Criminal Recidivism Prediction

Alanna Manfredini

December 6, 2023

1 Background

1.1 Introduction

Compared to other democracies around the world, America has one of the highest incarceration rates. In fact, the US prison population has grown 500% within the last 30 years [1]. Naturally, this massive increase in incarceration has put a strain on the American Justice system, and, in North Carolina, has led to average disposition times for non traffic-misdemeanours and felonies of 172 and 259 days respectively [2].

Due to this, it would be highly advantageous for the criminal justice system to only keep those who are likely to commit a future crime, recidivate, incarcerated. This would reduce the number of people in jail, and also reduce the number of preventable recidivism cases from returning back into society, committing a crime and having to be reprocessed by the Criminal Justice System. With the increase of the accuracy of predictions by machine learning and AI models, it is important to test whether this process could be streamlined with technology by using individuals' criminal histories to determine whether said person is likely to commit another crime, and, if not, set them up for early release. This streamlining is crucial as it could greatly reduce the strain on the Criminal Justice System, thus improving the quality of the remaining trials; can free those who are wrongly incarcerated; and can prevent future crime from happening. The possibility of early release could also have the knock on effect of increasing good behaviour during prison sentences. The situations analysed in this paper are 'general two year','general six month', 'drug two year', 'property two year','misdemeanor two year', 'felony two year', 'violent two year', 'drug six month', 'property six month', 'misdemeanor six month', 'felony six month', 'violent six month', 'any recid'

1.2 Data Description

The "Broward_data.csv" data for this project was found open source at https://github.com/BeanHam/2019-interpretable-machine-learning/tree/master/broward/data. The data was previously processed by machine learning researchers to only include the data with which the Arnold PSA can be calculated. Additionally, the data was processed to create 6 month and 2 year recidivism outcome columns, exclude minor offenses and exclude any non-realistic data such as people being over the age of 150. More detailed explanations of how the data was processed prior to being uploaded in the Github source is available in section 11.2 of the original paper [3].

To visualise the data, the data was divided into male and female and each of the non-binary features were plotted against each other, Figure 1 and 2. This provided insight into whether certain features were expected to be more highly correlated than other features. Additionally, it was easy to identify outliers in the data, for example, it is possible to tell that in the male data, one individual was documented as having his "first charge" at age 0. This is almost certainly impossible, but since it was a single outlier, the datapoint was left in as it seemed likely that having another datapoint was more likely to improve the fit than the single outlier would disturb the fit. Another potential mistake that could be easily identified by the pairwise plotting was that one individual had 218 charges. This is much higher than the second highest which sits at approximately 80. A final noteable outlier was the male who had 60 counts of voyeurism, 10x higher than the second highest count. Within the female dataset, two women had over 50 total convictions, which is considerably higher than the average of around 20. These datapoints were left in, because although they were much higher, the number of charges was still within the realistic realm.

During the experiments, the data was randomly shuffled and then divided into an 80-20 training test split. The random shuffling was to ensure that there were no inherent biases in the data that would become evident when split linearly. The test data was not used at any point throughout the experiment except initial plotting of datapoints and final accuracy scoring.

The full pairwise plots are included in Figure 1 and 2. In these plots every non-binary feature is plotted from top to bottom. For the female plots the features are 'age at current charge', 'age at first charge', 'p charges', 'p probation', 'p felprop viol', 'p felassault', 'p misdeassault', 'p weapon', 'p fta two year', 'p fta two year plus', 'p pending charge', 'p felony', 'p misdemeanor', 'p violence', 'total convictions', 'p arrest', 'p property', 'p traffic', 'p drug', 'p dui', 'p domestic', 'p stealing', 'p trespass'.

For the male plots the features are 'age at current charge', 'age at first charge', 'p charges', 'p probation', 'p juv fel count', 'p felprop viol', 'p murder', 'p felassault', 'p misdeassault', 'p sex offense', 'p weapon', 'p fta two year', 'p fta two year plus', 'p pending charge', 'p felony', 'p misdemeanor', 'p violence', 'total convictions', 'p arrest', 'p property', 'p traffic', 'p drug', 'p dui', 'p domestic', 'p stalking', 'p voyeurism', 'p fraud', 'p stealing', 'p trespass'.

It is interesting to note that the female dataset had no counts of voyeurism greater than one. A small snapshot is included of the male plot for ease of viewing.

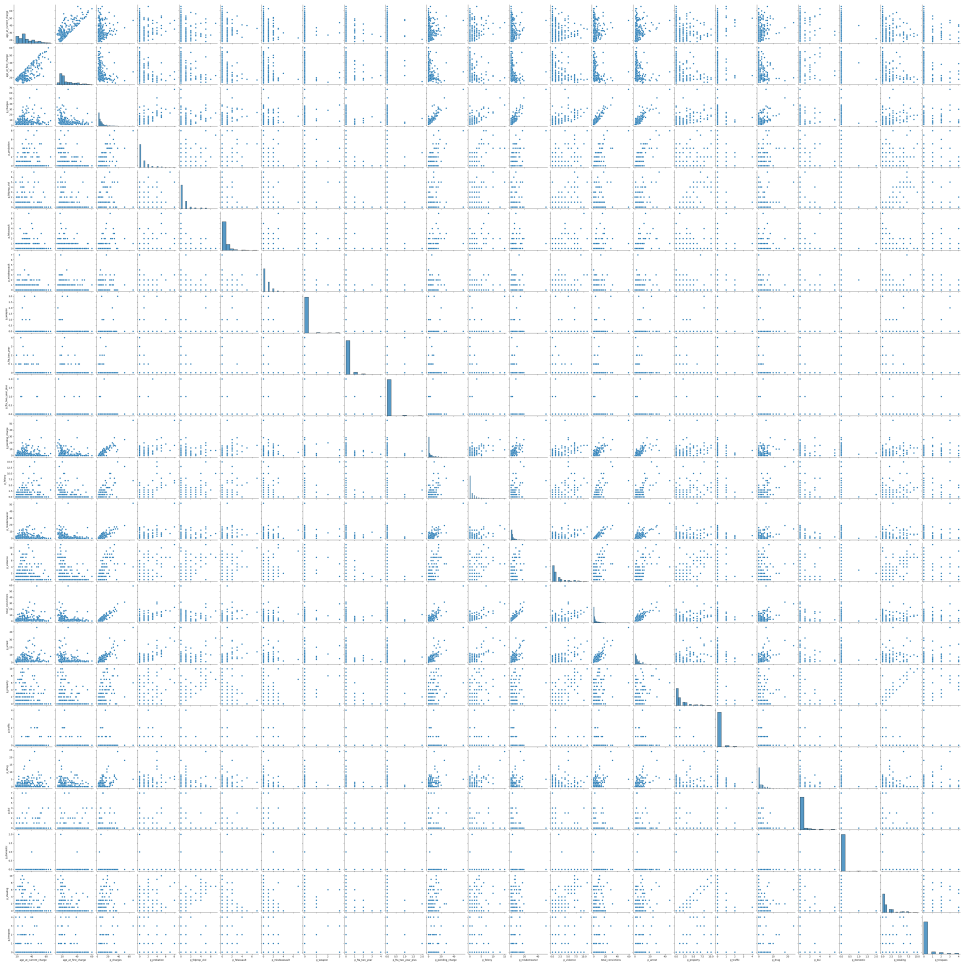

Figure 1: Female Full Pairwise Feature Plot

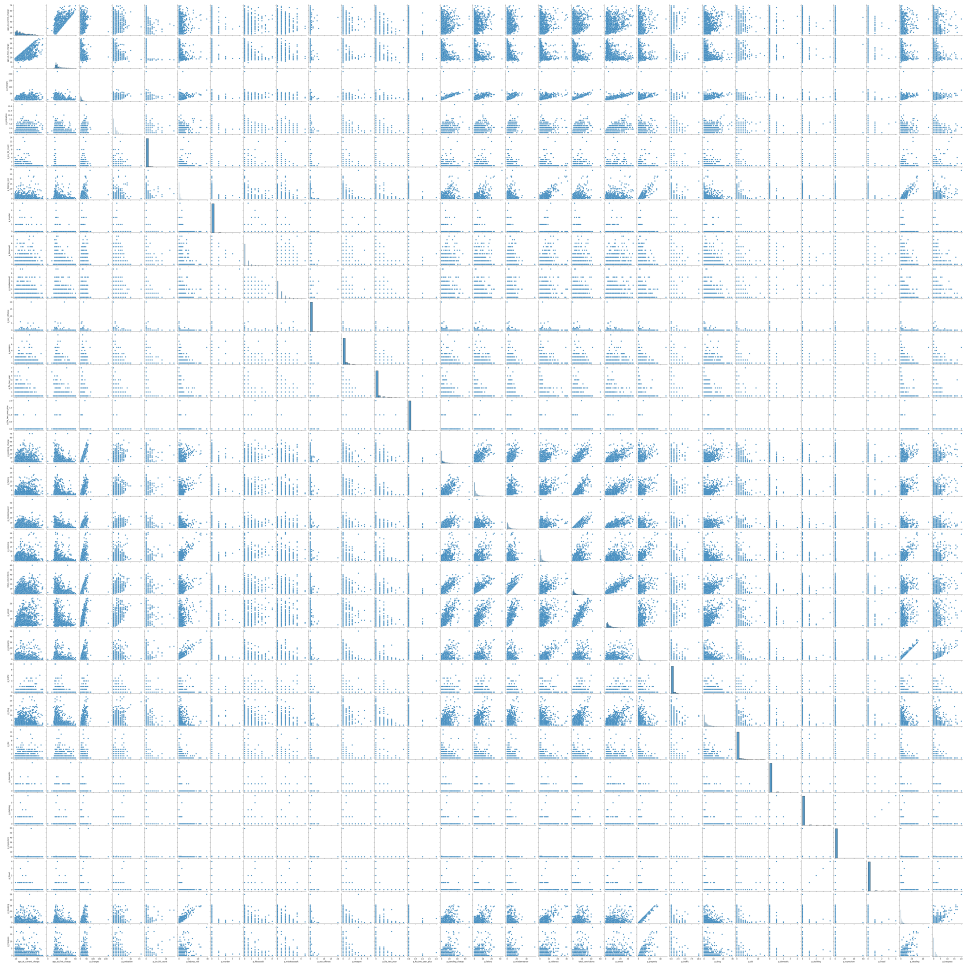

Figure 2: Male Full Pairwise Feature Plot

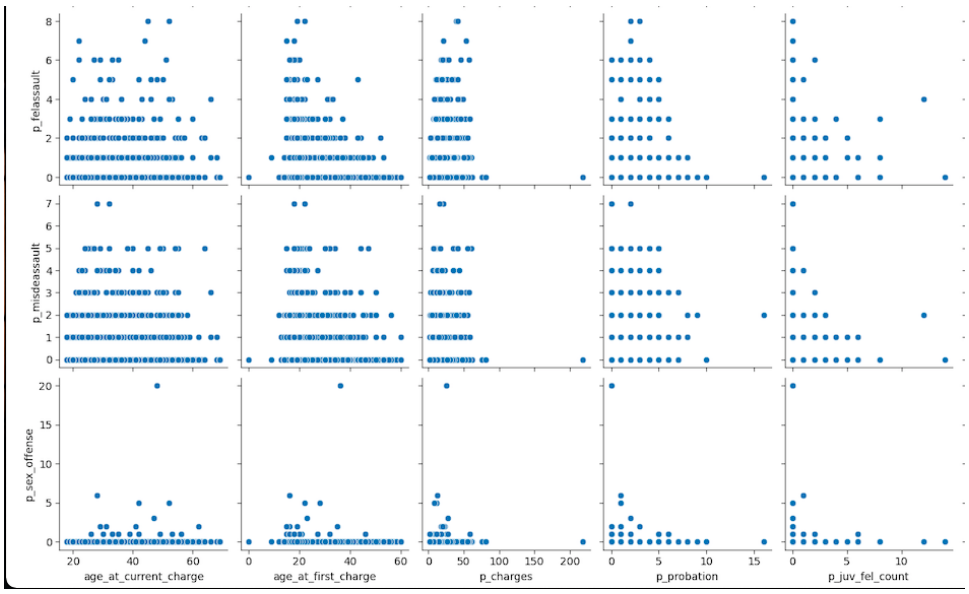

Figure 3: Male Partial Pairwise Feature Plot

1.3 Previous Work

The work of this project is based off "In Pursuit of Interpretable, Fair and Accurate Machine Learning for Criminal Recidivism Prediction Pursuit " [3]. This paper aims to build upon the results of the cited paper; it uses the same dataset, but intends on determining whether there are other algorithms which can achieve better outcomes for prediction. Both papers aim to use interpretable machine learning algorithms which return results that are not biased towards certain demographics. The algorithms chosen in this paper aim to determine whether less complex, more linear models can just as effectively predict criminal recidivism. Unlike the cited paper, which explores using decision trees and random forests, methods that split the data by a hierarchy of features, the models in this paper use algorithms which address all features at the same time. This could mean that any closely correlated features enhance the fit rather than are masked as are in a decision tree. Additionally, the models in this paper are primarily constructed to work with continuous variables, as is in the Broward Dataset.

