Profiling Serial Killers Using Multiple Supervised Machine Learning Approaches

Simon M. Mariani (s.m.mariani@students.uu.nl)

June 2020, Utrecht University

Abstract

Criminal profiling has gained a lot of recognition over the years. Profiling is done by experts who use information from a crime scene, to create a serial killer profile. Such a profile consists of serial killer attributes and can include: the gender, race and possible previous activities of the killer. The paper proposes a framework that combines multiple well-knows supervised machine learning techniques to create such a profile. The majority of the proposed approaches obtained a balanced accuracy over 72%, and a predictive accuracy over 80%. The proposed approaches also performed well on a set of other databases, including a single-victim homicide database where it reached a balanced accuracy over 72% and a predictive accuracy over 77%

1 Introduction

After the 80's in the U.S. the total number of homicides has decreased, the FBI claims that this decrease is correlated with a decrease in serial killers. The decrease in solved homicides however, suggests otherwise [1, 2]. Thomas Hargrove, the founder of the murder accountability project, has uncovered many problems within police departments regarding possible serial killers and estimates the number of active serial killers a lot higher [1, 2]. There are 3 categories of killers: serial killers, mass murderers and spree killers [3]. Serial killers commit multiple murders where the victims have no relationship. This makes it incredibly hard to identify suspects. Mass murderers on the other hand, commit multiple murders that are related to each other, this makes them a lot easier to find than serial killers. Spree killers commit multiple murders in such a short time frame that the murders are handled as one incident. It can also occur that a person is placed in more than one category. The focus of this paper is on serial killers.

Because it is so difficult to tie a person to these non-related murders, criminal profiling has become indispensable in the process of finding these killers [4, 5, 6]. It is not an overstatement that in many previous investigations, the only methods law enforcement institutions had to track down these killers, was profiling them and finding suspects to fit the profile. Profiling is done using known information of previous serial killers and present information of the crimes under investigation. Because profiling is so reliant on information gathered from studying other serial killers, it is not surprising that a collaboration exists for the purpose of collecting and preserving information about serial killers, namely the Serial Homicide Expertise and Information Sharing Collaborative (SHEISC). This collaboration was founded by Enzo Yaksic in 2013 and was built on another database, namely the Radford Serial killer Database (Radford SKDB). Both databases contain extensive information of over 3000 known serial killers. With the use of several machine learning it seems that this data can be used effectively for the purpose of profiling. The problem statement of our project can be formulated as follows:

Given a dataset about serial killers, can we successfully predict the following attributes about the killers:

1- 'Sex' (male/female)
2- 'WhiteMale' (male and white)
3- 'WhiteMale20s' (male, white and in his 20s)
4- 'WorkedCop' (worked in some form of law enforcement)
5- 'Mid to Late 20s' (age range from mid 20s to late 20s)
6- 'Killed Prior to Series' (killed before the current killings)
7- 'Organizational' (can be classified as organizational)
8- 'partner' (had a partner for the killing(s))
9- 'race' (white/black/hisp/asian/native-american/aboriginal)

These features are columns in the DataBase from the SHEISC, which we were able to use with special thanks to Enzo Yaksic.
Predicting these variables is usually done by experts, who have gathered information and experience over the years. In our approach, we apply several supervised machine learning algorithms to a data set in order to achieve the same result.
Before a Database can be used effectively, some preprocessing is required. The columns that will be used as labels cannot contain any missing values. For the training data some missing values are allowed. However, if a column simply contains to many missing values, it will be removed. The remaining missing values will need to be imputed, in order for our models to perform well.
The suggested machine learning methods are: Logistic Regression, Support Vector Machines, Neural Networks, Bayesian Networks and several Classifier Ensembles which include random forests and voting ensembles. They will all

need to be optimized individually before they are combined into new models.

The paper is organized as follows: Section 2 contains information about related work. Section 3 provides an overview with information about the techniques used in this study. Section 4 explains our framework for solving the problem statement. Section 5 presents the evaluation procedure and discuss the obtained results. Section 6 concludes the study.

2 Related Work

Although criminal profiling has been around for quite a long time and there is a lot of theory about it. There are not a lot of people who combined it with machine learning. Bayesian approaches have been tried in [14, 15] with similar databases. An example thesis on this topic is [15], and [14] a paper on this topic from the same group of people. Both of these papers concerned themselves with the prediction of several offender characteristics in order to create an offender profile. The Bayesian approach both papers took, provide useful insights for Bayesian learning. The paper [14] concerned itself with predicting psycho-behavioral profiles of single-victim single-offender homicides. The results were remarkable as the Bayesian network achieved a higher accuracy on new cases than the experts. The database in [15] consisted of homicide offenders in the UK from the 1970s to the early 1990s and showed similar results as [14]. In both studies, an initial Bayesian network was created using expert knowledge and then further improved with the K2' structure learning algorithm. These two studies strongly relate to this one, as they try to predict offender features using some supervised machine learning technique. In order to do so, they first established if the database was sufficient. The sufficiency of a database depends on the number of variables, their domain, and the underlying probability distributions [14].
After concluding the sufficiency, both studies proceeded in constructing an initial Bayesian structure, based on expert knowledge and heuristics. This structure was then improved using the K2' learning algorithm, which showed significantly better results than the original K2 structure learning algorithm [15].
This structure was used to predict variables about single-victim homicides such as: 'prior record of violence', 'unemployed' and 'gender'. This is rather similar to what this study is trying to achieve. However, the focus of this study is not solely on Bayesian networks but on a range of methods, including Bayesian Networks.
A discussion point of these studies is the metric that they used, namely the predictive accuracy. The predictive accuracy is the percentage of correctly classified instances and does not take unbalanced classes into account. E.g. when a certain class is to be labeled positive ('1') in 80% of the cases, a classifier that will predict everything as positive will achieve a predictive accuracy of 80%, while not classifying any of the negative cases. While they do not mention imbalance in their data sets, it seems highly unlikely that the data used was balanced enough for the predictive accuracy to be a sufficient metric.

Many other studies that combine criminal data and machine learning are concerned with the prediction of crime before it happens [17, 18]. These studies generally make use of demographic, and temporal data to find crime prominent times and locations in order to focus police resources. Other studies try to find crime series that are likely to be committed by the same offender [19, 20]. Thomas Hargrove [20] used the Modus Operandi of homicide offenders, and the clearance rates to find so called 'murder clusters'. A 'murder cluster' is a cluster of unsolved homicides which are likely to be committed by the same offender.
These approaches are different because they are not focused on finding the personal features of an offender. They focus on finding crimes before they happen, or linking crimes to an unknown offender.

3 Preliminaries

3.1 Handling the Missing Data

The main data set is the one explained in Section 1 from the SHEISC. It has 180 columns, and 4802 rows where each represent a serial killer. However, many columns contain a lot of missing values which makes these columns unusable. There are merely 82 columns that contain less than 20.8% null values, 52 columns that contain less than 10.4% null values, and 22 columns that contain less than 1% null values (out of the entire data set with 180 columns). Because of the missing values and the unusable columns, the used data set contains less than 180 columns. Handling the missing values can be done using different techniques that will be explained in the following subsections.

3.1.1 Grouping variables

Finding one or more variables based on expert knowledge that usually occur together, and then taking the conjunction of these columns to form new column. This approach can work for some columns, but the data indicates that there are either a lot of missing values for a killer, or almost no missing values for a killer. This means that other variables cannot make up for the absence of a variable because they are often both absent or both present.

