Vincent Pepe, Tyler O’Connor, and Parth Shah

**Final Report**

1. **Background:**

In just the United States, there are approximately 550,000 new diagnosed cases of heart failure each year. This ever-growing number is accompanied by an even scarier number of yearly deaths: 287,000. As we are in a global pandemic where the virus mostly affects patients with pre-existing conditions, we thought it would be a great idea to look at some of the main factors that lead to heart failure. The dataset, which we found on kaggle, consists of 299 observations and 11 variables. We plan to use 11 of the variables to try and predict the survival rate of those 300 patients.

1. **Data Description:**

As we previously stated, this dataset consists of 299 observations and 11 variables. Because we are trying to predict survival rates, the dependent variable we will be using in the models is DEATH\_EVENT which is a binary variable. The class distribution is moderately imbalanced with 68% of the records labelled as 0. The other ten variables are made up of both binary variables as well as continuous variables. Due to many of the variables containing detailed medical terms, we will add brief descriptions to what each variable measures. The continuous variables are as follows: age, creatinine\_phosphokinase (level of the CPK enzyme in the blood), ejection\_fraction (percentage of blood leaving the heart at each contraction), serum\_creatinine (level contained in blood), serum\_sodium (level contained in blood), and platelets. The remaining four independent variables we are using are boolean variables: diabetes, anaemia, high\_blood\_pressure, and sex.

1. **Exploring The Data:**

Before diving into any complex model, we understand it is beneficial to have as much knowledge on the data as possible. Before starting anything, we ran the skim() function on the dataset which allows the user to get the descriptive statistics of the dataset, which include stats like variable class, number of missing variables, mean, quartiles, etc. This showed us that there were five variables classified as numeric instead of factors, so we changed those classes. Next, we wanted to make sure none of the independent variables had a high correlation with one another. We created a correlation matrix (will be shown at the end of the report) and came to the conclusion that only two independent variables with any observable correlation at all were sex and smoking. Now this doesn’t necessarily assume causation, but it is nice to know before starting any model. When we viewed our data, we observed that many of the patients who smoked were male, which makes sense when you consider that men globally are far more likely to use tobacco than women. Next we ran a basic linear regression to see which covariates were significant when predicting our response variable death\_event. We concluded that four variables: age, ejection\_fraction, time, and serum\_creatinine were the significant covariates when predicting death\_event, so we used those to create the predictive models. We then split our data into training and testing sets, then moved on to creating the models.

1. **Preprocessing**

Before creating the models, we used an R function called recipe. Recipe allows the user to perform any type of preprocessing all in one statement, which makes building any model extremely easy. We created this recipe on the training data, and performed a downsample, created dummy variables for any that needed them, removed all variables with a single value, normalized the data, and then ran a prep function that estimates the required parameters from a training set that can be later applied to other data sets. Next we ran the juice function that downsizes the response variable, giving us an equal amount of factor events (0 and 1 in our case).

1. **Creating the Models**

**5.1 Logistic Regression**

The first model we ran was a logistic regression, using the glm package as our engine. We then created a workflow which aggregates our fitted model and recipe, allowing us to further explore the model and easily create graphs and plots. We will explore the results for each model in the “exploring results” section.

**5.2 Neural Network**

The second model we ran was a neural network, using the keras package as our engine. The neural net had 100 epochs, 5 hidden units, and a dropout of 0.1. The keras package does not mesh well with the tidymodels package that was used to create a workflow, so instead we ran the model as usual.

**5.3 Support Vector Machine**

Since SVM models are sensitive to outliers all features are standardized (mean of zero and unit variance) in the preprocessing step. Four different kernels *linear, polynomial (degree=3), rgb* and *sigmoid* are fitted on the training set. Initial results with default parameters in Python’s *sklearn* class (*Linear: C=1.0; Poly: C=1.0, d=3; Rbf: C=1.0, gamma=0.1; sigmoid: C=1.0)* are fitted on the dataset.

**5.4 PCA + SVC**

Dimensionality reduction on the dataset is performed using principal component analysis. First two principal components, explaining a combined variance of ~58%, are used for performing SVM models. The scatter plot of the principal components by classes is shown in the appendix

**5.5 XGBoost**

Our final model was an XGBoost model, using the xgboost package as our engine. We set the parameters to contain 1000 trees and tune the other five variables to make sure they are proportional to give us the best overall prediction. We then created a latin hypercube that gave us great insight into the optimal tree depth, loss reduction, learn rate, etc. Next we created a workflow to append our model and recipe.