2 Methods

Four algorithms were chosen to fit to the data. Since there were relatively few datapoints, 251 for women and 1703 for men, the algorithms that were chosen because they were efficient, handled outliers well or did not make assumptions about the data. Additionally, most of the algorithms chosen were ideal for predicting binary outcomes rather than continuous ones. All algorithms were tuned using GridSearchCV [4]. To increase the comparability of the models, all models were evaluated based off the accuracy of their predictions.

The first algorithm chosen was a Support Vector Machine [4]. Support Vector Machines split classes in the data using a hyperplane; the data can be transformed to higher dimensions using a kernel. This effectively means the data can be split with a shape that is not a hyperplane. This type of algorithm is good at efficiently finding the split between two classes in high dimensions. After looking at the data, it was determined that it was unlikely the outliers would be the support vectors and thus this algorithm was likely to be uneffected by said outliers. Additionally, since it is a simple model, it is easy to interpret the results and determine the influence of the various features. This is especially important in Criminal Recidivism where the outcome of the algorithm can have a life-changing positive or negative influence on both the convicted and potential future victims. This model was implemented by optimising with respect to the kernel, C and gamma hyperparameter. The algorithm was trained with the linear and rbf kernel. These were chosen because they are the most rigid and the most flexible kernel respectively. This means that the optimal choice of kernel for each situation provides more insight into the structure of the data and thus can be more easily compared to future algorithms. The C and gamma values, 0.1, 1, 10, 100 for both, were chosen because they span the expected range for C and gamma.

The second algorithm implemented was an Adaboost additive model algorithm [4]. Adaboost combines many weak learners, in this case SVMs, into a strong learner. This makes it robust to noise and can create a very complex algorithm based off simple algorithms. The base estimator was chosen to be an SVM so it would be easy to compare with the first algorithm. Theoretically, the use of many weak learners should produce a better outcome than a single SVM. A tradeoff between the two algorithms is that the computational efficiency of the Adaboost model is considerably lower than the single SVM. Whilst this is the case, in situations such as Criminal Recidivism computational inefficiency is a small trade-off for an improvement in the outcome. The Adaboost hyperparameters that were tuned were the number of estimators, 10, 50, 100; and learning rate, 0.001, 0.01, 0.1, 1.0. These were chosen because they span the realistic values for each parameter. Additionally, the C value of the base estimator was tuned in an external loop. This was done by implementing the Adaboost algorithm for C = 0.1, 1 and 10, taking the accuracy of each and choosing the Adaboost model with the best C for each recidivism situation.

The third algorithm was a K-nearest neighbours (KNN) algorithm [4]. KNN determines the classification of the testing data by looking at the closest "k" points and determining which is the more common class within this subset. This algorithm was chosen because it is also robust to outliers and does not make any assumptions about the underlying distribution. Since KNN works to merely classify based on the closest surrounding datapoints, if there are some features which have a poor delineation between the classes, the KNN algorithm should still have success classifying the test data.

The number of neighbours, the weight and p were the hyperparameters chosen to train. The span for the number of neighbours had to be quite large, because the female and male models were determined using the same function; since the size of the respective datasets were vastly different, the number of neighbours had to be appropriate to both sizes. The p values were chosen to see whether different distance metrics would have different effects on the outcome of the algorithm. This can be compared to the hyperparameters in different models, further elucidating whether the classes are linearly spread out amongst the features or not. Finally the weights were tuned to highlight the spread of the data and determine whether the classes change quickly without a clear delineation, thus making distance important or whether the classes are well separated, thus meaning that any of the close N neighbours are relevant.

The final algorithm was a Ridge Regression model [4]. This choice was less theoretically ideal than the previous choices, especially since it is mostly used as a regressor, not a classifier, but was made because Ridge Regression is good at handling multicollinearity. After plotting the initial data it was clear that a lot of the features were very highly correlated, for example, the number of arrests, charges and convictions is obviously going to have a strong relationship. Secondly, ridge regression was chosen because it is good at handling noise and small datasets. The final hyperparameters tuned were the alpha and the solver for the Ridge Regression model. These hyperparameters were chosen after some experimentation. Initially only numbers surrounding the common alpha value of 1 were chosen. However, after implementation, all of the alphas chosen were highest alpha provided to the GridSearch algorithm. This clearly indicated that the alpha should be higher for the model and thus the possible values was expanded to include higher numbers. This was effective, because after implementation there was a spread of alpha values that created the most accurate model. The solvers which most closely matched previous modes or which were more stable were chosen to iterate on.

3 Experiments

3.1 Results

	General	Drug	Property	misdemeanour	Felony	Violent
Female Train	0.756	0.930	0.920	0.766	0.886	0.861
Female Test	0.78	0.92	0.92	0.86	0.84	0.86
Male Train	0.629	0.902	0.961	0.949	0.959	0.957
Male Test	0.594	0.909	0.962	0.953	0.965	0.953

Table 1: SVM Accuracy - 2 Year (Rounded for Display Purposes)

	General	Drug	Property	misdemeanour	Felony	Violent
Female Train	0.836	0.965	0.955	0.896	0.945	0.940
Female Test	0.88	0.96	0.96	0.96	0.92	0.96
Male Train	0.954	0.961	0.987	0.977	0.910	0.910
Male Test	0.953	0.953	0.988	0.974	0.891	0.915

Table 2: SVM Accuracy - 6 Month (Rounded for Display Purposes)

	General	Drug	Property	misdemeanour	Felony	Violent
Female Train	0.662	0.930	0.920	0.751	0.886	0.861
Female Test	0.7	0.920	0.920	0.86	0.84	0.86
Male Train	0.644	0.903	0.914	0.715	0.825	0.787
Male Test	0.597	0.909	0.903	0.756	0.803	0.768

Table 3: SVM Adaboost Accuracy - 2 Years (Rounded for Display Purposes)

	General	Drug	Property	misdemeanour	Felony	Violent
Female Train	0.841	0.965	0.955	0.896	0.945	0.940
Female Test	0.88	0.96	0.96	0.96	0.92	0.96
Male Train	0.776	0.961	0.949	0.869	0.910	0.910
Male Test	0.774	0.953	0.947	0.885	0.891	0.915

Table 4: SVM Adaboost Accuracy - 6 Month (Rounded for Display Purposes)

	General	Drug	Property	misdemeanour	Felony	Violent
Female Train	1.0	0.930	0.920	0.746	0.891	0.886
Female Test	0.68	0.94	0.92	0.86	0.84	0.86
Male Train	0.627	0.902	0.914	1.0	1.0	1.0
Male Test	0.582	0.909	0.903	0.765	0.803	0.782

Table 5: KNN Accuracy - 2 Year (Rounded for Display Purposes)

	General	Drug	Property	misdemeanour	Felony	Violent
Female Train	1.0	0.965	0.960	0.900	0.950	0.945
Female Test	0.88	0.96	0.96	0.96	0.92	0.96
Male Train	0.999	0.961	0.953	0.869	1.0	0.914
Male Test	0.776	0.953	0.95	0.886	0.894	0.915

Table 6: KNN Accuracy - 6 Month (Rounded for Display Purposes)

	General	Drug	Property	misdemeanour	Felony	Violent
Female Train	0.692	0.935	0.920	0.766	0.886	0.861
Female Test	0.68	0.92	0.92	0.84	0.84	0.86
Male Train	0.642	0.902	0.913	0.713	0.823	0.784
Male Test	0.571	0.909	0.9	0.765	0.788	0.765

Table 7: Ridge Regression Accuracy - 2 Year (Rounded for Display Purposes)

	General	Drug	Property	misdemeanour	Felony	Violent
Female Train	0.841	0.965	0.955	0.896	0.945	0.940
Female Test	0.88	0.96	0.96	0.96	0.92	0.96
Male Train	0.776	0.961	0.949	0.869	0.911	0.910
Male Test	0.776	0.953	0.957	0.885	0.891	0.915

Table 8: Ridge Regression Accuracy - 6 Month (Rounded for Display Purposes)

3.2 Hyperparameter Selection

The best hyperparamters were chosen using GridsearchCV [4]. The cross validation was executed with five random folds. To avoid overfitting the data, the hyperparameter tuning was done with cross validation on hyperparameters relevant to loss and regularisation.

Recidivism	**kernel**	**C**	**gamma**
Female Parameters			
General 2yr	linear	10	-
General 6mo	linear	0.1	-
Drug 2yr	rbf	0.1	0.1
Property 2yr	linear	0.1	-
Misdem 2 yr	linear	0.1	-
Felony 2yr	rbf	0.1	0.1
Violent 2yr	rbf	0.1	0.1
Drug 6mo	rbf	0.1	0.1
Property 6mo	linear	0.1	-
Misdem 6mo	linear	0.1	-
Felony 6mo	linear	0.1	-
Violent 6mo	rbf	0.1	0.1
Male Parameters			
General 2yr	linear	0.1	-
General 6mo	rbf	1	1
Drug 2yr	linear	0.1	-
Property 2yr	rbf	1	0.1
Misdem 2 yr	rbf	1	1
Felony 2yr	rbf	1	1
Violent 2yr	rbf	1	1
Drug 6mo	linear	0.1	-
Property 6mo	rbf	1	1
Misdem 6mo	rbf	1	1
Felony 6mo	linear	0.1	-
Violent 6mo	linear	0.1	-

Table 9: Support Vector Machine - Hyperparameter Selection

Recidivism	**C**	**Number Estimators**	**Learning Rate**
Female Parameters			
General 2yr	0.1	10	0.01
General 6mo	1	10	0.1
Drug 2yr	0.1	10	0.001
Property 2yr	0.1	10	0.001
Misdem 2 yr	10	100	0.01
Felony 2yr	0.1	10	0.001
Violent 2yr	0.1	10	0.001
Drug 6mo	0.1	10	0.001
Property 6mo	0.1	10	0.001
Misdem 6mo	1	10	0.001
Felony 6mo	0.1	10	0.001
Violent 6mo	0.1	10	0.001
Male Parameters			
General 2yr	10	50	0.01
General 6mo	0.1	50	0.01
Drug 2yr	0.1	10	0.001
Property 2yr	0.1	100	0.01
Misdem 2 yr	1	50	0.001
Felony 2yr	10	100	0.01
Violent 2yr	10	100	0.01
Drug 6mo	0.1	10	0.001
Property 6mo	0.1	10	0.001
Misdem 6mo	0.1	10	0.001
Felony 6mo	1	100	0.1
Violent 6mo	0.1	10	0.001

Table 10: SVM Adaboost - Hyperparameter Selection

Recidivism	**Number Neighbours**	**p**	**Weights**
Female Parameters			
General 2yr	25	3	Distance
General 6mo	10	10	Distance
Drug 2yr	2	0.5	Uniform
Property 2yr	10	3	Uniform
Misdem 2 yr	10	1	Uniform
Felony 2yr	5	100	Uniform
Violent 2yr	2	1	Uniform
Drug 6mo	2	0.5	Uniform
Property 6mo	2	2	Uniform
Misdem 6mo	2	2	Uniform
Felony 6mo	5	2	Uniform
Violent 6mo	2	0.5	Uniform
Male Parameters			
General 2yr	50	1	Uniform
General 6mo	25	5	Distance
Drug 2yr	10	5	Uniform
Property 2yr	10	0.5	Uniform
Misdem 2 yr	50	10	Distance
Felony 2yr	25	3	Distance
Violent 2yr	25	1	Distance
Drug 6mo	10	0.5	Uniform
Property 6mo	5	5	Uniform
Misdem 6mo	10	0.5	Uniform
Felony 6mo	10	1	Distance
Violent 6mo	10	1	Uniform