3.1.2 Impute with statistical aspects

Replacing the missing values with statistical aspects of that column such as the mean, median or mode (or instead of the mode some other constant value) [7]. This does not represent the correlation between different attributes very well, but it is fast and does not require a lot of work before the data set can be utilized. The benefit of using the mode or some other constant values, is that the imputed values will fall withing the range of the already existing values. This makes it suitable for Bayesian Learning, because the classes in the training data must be the same as in the test data. This is also achieved by leaving the missing values in the data as Bayesian Networks generally handle missing data well.

3.1.3 Statistical Methods

Statistical Methods such as Multivariate Imputation by Chained Equation (MICE) [7, 8]. This method is a Markov chain Monte Carlo (MCMC) method where the initial samples are randomly drawn from the observed data. Then these variables will be imputed one by one for a given N times, using the conditional probabilities. The aim of this algorithm as formulated in [7], consists of three main objectives: (1) account for the process that created the missing data, (2) preserve the relations in the data and (3) preserve the uncertainty of these relations.
Another statistical method that can be used for imputing missing values, is a Bayesian Network. When a Bayesian Network is constructed and learned with data containing missing values, the Network can be used for imputing these missing values.

3.1.4 machine learning methods

Imputing the missing variables using different ML methods such as K-nearest neighbors [9], which has gained more acceptance for this purpose over the years, or another ML algorithm that can be used for predicting variables. The latter will be explained in later sections because the ML algorithms used for predicting the variables from the research statement can also be used for imputing the missing values.

3.2 Imbalanced classes

Unbalanced classes can be a problem when trying to predict labels because the ML algorithm will not have enough examples to learn the deprived class from. Another reason why imbalanced classes are a problem is because of discrimination in the data [23, 24]. When the majority of a class is for example 'Male', the algorithm will learn a bias for males. The more imbalanced a class, the bigger the bias. Some approaches to deal with class imbalance are suggested:

3.2.1 Massaging

Simply changing some class labels from positive to negative and the other way around [23]. The labels that will be relabeled are chosen according to a ranking system, such as a probabilistic ranking. Labels that are close to the decision boundary are more vulnerable to discrimination and are therefore suited for relabeling. The obvious disadvantage of this method is that the data is altered directly and (although close to the decision boundary) wrongly.

3.2.2 Reweighing and Cost Sensitive Learning

When reweighing the weights, the deprived classes will have an increased influence on the learning process, and the favored classes a decreased influence. The weights will be adjusted inversely proportional to the class frequencies in the input data [22]:

$$numberOfSamples/(numberOfClasses*classCount).$$

Cost Sensitive Learning (CSL) is similar to reweighing the classes, because both approaches directly change the influence of the classes on the model. Instead of changing the weights, CSL adds a cost function based on the confusion matrix [24, 25], an example is shown in Figure 1.

		ACTUAL	
PREDICTED		Positive class	Negative class
	Positive class	True positive (TP)	False positive (FP)
	Negative class	False negative (FN)	True negative (TN)

Figure 1: A Confusion Matrix [25]

Every classification has a cost bound to it. Let us take the same example as in the related work section: A data set where 80% is labeled positive, and 20% is labeled negative. We could say that in this case, a False Positive (FP) is worse than a False Negative, therefore the cost of a FP is a lot higher than that of a FN. This causes a FP to have a bigger effect on the model than a FN. Similar to reweighing, the direct influence of the data is either amplified or reduced without changing the data itself. Both reweighing and CSL seem suitable for dealing with the unbalanced classes as they do nunerot change the data, but merely change the influence of the data.

3.2.3 Modifying thresholds

A fairly simple but effective approach is modifying the threshold. By changing the threshold for when a sample is labeled positive or negative, the bias in the data can be countered by a bias in the opposite direction [23]. A sample is more likely to be labeled as the favored class when the threshold assumes that they are equally likely to occur (i.e. 0.5 in probabilistic models). Increasing the threshold for the favored class will 'level the playing field'.

3.2.4 sampling

As the name suggests, sampling means taking a sample of the data and using this sample as the training data [25]. There are two forms of sampling: over-sampling, and under-sampling. When over-sampling a data set, we increase the size of the data set by adding samples to it. The simplest way to do this is by adding copied instances of the deprived class, this does not add any new information. Another technique for

adding samples is Synthetic Minority Oversampling Technique (SMOTE). The SMOTE sampler takes a random data point in the feature space of the deprived class, and its k-nearest neighbours (generally $k = 5$). Then, it randomly picks one of these k samples, and randomly generates a new data point between these two samples. This way new information is added to the data set.
under-sampling on the other hand removes instances of the favored class. This is a good approach when the data set is large and it is advantageous to decrease the sample space. A disadvantage however, is that information is lost. With a small data set, over-sampling is likely to be more beneficial if it does not increase the learning time too much. Under-sampling would only be a good option if the remaining examples are enough to create an accurate model.

3.3 Learning Approaches

We will use supervised machine learning approaches, to predict the attributes mentioned in the problem statement. This section will go over all of them and explain the theory behind them.

3.3.1 logistic regression

Linear classifiers are generally not that mathematically complex, and can be explained in a few steps [10]. A logistic model generates a signal $S(x)$ that can be defined as

$$h(x) = w^T\mathbf{x},$$

where w^T is the transposed weight vector and $\mathbf{x}$ the input vector. Secondly, this signal is used as input in a logistic function that restricts the output to a probability range $[0,1]$

$$h(x) = \theta(w^T\mathbf{x})$$
$$\text{with } \theta(s) = \frac{e^s}{(1+e^s)}.$$

Other soft threshold functions can also be used. The higher the predicted probability of a class, the more likely that the label is to be classified as that class. A more formal definition of how we interpret the output can be written down as

$$P(y|x) = \begin{cases} h(x) & \text{for } y = +1 \\ 1 - h(x) & \text{for } y = -1 \end{cases}$$

Now that we have a probability for each label, we need to formalize an error measure (or loss function), so that we can update the weights accordingly. For logistic regression we want to maximize the probability of classifying the labels correctly: $\prod_{n-1}^{N} P(y_n|x_n)$. We can do this with the method of maximum likelihood, creating the following function that will increase when the probability is lower, making it suitable as an error measure [10]. This error measure is defined as

$$E_{in}(w) = -\frac{1}{N}\sum_{n=1}^{N} \ln(1 + x^{-y_n w^T X_n})$$

Note that we can also use other functions as error measures such as the binary cross entropy loss that will be explained in section 4.3.3, but for now we will take this error measure as en example. To minimize the error we need to solve the gradient vector of the weights: $\nabla E_{in}(w) = 0$. Because this is not feasible to do analytically, we need some algorithm to solve it, or rather approximate it. Numerous algorithms have been developed for this purpose which almost all built upon Gradient Descent. After every iteration, i.e. after classifying a data point, the weights are updated using an update rule

$$W_{new} = w(0) + \eta\hat{v}$$

here $w(0)$ is the current weight vector, η the step size and $\hat{v}$ a unit vector such that

$$\hat{v} = -\frac{\nabla E_{in}(w(0))}{||\nabla E_{in}(w(0))||}$$

The unit vector $\hat{v}$ can be calculated analytically and holds the gradients for all the weights and thus determines the direction in which the weights should change. The step size η determines how fast the algorithm learns and must be chosen with care. If η is to small, it might take too long before the algorithm finds a local minima, if it is to large the minima might not be found at all. A step size in the middle, or one that starts large but decreases over time works best.
A 'solver' is a combination of error functions and weight update algorithms, different solvers have different approaches for optimizing $\nabla E_{in}(w) = 0$ although the general concept remains the same.

