on following the process, I have opted to not scale the data prior.

For the **ANN** model, I have opted to use Sklearn MLP Classifier package as this algorithm is a neural network algorithm that has been designed to solve classification analysis. For this model, I ran it 100 times within a loop to find the optimum number of units. The research paper did not specify the range of units and layers but reported as 1 layer with 100 units was the best result. I was able to find the range of units from the Lua script but could not find the range of hidden layers. I ran my model with the following range of units (5, 10, 25,50,75,100,125,150,175,200,225,250,275,300). I was unable to get my results close to table 4. I think this could be due to the differences in the package between Python and Lua. Another reason could be due to the small sample size, Neural Network based models tend to work better when you are dealing with large datasets.

To conclude, most models that were split 80%-20%, I was able to replicate close to table 4 result except for OneRule. The models that were split 60%-20%-20%, I was not able to replicate close to table and this is due to not having much clarity around the hyperparameter tuning and validation methods in the research paper.

Task 2

Design and develop your own ML solution for this problem. The proposed solution should be different from all approaches mentioned in the provided article. This does not mean that you must have to choose a new ML algorithm. You can develop a novel solution by changing the feature selection approach or parameter optimisations process of used ML methods or using different ML methods or different combinations of them. This means, the proposed system should be substantially different from the methods presented in the article but not limited to only change of ML methods. Compare the result with reported methods in the article. Write a technical report summarising your solution design and outcomes.

Task 2.1: Objective

The proposed solution for predicting a patient's survival or death with relation to heart disease is a voting ensemble that is comprised of a Random Forest, Support Vector Machine and Logistic Regression model. The reason why I have decided to build a voting classification model is because of its un-biased nature. I believe when trying to solve a problem, we should be diverse in our approach, and by using a voting classifier I am able to combine predictions from 3 very different models that each has its own strengths. The voting classifier works by looking at the predictions of all 3 models and using the mode or majority value of the combined predictions.

I have decided to use a Random Forest model as this is an ensemble model that is quite powerful and robust to noisy datasets. The Random Forest model combined with a voting ensemble would be suitable to scale to real life data that often has missing values. I've also picked Random Forest as it is robust to outliers and handles the class imbalance problem well, both issues are present in the dataset that we are working with.

I have opted to use SVC in my ensemble because this model works well with the nonlinear data that we are working with by using the kernel trick. SVC is also useful when there appears to be data distribution that is not normally distributed, and the data that we are working with, some features appear to be skewed. SVC is also robust to outliers and is not prone to overfitting issues.

The third model that I have decided to use in the ensemble is the logistic regression model. I have opted to use this model as this is classical model that is often used to solve binary classification problems. Logistic regression makes no assumptions around the class distributions in our dataset, and by adjusting the regularization L1 and L2, its robust to over fitting issues as well.

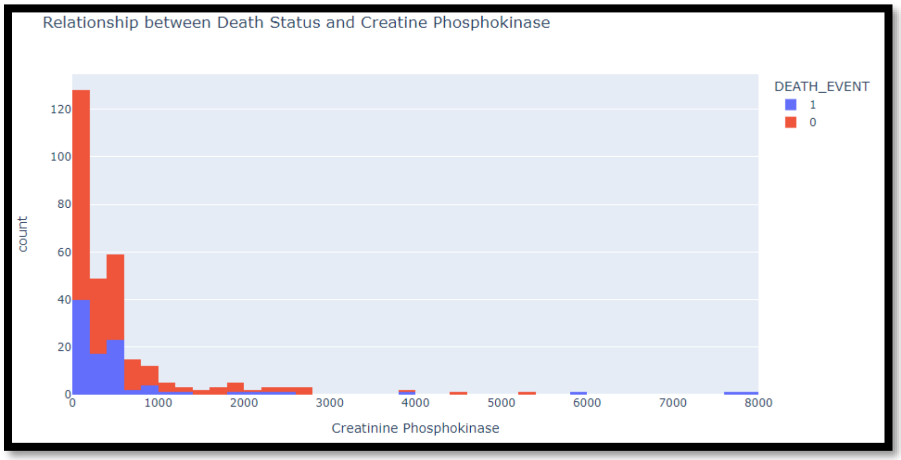
I believe by combining a tree-based model, a regression based model, and a support vector machine based model, we are maximizing on the advantages of each method and will be able to produce more accurate results and will be able to limit the spread of the predictions making the model more robust.

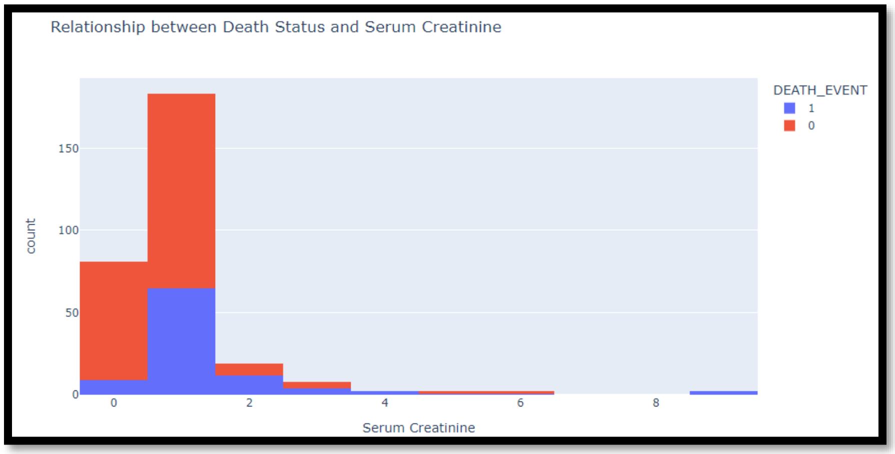
Task 2.2: Model Building

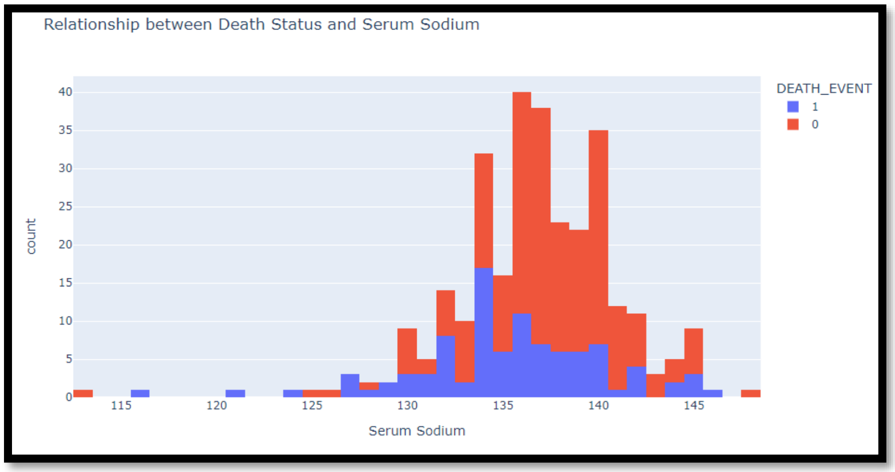
Exploratory Analysis and Data Pre-processing