Table 11: KNN - Hyperparameter Selection

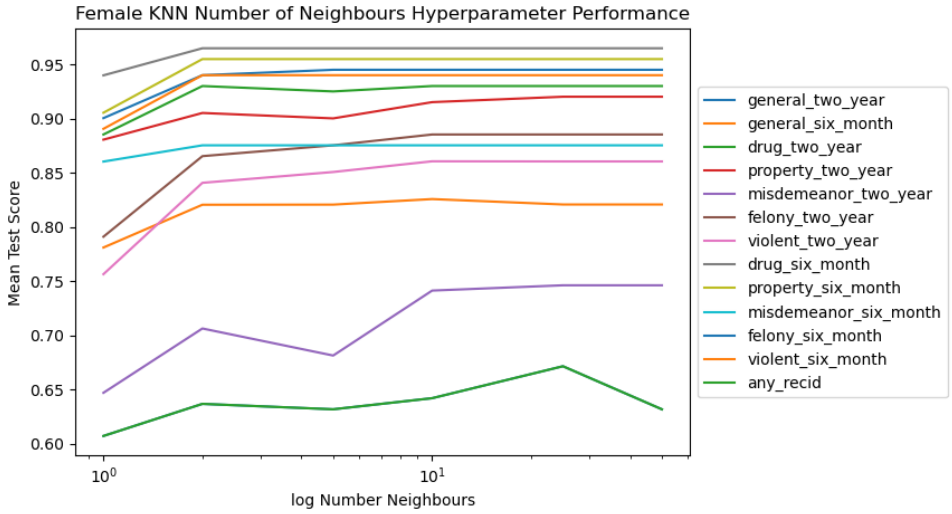

Figure 4: Hyperparameter variation

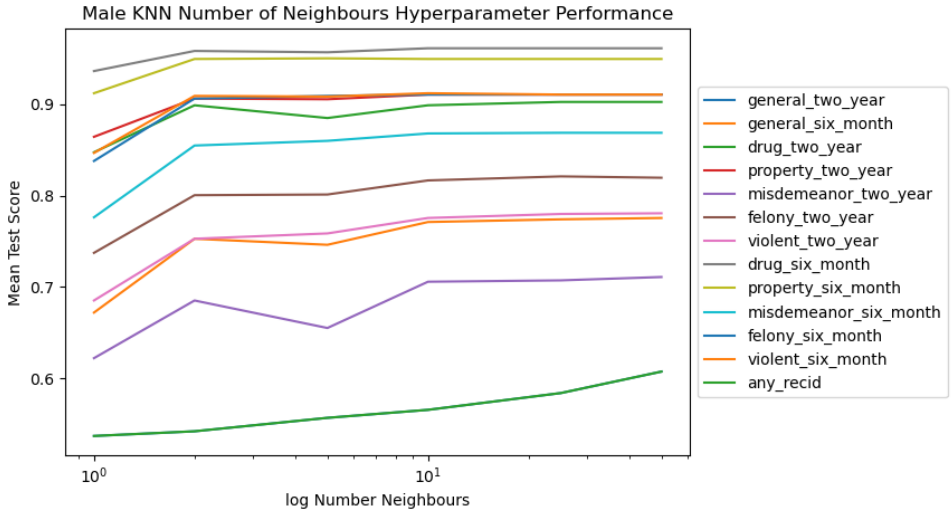

Figure 5: Hyperparameter variation

It is interesting to observe that the hyperparameters do not have much impact except for on the models which already have a very low Mean Test Score. At the lower values of number of neighbours, Figure 4, 5, has a strong impact even on the higher scoring models, but not as much as on the lower scoring models. This makes sense because the number of neighbours is very important to how well the fit is, especially if the dataset has a lot of outliers, like this one does.

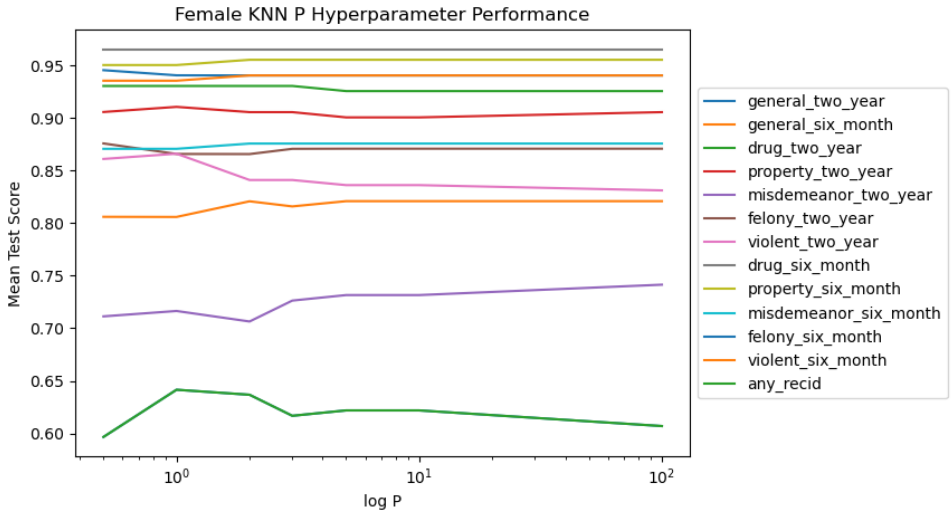

Figure 6: Hyperparameter variation

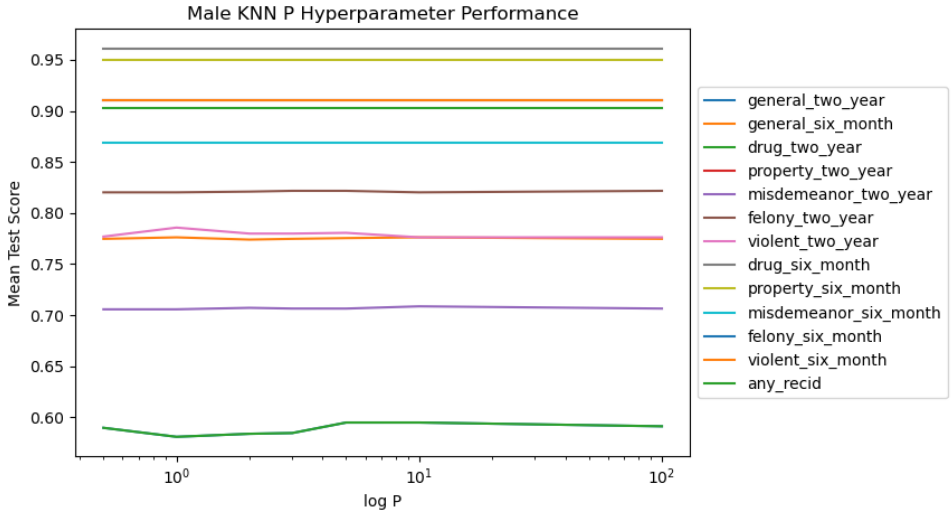

Figure 7: Hyperparameter variation

Recidivism	**Alpha**	**Solver**
Female Parameters		
General 2yr	1000	LSQR
General 6mo	1000	LSQR
Drug 2yr	1000	LSQR
Property 2yr	1000	SVD
Misdem 2 yr	1000	Cholesky
Felony 2yr	10000	LSQR
Violent 2yr	10000	LSQR
Drug 6mo	10000	LSQR
Property 6mo	1000	Cholesky
Misdem 6mo	1000	LSQR
Felony 6mo	1E6	SVD
Violent 6mo	1E6	LSQR
Male Parameters		
General 2yr	1000	LSQR
General 6mo	10000	Cholesky
Drug 2yr	1000	SVD
Property 2yr	10000	LSQR
Misdem 2 yr	1000	SVD
Felony 2yr	10000	LSQR
Violent 2yr	10000	Cholesky
Drug 6mo	1000	SVD
Property 6mo	10000	Cholesky
Misdem 6mo	10000	Cholesky
Felony 6mo	10000	Cholesky
Violent 6mo	10000	SVD

Table 12: Ridge Regression - Hyperparameter Selection

3.3 Visualisation

The figures included in this section are the ROC curves for the female and male test models. The train ROC curves are included in the Appendix.

The ROC curves do not show particularly good performance. This is especially true for the Male curves. Some models perform much better than others, and some even appear to be inverted below random. This is not a problem, because the model can merely be flipped. The female SVM ROC has all the data either along the random line or a single line above it. This is interesting because it highlights that the different outcomes can be predicted in a very similar way.

