An Integrated Credit Scoring Approach Using CART And Bayesian Network Modeling

Flinn Dolman

MSc Computational Finance

University College London

Declaration

Flinn Dolman
September 8, 2018

Abstract

This project explores the theory and implementation of a credit risk scoring system that performs analysis on agents (small businesses) that interact on a decentralised network. The system works by combining two powerful techniques in Machine Learning: Decision Trees and Bayesian Network learning. The project begins with a simple primer on specialist terms and ideas in Machine Learning that are prerequisite knowledge in order to fully appreciate the work. Next, the mathematics behind Decision Tree based methodologies and Bayesian Networks is developed and explored. These techniques are then applied to address the problems faced when building a credit scoring system that fully utilises the benefits of all lenders being network agents. Random Forests and Gradient Boosted Decision Trees are both employed, results are then validated and compared in order to perform a more traditional analysis of credit risk indicators supplied by candidate borrowers. Bayesian Network learning is used to estimate statistical dependencies between agent action and interaction histories and are compared to a set of assumptions about how a bad agent may act. Ultimately, both concepts are tied together to create an integrated system that takes into account both agent trustworthiness (Bayesian Networks) and traditional credit risk indicators.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In order for the financial system to properly function it is of the utmost importance that there exist systems that can accurately quantify risk. One type of risk: Credit Risk poses great threat not just to lenders but as evidenced by the financial crisis to the very structure of global economies. Luckily, recent advance in computational power is allowing for far richer and in depth analysis of data than ever before. This advance, along with the new wave of innovation in computational statistics that it has brought along with it is creating a sort of arbitrage whereby the concepts and technologies that more accurately solve problems such as credit scoring exist, but they have yet to be formally implemented by anyone, researcher or industry specialist alike. In this project we aim to do just that. We will take concepts whose research in most cases dates back into the 20th century and combine them with other technologies and computational resources in the hopes of constructing a flexible and modern integrated approach.

1.1 This Project

The purpose of this project is to design and implement a credit scoring system. This credit scoring system will act as the credit scoring component of a decentralised lending platform that allows lenders and borrowers to interact in a verifiable, secure and trustless capacity. We go into more details below.

1.1.1 Credit Network

For the purpose of understanding this project, the reader need only understand a few core details about the proposed 'Credit Network'. The Credit Network is a decentralised financial services marketplace on the Blockchain designed to solve the trust problem and close the credit gap faced by many small businesses around the world. The basic idea is that lenders would be far more compelled to provide secured and unsecured loans to a borrower if there existed a verified store of their personal finances as well as their interactions with other businesses. The network facilitates this by providing a platform where potential borrowers can upload and have attested to financial records and PII. This information is cryptographically secured and stored in a 'vault' which exists in a decentralised storage system named IPFS[1]. As a means for potential lenders to interpret the vast amounts of data that borrowers could potentially submit, there will exist a segregated credit scoring service which serves the purpose of interpreting and mapping a score to data that a borrower submits to it. Lenders can then use this score similarly to how they might use a credit score as part of their typical lending workflow. The thing that makes such a scoring system unique, is the ability for it to take into account a borrowers network activity in addition to the more traditional indicators that the borrower may submit (such as PII/past credit history). As a product of the way the network functions network activity data will exist in a verifiable and unalterable fashion which creates interesting opportunities for a more in depth analysis of trustworthiness as a consideration in granting a loan. We will come back to how this is possible later on in 4.3.

1.1.2 How do we solve this

Traditional credit scoring typically opts for models such as Artificial Neural Networks, Logistic Regression[2]. Let us consider each individually.

1.1.2.1 Artificial Neural Networks

A lot can be said about Artificial Neural Networks (typically referred to as just Neural Networks). For a more detailed overview I encourage the reader to look at [3]. From our perspective, Neural Networks are a class of computational models which aim to learn a functional mapping of input to output. They represent such a function as a set of weights which are iteratively updated over time according to a set of training data D. Whilst these models were conceptualised back in the 1940's[4], modern advances in computing has meant they have become almost synonymous with the field of AI and at the time of writing their use for solving almost any AI driven problem is ubiquitous.

1.1.2.2 Logistic Regression

Logistic regression is most intuitively imagined as a generalisation of Linear Regression to classification. Regression schemes work by determining a functional representation that best represents the data D according to a set of weights much like a Neural Network as above. The difference is that in a Neural Network the overall functional representation is defined by a nested structure of other functional representations (according to a graphical model, i.e. the network). Regression, at its root is a process for determining weights for a simple functional representation and thus if one wanted to, weights in a Neural Network could be updated with Regression. Based on this definition of Regression, Logistic regression is just an extra step whereby the output of the functional representation found according to Regression is 'squashed' into an interval of $[0, 1]$ through application of the Sigmoid function[5].