1. **Exploring Results**

Before fitting our trained models onto the test data, we ran cross validation on the dataset, with our stata as our response variable. Each model was then re-fit using these validation splits.

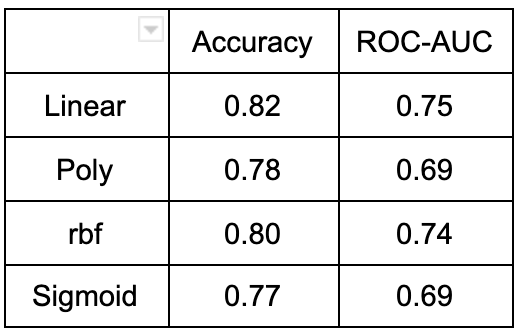
**6.1 Logistic Regression**

After re-running our model with the cross-validated data, we checked the accuracy and roc\_auc of the logistic regression model. The mean accuracy was .817 and the mean roc\_auc was .89. For a dataset with a small number of samples we thought the initial statistics were performed well. After selecting the best roc\_auc, we finalized the workflow using that roc\_auc and fit the model to the testing data. The accuracy and roc\_auc wasn’t as high as we would have liked, accuracy of .784 and an roc\_auc of .858, but overall we think it performed well. Out of the 74 observations, death\_event was correctly predicted 58 times. There will be a plot of variable importance below in the plot section to show the difference between each model.

**6.2 Neural Network**

After fitting the resampled data to the neural network, we learned that the accuracy of the model was 0.81 and the roc\_auc was 0.87. We were astonished to learn that the logistic regression model outperformed the neural network as neural networks typically outshine most models. This anomaly can be attributed to the small number of observations in the dataset, as neural networks typically don't perform as well on smaller datasets.

**6.3 Support Vector Machine**

Initial results with default parameters in Python’s *sklearn* class gave the following results on the test set where *C* indicates regularization.

* Linear: C = 1.0
* Poly: C = 1.0, degree = 3
* Rbf: C = 1.0, gamma = 0.1
* Sigmoid: C = 1.0

For finding the right set of hyperparameters gridsearch was implemented using *GridSearchCV* class with a range of parameters for *regularization* *(0.01, 0.05, 0.50, 1, 10), kernels (linear, poly, rbf, sigmoid)* and *polynomial degree (3, 5, 10).* A 5-fold cross validation totalling 300 fits is performed. The best model was a *linear* kernel with a 0.5 *regularization* giving an accuracy of 0.82 and roc-auc score of 0.76.

**6.4 PCA + SVC**

**PCA with SVM Linear Kernel:** Increasing the regularization parameter *C* from 0.5 to 1000 it is observed that the model starts performing better from *C=10* with no improvement in score when *C* is increased to 1000. [Appendix: PCA + SVC Linear]

**PCA with SVM Poly Kernel:** Increasing the polynomial degree *d* from 3 to 10, the model is observed to start overfitting with higher degrees. Increasing the regularization parameter *C* didn’t contribute to the increase or decrease in the performance of the model. [Appendix: PCA + SVC poly]

**PCA with SVM rbf Kernel:** The increase in *gamma (01 -> 5)* leading to instances having a smaller range of influence is observed to heavily overfitting the dataset. The increase in regularization parameter *C* with *gamma 0.1* leads to a good performing model with an roc-auc train score of 0.78 and test score of 0.81. [Appendix: PCA + SVC rbf]

**6.5 XGBoost**

For the XGBoost model, we first tuned our latin hypercube and fitted the resampled validation splits to the model. We next selected the best roc\_auc and finalized our xgboost workflow using that roc\_auc. The variable importance graph was very different when compared to the logistic regression model, with time by far being the most significant predictor. The final prediction gave us an accuracy of 0.81 and an roc\_auc of .938, outperforming all of the models in the roc\_auc category. Out of the 74 observations, death event was correctly predicted 60 times, which is only two more than logistic regression, but with a larger dataset we think that margin would be much larger.