Prior to deciding on which models to use, I first analysed the dataset that we are working with. I have found that the following features to be quite skewed:

- Creatinine Phosphokinase
- Serum creatinine
- Serum Sodium

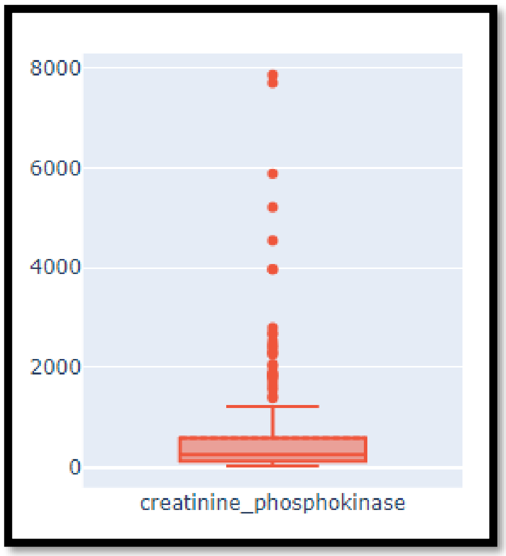


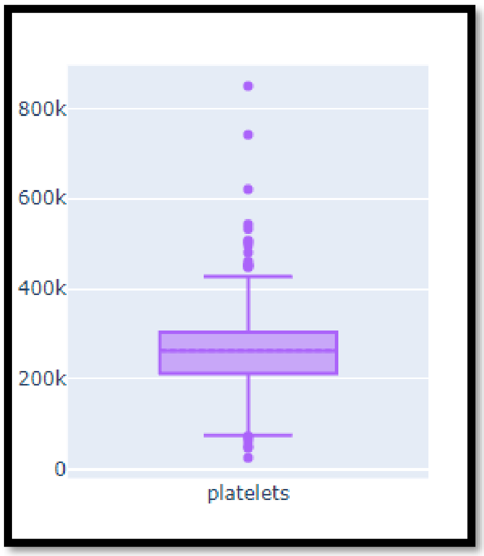


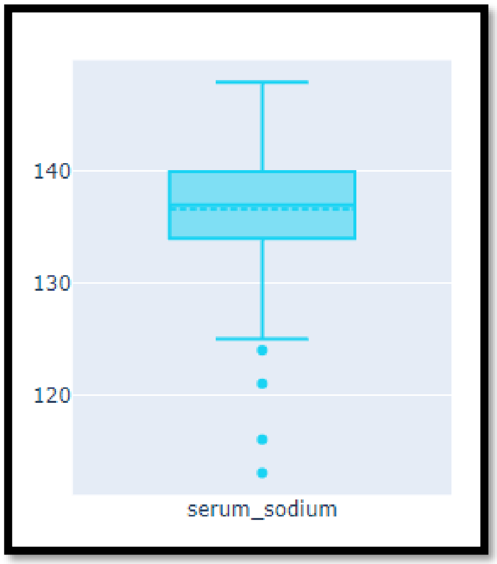


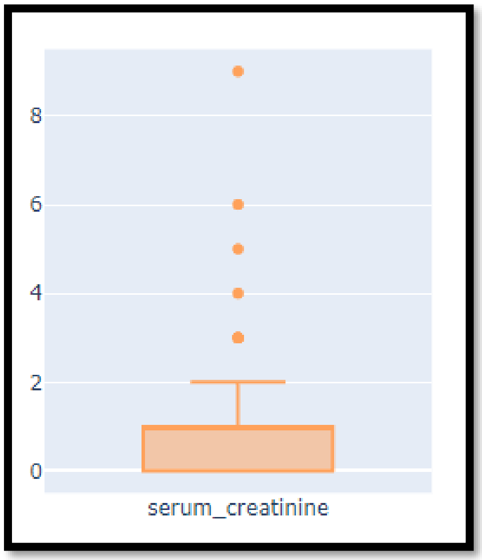
Our target variable "Death Event' also suffered a class imbalanced problem and there was outliers present in our dataset for the following features:

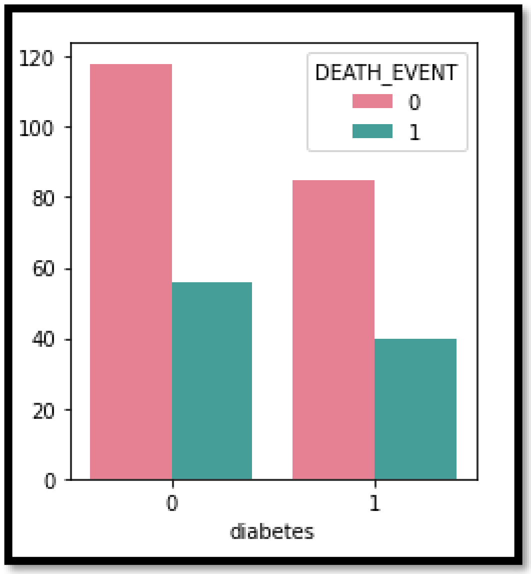
- Creatinine Phosphokinase
- Platelets
- Serum Creatinine
- Serum Sodium











Based on the findings around the data that we are working with, it justifies the decision to use Random Forest, SVC and Logistic Regression for this task as these models are robust to outliers, imbalanced data and skewed data.

Before building the model, I first checked if there was any high correlation with the features, and there did no appear to be any features with multicollinearity issues that we needed to handle.