1.1.2.3 The drawbacks of these models

Whilst these models are widely implemented, I feel as if it may be time to explore other options. Neural Networks, whilst powerful predictive tools suffer from being for the most part a black box. Their nested structure coupled with the vast amounts of data they must consume make it difficult for a researcher to give semantics for why their trained network is making a certain decision. In the context of credit scoring I feel as if it would be far more helpful for every party involved if it is easily interpretable as to why a certain decision has been made. Logistic regression on the other hand doesn't suffer to the same extent from this problem. This is at the cost of effectively shedding off a lot of the value that a more complex model can provide in terms of accuracy and predictive power. In an ideal world we would construct a model that provides a relatively simple and interpretable interface but also has high predictive power. This leads us on to the next section which details the approach I have opted for.

1.2 Our model

I have opted to design an integrated approach that combines two concepts in machine learning. Decision Tree based classifiers and Bayesian Networks.

1.2.1 Structure

For now it is not important the reader understands all details as to how these approaches work from a technical perspective.

The model is two legged. The first leg is a more traditional credit scoring model in the form of a Decision Tree based classifier. This Decision Tree based classifier will be trained on a more traditional credit scoring dataset and will provide the backbone of estimation. The second leg involves an analysis of conditional dependencies in network activity as inferred by means of a Bayesian Network. In our case the covariates considered will be agent action histories. Given some predefined understanding of what constitutes a good actor and what constitutes a bad actor we will then be able to obtain a probabilistic estimation as to how an agent will act. This will be used as an estimation of trustworthiness and be considered alongside the results of the Decision Tree model in a weighted fashion as to give a more informed credit score estimation.

1.2.2 Previous work

As individual concepts both Decision Tree based classifiers and Bayesian Networks have been used to model credit risk historically.

1.2.2.1 Decision Tree based classifiers

Notable work using Decision Tree based classifiers to determine credit risk includes [6] and [7]. The former differs from our analysis as It focuses on consumer credit risk modeling. Additionally the paper uses ROC AUC (2.1.3.1) as its sole performance metric whereas we will opt to also consider Recall (2.1.2.1). Similarly, the latter also corresponds to analysis of consumer based credit risk but does so in a way that explores the consumer credit risk paradigm and common practices themselves rather than only being concerned with arriving at results. The analysis portion of this paper focuses on credit risk data obtained via a bank as opposed to our work which will rely upon open source data provided by LendingClub[24].

1.2.2.2 Bayesian Network

Prior notable work relating to this project includes [8] and [9]. The former, whilst sharing a domain with our research (credit risk) differs because our use of this methodology will revolve around using this technique to model statistical dependencies between agents on a network rather than modeling dependencies between traditional indicators of credit risk. The latter is similar to our work in the sense that it tries to determine the relationships between various components in a network but differs because it is unrelated to credit risk.

1.2.2.3 Differentiation

The differences already discussed above are all relevant but neglect the main point of divergence between this project and research done before it. We are aiming to combine **both** Decision Trees and Bayesian Networks to create an integrated approach custom built to fit the specific demands of the unique network topology we are faced that is capable of leveraging the power provided by a‘’traditional’‘ credit scoring methodology such as a Decision Tree based model. As such, we are presenting an entirely unique and original solution to a problem that exists as a result of the inception of the Credit Network.

1.2.3 The following chapters

The next chapter will detail all prerequisite knowledge required in order to understand the terminology and ideas in this project (2). We will then explore the mathematics and intuition behind Decision Trees (3) followed by Bayesian Networks (4). After that these methodologies will be utilised to create an integrated credit scoring system (5). This system will act as a POC for credit scoring in the Credit Network.

Chapter 2

Learning framework

In this chapter we consider some tools and concepts that will aid in interpretation of the model analysis we will perform later on. A reader familiar with fundamental concepts in machine learning should feel free to skip this section or simply come back to it for reference whilst they read. First lets define some frequently reoccurring terms.

2.0.1 Definitions

2.0.1.1 Machine Learning model

Whilst this may be obvious from context, what is meant by a machine learning model is an approach for taking input and determining output. Throughout this document the word ‘model‘ will be used to mean machine learning model unless otherwise specified.

2.0.1.2 Features

Features, also known as independent variables attributes or predictors are simply the columns values corresponding to each value in a dataset. For example, if we have a biometric dataset composed of information about many different patients the features would be the values corresponding to height, weight, eye colour etc.

2.0.1.3 Labels

Labels, also known as dependent variables or classes are the variables in a dataset corresponding to information that we are trying to find out. For example, in the previously described biometric dataset the labels could be whether or not the patients have cancer.