3.3.2 Support Vector Machine (SVM)

This is another linear classification algorithm that has proven itself to be effective for many problems [11]. Like other linear classifiers, SVM separates data with a hyper plane by adjusting its weights. SVM distinguishes itself by using a margin to soften the boundaries by which the data is separated. This way it generalizes over the data, and is more resilient towards data it has not seen before. The hyper-plane that separates the data is $w^T\mathbf{x} - b = 0$, where w^T is the weight vector and $\mathbf{x}$ the input vector. Each data-point either has the label 1 or -1, therefore the initial hyper-planes that will help us separate the data are $w^T\mathbf{x} - b_1 = 1$ and $w^T x - b_2 = -1$. The distance between these two parallel hyper-planes can be calculated using the formula for the standard distance

$$d = \frac{(b_2 - b_1)}{\sqrt{w^+1}}$$

After rewriting this formula, we obtain a problem that we can minimize in order to maximize the distance between the two hyper-planes, also called the margin error

$$E_{margin} = \frac{||w||^2}{2}$$

The problem is that we want to minimize the margin error, but still make accurate predictions. Therefore, we want to minimize margin error subject to

$$P(y|x) = \begin{cases} 1 & w^T\mathbf{x}_i + b >= 1 \\ -1 & y = w^T\mathbf{x}_i + b <= -1 \end{cases}$$
$$=$$
$$\text{y}(\text{w}^T\mathbf{x} + b) >= 1$$

When a data point is labeled 1, the above formula shows that the classification should be bigger or equal to 1, while a data-point that is labeled -1 should be smaller than, or equal to -1. This approach assumes that the data is perfectly separable, also called hard-margin SVM. The approach can be formalized as an optimizaion as problem as follows:

Minimize $\frac{||w||^2}{2}$
subject to: $y(w^T\mathbf{x} + b) >= 1$

Figure 2 explains the mai idea of the SVM approach.

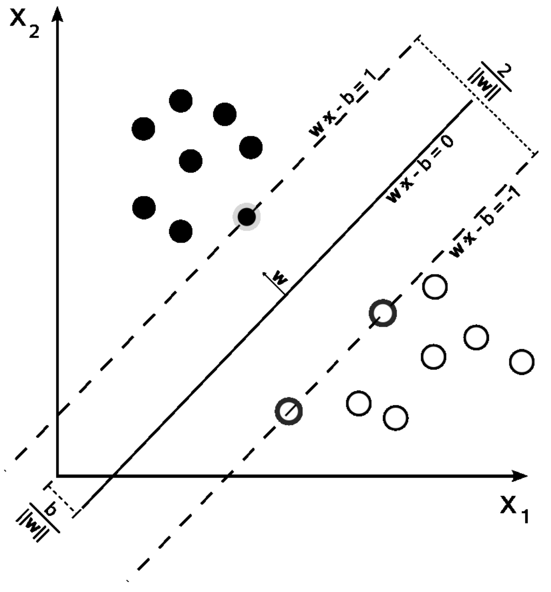

Figure 2: A separating hyperplane in a SVM [35]

In Figure 2, we can see that the data is perfectly separable. With real world data this cannot be the case. Therefore, we add an error to the optimization formula that indicates how much 'slack' is allowed

Minimize $\frac{||w||^2}{2} + e(C)$
subject to: $y(w^T\mathbf{x} + b) + C >= 1$,

where c is a regularization constant that can be set beforehand to increase or decrease the amount of regularization. This is called soft-margin SVM, and it is the default SVM. Another way of adapting the hard-margin SVM so that it can classify non-linearly separable data is to use kernel functions to map the data into a higher dimensional space where the data is linearly separable.

One commonly used loss function is the 'hinge loss':

$$C * \sum_n^i max(0, 1 - y_i(w^T\mathbf{x}_i + b))$$

We can see that when the C is large, the error becomes more important, causing the classifier to focus on the error measure, and therefore creating smaller margins. A small C would cause the classifier to focus on the margins at the cost of the error, creating bigger Margins. The 'squared hinge loss' is also commonly used and is obtained by simply squaring the output of the hinge loss function.

3.3.3 Neural networks

Neural Networks are still used in many problems [12, 21]. They generally have a much higher complexity than regression algorithms, which comes with its pros and cons. The high complexity allows it to capture relations between variables that regression algorithms cannot, yet reaching that point is not trivial. When a Neural Network becomes big, it also tends to take a lot of time to learn from the data before it performs well. How a neural network works and what the topology looks like is presented in Figure 3.

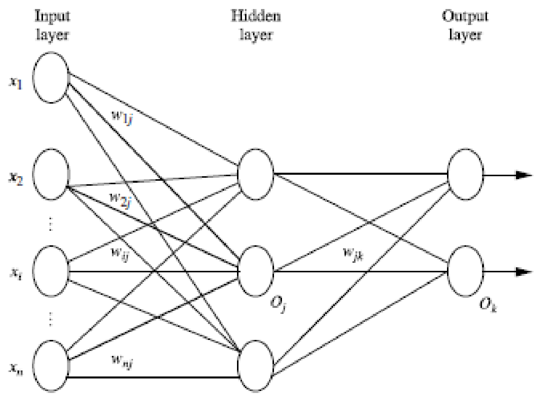

Figure 3: A Neural Network [21]

A Neural Network is a multi-layer linear classifier. In the linear algorithms, we discussed earlier (Logistic regression and SVM), there is one input layer with multiple nodes, and one output layer consisting of only one node. Besides the input and output layer, Neural Networks can include one or more hidden layers, which are layers between the input and the output layer. Neural Networks allow for any number of layers, as well as any number of nodes per layer with exception of the input layer, which should always match the size of the input vector **x**.
As shown in Figure 3, every edge has a corresponding weight, this weight is multiplied with the output value of the previous node. Then, the values of all the incoming edges are summed. The incoming value of a node can be defined as: $h(x) = \sum_{n=1}^N w^T\mathbf{x}$. This value is used as input to a a non-linear activation function for every node. Some examples of activation functions are:
- Relu ($max(0,x)$)
- Softplus ($log(\exp(x)+1)$)
- Sigmoid functions such as the one from section 3.3.1 ($1/(1+\exp(-x))$).