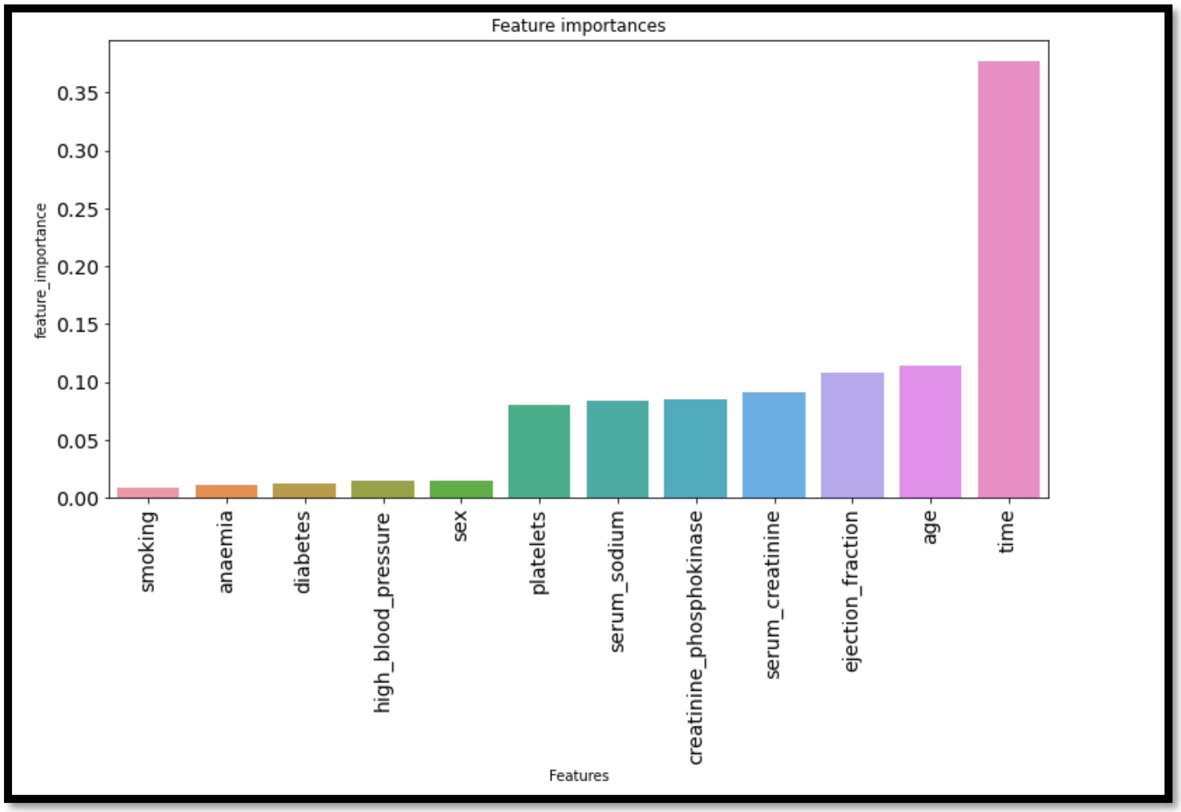
I then decided to use a Robust Scaler to normalise the data that we are using. As we have outliers in the data set, this scaling method is robust to outliers, it works by removing the median value that can be distorted if there are outliers and scales the data using the interquartile range between the 1st and 3rd quartiles. I split the data set into 70% train set and 30% test before running the data through the scaler to make sure there is no data leakage.

Feature Importance

Upon splitting and scaling the data set, I then proceeded to see what features are important to be used with the final model. I decided to use a Random Forest model to decide this as we will be using the random forest in our ensemble model. I fitted the data into a random forest model with the default values as the hyperparameter, alongside the default value, I specified class_weight as balanced to mitigate the class imbalanced issue that we are dealing with in this dataset.

From the default model, **Time** appeared as the most important feature, followed by **Age** and **Ejection Fraction**. I have decided to use **Time, Ejection Fraction** and **Serum**

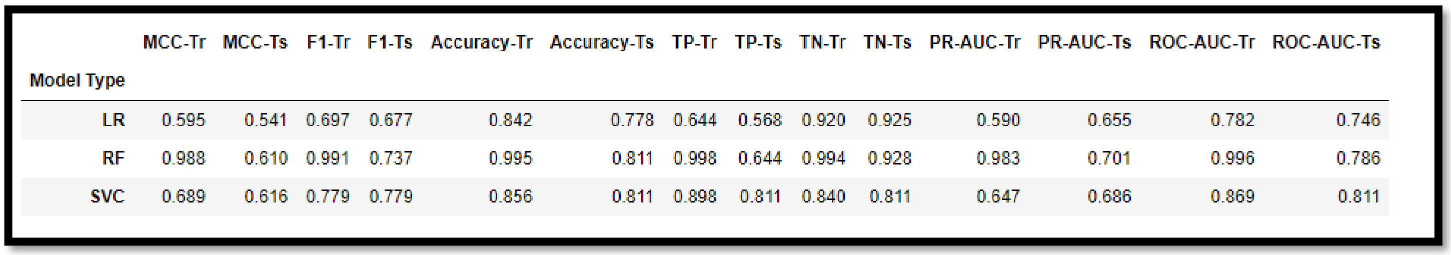
Creatinine as the features in my final model. I have omitted age, as from my testing what I found was that including age decreased the performance of the SVC and Logistic Regression models tremendously. I have decided to include serum creatinine as a feature as this was the 4th best feature, and the research paper has reported that this is one of the more powerful predictors.



Estimator Models Initial Testing

I first ran all 3 models with the default parameters within a 100 run loop and got the mean values across all metrics that we are evaluating. Alongside the default parameters, I included a class weight = balanced as a parameter for the Random Forest and SVC model.

The table below outlines the average results between the train and test datasets using the default values:



Model Type	MCC-Tr	MCC-Ts	F1-Tr	F1-Ts	Accuracy-Tr	Accuracy-Ts	TP-Tr	TP-Ts	TN-Tr	TN-Ts	PR-AUC-Tr	PR-AUC-Ts	ROC-AUC-Tr	ROC-AUC-Ts
LR	0.595	0.541	0.697	0.677	0.842	0.778	0.644	0.568	0.920	0.925	0.590	0.655	0.782	0.746
RF	0.988	0.610	0.991	0.737	0.995	0.811	0.998	0.644	0.994	0.928	0.983	0.701	0.996	0.786
SVC	0.689	0.616	0.779	0.779	0.856	0.811	0.898	0.811	0.840	0.811	0.647	0.686	0.869	0.811

The model with **highest test MCC value was SVC with a value of 0.616** followed by Random Forest and Logistic Regression**. SVC outperformed Random Forest with the F1 value of 0.779**. In terms of **Accuracy, both SVC and Random Forest scored a high accuracy of 0.811**. **SVC had the highest TP rate of 0.811** followed by Random Forest and Logistic Regression. **Random Forest had the highest TN rate with a value of 0.928** followed by Logistic Regression and SVC performed the worse in this regard. **Random Forest performed best with the PR- AUC rate with a value of 0.702** followed by SVC. **SVC performed the best with the ROC-AUC rate, with a value of 0.811**. Across all metrics except for TN rate, logistic regression model performed the worse. The logistic regression model was able to predict more accurately if a patient had died.