In the case where the set of possible labels has size two (i.e. yes cancer or no cancer) and we are trying to decide which label most accurately describes the features for a given data point we say that we are performing 'binary' classification. A simpler way of representing binary labels is by mapping label values to 0 and 1 in a binary fashion. Non binary classification is a broader topic and generally considered more complex to analyse. It is better known as multi class classification.

2.0.1.4 Classifier

A classifier is the model that performs the classification of the data. A typical setup involves the classifier being given a set of data from which it 'learns' some sort of trend or relationship. Then when a researcher wishes to determine the label of a new data point they simply give the classifier the data point. It will output the label it thinks is most likely.

2.0.1.5 Training and test data

Given the classifier setup above, the training data is the data provided to the classifier in order for it to 'learn'. The test data is typically a small unseen sample of data drawn from the dataset prior to training taken place. The test data then serves the role of testing how well the classifier generalises to predicting previously unseen data.

2.0.1.6 Threshold

Classifiers we will concern ourselves with output probability values corresponding to how likely a class is given the data you inputted. Typically if the value of such a probability for a certain class is greater than 0.5 (50%) you would want the classifier to output that class. Put a different way, your classifier in this case is telling you it thinks there is a chance greater than 50% that the class it is describing corresponds to the data you provided. This interpretation expresses an inbuilt notion of certainty within a classifier. In fact, this is what a threshold is. The threshold value for a classifier is the value that the classifier has to predict a class with greater certainty than in order for the model to output that class value. In the case of what we described above, the threshold value was 0.5.

2.0.1.7 Overfitting and underfitting

Overfitting refers to a classifier whose inferences too closely follow the structure of the training set as opposed to the structure of the more broad set of all data. In other words a classifier overfits if it doesn't generalise well to unseen data. Underfitting refers to a classifier whose inferences do not fully grasp the underlying structure of the data. In other words a classifier underfits when it cannot make accurate predictions.

2.0.1.8 Variance and bias

Variance describes the propensity of a classifiers to be sensitive to small changes in the training set. A classifier that is overfitting has high variance. Bias describes the propensity of a classifier to make erroneous predictions. A classifier that underfits has high bias.

2.0.1.9 Error 'scoring'/loss function

These functions describe how prone a classifier is to making mistakes. Such a metric is important as it provides an easier to optimise definition of what the goal of learning is. The goal we are referring to is minimising the loss function. This is a useful reframing of the learning problem because there exists great schools of literature on function minimisation. One popular example of a loss function is mean squared error (MSE) which is defined as follows

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.1}$$

where $n, Y_i, \hat{Y}_i$ are the number of test points, actual outcomes and the predicted outcomes respectively.

2.0.1.10 Objective function

An objective function is simply a generalisation of the idea of a loss function. As stated before, loss functions are convenient because function minimisation is well studied. That is not to say however that one cannot frame learning differently, for example in terms of a function to be maximised. The term objective function captures this. It is a function that one uses to enact their objective for example minimisation of loss.

2.0.1.11 Regularisation

Regularisation is an attempt to reduce overfitting in a classifier. Loosely, the idea revolves around penalising classifiers that learn complex trends in the training data in an attempt to discourage it from happening. For example, take MSE from before. Regularisation adds a penalty function ρ such that MSE becomes:

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 + \lambda\rho(\hat{Y}_i). \tag{2.2}$$

Where λ is known as the regularisation parameter. In this project we use two forms of $\rho(.)$ known as L_1, L_2. They are defined as

$$\rho(\omega) = \sum_{j=1}^{d}|\omega_j| \tag{2.3}$$

and

$$\rho(\omega) = \sum_{j=1}^{d}\omega_j^2 \tag{2.4}$$

respectively. Here ω_j corresponds to the jth dimension of the weight vector ω in $\hat{Y}_i = \omega^T X$ where X is the input data.

2.1 Evaluative measures

Let us consider some concepts that allow us to easily interpret the effectiveness of the models we create.

2.1.1 Confusion matrix

A confusion matrix is a matrix whose rows correspond to actual values and whose columns correspond to the predicted outcome (0,1 respectively in a binary setting). The actual values in the matrix correspond to the occurence frequencies of the events described. Below is an example.

Prediction outcome

		p	**n**	**total**
actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

The purpose of a confusion matrix is to give an easily readable representation of when a classifier correctly or incorrectly classifies and whether the inference was to class 1 or class 0.

2.1.2 TPR and FPR

TPR stands for true positive rate and FPR stands for false positive rate. These values can be easily identified from the entries in the confusion matrix as they correspond to the x and y positions respectively. They are related to a set of terms that are of interest in data analysis.