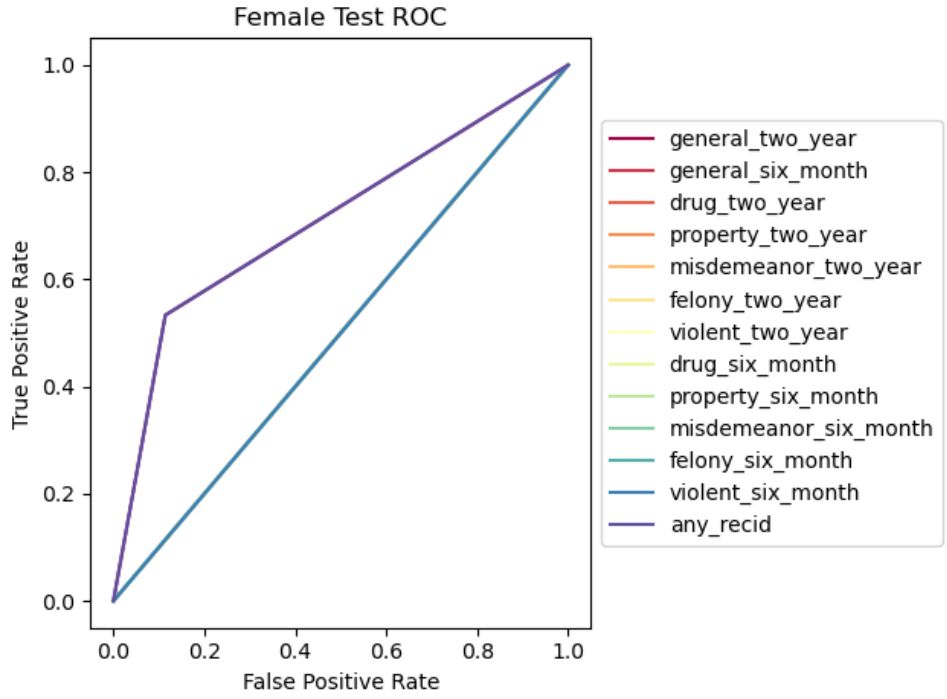

Figure 8: Test ROCs

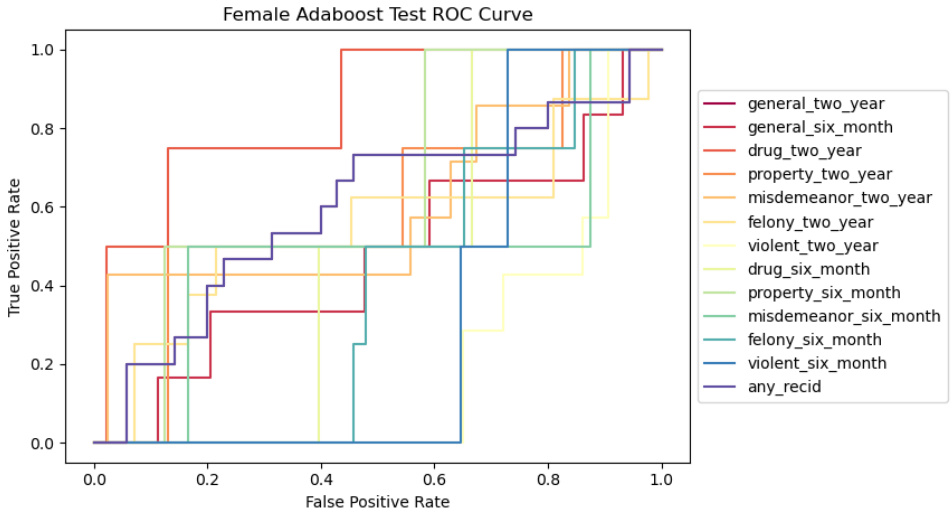

Figure 9: Test ROCs

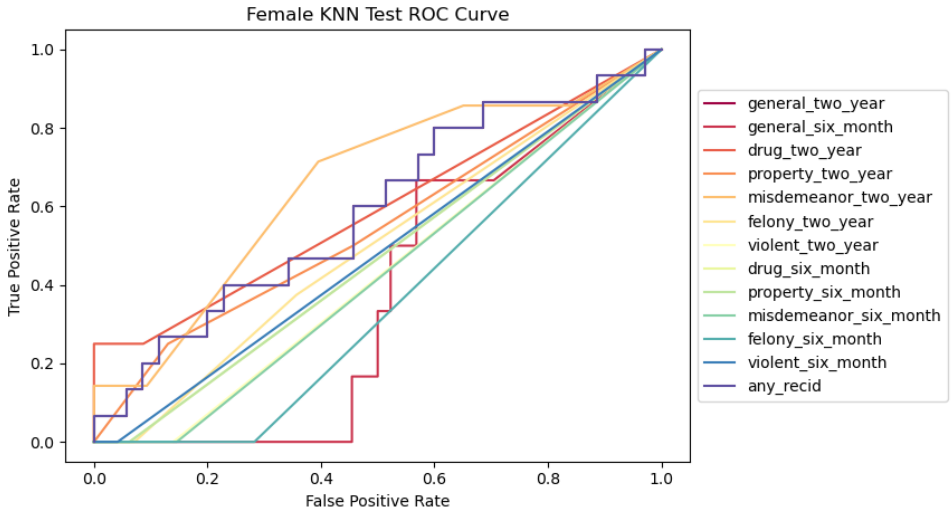

Figure 10: Test ROCs

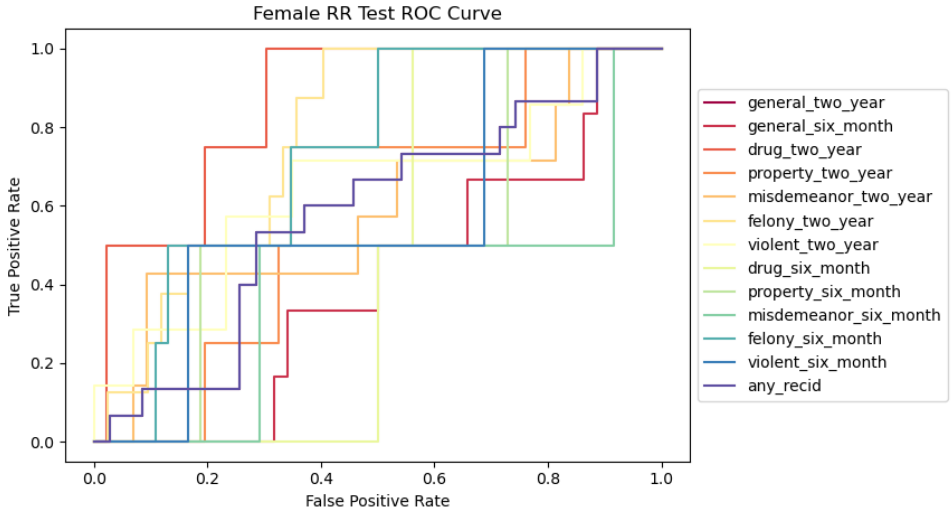

Figure 11: Test ROCs

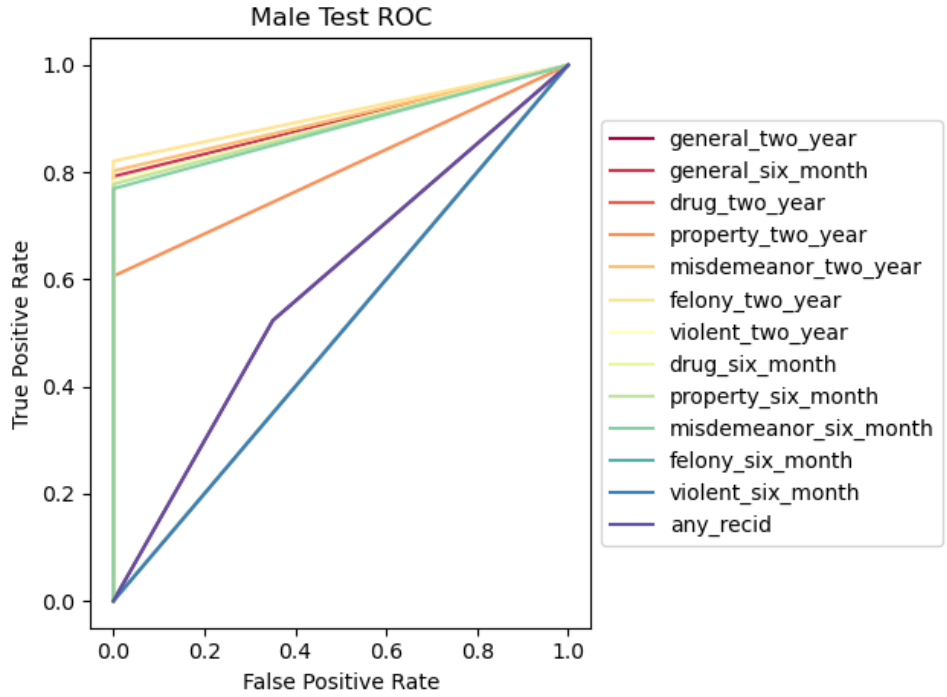

Figure 12: Test ROCs

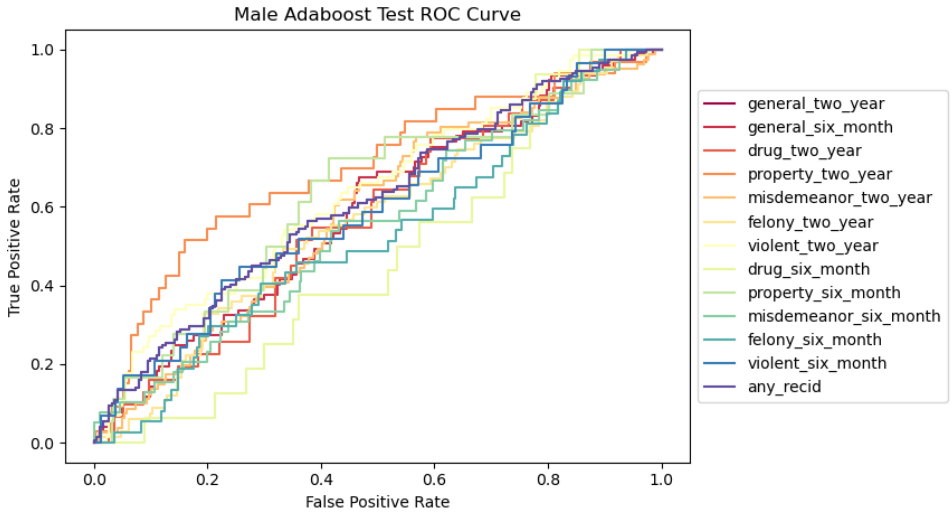

Figure 13: Test ROCs

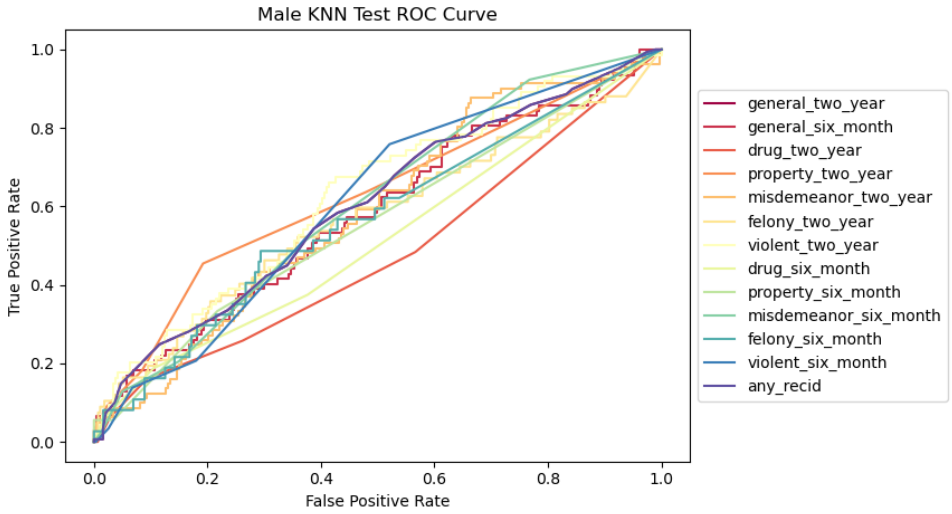

Figure 14: Test ROCs

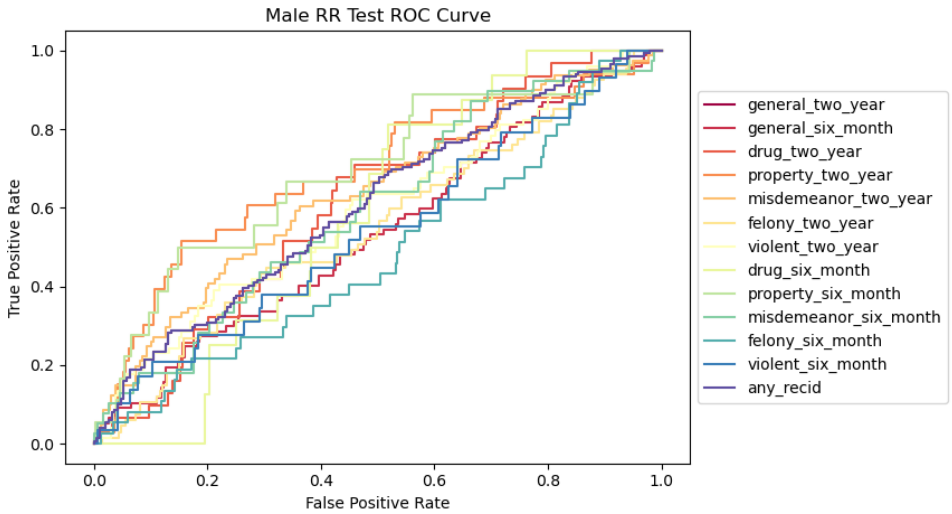

Figure 15: Test ROCs

3.4 Variable Importance Analysis

The variable importance for the Adaboost and Ridge Regression models are plotted below. The x axis of the plots is each feature in order of 'age at current charge', 'age at first charge', 'p charges', 'p incarceration', 'p probation', 'p juv fel count', 'p felprop viol', 'p murder', 'p felassault', 'p misdeassault', 'p famviol', 'p sex offense', 'p weapon', 'p fta two year', 'p fta two year plus', 'current violence', 'current violence', 'p pending charge', 'p felony', 'p misdemeanor', 'p violence', 'total convictions', 'p arrest', 'p property', 'p traffic', 'p drug', 'p dui', 'p domestic', 'p stalking', 'p voyeurism', 'p fraud', 'p stealing', 'p trespass', 'six month', 'one year', 'three year', 'five year'.

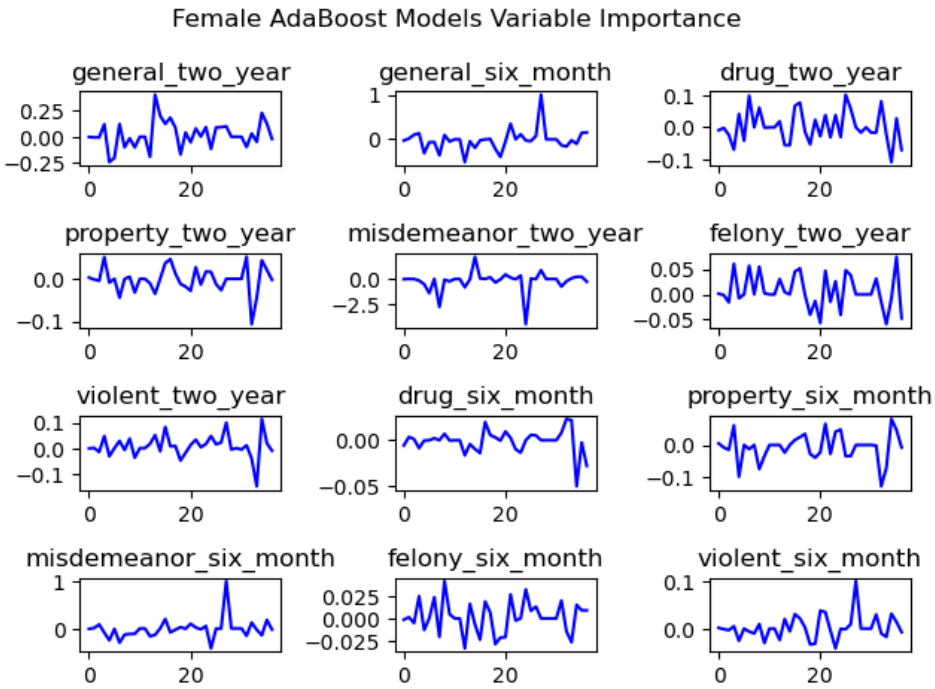

Figure 16: Variable Importance 1

In the female, Figure 16, and male Adaboost model, Figure 17, each model appears to favour different features. Interestingly, for male misdemeanor 2 years and felony 2 years, "current violence" and "p weapon" were almost the entire model. The rest of the features were around the order of 1E-6. This is unsurprising, because the original data had strong correlation between these features. The female model was even less conclusive with variable importance. All the variables have different scales of importances in each model and no features appear to be outliers for any models.

Similarly, the Ridge Regression models, Figure 18, 19, do not have any clear outliers in variable importance. In the female models for both general models, both misdemeanors and drug 6 months, the most important variable was "p probation". This is a surprising trend that is more than likely due to social structures.