There is a huge difference in performance across train and test sets for the Random Forest model indicating some overfitting occurring there.

Estimator Models Hyper Parameter Tuning

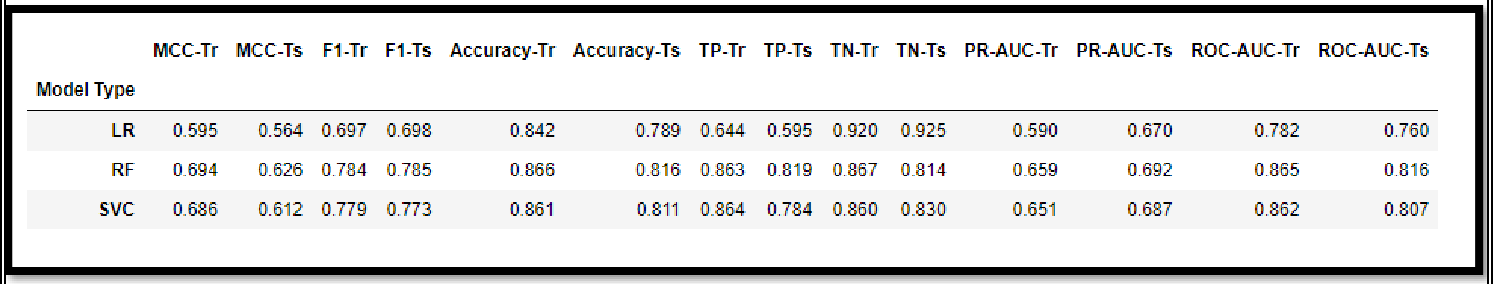
After running the models using the default settings, the next step would be tuning the hyper parameters of these models to achieve an optimum performance. I opted to use Sklearn's Grid Search CV to find the best parameter. This package has an inbuilt cross validation in it to ensure that we are not finding parameters that can over fit a model. As we are tuning more than 1 hyper parameter at a time, I have opted to not do this within the 100 times loop as it will be computationally expensive to do so. Below is a breakdown of the parameter values that were grid searched:

- SVC model:
  - C: [1,10,20,30,40,50,60,70,80,90,100]
  - Gamma: [0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]
  - Kernel: rbf
  - Class Weight: Balanced
  - **Best Parameter Combination: C = 30, Gamma = 0.07**

- Random Forest:
    - Max Depth: [2,4,6,8,10,12,14]
    - Min Samples Leaf: [2,4,6,8,10,12,14,16,18,20,22,24,26,28,30]
    - Min Samples Split: [2,4,6,8,10,12,14,16,18,20]
    - N Estimators: [100,500,700,1000]
    - Class Weight: Balanced
    - **Best Parameter Combination: Max Depth = 12, Min Samples Leaf = 16, N Estimators = 500, Min Samples Split = 2**

- Logistic Regression:
    - Penalty: [l1, l2]
    - Solver: [liblinear, lbfgs]
    - **Best Parameter Combination: Penalty = l1, Solver = liblinear**

After running the grid search, I then ran the optimum parameters for each model 100 times to review the results, and the grid search did improve the over fitting issue with the Random Forest as we are no longer observing variances around 30% more between train and test values, and the grid search did improve the performance of the logistic regression and SVC models a tiny bit. The table below outlines the results after grid search:



	MCC-Tr	MCC-Ts	F1-Tr	F1-Ts	Accuracy-Tr	Accuracy-Ts	TP-Tr	TP-Ts	TN-Tr	TN-Ts	PR-AUC-Tr	PR-AUC-Ts	ROC-AUC-Tr	ROC-AUC-Ts
Model Type														
LR	0.595	0.564	0.697	0.698	0.842	0.789	0.644	0.595	0.920	0.925	0.590	0.670	0.782	0.760
RF	0.694	0.626	0.784	0.785	0.866	0.816	0.863	0.819	0.867	0.814	0.659	0.692	0.865	0.816
SVC	0.686	0.612	0.779	0.773	0.861	0.811	0.864	0.784	0.860	0.830	0.651	0.687	0.862	0.807

<u>Ensemble Model Testing</u>

Upon completing the grid search, I then used the optimum parameters for each estimator model and fitted that to Sklearn's Voting Classifier model and ran the voting classifier model 100 times to see if the results are good. With this test run, I was able to beat all the metrics in table 4 marginally and there did not appear to be any huge variances between the train and test data sets.

Final Ensemble Model

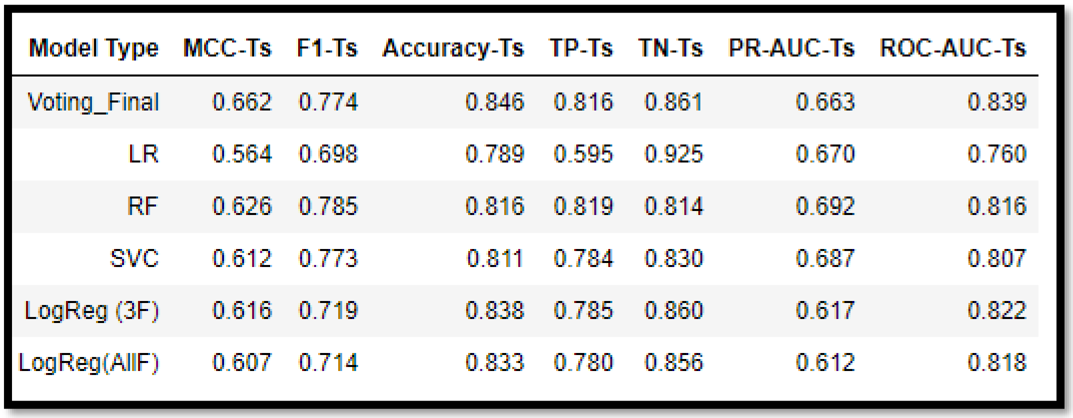
To ensure that the model designed is robust and performs well, I created a loop that runs 100 times, and within this loop I have included the following 2 pre-processing steps:

- Data Split: 70% Train and 30% Test shuffled splits with the target variable stratified to reduce the impact of the class imbalance problem that is present in the data set.
- Scaled data within loop using Robust Scaler.

This loop randomises the data that is used for each run, thus providing more accurate results. I fitted the tuned estimator models within the VotingClassifier in this final model.