1. **Conclusion**

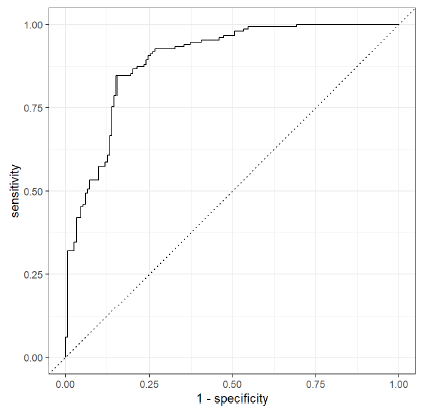
Overall, the XGBoost model performed the best with a final accuracy of 0.81 and an roc\_auc of .938. We definitely ran into some problems throughout the project, but overall we think all the models performed very well on such a small dataset. If we were to revisit this in the future, we would look for a larger dataset to test these models on to get a more accurate reading.

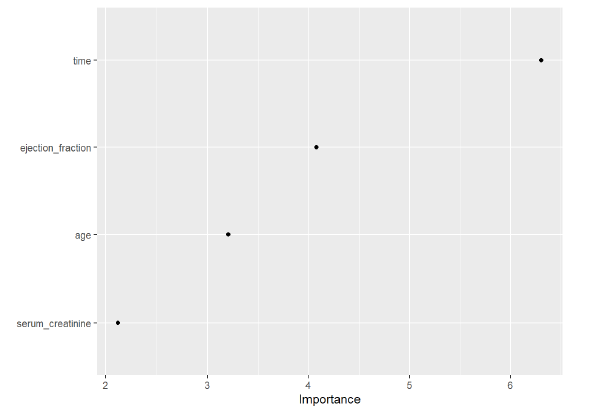
1. **Plots**

**8.1 Logistic Regression**

Initial accuracy, roc\_auc

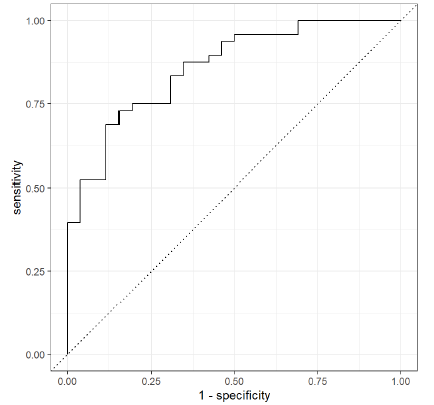


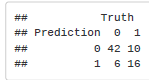
Initial sensitivity vs. specificity plot

Variable importance

Final accuracy, roc\_auc

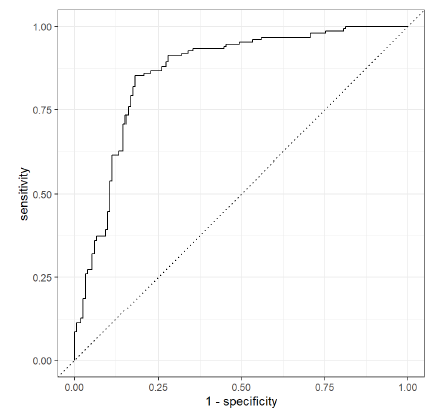
Final confusion matrix and sensitivity vs. specificity