2.1.2.1 Sensitivity/Recall

Sensitivity, or as we shall refer to it later on, recall is defined as follows

$$\frac{a}{a+b} \tag{2.5}$$

where a, b are the number of true positives and false negatives respectively.

2.1.2.2 Specificity

Specificity is defined as follows

$$\frac{c}{c+d} \tag{2.6}$$

where c, d are the number of true negatives and false positives respectively.

2.1.2.3 Accuracy

Accuracy is defined as follows

$$\frac{c+a}{N} \tag{2.7}$$

where N is the total number of classifications. a, c are defined as before.

2.1.2.4 TPR and FPR equivalances

The TPR is equivalently known as the sensitivity as defined above. The FPR is equal to $1 - specificity$. Collectively, these terms provide a powerful description of how data is classified by a candidate model.

2.1.3 Reciever Operating Curve

The receiver operating characteristic curve or ROC curve, is simply a plot of the TPR vs FPR for a range of classifier thresholds. The diagram below is an example of this.

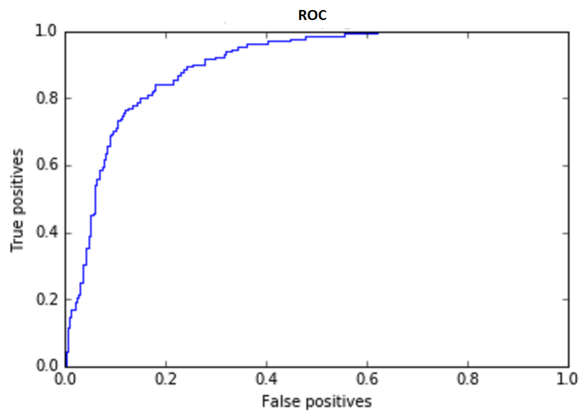


FIGURE 2.1: Example ROC curve

Clearly an ROC curve as described is only helpful when analysing binary classification models. There exist generalisations of the concept to models where the number of classes considered exceeds two. I encourage the interested reader to look at[11].

2.1.3.1 AUC

The diagram demonstrates that a better classifier will correspond to an ROC curve that closely hugs the top left of the graph (this is because doing so implies the best ratio of TPR to FPR). An alternative but equally valid way of expressing this idea is to say that the larger the area under the curve or the AUC as its better known, the better the overall performance of the classifier.

2.2 Validation methods

Another important concept in classification is that of validation. Validating in this context simply means affirming that your model acts in the way you expect it to.

2.2.1 Test set

The most rudimentary form of validation is use of a test set. We already briefly touched on the value a test set provides in the definitions at the beginning of this chapter. Building on what we discussed, a test set allows the researcher to easily identify if their model has succumbed to overfitting (in other words it has high variance).

2.2.2 K fold cross validation

An important idea (in what will become a recurring theme throughout this document) is the idea of repeated calculation of a value followed by averaging. In the case of cross validation we split our whole dataset into a set of K folds. Next, in a set of rounds each fold is treated as a test set whilst the rest of the data is used for training. A loss function is used to calculate the error of the classifier on each test set and these values are averaged producing whats known as a cross validation score. In theory this score provides a more accurate depiction of the classifiers true error than a single calculation of error would.

Chapter 3

Decision Tree Based Methods

In this chapter we begin by explaining what a Decision Tree based model is on a technical level and the techniques that exist for optimising their performance both in a computational and predictive sense.

3.1 CART

CART is an acronym for Classification And Regression Trees. Tree based models work by generating a tree like structure that infer an output by considering how the input compares to a set of rules it creates from some training data. These rules are simply boundaries in the space that contains all of the training data. For example, consider the two dimensional space with training data $\{(0,1),(0,2),(-0.5,1.5),(1,2),(1,3),(1.5,2.5)\}$ and corresponding class labels $\{(red),(red),(red),(blue),(blue),(blue)\}$.

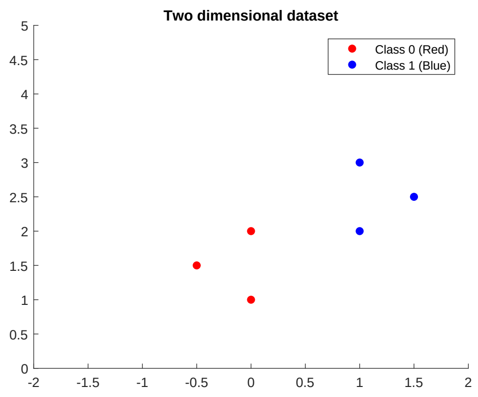


FIGURE 3.1: Example 2D dataset

We could draw a boundary separating these points at $x = 0.5$. Such boundaries are called cuts. Our 'tree' would then be referred to as a decision stump. In other words, it would look at the input and classify according to which side of the boundary the input was. The class on either side is determined from the average of the classes of the training data on either side. For example given our data and the boundary specified above, the input: $(0.3, 2.3)$ would be classified as *red*.