Before we can optimize the network, we need a way to determine the performance of the network on an input tuple. To handle this, we need to define a function that can quantify the difference between the actual and the predicted classes so that the algorithm would be adjusted to achieve better accuracy. A loss function, as we saw before is a function that determines how far apart a set of predictions is from a set of corresponding labels in some sense. We interpret this as how wrong (or right) our classification is. Two popular loss functions are the mean squared error ($MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$), and the binary Cross-Entropy loss. But why do we need to calculate it for a set of classifications? This is the case, because if we would run the backpropagation algorithm for every tuple separately, the processing time would increase greatly. Therefore we take a so called batch, which is a set of tuples from the data for which we calculate the average loss. Then we apply back-propagation with the average loss of the batch. This greatly shrinks the time needed to train a Neural Network while all the data still influences the learning process.
Binary Cross Entropy Loss is a bit more difficult than the mean squared error so it requires some further explanation. We want our loss to be bigger when our classification is less accurate. When the output is a probability, we need the loss to be big when we are far from the correct label, and small when we are very close. Fortunately, the negative logarithmic function enables us to do this. E.g. for a positive label y, the negative log of the probability of our classifier classifying it as positive $-log(p(y))$, will be small if $p(y)$ is big, and big if $p(y)$ is small. This is also called: log loss. To calculate the loss of a set of classifications with size N we use the formula

$$H_p(y) = -\frac{1}{N}\sum_{i=1}^{N} y_i * log(p(y_i)) + log(1-p(y_i))$$

Now that we have our loss, the only thing that is left is to determine how we are going to use it to adjust the weights. Similar to Logistic Regression we need an algorithm to approximate it for us. Again, gradient descent (explained in section 3.3.1) can be used for updating the weights.
Deep Learning is the successor of Neural Networks, and is being widely applied to a variety of problems. Deep Learning Neural Networks are generally much bigger than Neural Networks, and sometimes require super computers, or a multi-node setup to be ran within a reasonable amount of time. Because our data has a lower complexity than the data for which deep learning is generally used, Deep Learning is not considered.

3.3.4 Bayesian Networks

The goal of a Bayesian network, is to model the joint probability distribution of a set of variables [14, 15, 26]. The Bayesian network consists of a Directed Acyclic Graph (DAG), where every node represents a variable from a data set, and every edge represents a dependency between these variables. Every node also has a corresponding Conditional Probability Table (CPT), that is either implemented manually or learned from data. The probability of a variable is then defined as

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i | parents(x_i))$$

The probability for every node is represented in its Conditional Probability Table. The number of rows of this table is bounded by $O(2^k)$, where k is the number of parents. This means that the total number of rows in the Bayesian network is $O(n * 2^k)$, where k is the maximum number of parents for a node. Therefore $O(2^n)$ is the complexity for the full joint distribution. To lower the complexity, it is important to keep the number of parents to a minimum. A simple example of such a Network is shown in Figure 4.

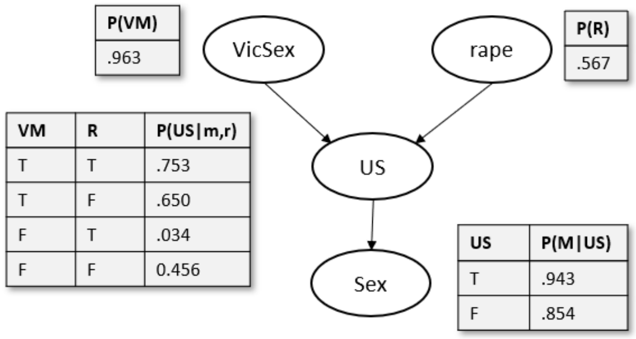

Figure 4: balanced accuracy of the Homicide data

The main benefit of a Bayesian Network is that it can be constructed with domain knowledge, and is thus very interpretable. After creating an initial Graph randomly or with domain knowledge, the graph can be further improved with graph learning algorithms. There are three types of graph learning algorithms: Constraint estimator, score based and hybrid graph learning algorithms.
Constraint based graph learning performs statistical tests between variables in a data set, in order to find variables that are dependant. Then, it creates edges between the corresponding nodes that are found to be dependent of each other.
Score based algorithms are in essence local search algorithms. For every DAG a structure score can be calculated, for which the higher the score, the better the DAG. Generally, Bayesian scoring functions are used for this purpose [26]. The algorithm starts from an initial state with a corresponding DAG, and score. The initial DAG can be random, or created from expert knowledge. Then, the neighboring DAGs are given a score, and the algorithm shifts its state to the neighbouring state that achieved the highest score from all the neighbors, and is higher than or equal to the current score. This process is repeated until a local optima is found, i.e. all the neighbors have a lower score than the current state. Local search algorithms can be greatly improved by adding some randomization to it, e.g. stochastic hill climbing search.
A hybrid algorithm makes use of both of these approaches. First a constraint based algorithm is used to create an initial DAG, this DAG will then be the initial state of the score based algorithm. By using a hybrid algorithm, the search space is restricted to a subset of nodes, in order to reduce the time complexity for the score based algorithm.

After the graph is constructed, each node needs a corresponding Conditional Probability Distribution (CPD). These CPDs are learned by feeding the data to the network and adjusting the CPDs for every new data point.

3.3.5 Classifier ensembles

Classifier ensembles either combine a set of different classifiers, or use multiple variations of the same classifier to create a more accurate classifier [13]. Some widely used forms include: voting, stacking, boosting, bagging, model averaging and forecast combining. Voting, is the simplest of these methods: Given a set of classifiers, each classifier votes for a target class (their prediction is their vote), and the class with the majority of the votes gets chosen.
Bootstrapped aggregating (Bagging), makes use of voting and is widely used, e.g the random forests classifier. Bagging, uses a data set of N samples, and performs N trials, where each sample has a probability of $1/N$ to be drawn, thus creating a data set of length N with randomly drawn samples from the original data set. Then, this data set of N random samples is used to train a model. This entire process is repeated M times, to create M different classifiers. With an unseen sample, the output class would be determined by the class that gets the majority of the votes from these M classifiers.
In the case of random forests, we would have M number of trees, which have all been trained on N random samples from a data set with size N. Bagging decreases the bias, and increases the variance, making it remarkably effective in enhancing the performance of simple classifiers such as decision trees, and creating random forests.
A decision tree itself can be made in two steps (excluding additional contraints). First, a root node must be found by finding the best node to split into two new nodes. This is done by finding the feature with the lowest level of total impurity. The total impurity indicates how well a feature separates the data points, regarding the label that we want to learn. A node separates the data better regarding a label, if it divides the data more unevenly regarding that label. To calculate the total impurity we need to calculate the impurity of the two nodes that a feature would split into (in case of binary classes). For 'gini impurity', the formula for measuring the impurity of a leaf node is

$$1 - P(1)^2 - P(0)^2$$

Other impurities can also be used such as the 'entropy', which is the same as the binary entropy explained before. To calculate the impurity of a feature we take the weighed average of the impurity of the two leaf node:

$$\frac{datapoint_1}{sumOfPoint_{12}} * impurity_1 + \frac{datapoint_2}{sumOfPoint_{12}} * impurity_2$$

The feature with the lowest total impurity will be the root node. The second step is to iteratively construct the rest of the the tree, by finding the feature with the lowest total impurity for each node, until we either have pure leaves, or we have reached some stopping criteria. Generally there are 4 stopping criteria:
1- Max depth: sets a maximum depth for the entire tree.
2- Minimum samples per split: sets the minimum number of samples a node should have before it can be split.
3 - Minimum samples for a leaf: sets the minimum samples a leaf should contain, if a split would cause a leaf to have less samples, the split is not performed.
4- Maximum number of leaf nodes: sets the maximum number of leaf nodes allowed in the entire tree.
Note that when looking for the best split, the number of features that are evaluated can be set beforehand.