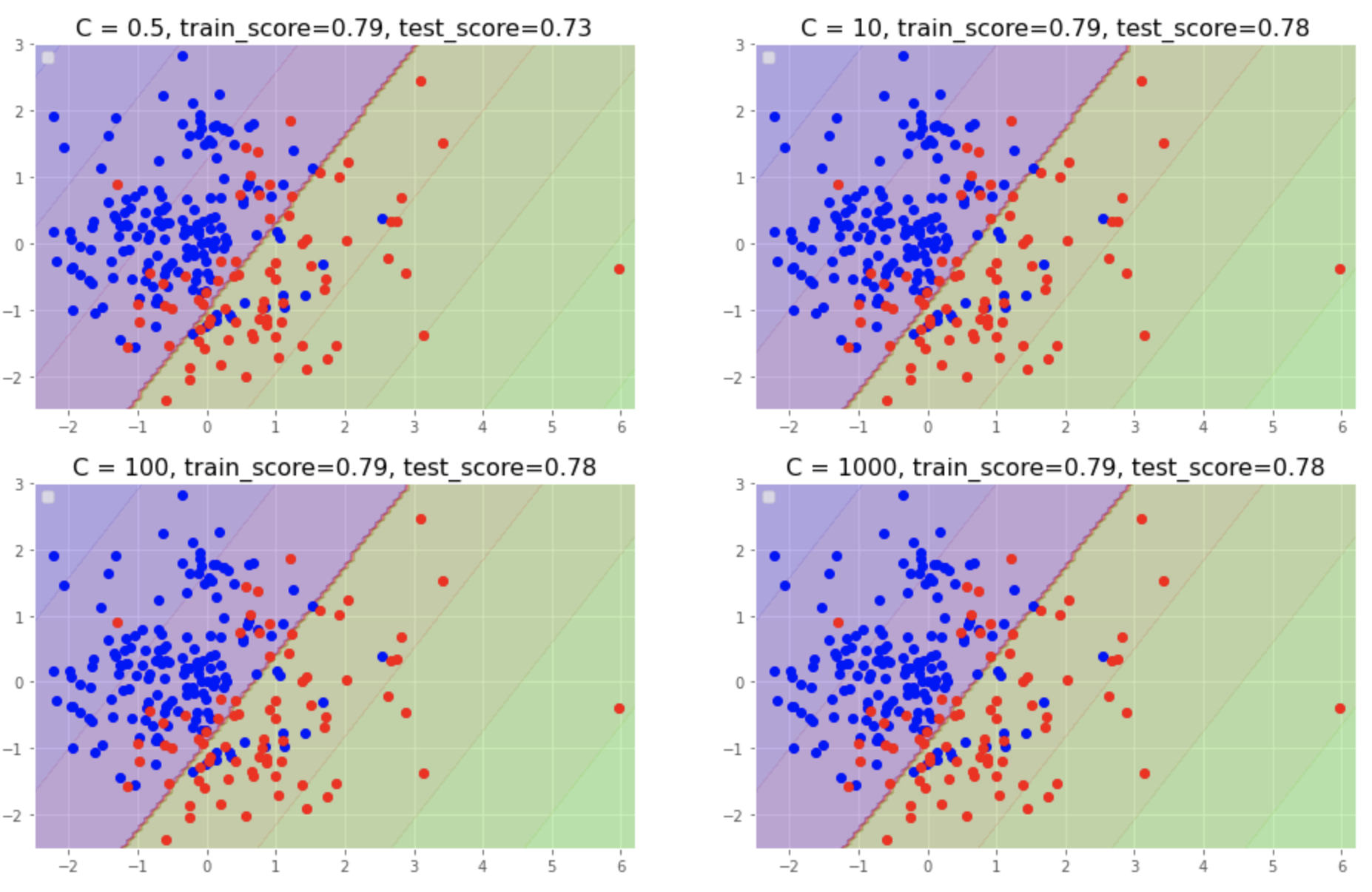


**8.2 Neural Network**

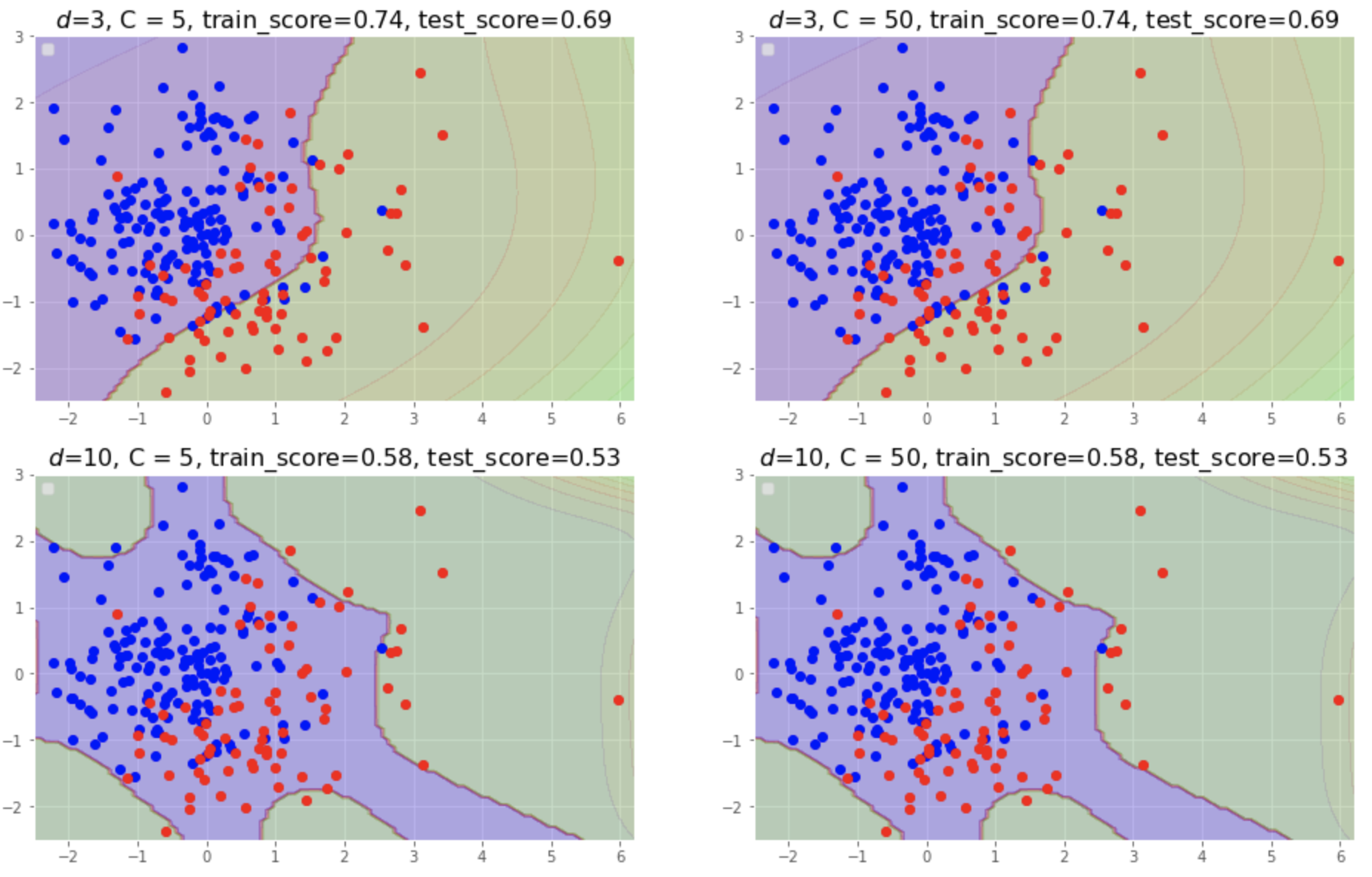
Accuracy and roc\_auc

Sensitivity vs. specificity

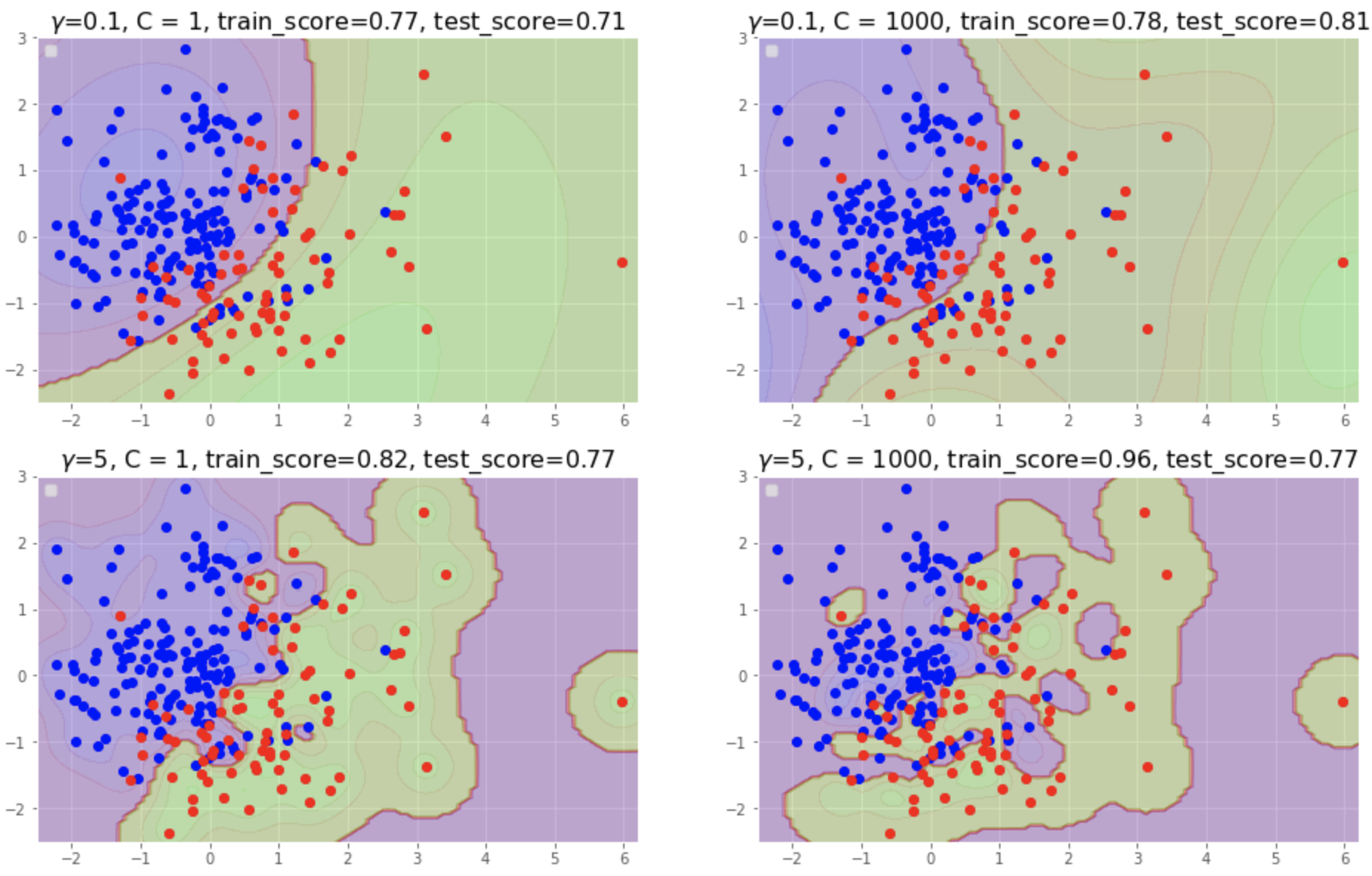