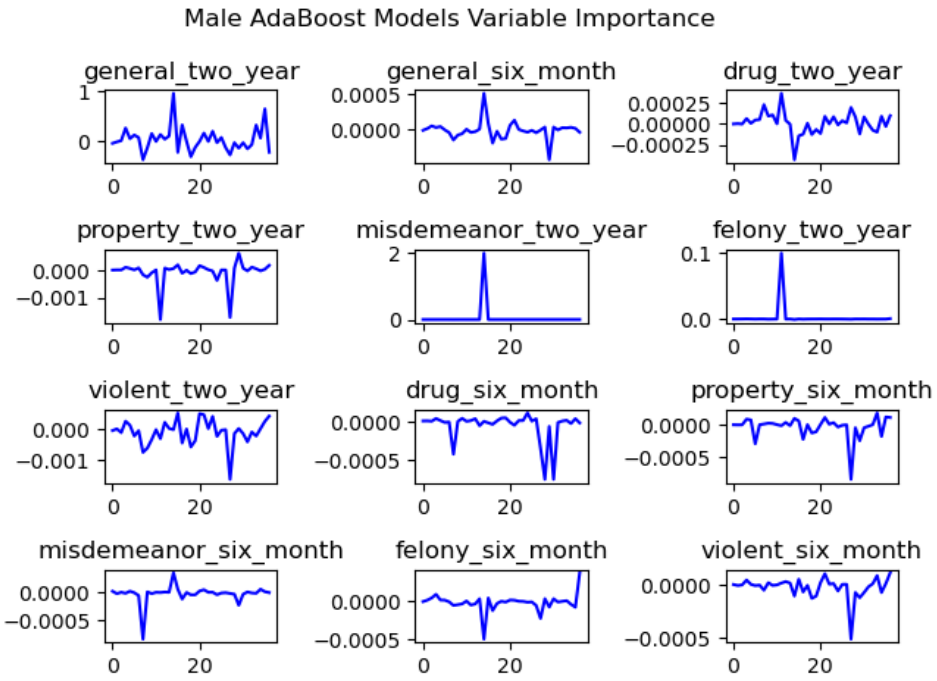

Figure 17: Variable Importance 2

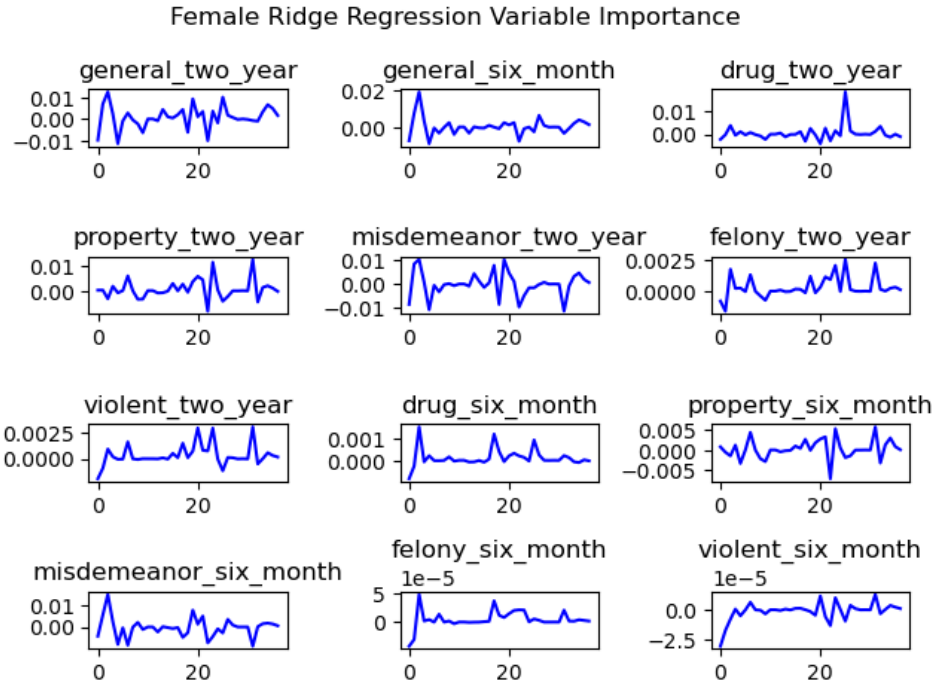

Figure 18: Variable Importance 3

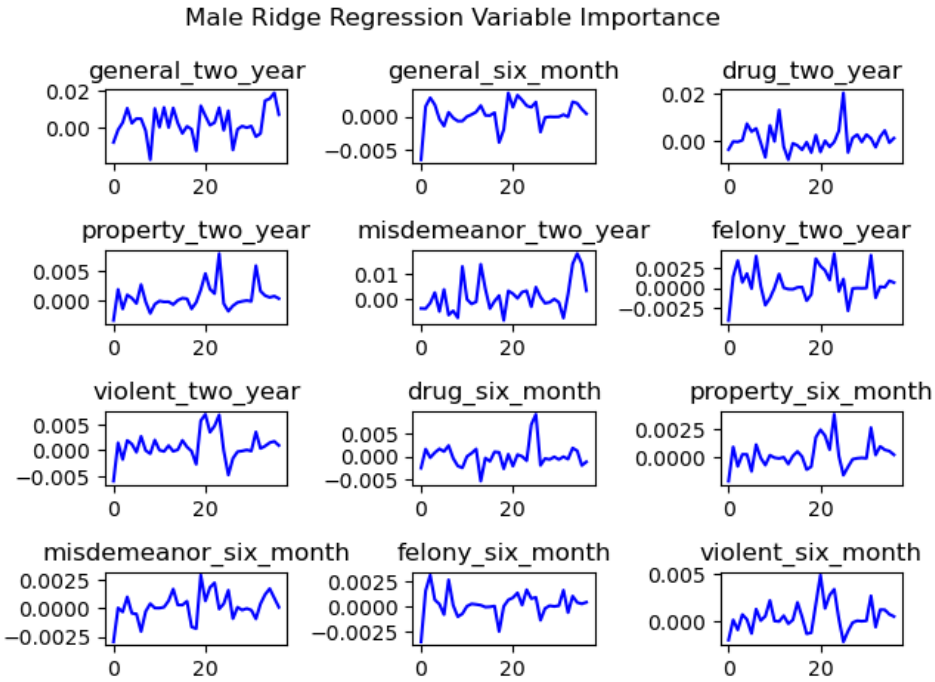

Figure 19: Variable Importance 4

4 Insight

4.1 SVM Model

The SVM model performed quite well on the dataset. Generally the accuracy for both the female and male test set was around 0.9. Additionally, most of the training accuracies were slightly above, but close to, the testing accuracies. This highlights that the models did not overfit. There were a couple of aberrations where the test accuracy was higher than the train accuracy. This can most likely be attributed to 'good luck' in the model, which was exacerbated by the limited number of data. This hypothesis is affirmed by the fact that the male testing data has much fewer testing results that were better than training results than the female testing data. The highest performing models were in the female drug, property, misdemeanour and violent models at 96% and the 6 month male Property model at 94.7%. The lowest performing model was the general 2 year model. For the female models it sat at 78% and for the male model it sat at 59.4%. In the female models, almost all of the C parameters were 0.1, except for the most broad outcomes, general 2 year. This highlights that the algorithm found that there did not need to be a large penalty for misclassification in any of the cases except the general cases. This makes sense as the general cases have much more data which can be influential and thus it is important to not overfit to the many nuances of each of the features. Additionally, in all the rbf models, the gamma is very low which aims to reduce overfitting to the data. Potentially because the general model is linear, since they are effectively a combination of all the other models, to make the model smoother the higher C compensated for the lack of rbf smoothing seen in some of the more specific models.

In the male models, the C is relatively low overall; however, interestingly, the general C value is amongst the smallest. Additionally the gamma for most of the rbf functions are higher than the female gammas. Both of these characteristics could be because the male dataset had more datapoints. Thus there was a more clear boundary between classes and the model could fit to that boundary without overfitting outliers by using a higher gamma. The C value may have been smaller for the general model because of the same reason; misclassifications with a tight fit were less likely to happen and less likely to be penalised.

4.2 Adaboost Model

The Adaboost model also had the general 2 year model perform the worst. This was at 70% and 59.7% respectively. Most of the other models performed well except for the male violent and misdemeanour 2 year models. The violent model's poor performance is most likely because an rbf kernel is needed. For most of the models, the optimal number of estimators was 10 for female and 50 for male. This difference is expected, because with more datapoints it is more likely to be more effective to have a lot of weak estimators using different datapoints rather than creating estimators with a large majority of the data. Finally the learning rate of most of the estimators was very low. Clearly it is important that the model does not overfit to the data. The average female learning rate was lower than the male, which could be attributed to the lack of datapoints and thus potential occurrences of underfitting to avoid being influenced by outliers.

4.3 KNN Model

The KNN had very high accuracy, especially when tested on the training data. There were multiple models which perfectly classified the training data, but this overfitting meant that the testing accuracy was lower than the other models. This being said, the KNN model still performed well on these sets of testing data. Unsurprisingly, the average number of neighbours for the male model was much higher than that of the female model. This is clearly due to the number of datapoints in each set. It is interesting that the 6 month property number of neighbours is considerably lower than the others. This could potentially be because the trends between features is so strong and thus having a high number of neighbours is unnecessary. During the initial plotting of the data, it was noticed that the prior stealing convictions and property recidivism were by far the most correlated features in the dataset. On the other hand, because of this high correlation, the small number of neighbours could have been because the other features introduced unnecessary outliers and thus it was important to only use the closest datapoints. Euclidean distance was rarely used, but was used solely within the

female models. Generally all of the models use the Minkowski distance metric, but there was no clear trends between models or even between the p value and the weight. All the female models except the most general used the uniform weight, which supports the hypothesis that the general models must incorporate as many features as possible, including proximity between datapoints, to make accurate classifications. The male models were much more sporadic, and, in fact, used distance more than uniform weight. This could potentially be because the data had less clear boundaries between any clusters that appeared and thus it is important to only choose neighbours that are closest. This is also most likely the case because the number of neighbours in the male models were higher and thus introduced more of the datapoints on the margin.

4.4 Ridge Regression

Unsurprisingly the ridge regression algorithm did not perform particularly well in the general cases, but did do quite a good job across the board. Apart from the general case, on the 2 year misdemeanour, felony and violent models, it performed the worst at around 75%. On the other models it was roughly around 90%. Since the model used was a regressor being used as a classifier, this lack of success was expected, however, overall the results aren't considerably lower than the other models. As discussed previously, the alpha value was a lot higher than initially predicted and sat around 10000 on average. This would do a good job of preventing overfitting, especially important on such a small dataset. The best solver chosen for each model was very different between models. This could be because some datasets are harder to predict and thus must be robust to outliers, using Cholesky, or potentially are trying to mimic more of the classification style of the SVM using least squares.

4.5 Comparisons

When comparing the accuracy of the SVM and Adaboost SVM models, the Adaboost models almost all performed slightly better than the pure SVM. This was expected and acts as a verification that there were no bugs in the method. There were certainly some outliers, but this can easily be attributed to the fact that their optimal SVM model had rbf kernels and this was not tested with the Adaboost model. A surprising observation that was made when comparing the SVM and Adaboost SVM models was that the C values of each model were very different. Clearly, adding many of the optimal estimator does not have as effective of a fit as adding many suboptimal estimators. This could be explained by the fact that adding suboptimal estimators gives more breadth than adding a lot of the same slightly stronger estimator. One caveat is that the Adaboost model was only implemented with a linear kernel and thus some of the rbf estimators which were optimal by themselves were not considered. Overall, the Adaboost SVM was superior to the SVM, but was highly computationally inefficient. The Adaboost algorithm took approximately 27 hours to run compared to 30 minutes for the SVM algorithm. The Adaboost algorithm only had improvements on the order of 1%. In future, it would be helpful to implement the Adaboost algorithm with the rbf kernel as well.

Overall the best models are summarised below, Table 13, 14. The SVM model did very well for the male data. It was interesting how many of the models performed exactly the same on the test set. This could potentially be because the margins for those datapoints were much more clear than the others and thus any decent algorithm would have success classifying that number of datapoints.

	General	Drug	Property	misdemeanour	Felony	Violent
Female	SVM	KNN	Any	SVM, ADA or KNN	Any	Any
Male	ADA	Any	SVM	SVM	SVM	SVM

Table 13: Best Algorithms - 2 Year

	General	Drug	Property	misdemeanour	Felony	Violent
Female	Any	Any	Any	Any	Any	Any
Male	SVM!	SVM	SVM	SVM	KNN	Any

Table 14: Best Algorithms - 6 Month

Please note for the 6 month Male General SVM improves on the others by nearly 20%!

Compared to the paper this paper was based on, the models performed less well. This being said, without the same amount of time spent precisely tuning the outcomes, these algorithms did very well on the test set, frequently producing accuracies over 95%. Since these algorithms are easily interpretable, easy to implement and are frequently linear models which do not mask the effect of certain variables, more work should be put into these methods to determine whether they are a viable solution to the Arnold PSA or the decision tree algorithms.

5 Errors and Mistakes

One mistake that was caught late in the process was that rather than updating the model that was to be used to determine the accuracy, I accidentally wrote "==" rather than "=". This meant that my analysis on the fitting of the model was incorrect. I discovered the problem when creating the Hyperparameter table when I was very surprised that the hyperparameters for both the female and male models were identical. The hardest part of the project was comparing all of the outcomes from all of the models.

6 Citations

References

[1] An end to mass incarceration: Crown Family School of Social Work, policy, and Practice (1970) An End to Mass Incarceration — Crown Family School of Social Work, Policy, and Practice. Available at: https://crownschool.uchicago.edu/ssa_magazine/end-mass-incarceration.html (Accessed: 05 December 2023)

[2] Smith, J. and Tom (2020) How long does it take to process a criminal case in North Carolina?, North Carolina Criminal Law. Available at: https://nccriminallaw.sog.unc.edu/how-long-does-it-take-to-process-a-criminal-case-in-north-carolina/ (Accessed: 05 December 2023).

[3] Wang, C., Han, B., Patel, B., Mohideen, F., Rudin, C. (2022). In Pursuit of Interpretable, Fair and Accurate Machine Learning for Criminal Recidivism Prediction. Journal of Quantitative Criminology, 38(3), 687-714.

[4] Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), pp. 2825-2830.