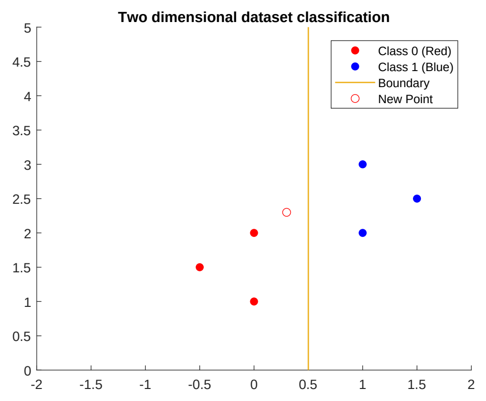


FIGURE 3.2: Example Decision Tree split of 2D data

In the case of higher dimensional training data, more questions are raised about on which dimension to make cuts and in fact how many cuts to make. The first step in solving this problem is to consider how certain tree structures effect an error score (given by an error function). Many such error functions exist and we will detail our selection later. A surface level solution is to then select the tree corresponding to the lowest error score. The issue is that is that it is computationally intractable to compute every cut to determine which Tree performs best. The solution is to take a heuristic Tree generated from a greedy algorithmic approach. The approach typically adopted is known as Recursive Binary Splitting.

3.2 Recursive Binary Splitting

Recursive Binary Splitting (RBS) is a greedy approach to the Tree splitting problem. Its solution is to select cutpoints that minimise the error of the Tree according to its current state. An approach to build a Tree using RBS would go as follows.
Given:

$$R_1(f,c) = \{X|X_f < c\},\ R_2(f,c) = \{X|X_f < c\} \tag{3.1}$$

where R_1, R_2 are the hyper rectangles created by the split. Then select (f, c) that minimises

$$\sum_{i:x_i \in R_1} (y_i - \bar{y_{R_1}})^2 + \sum_{i:x_i \in R_2} (y_i - \bar{y_{R_2}})^2 \tag{3.2}$$

or in words, consider all features f. Select a cutpoint c on the axis of an f that leads to the greatest reduction in Residual sum of squares (RSS) error.

The most obvious drawback to this approach is that a cut resulting in a small reduction in error might be followed by a cut leading to a large reduction in error. RBS would not take this into account thus the Tree generated is not guaranteed to be optimal. RBS makes this trade off in order to be able to work in $O(dm)$ computations where m is the number of samples in the training set and d is the dimensionality of the data (the order of f). This can be calculated by considering that for each possible variable to split on the search for the best cutting point x can be done in $O(m)$ computations. It is then a simple case of performing $O(m)$ operations d times in order to perform the best heuristic split. Thanks to this complexity it is computationally feasible to build Trees for even highly dimensional data.

3.3 Ensemble methods

The real power of Decision Tree based models is that they individually are relatively simple to generate when using a greedy splitting approach such as RBS above. This means that it is computationally viable to generate many such models and make inferences based upon the derived group opinions. There are two prominent schools of thought regarding utilising collective sources of knowledge in this way known as Bagging and Boosting.

3.3.1 Bagging

Bagging, which stands for bootstrap aggregation, is a process whereby collective decisions are made by way of randomly sampling the training dataset many times and creating a Tree for each sample. The decision itself is made by means of equal weighted summing of inferences made by each Tree. More explicitly and given Trees generated with RBS as explained in the previous section we define a Bagging procedure as follows:

1. Take a training dataset and resample it with replacement N times. Sample the same number of points: x for each iteration of resampling so that you end up with N samples of size x.

2. Train N models using each sample
3. Average over the sample according to: $\frac{1}{N}\sum_n \hat{f}^n(x)$ where each function in the sum corresponds to a model that you have trained.

Conceptually, considering a wide range of opinions when making a decision makes sense. Indeed, the process in bagging above of resampling the dataset could be seen as the same as how every voter in real world system is drawing an opinion (the sample) from the collective bank of knowledge (the dataset). Then by considering all of these opinions in tandem you hope that there is some sort of accurate portrayal of the underlining bank of knowledge and the right decision is made. How does this map to mathematics? The concept is also fairly simple. We assume that for n models we generate n normally distributed observations: $T_1,\ T_2, ..., T_n$ each with variance σ^2. If we can reduce the variances of the inferences drawn from such distributions then we will get more accurate results. One way to do so is by using a corollary of the Central Limit Theorem[12].