**8.4.1 PCA + SVC Linear**

****

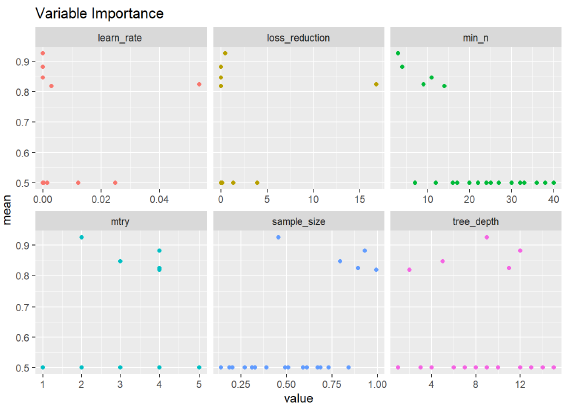
**8.4.2 PCA + SVC Poly**

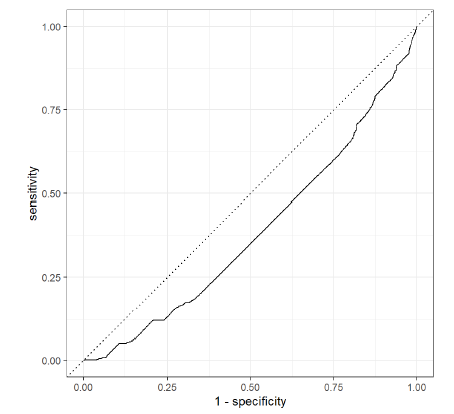
****

**8.4.3 PCA + SVC rbf**

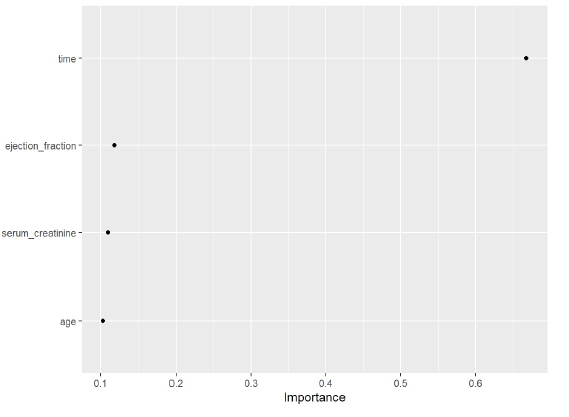
****

**8.5XGBoost**

Variable Importance

Initial sensitivity vs specificity

Variable Importance



Final accuracy and roc\_auc



Confusion matrix and Final sensitivity vs specificity