7 Appendix

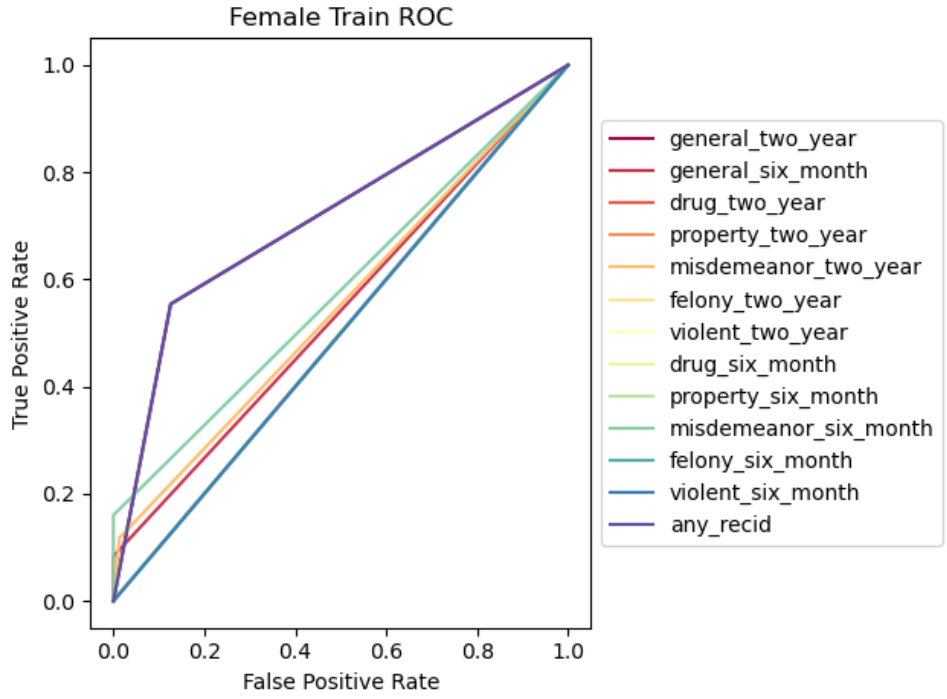

Figure 20: Train ROCs

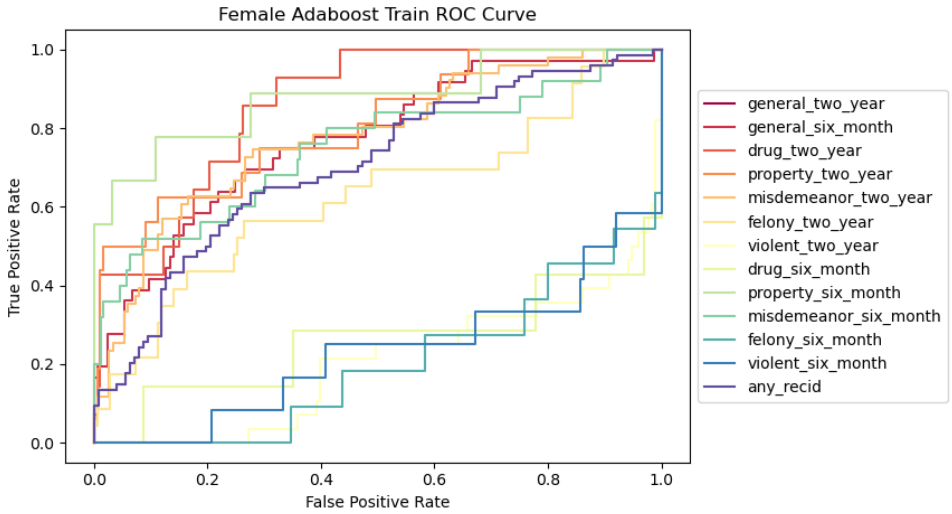

Figure 21: Train ROCs

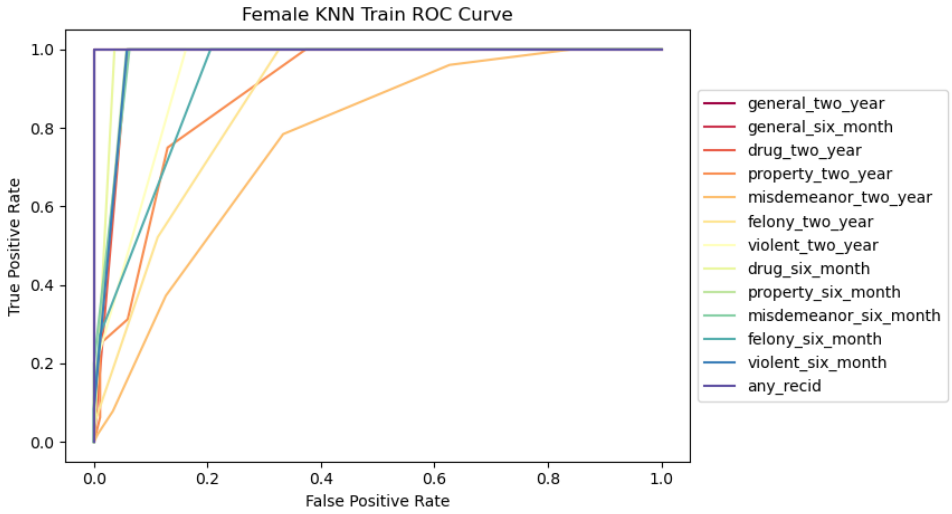

Figure 22: Train ROCs

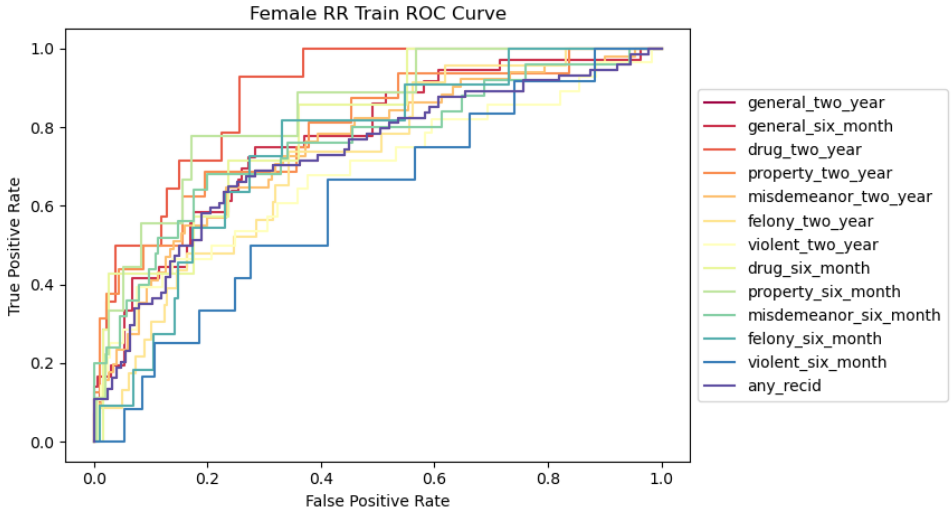

Figure 23: Train ROCs

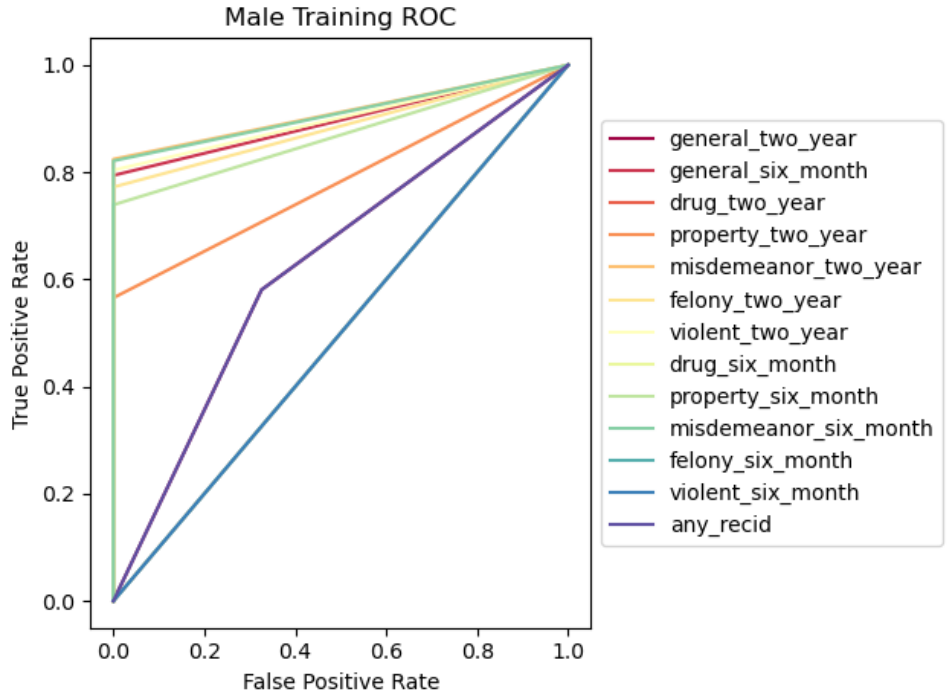

Figure 24: Train ROCs

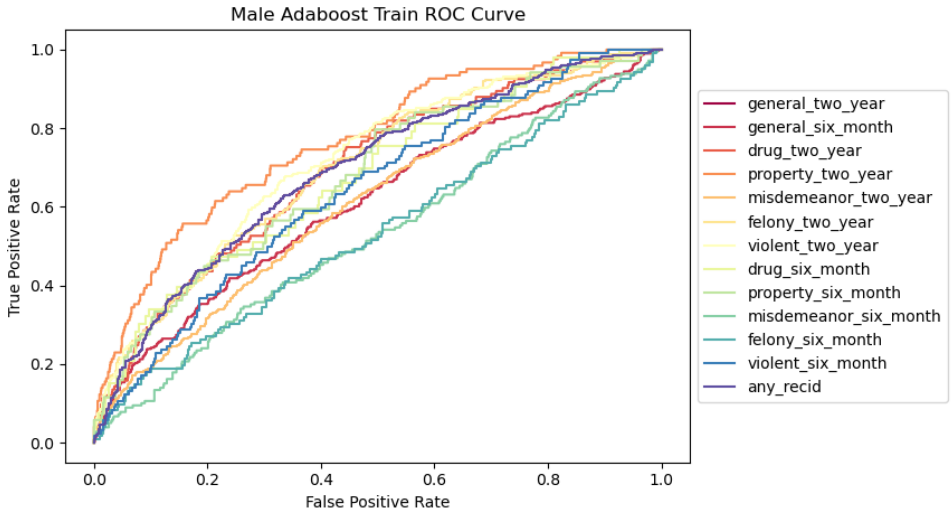

Figure 25: Train ROCs

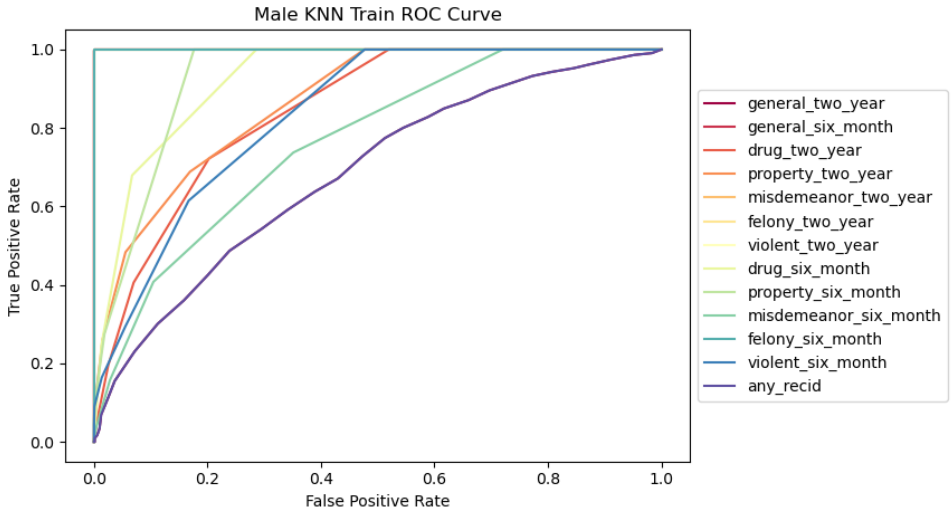

Figure 26: Train ROCs

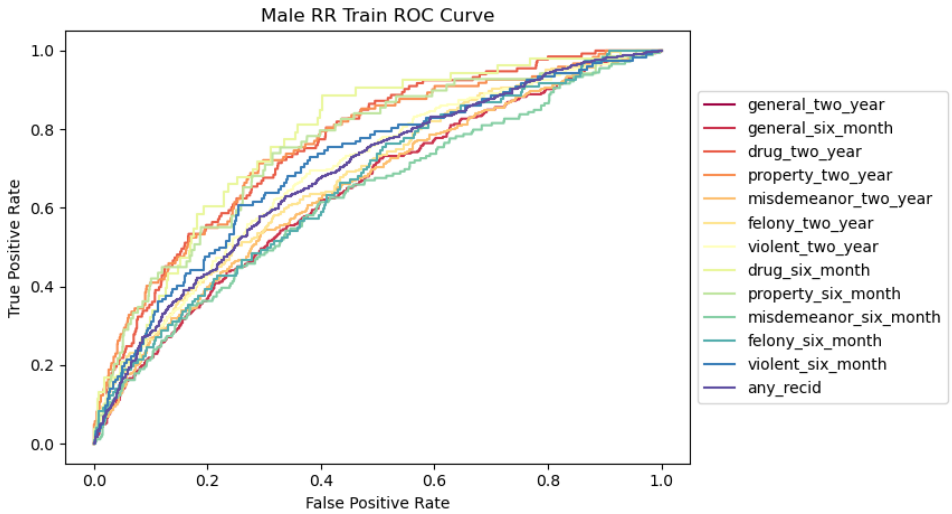

Figure 27: Train ROCs

8 Code

```
#!/usr/bin/env python
# coding: utf-8

# In[92]:


import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib

from sklearn import svm
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
import random

from joblib import Parallel, delayed
import joblib


# In[109]:


data = pd.read_csv("/Users/alannamanfredini/Documents/* ML/Final Project/broward_data.csv

#import data, shuffle it to ensure there are no hidden biases which could appear during
fem_dat = data[data['sex'] == 0].sample(frac=1).reset_index(drop=True)
male_dat = data[data['sex'] == 1].sample(frac=1).reset_index(drop=True)


# In[388]:


print(fem_dat.shape)
print(male_dat.shape)


# In[35]:


#cut out columns which may introduce racial bias, temporal bias or which do not provide
#for doing the pairwise comparison plot, cut out binary classification terms
male_pairwise_dat = male_dat.drop(['person_id','screening_date','sex', 'race'], axis = 1
male_pairwise_dat_nonbin = male_pairwise_dat.loc[:,male_pairwise_dat.nunique() > 2]

fem_pair_dat = fem_dat.drop(['person_id','screening_date','sex', 'race'], axis = 1)
fem_pair_dat_nonbin = fem_pair_dat.loc[:,fem_pair_dat.nunique() > 2]


# In[258]:
```

```
print(male_train.columns)


# In[36]:


# double checking for dodgy data - one man has first charge at 0
print(male_pairwise_dat[male_pairwise_dat['age_at_first_charge']<10])


# In[37]:


#sns.pairplot(fem_pair_dat[iji])


# In[38]:


"""
0.2 % of the male data is 340 datapoints
0.2 % of the female data is 50 dp

because the data was shuffled during import,
there is no worry about introducing biases by choosing the end of the dataset as our tes
"""

male_test = male_pairwise_dat.iloc[:340]
male_train = male_pairwise_dat.iloc[340:].reset_index(drop=True)

fem_test = fem_pair_dat.iloc[:50]
fem_train = fem_pair_dat.iloc[50:].reset_index(drop=True)


# In[40]:


# function to return a SVC model based off a single input (not Gridsearch)
def cropping_model(a,b,recid, kern):
    male_train_X = male_train.iloc[:,a:b]
    male_train_y = male_train.iloc[:,recid]

    model = svm.SVC(kernel = kern)
    model.fit(male_train_X, male_train_y)

    pred = model.predict(male_train_X)
    df = pd.DataFrame({'pred':pred, 'real' : male_train_y})
    return model, df


# In[41]:


#X val/ hyperparam tuning
hyperparams = {'kernel':('linear', 'rbf'), 'C':[0.1, 1, 10, 100], 'gamma':[0.1, 1, 10, 1
```

```
# RVM??


# In[42]:


#creating a function to make a model for the male training data
def grid_model(a,b,recid, param):
    male_train_X = male_train.iloc[:,a:b]
    male_train_y = male_train.iloc[:,recid]
    model = svm.SVC()
    classif = GridSearchCV(model, param)
    classif.fit(male_train_X, male_train_y)

    return classif


# In[35]:


#make models for all the possible male outcomes
models = [param_mod]

for mod in range(38,49):
    models.append(grid_model(0,37,mod,hyperparams))
    print(mod)


# In[52]:


# add a column to see whether the model can predict any form of recidivism
recid_outcome = male_train.iloc[:,37:].sum(axis = 1)
recid_outcome[recid_outcome>0] = 1
male_train['any_recid'] = recid_outcome

recid_outcome = male_test.iloc[:,37:].sum(axis = 1)
recid_outcome[recid_outcome>0] = 1
male_test['any_recid'] = recid_outcome
print(male_test)


# In[43]:


# create model for any recidivism
classif = grid_model(0,37,49,hyperparams)


# In[65]:


# save models
for mod in range(37,50):
    joblib.dump(models[mod-37], f'{mod}.pkl')
```

```
joblib.dump(classif, f'any.pkl')


# In[45]:


# create function for creating SVM models for female data
def fem_grid_model(a, b, recid, param):
    fem_train_X = fem_train.iloc[:,a:b]
    fem_train_y = fem_train.iloc[:,recid]
    model = svm.SVC()
    classif = GridSearchCV(model, param)
    classif.fit(fem_train_X, fem_train_y)

    return classif


# In[53]:


#add a column for all recidivism
recid_outcome_fem = fem_train.iloc[:,37:].sum(axis = 1)
recid_outcome_fem[recid_outcome_fem>0] = 1
fem_train['any_recid'] = recid_outcome_fem

recid_outcome_fem = fem_test.iloc[:,37:].sum(axis = 1)
recid_outcome_fem[recid_outcome_fem>0] = 1
fem_test['any_recid'] = recid_outcome_fem


# In[72]:


# save
for mod in range(37,50):
    joblib.dump(fem_grid_model(0,37,mod,hyperparams), f'fem_{mod}.pkl')


# In[54]:


# create datafframes that store the prediction and real results for each model next to ea
# store the best parameters for each model
train_results_df_fem = pd.DataFrame()
train_results_df_male = pd.DataFrame()

test_results_df_fem = pd.DataFrame()
test_results_df_male = pd.DataFrame()

params_df_male = []
params_df_fem = []
for i in range(37,49):
    model_m = joblib.load(f'{i}.pkl')
    model_f = joblib.load(f'fem_{i}.pkl')
```

```
    train_results_df_fem[f'real_{i}'] = fem_train.iloc[:,i]
    train_results_df_fem[f'{i}'] = model_f.predict(fem_train.iloc[:,0:37])

    train_results_df_male[f'real_{i}'] = male_train.iloc[:,i]
    train_results_df_male[f'{i}'] = model_m.predict(male_train.iloc[:,0:37])

    test_results_df_fem[f'real_{i}'] = fem_test.iloc[:,i]
    test_results_df_fem[f'{i}'] = model_f.predict(fem_test.iloc[:,0:37])

    test_results_df_male[f'real_{i}'] = male_test.iloc[:,i]
    test_results_df_male[f'{i}'] = model_m.predict(male_test.iloc[:,0:37])

    params_df_male.append(model_m.best_params_)
    params_df_fem.append(model_f.best_params_)

model_m = joblib.load(f'any.pkl')
model_f = joblib.load(f'fem_49.pkl')


train_results_df_fem[f'real_49'] = fem_train.iloc[:,49]
train_results_df_fem[f'49'] = model_f.predict(fem_train.iloc[:,0:37])

train_results_df_male[f'real_49'] = male_train.iloc[:,49]
train_results_df_male[f'49'] = model_m.predict(male_train.iloc[:,0:37])

test_results_df_fem[f'real_49'] = fem_test.iloc[:,49]
test_results_df_fem[f'49'] = model_f.predict(fem_test.iloc[:,0:37])

test_results_df_male[f'real_49'] = male_test.iloc[:,49]
test_results_df_male[f'49'] = model_m.predict(male_test.iloc[:,0:37])

params_df_male.append(model_m.best_params_)
params_df_fem.append(model_f.best_params_)


# In[362]:


# store the accuracies and plot the ROC curves for each model
names = male_train.iloc[:,37:].columns
models = [train_results_df_male, test_results_df_male, train_results_df_fem, test_results
mod_name = ['Male Training ROC', 'Male Test ROC', 'Female Train ROC', 'Female Test ROC']

accuracy = []

cmap = matplotlib.cm.get_cmap('Spectral')

mod_count = 0
for mods in models:
    fig, ax = plt.subplots()
    counter = 0
    for i in range(13):
        fpr = [0]
        tpr = [0]
        relev_df = mods.iloc[:,counter:counter+2]
```

```
        accuracy.append(metrics.accuracy_score(relev_df.iloc[:,0], relev_df.iloc[:,1]))

        neg = relev_df[relev_df.iloc[:,0] == 0]
        pos = relev_df[relev_df.iloc[:,0] == 1]


        fpr.append(neg.sum()[1]/neg.shape[0])
        tpr.append(pos.sum()[1]/pos.shape[0])

        fpr.append(1)
        tpr.append(1)


        ax.plot(fpr, tpr, label = names[i], color=cmap(i / 12))
        counter+=2
    ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.set_title(mod_name[mod_count])
    fig.tight_layout()
    fig.savefig(f'{mod_name[mod_count]}.png', bbox_inches = 'tight')

    mod_count += 1



print(fpr)
print(tpr)


# In[358]:


# split the accuracies so they are more legible
male_tr_acc = accuracy[:13]