Another ensemble which we propose, is one where the set of classifiers consists of the classifiers that we implemented. Three variants will be used. The first is a simple voting ensemble, where the label will be classified as the label with the majority of the votes.
For the second, we take the average probability of all our outputs, and classify the sample accordingly.
For the third we take a combination of the first and second ensemble classifier. We will either take the classification of a single classifier if its predicted probability is higher than a certain threshold, or we take the voting of all classifiers if none of the classifiers reach the threshold. The idea behind it is that when one classifier is really certain of a label, it probably captured a relation that the other classifiers did not. In the case that none of the classifiers are very certain about a label, the best is to let them vote for it.
For the final proposed ensemble we make use of Snorkels labeling model [33], which trains on several labeling models and tries to find the probability of predicting the correct label for each classifier [33, 34]. Then it uses these probabilities to re-weight and combine the outputs of the labeling functions.

3.3.6 Regularization

Regularization is a method to reduce the out of sample error. A lot of algorithms have the tendency to overfit on the data, which causes it to perform poorly on data it has not seen before [10, 11]. For the methods that are regression bases (Logistic regression, SVM and NN), three methods generally used: L_1 regression, aka LASSO regression, L_2 regression, aka RIDGE regression and Elasticnet, which is a combination of both.
All these approaches add an extra term to the loss/error function, creating the augmented error:

$$E_{aug} = E_{in} + \lambda a$$

Here E_{in} is the original error function. λ is a regularization constant, in the range $[0, 1]$, which can be altered to change the degree of regularization. a is the added loss. The difference between the three regularization methods lies in the term a.
The L_1 refers to the L_1 norm of a vector and L_2 to the L_2 norm. This is because a is equal to the L_1 and L_2 norm of the weight vector. Note that, for L_2 it is equal to the L_2 norm without the square root making it much larger than the actual

L_2 norm. For L_1 and L_2, regression the complete terms are therefore:

$$E_{augL_1} = E_{in} + \lambda \sum_{i=1}^{n} |w_i|$$
$$E_{augL_2} = E_{in} + \lambda \sum_{i=1}^{n} w_i^2$$

The main difference between these two forms of regularization, is that L_1 regularization shrinks the least important features, thus removing them completely. This is especially effective when the feature space is large.
Elasticnet combines the two methods, and divides the regularization constant λ into two terms to determine the influence of both forms of regularization:

$$E_{aug} = E_{in} + \lambda_1 \sum_{i=1}^{n} |w_i| + \lambda_2 \sum_{i=1}^{n} w_i^2$$

Here λ_2 can be defined as $1 - \lambda_1$.

4 Framework

The goal of this study is to find the best method to predict a certain feature of the serial killer profile. Our approach can be captured in two stages: The data preprocessing stage and the training stage. The data preprocessing stage includes determining the target columns, removing unwanted columns, transforming columns and imputing the missing values. The goal of this stage, is to obtain a data set which can be used by the training models.
In the training stage several models are implemented and optimized: Logistic regression, SVM, random forests, Neural Networks, Bayesian Networks. These models are individually optimized to obtain the highest possible balanced accuracy (section 5), and then combined into 4 ensembles: Voting Ensemble, probability ensemble, Voting/probability ensemble and a Snorkel Ensemble.

4.1 Preprocessing

First, we determined the percentage of missing values that we want to allow per column. Since there is no concrete theory about estimating such a percentage [7], we chose for an upper limit of 50% missing values, so that the majority of every column would always consist of real data.
After removing some columns we hand picked columns that could be used as labels. After picking the label columns based on the number of missing values and their profiling potential, we removed all the rows for which a label contained a missing value. This is necessary because we cannot generate labels for the data as it could result in an unfair evaluation of the model. For the rest of the columns that contain variables that cannot be known beforehand, but contain too many missing values to serve as label columns are removed as well.
Now that the data set consists solely of labels and training data, the missing values are imputed. One simple method, and one advanced method are used for imputing the data: imputing with a constant value, namely the median value, and k-nearest neighbors for imputation [22]. These approaches are evaluated by testing them against several learning approaches, and then evaluating them based on the balanced accuracy.

4.2 Training

The optimization of the models is not done per variable in order to maintain generality. For every feature in the serial killer profile a separate model is trained. Then, the model is optimized for this set of variables. A problem that arises is that for different variables, different parameters are optimal. Usually some variables requires more generalization and others requires less, this causes a trade off in score between these variables. Therefore, the models could perform better on individual variables, at the cost of the average balanced accuracy. After optimizing all the models, we combine the optimized models in several ways to create better performing classifier ensembles.

4.2.1 The software

For implementing Logistic regression, SVM and random forests, we used scikit-learn [22]. Scikit-Learn is an open source library for implementing machine learning algorithms in Python, and is widely used. It contains a wide range of machine learning algorithms of which the parameters can be modified. The Scikit-learn library also provides a built in method for weighing the input data when it is passed on to a model.
For constructing the Neural Network, we use Keras [29], which provides multiple easy ways to construct a Neural Network, and change its parameters. Keras uses a Tensorflow Backend which makes it very fast when compiled to run on the GPU.
For constructing the Bayesian Network, we used pgmpy [30]. Pgmpy makes it a lot easier to construct a Bayesian Network and includes some useful structure learning algorithms. It also provides easy ways to add edges to a graph, construct conditional probability tables manually and learn a Bayesian Network from data.
For one of the classifier ensembles we use [33]. Snorkel is an open source library for labeling unlabeled data and it can combine labeling functions in a variety of ways. Snorkel has a labeling model (explained in 3.3.5) that is especially suited for this purpose.
Finally, for implementing over-sampling we use imbalanced-learn [28], which focuses on dealing with imbalanced data, and provides multiple under, and over-sampling methods.

4.2.2 Dealing with imbalanced classes

For Logistic regression, SVM and random forests, over-sampling and input for dealing with imbalanced classes are tested against each other. Weighing the data is preferred as it does not increase the time needed to train the model.
For the neural Network and the Bayesian Network, weighing the input data and over-sampling the data will both be considered in order to deal with the imbalanced classes.

4.2.3 Construction and Optimization

After initializing the models, the remaining parameters are optimized based on their effect on the balanced accuracy. For Logistic regression and SVM, the results of all three regularization approaches from 3.3.6 are compared to each other and the best regularization method and constant are chosen. For SVM 'hinge loss' and 'squared hinge loss' are compared as well.
For the random forest, several parameters are evaluated: The number of estimators, aka the number of trees in the forest, The criterion for a split, the maximum depth of the trees in the forest, the minimum samples required for a split and the minimum samples required for a leaf.

Neural Networks need relatively more work than the other approaches. This is mainly due to the non-fixed implementation and freedom in constructing them. For constructing the Neural Network we start from a two layer network with different activation functions, and regularization methods. Then, we gradually extended the number of layers with different shapes (i.e. cylinder, funnel, hourglass, etc). After we manually find a Network that obtains decent predictions, we train it with different batch sizes, and number of Epochs, aka the number of times we feed the data to our Neural network.

For constructing a Bayesian Network the first step is constructing the graph. We try all three approaches for graph learning mentioned in 3.3.4, with an initial random graph. After the graph is constructed the corresponding CPTs will be learned from the data. Creating a Bayesian graph using heuristics and then learning the CPTs generally provides the best results. However, we chose not to heuristically construct a Bayesian Network because it requires expert knowledge and a lot of time to construct.