Task 2.3: Final Ensemble Performance

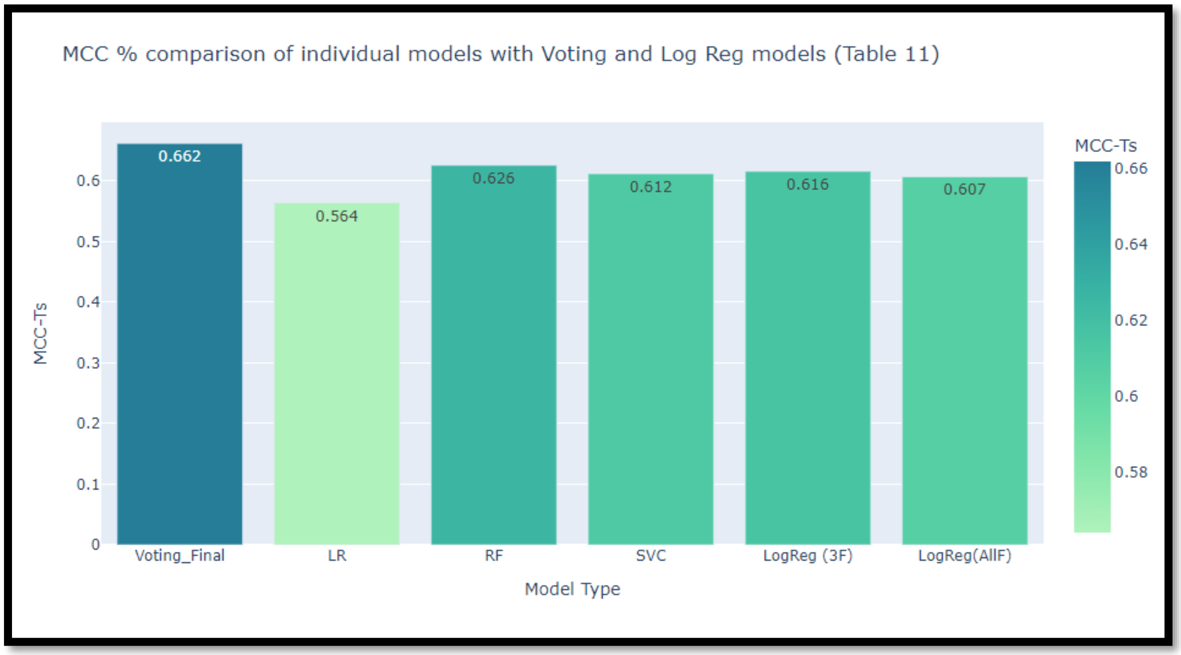
By using the 3 different tuned models within a voting classifier, I was able to beat the performance in table 11 across all metrics. We will analyse each metric below and do note that LogReg (3F) is the results from the logistic regression model using the 3 features by the researchers, and LogReg(AllF) is the logistic regression model by the researchers using all features. **Do note that the individual estimator models performance was captured in a separate loop, so we are not comparing like for like, but will give a good indication of areas that certain estimators are strong with.**



Model Type	MCC-Ts	F1-Ts	Accuracy-Ts	TP-Ts	TN-Ts	PR-AUC-Ts	ROC-AUC-Ts
Voting_Final	0.662	0.774	0.846	0.816	0.861	0.663	0.839
LR	0.564	0.698	0.789	0.595	0.925	0.670	0.760
RF	0.626	0.785	0.816	0.819	0.814	0.692	0.816
SVC	0.612	0.773	0.811	0.784	0.830	0.687	0.807
LogReg (3F)	0.616	0.719	0.838	0.785	0.860	0.617	0.822
LogReg(AllF)	0.607	0.714	0.833	0.780	0.856	0.612	0.818

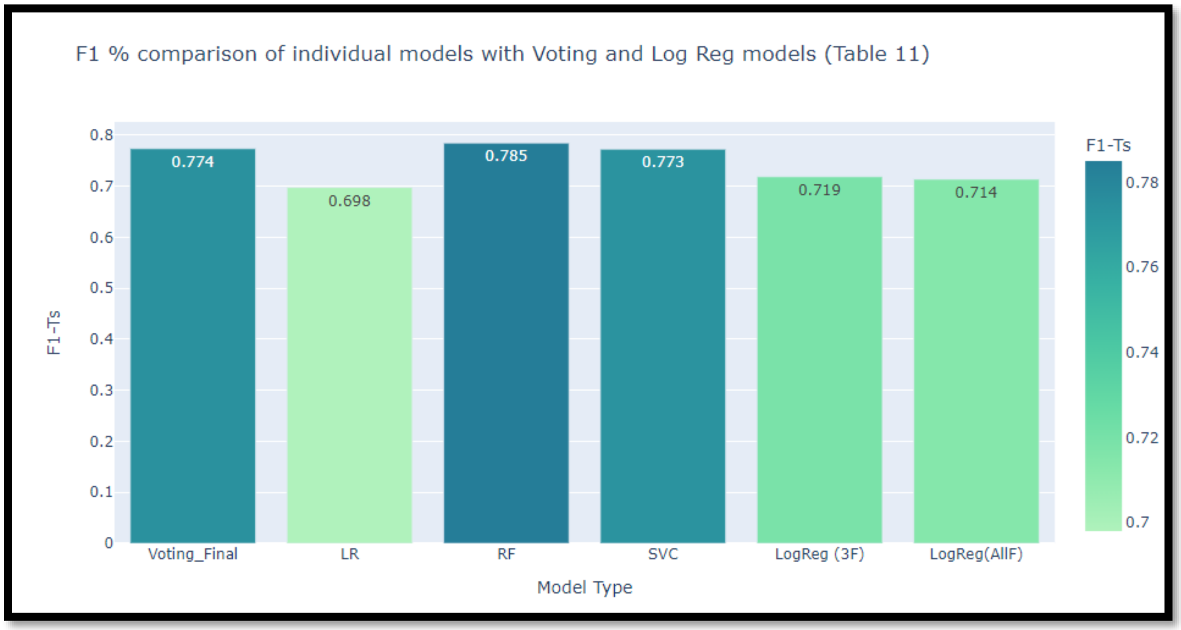
Matthews Correlation Coefficient:

Table 11 best result was a value of 0.616 for this metric that was from the logistic regression model with 3 features. **The voting ensemble model scored 0.662** for this metric so a slight improvement by 0.046. When comparing the Voting Ensemble between the models by the researchers and the individual models that are within the voting classifier, the voting classifier outperformed all, and my tuned logistic regression model performed the worse. What this means is that the voting classifier has leveraged off the stronger learners compared to the weak logistic regression model.