male_te_acc = accuracy[13:26]
fem_tr_acc = accuracy[26:39]
fem_te_acc = accuracy[39:]


# In[104]:


# print accuracies
print(male_tr_acc)
print('************')
print(male_te_acc)
print('************')
print(fem_tr_acc)
print('************')
print(fem_te_acc)
print('************')


# In[105]:
```

```
# print out the Gridsearch parameters
print("male params")
print(params_df_male)

print("fem params")
print(params_df_fem)


# NEXT MODEL - ADABOOST USING SVM

# In[170]:


# print data for next model
print(male_train.shape)
print(male_test.shape)
print(fem_train.shape)
print(fem_test.shape)

DataSets = [fem_train, fem_test, male_train, male_test]


# In[110]:


from sklearn.ensemble import AdaBoostClassifier


# In[222]:


# create function which returns the adaboost model for SVC with gridsearch
def adaboost_mod(data_x, data_y, kernel, C, gamma):

    # Create the base estimator (SVM)
    if kernel == 'linear':
        base_estimator = svm.SVC(kernel=kernel, C = C, probability=True)
    if kernel == 'rbf':
        base_estimator = svm.SVC(kernel=kernel, gamma = gamma, probability=True)

    # Create the AdaBoost model
    model = AdaBoostClassifier()
    GS_param = {'estimator':[base_estimator], 'random_state':[0], 'n_estimators':[10,50,1
    clf = GridSearchCV(model, GS_param)

    # Train the model
    clf.fit(data_x, data_y)

    return clf


# In[225]:
```

```
train_sets = [fem_train, male_train]
C = [0.1,1,10]

Cs = []
modl = []
counter_train = 0

# use gridsearch to determine the best Adaboost parameters... within the loops create th
    #the best C. This is effectively a second loop of gridsearch for the base estimator
for train in train_sets:
    if counter_train == 0: name = "fem"
    else: name = "male"
    for recid in range(37,50):
        print('Next model:', recid)
        small_c = adaboost_mod(train.iloc[:,:37], train.iloc[:,recid], 'linear', 0.1, 'x
        print('small finished')
        med_c = adaboost_mod(train.iloc[:,:37], train.iloc[:,recid], 'linear', 1, 'x')
        print('med finished')
        big_c = adaboost_mod(train.iloc[:,:37], train.iloc[:,recid], 'linear', 10, 'x')
        print("large finished")

        models = [small_c, med_c, big_c]

        scores = np.array([small_c.best_score_, med_c.best_score_, big_c.best_score_])

        #create a list of the best C for each model
        Cs.append(C[scores.argmax()])
        modl.append(models[scores.argmax()])

        #save the model
        joblib.dump(models[scores.argmax()], f'Ada_{name}_{recid}.pkl')
    counter_train += 1


# In[240]:


# load models from saved
modl = []
counter_train = 0
for train in train_sets:
    if counter_train == 0: name = "fem"
    else: name = "male"
    for recid in range(37,50):
        modl.append(joblib.load(f'Ada_{name}_{recid}.pkl'))
    counter_train += 1


# In[242]:


#split the Cs and model lists into separate genders for easy interpretation
fem_Cs = Cs[:int(len(Cs)/2)]
male_Cs = Cs[int(len(Cs)/2):]

fem_models = modl[:int(len(modl)/2)]
```

```
male_models = modl[int(len(modl)/2):]


# In[243]:


# Create a general function to make an ROC when given the predicted values
names = male_train.iloc[:,37:].columns
cmap = matplotlib.colormaps['Spectral']
def ROC(figu, axi, model, data, real, i):
    pred = model.predict_proba(data)[:,1]

    fpr, tpr, thresholds = metrics.roc_curve(real, pred)

    axi.plot(fpr, tpr, label = names[i], color=cmap(i / 12))


# In[373]:


# print the best hyperparameters for comparison
for items in male_models:
    print(items.best_params_)


# In[361]:


# create and ROC curve for each model
dataset_idx = 0
# one plot for each gender
for genders in range(2):
    if genders == 0: modl = fem_models; gen = 'Female'
    else: modl = male_models; gen = 'Male'
    print(f"Genders {gen}")

    #one plot for test and train
    for te_tr in range(2):
        if te_tr == 0: calc = 'Train'
        else: calc = 'Test'

        print(f"Test? {calc}")

        fig, ax = plt.subplots()

        relv_DS = DataSets[dataset_idx]
        counter = 0

        for mod in modl: # for the models for each outcome make an ROC curve
            print(f"Model {mod.best_params_}")
            feat = relv_DS.iloc[:,:37]
            expected = relv_DS.iloc[:,counter+37]
            ROC(fig, ax, mod, feat, expected, counter)

            counter+=1
```

```
        ax.set_xlabel("False Positive Rate")
        ax.set_ylabel("True Positive Rate")
        ax.set_title(f"{gen} Adaboost {calc} ROC Curve")
        fig.tight_layout()
        ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
        fig.savefig(f"{gen} Adaboost {calc} ROC Curve.png", bbox_inches = 'tight')
        dataset_idx += 1


# In[335]:


#Determine the accuracy score for each of the test and train models
ada_acc = []
dataset_idx = 0
for genders in range(2):
    if genders == 0: modl = fem_models; gen = 'Female'
    else: modl = male_models; gen = 'Male'
    print(f"Genders {gen}")

    for te_tr in range(2):
        if te_tr == 0: calc = 'Train'
        else: calc = 'Test'

        print(f"Test? {calc}")

        relv_DS = DataSets[dataset_idx]
        counter = 0

        for mod in modl:
            print(f"Model {mod.best_params_}")
            feat = relv_DS.iloc[:,:37]
            expected = relv_DS.iloc[:,counter+37]
            ada_acc.append(metrics.accuracy_score(expected, mod.predict(feat)))

            counter+=1

        dataset_idx += 1


# In[336]:


print(ada_acc[:13])
print(ada_acc[13:26])
print(ada_acc[26:39])
print(ada_acc[39:])


# NEXT MODEL - KNN

# In[ ]:


from sklearn.neighbors import KNeighborsClassifier
```

```
# In[297]:


# Similar to previously, create a function for gridsearch on KNN model
def KNN_mod(data_x, data_y):
    GS_param = {'n_neighbors':[1, 2, 5, 10, 25, 50], 'weights':['uniform','distance'], '

    # Create the base estimator
    neighbours = KNeighborsClassifier()
    clf = GridSearchCV(neighbours, GS_param)

    # Train the model
    clf.fit(data_x, data_y)

    return clf


# In[217]:


# ignore warnings
import warnings
warnings.filterwarnings("ignore", message=".*Minkowski metrics are not distance metrics.*


# In[298]:


# create a model for each recidivism outcome
train_sets = [fem_train, male_train]
modl_KNN = []
counter_train = 0
for train in train_sets:
    if counter_train == 0: name = "fem"
    else: name = "male"

    print("Gender: ", name)
    for recid in range(37,50):
        print("Output", recid)
        mod = KNN_mod(train.iloc[:,:37], train.iloc[:,recid])
        modl_KNN.append(mod)

        joblib.dump(mod, f'KNN_{name}_{recid}.pkl')
    counter_train += 1


# In[299]:


#split models for ease of parsing
KNN_fem_models = modl_KNN[:int(len(modl_KNN)/2)]
KNN_male_models = modl_KNN[int(len(modl_KNN)/2):]


# In[363]:
```

```
# create an ROC curve for each model
dataset_idx = 0
KNN_acc = []
for genders in range(2):
    if genders == 0: modl = KNN_fem_models; gen = 'Female'
    else: modl = KNN_male_models; gen = 'Male'
    print(f"Genders {gen}")


    for te_tr in range(2):
        if te_tr == 0: calc = 'Train'
        else: calc = 'Test'

        print(f"Test? {calc}")

        fig, ax = plt.subplots()

        relv_DS = DataSets[dataset_idx]

        counter = 0
        for mod in modl: # for the models for each outcome make an ROC curve
            print(f"Model {mod.best_params_}")
            feat = relv_DS.iloc[:,:37]
            expected = relv_DS.iloc[:,counter+37]
            ROC(fig, ax, mod, feat, expected, counter)

            counter+=1

        ax.set_xlabel("False Positive Rate")
        ax.set_ylabel("True Positive Rate")
        ax.set_title(f"{gen} KNN {calc} ROC Curve")
        fig.tight_layout()
        ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