4.2.4 Combining the models

The final step is combining the optimized models that made decent predictions into new ensemble models. The suggested ensemble models are: Voting Ensemble, probability ensemble, Probability Voting Ensemble and a Snorkel Ensemble. The separate models are already optimized, meaning that for the Ensembles we only need to combine them.

5 Evaluation and discussion

In this section we will discuss the metrics used for evaluating our models, the final models, and the results from these models. The last section of this part contains information about other data sets on which we tested the models.

5.1 Metrics

For evaluation, the main metric is the balanced accuracy. Because of the imbalanced classes, a lot of metrics such as regular accuracy will give a poor representation of the models performance. The balanced accuracy however, takes the average of the percentage of correctly classified positive instances (sensitivity), and the percentage of correctly classified negative instances (specificity). Given that out classes are unbalanced, this metric will give the best representation of success for our models. A formal definition is given below

$$A_{bal} = \frac{1}{2} * \frac{TP}{TP+FN} + \frac{TN}{TN+FP}$$

In addition to the balanced accuracy we will also use the predictive accuracy. The predictive accuracy is simply the percentage of correctly classified instances:

$$A_{pre} = \frac{NRcorrectlyClassified}{NRtotal}$$

Our models have been tuned for maximizing the balanced accuracy. The predictive accuracy in Figure 5, shows the corresponding scores without any change to the parameters. Using both metrics will give a good insight in the overall performance.
To ensure that we did not only test for our in-sample error and that the model has seen enough data, we used k-fold Cross Validation. We took 4 folds where every fold was 25% of the data, then we took 3 of these folds to train on, and 1 to test on. We repeated this process for all the folds, so that at the end all the data has been trained on, and tested on. The scores shown in Figure 5 and 6 show the average balanced accuracy over all the variables.

One interesting observation is that when we do not normalize all the columns the results almost uniformly decreased with some increased differences. Because this reduces interpretability, we chose to normalize that data at the cost of the score.

5.2 The Models

For Logistic regression, SVM and random forrests, weighing the input data as well as over-sampling the data showed a great increase in the balanced accuracy. Because over-sampling has no specific benefit over weighing the input data, weighing the input is a better option since it does not increase the run time. For logistic regression, L_1 regularization achieved a slightly better performance than L_2 regularization. For SVM however, L2 regression showed better results. Both of the models used regularization constant $\lambda = 0.1$, and both did not benefit from Using Elasticnet. SVM performed better with the squared hinge loss.
The random forests classifier performed well with 1000 trees with no maximum depth, 5 minimum samples per split, and 5 minimum samples per leaf. Cross entropy as criterion, worked better than using Gini.

When looking for the best topology for the Neural Network, we found that the best topology remained the first one we used, with just two layers. The activation function of the input layer benefited with softplus. For the output layer the sigmoid activation worked the best with L_1 regularization. For sampling the data before passing it to the Neural Network, we used SMOTE. After 10 Epochs the balanced accuracy did not increase. With a batch size of 32, the model was still fast and the results were good. Decreasing the batch size did not obtain better results, yet increased the learning time.

When using structure learning for constructing the DAG of the Bayesian network, hybrid learning obtained the highest score, yet it still performed poorly. This is most likely caused by the great complexity due to the large feature space. The Bayesian Network needs to be constructed manually with care if it is to perform well on a feature space of this magnitude. Because Hybrid Learning raised some additional issues, we chose to create a linear shaped Bayesian Network which also turned out to perform better. For the Bayesian Network we were forced to use a basic random-sampler instead of SMOTE, as Bayesian Networks do not scale well when the input data contains a lot of categories.

When selecting models for the Ensemble methods we took all of the models except for the Bayesian model which performed poorly. The optimized models are not altered before combining them into Ensembles.

5.3 Results

For every model, the average score over the variables is calculated. This section contains the results of all our models. It also explains the results per metric, i.e. balanced accuracy and predictive accuracy.

5.3.1 balanced accuracy

The balanced accuracy of the models are shown in Figures 5 and 6. Because the 'Race' variable contains 6 classes while the other features are all binary, we chose to evaluate that variable separately (Figure 6).
The baseline of the balanced accuracy for the features excluding 'Race' is 0.5. The baseline for the 'Race' variable however, is 0.167. We can see that there is not much fluctuation in the balanced accuracy between the models without the 'Race' variable. This is an indication that we have reached some kind of maximum for the balanced accuracy.
The error bars in Figure 5 show the standard deviation of predictions for the different variables. These bars indicate that some algorithm have more fluctuations in their predictions regarding the different variables. The Random Forrest model for example, predicted over 0.82 on two variable, yet just under 0.55 on two other variables. As mentioned before, this is most likely caused by the different properties of the variables, and therefore different requirements for optimizing them.
The four models with the highest score do not show a lot of difference in their performance compared to the other well performing models (random forest, logistic regression and SVM). They also include some of the most time intensive models that we compared, which should be taken into account. The difference between the Neural Network and the Snorkel ensemble is only 0.05, yet the time consumption is significantly lower. Therefore we can say that the best performing model is the Neural Network.

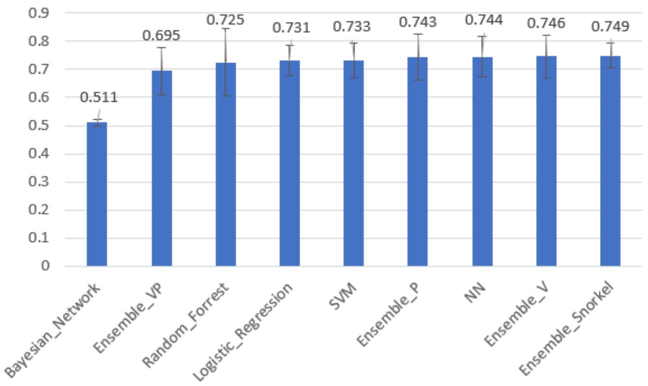

Figure 5: The average balanced accuracy per algorithm with the standard deviation as error bars

For the 'Race' variable the results are remarkable given the low baseline for the balanced accuracy (0.167). The Neural Network however, performed poorly. This is most likely due to the fact that Neural Networks are to be structured differently when the classification problem is multinomial. Our Neural Network only has one output node, while with multinomial classification every class should have its own output node. We did not implement this to maintain generality over all of the variables. The Voting probability ensemble also performed way below average. This is most likely because the threshold is not suited for multinomial classification. When there are several classes, the probability per class will be lower than when there are only two classes. When testing the model with a threshold set for multinomial classification it performed a lot better. We did not implement this because it is out of the scope of out research. The 4 models with the highest scores again include some of the most time intensive models. The difference in score between the best performing model (probability ensemble) and the second best performing models (random forest), is only 0.06. There is however a significant time increase when choosing for the random forest, which is why we consider this to be the best model for predicting the 'Race' variable.