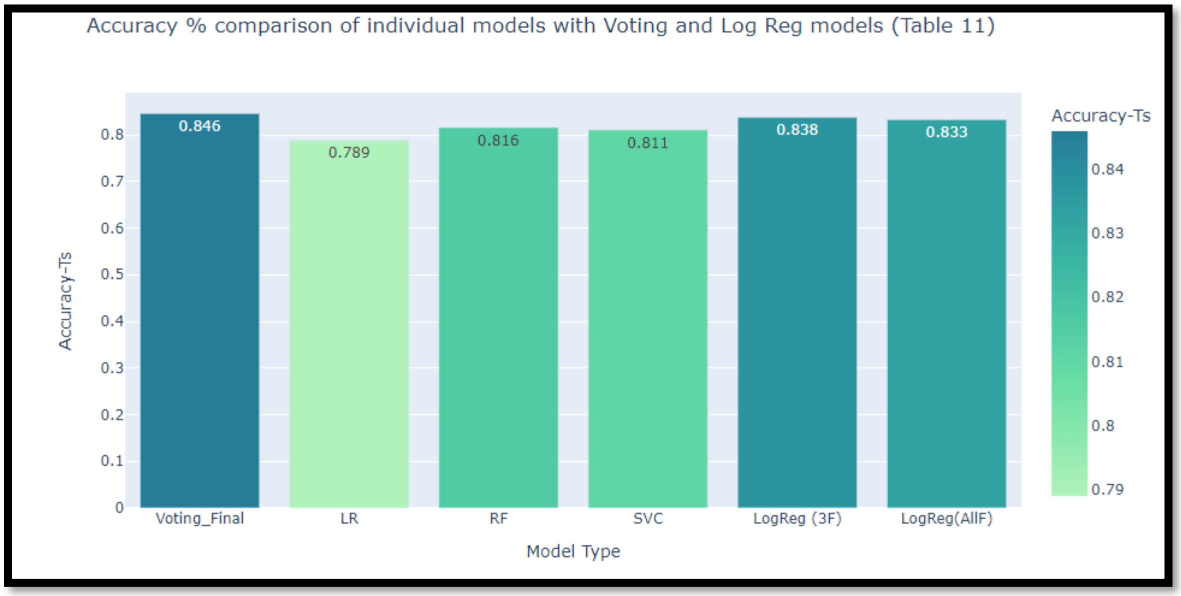
F1 Score:

The Voting Classifier outperformed both regression models in table 11 for the F1 score as well. The best result in table 11 was 0.719, and the **voting classifier scored 0.774**for this metric, so a slight improvement by 0.055. When comparing the results between the ensemble and the estimator models, the Random Forest and SVC model outperformed the voting classifier in this area, with Random Forest having the highest F1 score. This means that overall, the Random Forest model had better ability to classify true positive and true negative values.



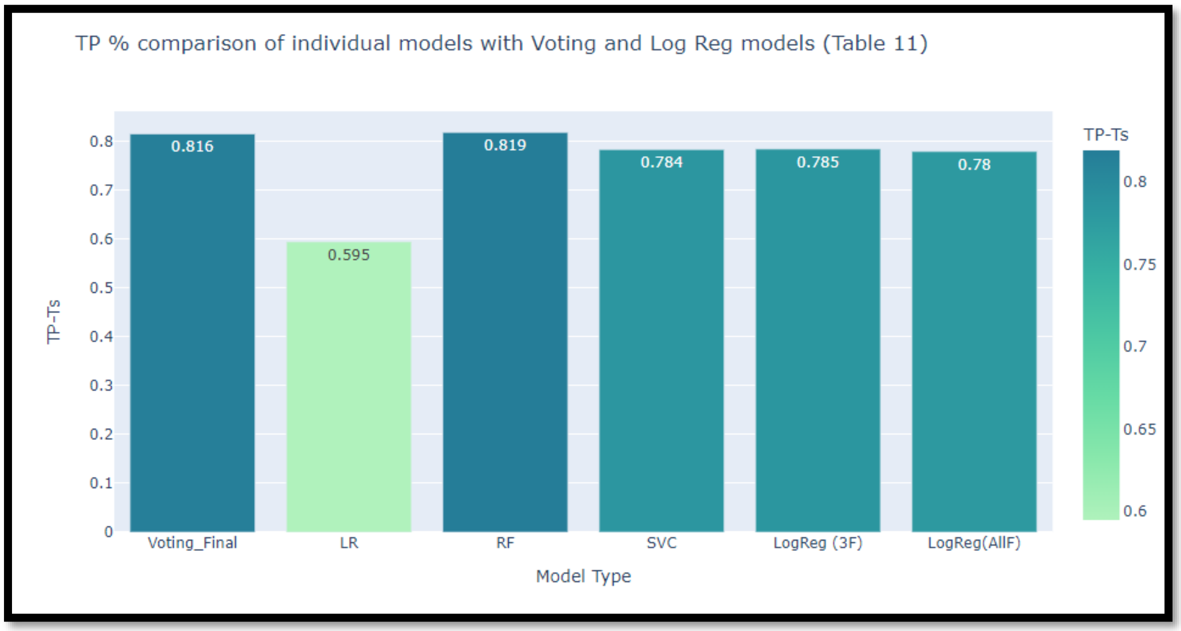
Accuracy:

The Voting Classifier was also able to marginally beat the best regression model value of 0.838 with a **value of 0.846**, so we can say the accuracy performance is very similar. When evaluating with the individual models, the voting classifier outperformed all individual models as well.



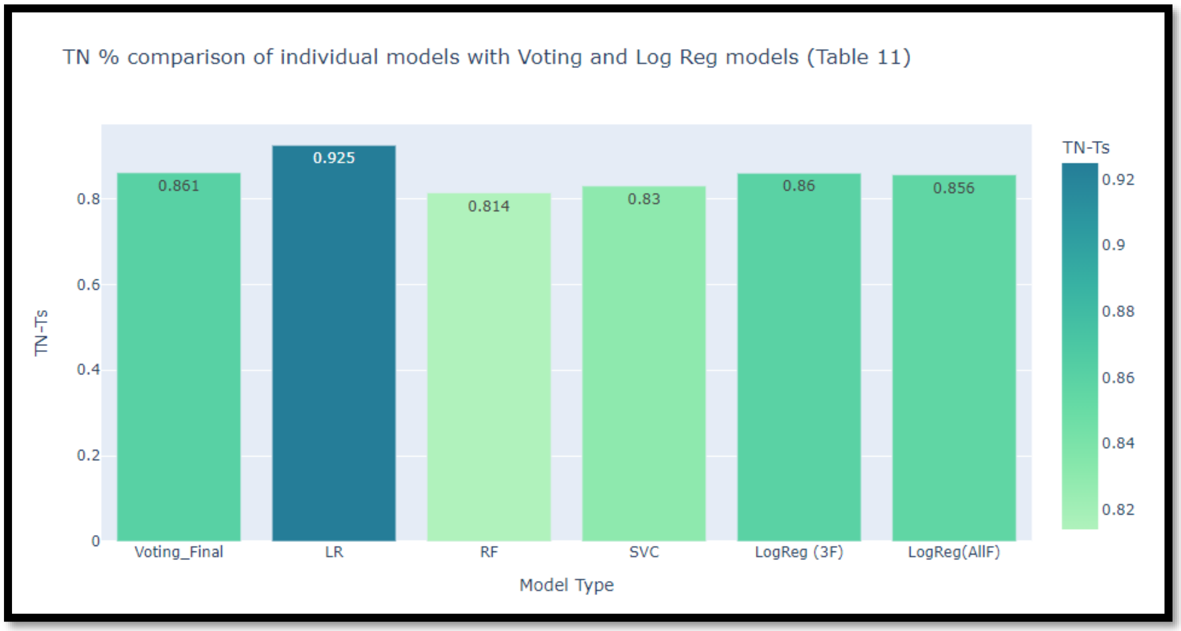
True Positive Rate:

The voting classifier has been able to outperform the best regression model in table, where the regression model scored 0.785, and the **voting classifier scored 0.816**, so an improvement of 0.031. Again the individual Logistic Regression model fitted into the ensemble has performed the worse with this metric, and the Random Forest performed similar to the voting classifier, which means this model is contributing the most to the improvement of this metric.



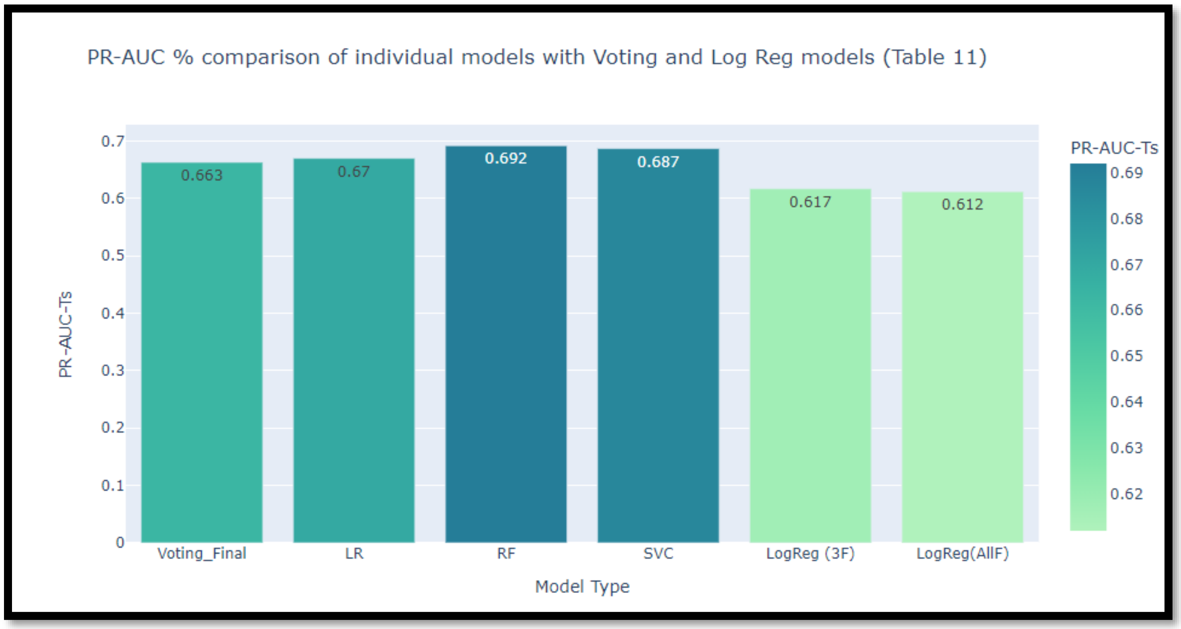
True Negative Rate:

The true negative rate of the voting classifier compared to the regression model in table 11 is very similar, the voting classifier have seen an improvement of 0.01 with a **performance of 0.861 for this metric** compared to the best value of 0.860 in table 11. When analysing the estimator model results, the logistic regression model out performed all estimator models and the voting classifier and also both logistic regression models of the researchers with a value of 0.925, meaning that this model is able to predict our target variable of patient status of Death really well.



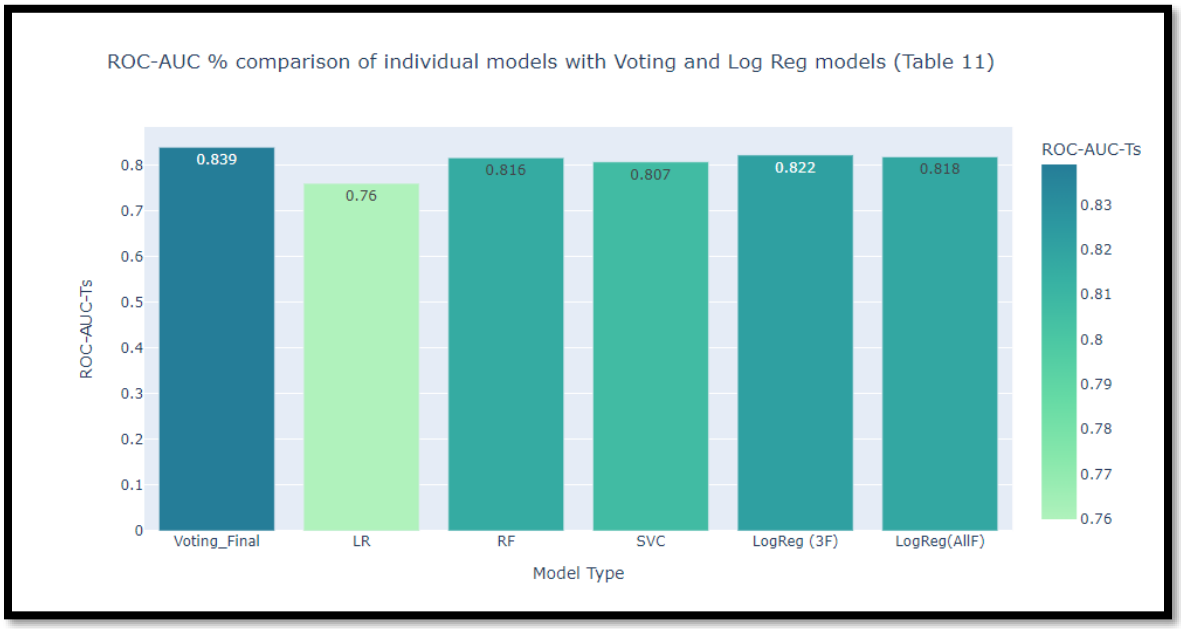
PR AUC:

The voting ensemble was able to outperform table 11's PR AUC results with a **value of 0.663** compared to table 11's best value of 0.617, so an improvement of 0.046. However when looking at all 3 estimator models, all of them had higher scores than the voting ensemble, with Random Forest performing the best.