Corollary 3.1. *The distribution given by the average of observations T_i has variance σ^2/N.*

In other words, by summing inferences of many models trained on random samples of the training dataset we can reduce the variance by a factor N (the number of samples). This sounds great and on paper it is, however it is also idealistic. We are assuming that our Trees produce results independently which is a very strong assumption. Assuming this more realistic scenario let us derive a more accurate estimate for the variance.

Theorem 3.2. *The variance of the average of the sum of n random variables X_i with variance σ^2 and correlation $\rho > 0$ is given by $\sigma^2(\rho + \frac{1-\rho}{n})$.*

Proof. Starting from the definition of variance we have:

$$var(\sum_{i=1}^{n}) = E([\sum_{i=1}^{n}]^2) - [E(\sum_{i=1}^{n} X_i)]^2 \tag{3.3}$$

rewriting each term on the RHS:

$$E([\sum_{i=1}^{n}]^2) = \sum_{i,j}^{n} E(X_iX_j), [E(\sum_{i=1}^{n} X_i)]^2 = \sum_{i,j}^{n} E(X_i)E(X_j) \tag{3.4}$$

and substituting back into the original equation gives:

$$var(\sum_{i=1}^{n}) = \sum_{i,j}^{n}(E(X_iX_j - E(X_i)E(X_j))) = \sum_{i,j}^{n} cov(X_i, X_j). \tag{3.5}$$

Given variance σ^2 and correlation ρ this means we have:

$$var(\sum_{i=1}^{n}) = n\sigma^2 + n(n-1)\sigma^2\rho = n^2\sigma^2\rho + n(1-\rho)\sigma^2. \tag{3.6}$$

finally dividing through by n^2 to compute the average gives

$$\sigma^2\rho + \frac{(1-\rho)\sigma^2}{n} \tag{3.7}$$

as required. □

This proof implies that if we can somehow generated decorrelated Trees then we should be able to reduce our variance and get closer to the idealistic variance value. The most well known approach for doing so is known as random forests[13]. The idea is to make a slight alteration to how the branches are formed (how each Tree is split). Instead of considering how every feature would effect the error when deciding on a cutting point instead a subsample $f = \sqrt{p}$ of possible features is chosen and splitting is done on one of these instead. By randomly selecting the features to do the splitting on at each step covariance between Trees is reduced. This is because if there was a single feature who cutting on always resulted in large reductions of error this feature would be cut upon by many of the Trees, increasing their covariance.

3.3.2 Boosting

Boosting is a general name for a set of algorithms whereby weights which optimally combine a simple set of 'rule of thumb' learners is iteratively determined. Popular types of simple learners for Boosting are Decision Trees (likely stumps for the sake of simplicity) and linear models (Regressors). A particularly effective type of Boosting is Gradient Boosting. Two popular Python libraries for Gradient Boosting are XGBoost[14] and Lightgbm[15]. Popularity of these libraries is in main part thanks to their implementation of efficient parallelisation of the model weight finding process. In the case of XGBoost this is done by using an approach referred to as shotgun coordinate descent[16]. By doing so XGBoost can optimise the model being trained in a quick and effective manner. For our application we are interested training Gradient Boosted Decision Tree models (which we shall refer to as *GBDT*, *GB models* or *GBM*). Let us consider the mathematics behind training a model using this approach.

3.3.2.1 Gradient Boosted Decision Trees

As mentioned before decsions made by models based on Boosting are calculated based on a linear sum of the outputs of a set of classifiers (Decision Trees for GBDT.) Let us take $t_i : X_i \rightarrow y_i$ where X_i, y_i is the set of all feature vectors (or datapoints excluding labels) and the set of their corresponding labels respectively. t_i is simply a function (in our context a Tree) that maps feature vectors to a predicted label.

3.3.2.2 Rounds

Gradient Boosting models are optimised over a set of k iterations or 'rounds' such that the final prediction the model makes is

$$\hat{y}_i^k = \sum_{r=1}^{k} t_r(x_i). \tag{3.8}$$

Given a model of this structure the question of how to optimise/update the model is just a proxy to the question of how to select Trees at each round. One approach is to select Trees according to optimisation of an 'objective' function composed in the following way

$$obj^{(k)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^k) + \sum_{i=1}^{t} \Omega(t_i) \tag{3.9}$$

where the term $\Omega(t_i)$ is a regulariser. The objective function at the kth round can be rewritten in terms of the $k-1$th round

$$obj^{(k)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(k-1)} + t_i(x_i)) + \Omega(t_i) + c \tag{3.10}$$