        fig.savefig(f"{gen} KNN {calc} ROC Curve.png", bbox_inches = 'tight')
        dataset_idx += 1


# In[339]:


#determine the accuracy of each model on both test and train data
KNN_acc = []
dataset_idx = 0
for genders in range(2):
    if genders == 0: modl = KNN_fem_models; gen = 'Female'
    else: modl = KNN_male_models; gen = 'Male'
    print(f"Genders {gen}")

    for te_tr in range(2):
        if te_tr == 0: calc = 'Train'
        else: calc = 'Test'

        print(f"Test? {calc}")
```

```
        relv_DS = DataSets[dataset_idx]
        counter = 0

        for mod in modl:
            #print(f"Model {mod}")
            feat = relv_DS.iloc[:,:37]
            expected = relv_DS.iloc[:,counter+37]
            KNN_acc.append(metrics.accuracy_score(expected, mod.predict(feat)))

            counter+=1

        dataset_idx += 1


# In[340]:


# print accuracies
print(KNN_acc[:13])
print(KNN_acc[13:26])
print(KNN_acc[26:39])
print(KNN_acc[39:])


# In[364]:


# plot the accuracy of each model compared to the changing hyperparameters
pot_n_neighbours=[1, 2, 5, 10, 25, 50]
pot_p=[0.5,1,2,3, 5, 10, 100]
p_loc = np.array([14,16,18,20,22,24,26])


dataset_idx = 0
for genders in range(2):
    if genders == 0: modl = KNN_fem_models; gen = 'Female'
    else: modl = KNN_male_models; gen = 'Male'
    print(f"Genders {gen}")

    fig1, ax1 = plt.subplots()
    fig2, ax2 = plt.subplots()

    i = 0
    for mod in modl: # for the models for each outcome make an ROC curve

        data = pd.DataFrame(mod.cv_results_)
        nn_dat = data.loc[(data.index % 14) == 4]
        if gen == 'Female': p_dat = data.loc[p_loc]
        else: p_dat = data.loc[p_loc+ 14*3]

        ax1.semilogx(pot_n_neighbours, nn_dat['mean_test_score'], label = names[i])
        ax2.semilogx(pot_p, p_dat['mean_test_score'], label = names[i])
        i += 1


    ax1.set_xlabel("log Number Neighbours")
```

```
    ax1.set_ylabel("Mean Test Score")
    ax1.set_title(f"{gen} KNN Number of Neighbours Hyperparameter Performance")
    fig1.tight_layout()
    ax1.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    fig1.savefig(f"{gen} KNN Hyperparameter Performance - NN.png", bbox_inches = 'tight')

    ax2.set_xlabel("log P")
    ax2.set_ylabel("Mean Test Score")
    ax2.set_title(f"{gen} KNN P Hyperparameter Performance")
    fig2.tight_layout()
    ax2.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    fig2.savefig(f"{gen} KNN Hyperparameter Performance - P.png", bbox_inches = 'tight')
    dataset_idx += 1




# NEXT MODEL - RIDGE

# In[273]:


from sklearn.linear_model import Ridge


# In[381]:


# create a function to create a gridsearch for ridge regression
def Ridge_mod(data_x, data_y):
    # linear regression alpha also chosen for comparison
    GS_param = {'alpha':[0, 1, 100, 1000, 10000, 1000000], 'solver':['svd', 'cholesky',

    # Create the base estimator
    RR = Ridge()
    clf = GridSearchCV(RR, GS_param)

    # Train the model
    clf.fit(data_x, data_y)

    return clf


# In[382]:


# create and save best ridge regression models
modl_RR = []
counter_train = 0
for train in train_sets:
    if counter_train == 0: name = "fem"
    else: name = "male"

    print("Gender: ", name)
    for recid in range(37,50):
```

```
        print("Output", recid)
        curr_mod = Ridge_mod(train.iloc[:,:37], train.iloc[:,recid])
        modl_RR.append(curr_mod)

        joblib.dump(curr_mod, f'RR_{name}_{recid}.pkl')
    counter_train += 1


# In[383]:


# split for easy parsing
RR_fem_models = modl_RR[:int(len(modl_RR)/2)]
RR_male_models = modl_RR[int(len(modl_RR)/2):]


# In[384]:


#function for ROCs for the models
names = male_train.iloc[:,37:].columns
cmap = matplotlib.colormaps['Spectral']
def ROC_RR(figu, axi, model, data, real, i):
    pred = model.predict(data)

    fpr, tpr, thresholds = metrics.roc_curve(real, pred)

    axi.plot(fpr, tpr, label = names[i], color=cmap(i / 12))


# In[385]:


# create the ROC for both genders and test/train
dataset_idx = 0
RR_acc = []
for genders in range(2):
    if genders == 0: modl = RR_fem_models; gen = 'Female'
    else: modl = RR_male_models; gen = 'Male'
    print(f"Genders {gen}")


    for te_tr in range(2):
        if te_tr == 0: calc = 'Train'
        else: calc = 'Test'

        print(f"Test? {calc}")

        fig, ax = plt.subplots()

        relv_DS = DataSets[dataset_idx]

        counter = 0
        for mod in modl: # for the models for each outcome make an ROC curve
            print(f"Model {mod.best_params_}")
            feat = relv_DS.iloc[:,:37]
```

```
            expected = relv_DS.iloc[:,counter+37]
            ROC_RR(fig, ax, mod, feat, expected, counter)

            counter+=1

        ax.set_xlabel("False Positive Rate")
        ax.set_ylabel("True Positive Rate")
        ax.set_title(f"{gen} RR {calc} ROC Curve")
        fig.tight_layout()
        ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
        fig.savefig(f"{gen} RR {calc} ROC Curve.png", bbox_inches = 'tight')
        dataset_idx += 1


# In[386]:


# determine the accuracy of each model
dataset_idx = 0
RR_acc = []
for genders in range(2):
    if genders == 0: modl = RR_fem_models; gen = 'Female'
    else: modl = RR_male_models; gen = 'Male'
    print(f"Genders {gen}")


    for te_tr in range(2):
        if te_tr == 0: calc = 'Train'
        else: calc = 'Test'

        print(f"Test? {calc}")


        relv_DS = DataSets[dataset_idx]

        counter = 0
        for mod in modl:
            print(f"Model {mod.best_params_}")
            feat = relv_DS.iloc[:,:37]
            expected = relv_DS.iloc[:,counter+37]
            pred = pd.DataFrame(mod.predict(feat))
            pred[pred[0] > 0.5] = 1
            pred[pred[0] < 0.5] = 0

            RR_acc.append(metrics.accuracy_score(expected, pred ))

            counter += 1
        dataset_idx += 1


# In[387]:


# print accuracy
print(RR_acc[:13])
print(RR_acc[13:26])
```

```
print(RR_acc[26:39])
print(RR_acc[39:])


# In[446]:


#creating plots for vriable importance for ridge regression
dataset_idx = 0
rge = range(37)
RR_acc = []


for genders in range(2):
    if genders == 0: modl = RR_fem_models; gen = 'Female'
    else: modl = RR_male_models; gen = 'Male'
    print(f"Genders {gen}")
    fig, ax = plt.subplots(4, 3)
    counter = 0
    for mod in modl:
        if counter == 12: break
        print(f"Model {mod.best_params_}")
        y = mod.best_estimator_.coef_
        #print(y)
        ax[int(counter/3), counter%3].plot(rge, y, color = 'b')
        ax[int(counter/3), counter%3].set_title(names[counter])

        counter+=1


    fig.suptitle(f"{gen} Ridge Regression Variable Importance")
    fig.tight_layout()
    fig.savefig(f"{gen} Ridge Reg Variable Importance.png", bbox_inches = 'tight')


# In[454]:


#creating plots of the variable import in the Adaboost - this is using the same paramete
dataset_idx = 0
rge = range(37)
RR_acc = []

fem_c = [0.1,1,0.1,0.1,10,0.1,0.1,0.1,0.1,1,0.1,0.1]
male_c = [10,0.1,0.1,0.1,1,10,10,0.1,0.1,0.1,1,0.1]
for genders in range(2):
    if genders == 0: dat = fem_train; gen = 'Female'; c = fem_c
    else: dat = male_train; gen = 'Male'; c = male_c
    print(f"Genders {gen}")
    fig, ax = plt.subplots(4, 3)
    counter = 0
    for i in range(12):
        #print(f"Model {mod.best_params_}")
        mod = svm.SVC(kernel = 'linear', C = c[i])
        mod.fit(dat.iloc[:,:37], dat.iloc[:,37+i])
        y = mod.coef_
```

```
            print(i)
            ax[int(counter/3), counter%3].plot(rge, y[0], color = 'b')
            ax[int(counter/3), counter%3].set_title(names[counter])

            counter+=1


        fig.suptitle(f"{gen} AdaBoost Models Variable Importance")
        fig.tight_layout()
        fig.savefig(f"{gen} Ada Variable Importance.png", bbox_inches = 'tight')


# In[ ]:
```