ROC AUC:

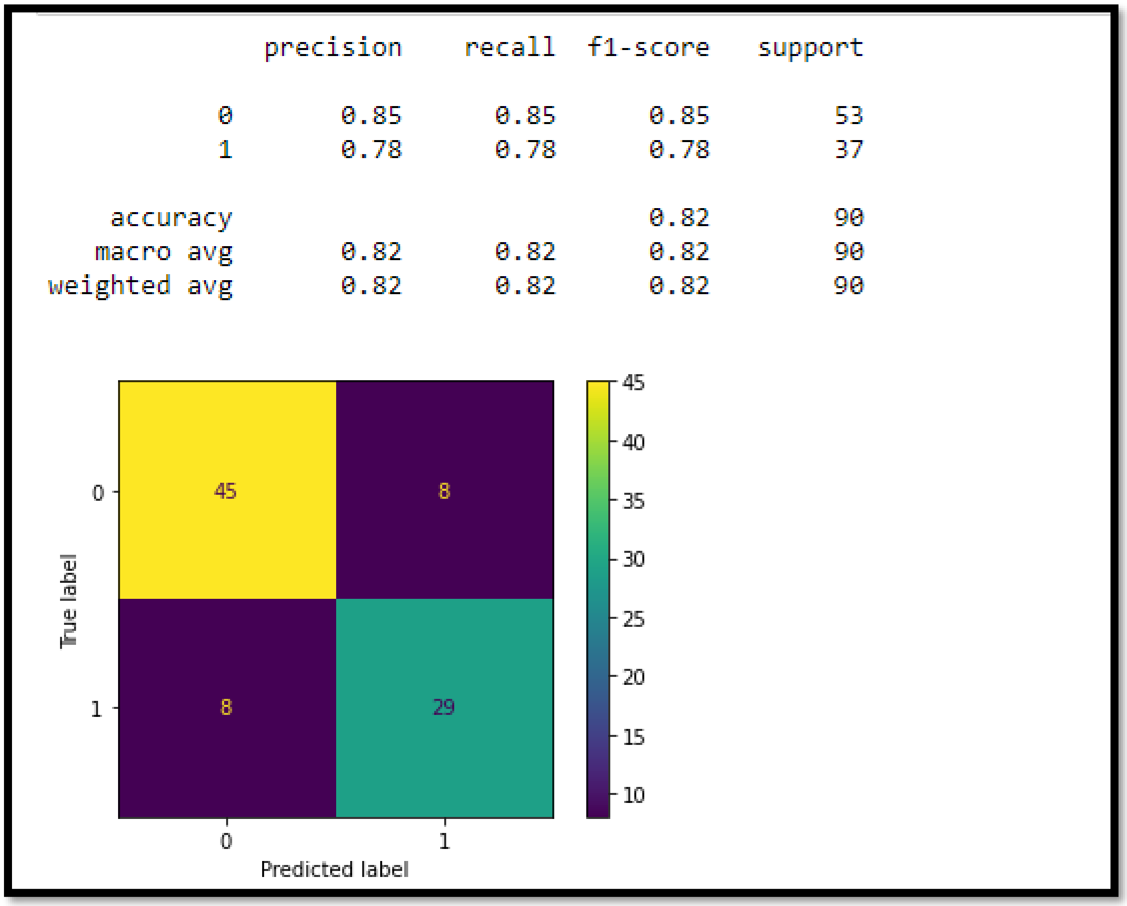
The voting ensemble was also able to beat table 11s results marginally for this metric. **The ensemble scored 0.839** compared to the best result of 0.822, so an improvement of 0.017. Having a high 80% and above ROC AUC means that the model can classify class 0 to 0 and class 1 to 1 accurately and is not due to random chance. The voting ensemble also outperformed all individual estimator model in this regard.



Final Run Confusion Matrix Analysis:

To understand the models ability to predict the classification of 0 and 1 accurately, I ran the model again just 1 time and got out the confusion matrix. From the confusion matrix, we can observe that the model is better at predicting label 0 compared to label 1, and this is due to the class imbalance that we have as there are more 0 labels in the data set compared to label 1. However the performance between the 2 is not bad, the model was able to predict 85% correctly the labels 0, and 78% correctly the labels 1.

Of the test data set that we used, the model classified 29 of 37 patients that have died correctly, and model predicted 45 of 53 patients that have survived correctly. The model misclassified 8 patients that died as survived and 8 patients that survived as died.



```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85        53
           1       0.78      0.78      0.78        37

    accuracy                           0.82        90
   macro avg       0.82      0.82      0.82        90
weighted avg       0.82      0.82      0.82        90
```

Task 2.4: Conclusion

To conclude, overall, the ensemble model has beaten the results of all metrics for the logistic regression models that the researchers have developed to predict patient survival. Even though the improvement in most metrics is improved marginally, the most improvement we have seen is with the MCC scores which is the most important metric to evaluate when we are working with imbalanced classes datasets.

There are slight differences to my experimentation protocol compared to the researcher's protocol, the main difference is around the hyper parameter tuning. The paper does not mention what parameters were tuned for the logistic regression model in table 11, and in table 4, for the models that required tuning, only 1 parameter was tuned. To build the voting ensemble, I had tuned multiple parameters for each model that are being used in the voting ensemble.

I was able to get my results more than 85% for all metrics, but due the small sample size we are working with, the testing results came around 98%-100%, thus indicating the models were overfitting quite a bit, so to make sure the models are generalisable, I have decided to reduce the performance slightly but keep the range between train and test much closer. I believe if we had a larger sample size, the performance of the models could be optimised further.

Even though age came out as a top 3 predictor for the Random Forest model, what I have found that by including this feature, it reduced the performance of the model quite significantly. So, we can conclude that serum creatinine and ejection fraction are important indicators alongside time to predict patients' survival with heart disease.

When analysing the results of the voting ensemble alongside the individual estimators, it is evident that each individual estimator has its own strengths, for example, the logistic regression model that I have used, scored 92.5 with its true negative score and was the only model that scored above 90% and the Random Forest scored 82% with the true positive rate which was higher all models that we are comparing with.

I believe by using the voting classifier and combining multiple models into 1, the solution that I have produced is more robust and accurate with the predictions.

Bibliography

[1] D. C. a. G. Jurman, “Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone,” *BMC Medica Informatics and Decision Making,* 2020.