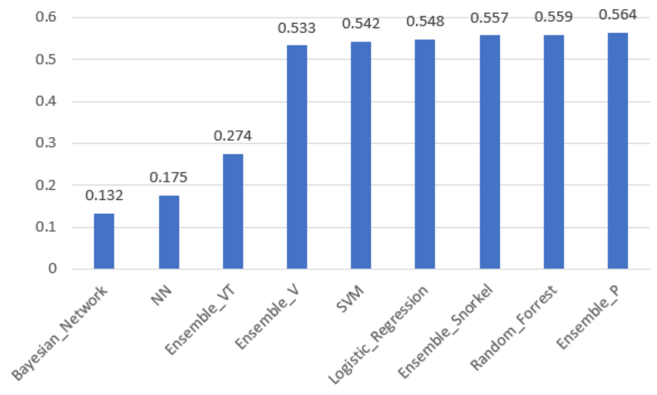

Figure 6: The balanced accuracy of the race variable

Overall most of the Ensembles obtained better scores than the rest of the classifiers. It is questionable if the best performing ensembles can be considered better, due to their increase time consumption. The increase score does however indicate their potential.
For future work it would be worthwhile to optimize the models used in the ensembles for specific variables, instead of optimizing them for the average score of the set of variables. This could be a great benefit if the ensembles would manage to utilize the models with respect to their strengths.

5.3.2 predictive accuracy

The predictive accuracy shown in Figure 7, shows higher scores than the balanced accuracy. This is no surprise when taking into account that the baseline for the predictive accuracy is 0.84 without the 'Race' variable. However, these scores are not optimized for the predictive accuracy, making this an indication that we generalized well in our models, since we are not far above, or below the baseline. Because the baseline is so high, a high score will not say much about how well the model performs. A low score on the other hand, is a good indicator that the behavior of a model is naive or random, which we can see is the case for the Bayesian network.

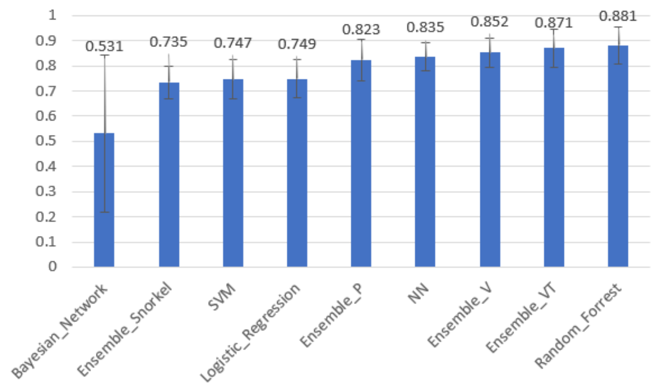

Figure 7: The average accuracy per algorithm with the standard deviation as error bars

For the Race variable, the baseline of the predictive accuracy is 0.3 which makes this metric more valuable. In Figure 7, we see that the results are far above this baseline, which shows that our models perform well on this variable. However, the predictive accuracy is still a lot higher than the balanced accuracy which is because some instances of the deprived class are still being neglected.
The random forest classifier obtained the highest predictive accuracy with a difference of 0.07 between the probability ensemble. Earlier we stated that the random forest classifier was better than the probability ensemble because of the decreased time consumption. Because random forest obtained the highest predictive accuracy this claim is fortified.

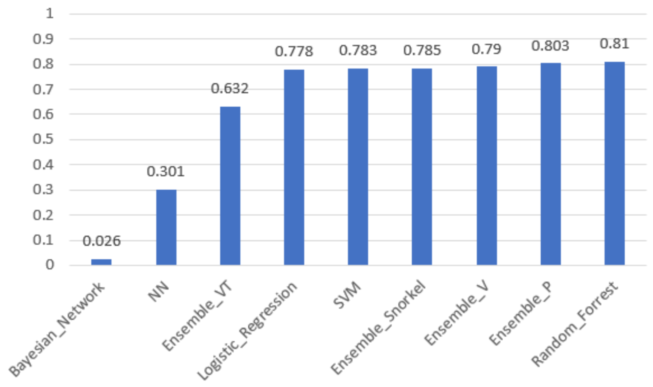

Figure 8: The predictive accuracy for the Race variable

We cannot deduct much when looking at the predictive accuracy of the set of variables without the 'Race' variable. For the 'Race' variable which is less imbalanced, random forest obtained the best score. Random forest only performed slightly better than the probability ensemble when looking at the predictive accuracy, and slightly worse when looking at the balanced accuracy. Because random forest is a lot faster, we again consider it to be the best classifier.

5.4 Other Datasets

For the purpose of generalization we also downloaded the following data sets:

1- Homicide data collected by the Murder Accountability Project [20], and used by Thomas Hargrove in his Murder Cluster Algorithm. This data was already processed a great deal. After processing, the data contained 44 columns and roughly 536 thousand rows, each representing a reported homicide. We only used a subset of 5000 sample because of the amount of time to train all the models.
As the label columns, we used the sex and race of the offender: 'OffSex' and 'OffRace'.

2- National crime victimization survey estimation of rape and domestic violence [31]. This data set is a collection of crime surveys from 1986 until 1990, regarding rape and domestic violence. After processing, the data contains 24 columns, and roughly over 665 thousand rows, each

representing an interview. Again we used a sub set of 5000 samples.
The label to predict is if there was any rape reported, in this case variable '45'. This column was even more imbalanced than our main data set, making it a good test set to evaluate how well we deal with the imbalance. Besides this property of the data, we are trying to predict a personal feature, which is strongly related to our problem statement, i.e. predicting serial killer features.

3- Basketball data from the open API of the NBA, processed and used in [32]. This data was retrieved form the NBAs open API and used to predict if a team would win based on the team composition. The data was already processed completely when we obtained it with special thanks to the author of [32]. It contained 1734 columns and 6567 rows, where each row represented a match between two teams. The label to predict was 'win'. This data is not related to criminal profiling, but it does provide a decent machine learning problem as it makes for a challenging classification problem.

5.4.1 balanced accuracy

The balanced accuracy for the other data sets are shown in Table 1. For the homicide data the Average baseline of the balanced accuracy is 0.35. The results shows that we are far above this baseline, indicating that we dealt well with the imbalanced classes and made accurate predictions.

For the sexual offense data the baseline of the balanced accuracy is 0.5. The results show that all of the models are above this baseline and that the models did not produce random or naive predictions. Even though the high imbalance (0.12% and 99.88%) the models performed well regarding the balanced accuracy.

The NBA data set has a balanced accuracy baseline of 0.5 as well. Our results show that the scores are above this, although not very high above it. Nonetheless, just as with the sex offense data we see that the models are not random nor naive.

Table 1: balanced accuracy of the Other data sets

Algorithm	Homicide	Sex-Off	NBA
NN	0.428	0.787	0.631
ensemble_PV	0.444	0.680	0.620
SVM	0.711	0.864	0.648
ensemble_V	0.711	0.788	0.648
ensemble_S	0.712	0.861	0.648
Logistic	0.713	0.878	0.645
Ensemble_P	0.719	0.666	0.650
random forest	0.727	0.625	0.652

Overall our models obtained acceptable results on these data sets. Again, random forest and the probability ensemble obtained some of the highest scores within reasonable time.

5.4.2 predictive accuracy

The results for the predictive accuracy of the other data sets are shown in Table 2. For the homicide data the average baseline for is 0.6843. The results in Table 2 show that most of the models perform well above the baseline. Take into account that for the 'Race' variable we are predicting 5 classes, which means that the baseline is much lower than the average of the two variables that we are trying to predict. This again caused the Neural Network and the voting threshold ensemble to perform poorly.