were c is a constant that can be determined by evaluation of the regulariser for the previous $(k-1)$ rounds. Thanks to the structure of the model and we can use a 2-dimensional Taylor series expansion and simplification to rewrite the equation as

$$obj^{(k)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(k-1)}) + g_i t_k(x_i) + \frac{1}{2} h_i t_k^2(x_i)] + \Omega(t_i) + c \tag{3.11}$$

with g_i, h_i defined as

$$g_i = \delta_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)}) \tag{3.12}$$

$$h_i = \delta^2_{\hat{y_i}^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)}) \tag{3.13}$$

i.e. the first and second order gradients of the loss function with respect to the prediction of the model for the last round. In the case of some loss functions such as MSE these derivatives can be solved for analytically. More generally e.g. loss in logistic classification numerical methods must be used. This is where XGBoosts use of parrallelised optimisation techniques (namely shotgun coordinate descent) comes into play.

3.3.2.3 Splitting

At this point we have defined the objective by which a ‘GBDT‘ model grades its success. We are yet to detail how the model is actually optimised according to this objective however. The approach is principally the same as any other Tree based ensemble model. Firstly $\Omega(t_i)$, the regulariser is precisely defined in terms of L_1 and L_2:

$$\Omega(t_i) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2 \tag{3.14}$$

where ω is the vector of scores for each leaf (terminal node) of the Tree, T is the number of leaves and γ, λ are regularisation parameters. The relative improvement in performance given a certain split in the kth Tree can be obtained by rewriting the objective function in terms of the leaves of the Trees. Explicitly

$$\frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma \tag{3.15}$$

G_L, G_R correspond to the sum of the set of g_i assigned to the left/right hand side of the split boundary respectively and H_L, H_R the same for h_i. In words this expression is the gain in the objective function generated by making the specified split. Accordingly RBS can be applied and a Decision Tree can be constructed.

3.3.3 Boosting vs Bagging

Bagging is associated with a low variance but high bias. On the other hand Boosting is typically associated with a low bias but high variance. I was unable to find conclusive evidence as to which (if either) performed better in general for our domain (credit risk) or any other. As a result we shall test and validate both and ultimately select the one that performs best.

Chapter 4

Bayesian Networks

In this chapter we will discuss Bayesian Networks and their application to data analysis for our application. Let us begin similarly to how CART was introduced in the previous chapter by explaining what a Bayesian Network is. We will then go into detail about how this can be applied to our model.

4.1 Bayesian Statistics

Before delving into what a Bayesian Network is lets first concern ourselves with the more broad field of Bayesian Statistics. The area is almost a philosophy as much as it is a topic in Statistics. The basic idea is that given some data and and some preexisting understanding of how a set of random variables are related one can generate a new understanding of how the variables are related and using this new understanding one can make more accurate statistical inferences. The initial understanding is known as the prior and the newly generated understanding is the posterior. Mathematically speaking we say:

$$p(\theta|x) \propto p(x|\theta)p(\theta). \tag{4.1}$$

Where $p(\theta|x)$ is the posterior, $p(\theta)$ is the prior and $p(x|\theta)$ is the likelihood of data x given label θ. The process of moving from the prior to the posterior is learning in the context of Bayesian statistics and this learning can be iteratively performed. In other words, as new data is found posteriors can be considered as new priors and the model can continually be updated into new more informed posteriors. The process of learning requires an initial assumption of a prior which is the main reason why Bayesian statistics has drawn criticism from some statisticians and philosophers[17]. The argument against the approach essentially boils down to assuming a hypothesis to be true before proving it to be so (assumption of the initial prior) is a contradiction in terms. Nonetheless

Bayesian methodologies have empirically been shown to be effective tools for inference and are widely used. The simplest of such tools for classification is known as the Naive Bayes classifier[18]. This classifier when trained approximates the probability of a class c given a set of features (a feature vector) x. Mathematically this probability is denoted $p(c|x)$. Typically, inferences are then made according to whats known as the *Maximum a Posteriori* (MAP)[19] rule. This is written as

$$\hat{y} = argmax_{c \in C} p(c|x) \tag{4.2}$$

where C is the set of all possible classes.