For the Sexual Offense data, the baseline is 0.9988. In most cases this makes it extremely hard to predict the deprived class, although the predictive accuracy can still be high. All The models obtained a high predictive and balanced accuracy, indicating that this is not the case for these models.

The baseline for the predictive accuracy of the NBA data is 0.5838. The results show that we are above this baseline, although again not very far above it. Because of the immense feature space, this is in any case a difficult classification problem.

Table 2: predictive accuracy of the Other data sets

Algorithm	Homicide	Sex-Off	NBA
NN	0.260	0.964	0.610
ensemble_PV	0.553	0.987	0.604
ensemble_V	0.736	0.963	0.649
ensemble_S	0.738	0.911	0.647
SVM	0.751	0.914	0.649
Logistic	0.753	0.938	0.644
random forest	0.768	0.999	0.655
ensemble_P	0.772	0.967	0.650

The predictive accuracy also shows that the random forest forest classifier and the probability ensemble obtained some of the the highest scores. Overall we can say that the models performed acceptable on these data sets.

6 Conclusion

To make a serial killer profile we proposed a framework that consists of optimizing single models and then using the optimized models in classifier ensembles. The optimization of the models was an empirical process and can be done analytically in the future. For evaluating the models, we look at the obtained balanced accuracy, and the predictive accuracy of our experiments. Almost all of the models obtained a score over 0.53 for the 'Race' variable, and over 0.72 for the rest of the variables. The balanced accuracy indicates that our models did not produce random or naive predictions, and above

the baseline. We can therefore say that the set of serial killer features is predicted with success.
We also see that the models performed acceptable on the additional data sets. Especially the homicide data is interesting, as the majority of the committed homicides fall within this category. The performance on our homicide data matched the performance on our serial killer data.
Additionally to the achieved scores of these models, the models that performed well do not provide issues with constructing a graph based on domain knowledge, and are resilient towards changes in the data.

References

[1] Rene Chun. (2019, October). *"Modern Life Has Made It Easier for Serial Killers to Thrive"*. The Atlantic.

[2] Joshua New. (2017, July 7). *"5 Q's for Thomas Hargrove, Founder of the Murder Accountability Project"*. Center for data innovation.

[3] Robert J. Morton, Mark A. Hilts. *"serial murder"*. Behavioral Analysis Unit-2, FBI

[4] Douglas, John E, Mark Olshaker. (1995). *"Mindhunter: Inside the FBI's Elite Serial Crime Unit"*. New York: Scribner.

[5] Douglas, John E., Mark Olshaker. (1999). *"The Anatomy of Motive: The FBI's Legendary Mindhunter Explores the Key to Understanding and Catching Violent Criminals"*. New York: Scribner.

[6] Robert K. Ressler, Tom Schachtman. (1992). *"Whoever fights monsters"*. St Martin's press

[7] Stef van Buuren. (2018). *"Flexible Imputation of Missing Data, Second Edition."* Taylor Francis Group, LLC

[8] Stef van Buuren, Karin Groothuis-Oudshoorn. (2011). *"mice: Multivariate Imputation by Chained Equations in R"*. Journal of Statistical Software

[9] Jianglin Huang, Jacky Wai Keung, Federica Sarro, Yan-Fu Li, Y.T. Yu, W.K. Chan, Hongyi Sun. (2017). *"Cross-validation based K nearest neighbor imputation for software quality datasets: An empirical study"*. The Journal of Systems and Software

[10] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, Hsuan-Tien Lin. (2012). *"Learning from Data. A short course"*. AMLBook

[11] Shan Suthaharan. (2016). *"machine learning Models and Algorithms for Big Data Classification"*. Springer, Boston, MA

[12] Yoshua Bengio Ian Goodfellow Aaron Courville.(2015, October 3). MIT Press *"Deep Learning"*.

[13] Nikunj C. Oza, Kagan Tumer. (2007, July 4). *"Classifier ensembles: Select real-world applications"*. Science Direct

[14] K. Baumgartner, S. Ferrari, G. Palermo. (2008). *"Constructing Bayesian networks for criminal profiling from limited data"*. Elsevier

[15] Kelli A. Crews Baumgartner. (2005) *"Bayesian Network Modeling of offender behavior for criminal profiling"*. Graduate School of Duke University.

[16] John W. Brahan, Kai P. Lam, Hilton Chan, William Leung. (1998). *"artificial intelligence crime analysis and management system"*. Elsevier

[17] Chao Huang, Junbo Zhang, Yu Zheng, Nitesh V. Chawla. (2018). *"DeepCrime: Attentive Hierarchical Recurrent Networks for Crime Prediction"*. CIKM

[18] Renjie Liao, Xueyao Wang, Lun Li and Zengchang Qin. *"A Novel Serial Crime Prediction Model Based on Bayesian Learning Theory"*. (2010). Ninth International Conference on machine learning and Cybernetics

[19] Tong Wang, Cynthia Rudin, Daniel Wagner, and Rich Sevieri. (-). *"Learning to Detect Patterns of Crime"*. Massachusetts Institute of Technology

[20] Thomas K. Hargrove. (2019). Murder accountability project. Retrieved from: http://www.murderdata.org/p/var-divelement-document_30.html

[21] Jiawei Han, Micheline Kamber, Jian Pei. (2012). *"Data Mining, Concepts and Techniques"*. Elsevier Inc

[22] scikit-learn. (2020). machine learning in Python. Retrieved from: https://scikit-learn.org/stable/index.html

[23] Faisal Kamiran, Asim Karim, Sicco Verwer, Heike Goudriaan. (-). *"Classifying Socially Sensitive Data Without Discrimination: An Analysis of a Crime Suspect Dataset"*

[24] Faisal Kamiran, Toon Calders. (2011). *"Data preprocessing techniques for classification without discrimination"*. Springerlink.com

[25] Gary M. Weiss, Kate McCarthy, and Bibi Zabar. (2007). *"Cost-Sensitive Learning vs. sampling: Which is Best for Handling Unbalanced Classes with Unequal Error Costs?"*. Fordham University

[26] Richard E. Neapolitan. () *"Learning Bayesian Networks"* Northeastern Illinois University Chicago, Illinois

[27] Prof. Carolina Ruiz. () *"Illustration of the K2 Algorithm for Learning Bayes Net Structures"* Department of Computer Science, WPI

[28] Imbalanced-Learn. (2020). Retrieved from: https://imbalanced-learn.readthedocs.io/en/stable/index.html

[29] Keras. (2020). Retrieved from: https://keras.io/

[30] pgmpy. (2020). Retrieved from: https://pgmpy.org/

[31] Ann L. Coker, Elizabeth A. Stasny. (2000) *"Adjusting the National Crime Victimization Survey's Estimates of Rape and Domestic Violence for 'Gag' Factors, 1986-1990."* ICPSR 6558

[32] Ibrahim EL Garmouhi. (2020) *"NBA Game Predictions based on a Team Composition Model."* Utrecht University

[33] Snorkel. (2020). Retrieved from: https://www.snorkel.org/

[34] Alexander Ratner, Braden Hancock, Jared Dunnmon, Frederic Sala, Shreyash Pandey, Christopher Ré. (2018) *"Training Complex Models with Multi-Task Weak Supervision."* Stanford University

[35] Romaric Pighetti, Denis Pallez, Frederic Precioso. (2015) *"Improving SVM Training Sample Selection Using Multi-Objective Evolutionary Algorithm and LSH"*