4.1.1 Derivation

To note where the 'naivety' in Naive Bayes comes from one simply needs to understand how values $p(c|x)$ are determined from the training data. First, from Bayes rule we have that

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)}. \tag{4.3}$$

If we assume inferences are determined according to MAP, note that the denominator of the expression has no impact on the results. Explicitly,

$$p(x) = \sum_{c \in C} p(c)p(x|c) \tag{4.4}$$

since $\sum p(c)p(x|c)$ is equal to

$$\sum_{c \in C} p(c, x). \tag{4.5}$$

This is constant so long as x is known. The fact that it is constant means that it has no impact on finding c as the *argmax* of expression (4.3). Given that we can discard consideration of $p(x)$ the problem now becomes calculating

$$\begin{aligned} & p(x|c)p(c) = p(c, x) = p(x_1, x_2, ..., x_n, c) \\ & = p(x_1|x_2, ..., x_n, c)p(x_2, ..., x_n, c) \\ & = p(x_1|x_2, ..., x_n, c)p(x_2|x_3, ..., x_n, c)p(x_3, .., x_n, c) \\ & = p(x_1|x_2, ..., x_n, c)p(x_2|x_3, ..., x_n, c)p(x_3|x_4, ..., x_n, c)...p(x_{n-1}|x_n, c)p(x_n, c)p(c) \end{aligned} \tag{4.6}$$

where x_i corresponds to the value of the ith feature. At this point the notion of naivety is introduced. In order to simplify the calculation we assume x_j and x_i for $j \neq i$ are conditionally independent given the class c. This allows us to reformulate our expression

$$p(x|c) = \prod p(x_i|c)p(c) \tag{4.7}$$

and finally gives us

$$p(c|x) = \frac{p(c)\prod_{i=1}^{n} p(x_i|c)}{Z}, \tag{4.8}$$

where Z is the evidence, or as we introduced it the constant $p(x)$.

4.1.2 Graphical Representations

We can represent Bayesian classifiers as graphical networks with directed edges. Doing so provides a rich interpretable layout from which it is easy to extrapolate results. This is the basic idea behind Bayesian Networks. In the next section we will provide a more mathematical definition. See below.

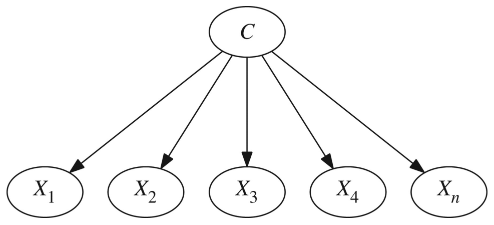


FIGURE 4.1: Graphical Naive Bayes

Note in the diagram above vertices represent input features and edges between them represent conditional probabilities (also known as parameters). In the case of Naive Bayes these edges exist only between the dependent variable and the input features. Mathematically what this means is that all the input features are conditionally independent given the value of the class c. In other words, this is the graphical interpretation of why Naive Bayes is 'Naive'. It is hard to imagine that this is ever the case in practice thus in almost every case of application, useful information is neglected. Luckily this is not an issue that plagues all Bayesian Networks as the Naive Bayes Network is simply a special case. We will now consider a more general class of Bayesian Networks where there also exist edges between input features.

4.2 Bayesian Networks

As touched in the previous section, a Bayesian Network is simply a graphical representation of a Bayes classifier. Formally we define a Bayesian Network as follows.

Definition 4.1. [20] A Bayesian Network is a pair $B = < G, \Theta >$.

The first component G is a directed acyclic graph whose vertices correspond to the random variables $X_1, ..., X_n$, and whose edges represent direct dependencies between the variables.

The second component of the pair θ, represents the set of parameters that quantifies the network. It contains a parameter $\theta_{x_i|\pi_{x_i}} = P_B(x_i|\pi_{x_i})$ for each possible value x_i of X_i, and π_{x_i} of π_{X_i} where π_{X_i} denotes the set of parents of X_i in G.

4.2.1 Learning

The goal of learning in the Bayesian Network setting is twofold.

4.2.1.1 Structure

Firstly, in the graphical sense we want to create a structure that accurately represents dependencies between the covariates (the terms covariates and input features will be used interchangeably from here onward). Accordingly, a job that constitutes part of learning is the process of creating the structure that most accurately represents the conditional dependencies of the training data provided.

4.2.1.2 Parameters

Given a structure the learning procedure will then need to infer the values of the conditional dependencies that are most likely given the training data provided. We will see that for one loss scoring function, the process of determining these parameters simply reduces to determining their conditional frequencies as in the training data.

4.2.2 Learning approach

Learned structures and their corresponding parameters are judged by means of a scoring function. This means that in a practical setting the process of finding the optimal Bayesian Network for a given dataset is equivalent to the procedure of optimising the values of the given scoring function. Traditionally in literature about Bayesian Networks the scoring function adopted is either the 'Bayesian scoring function' or MDL[20] which stands for minimal description length. For our application we will be using the latter on account of the nice mathematical properties it imposes on calculation of the parameters of the network.

Definition 4.2. The MDL scoring function of a network B given a training data set D_t, written $MDL(B|D_t)$, is given by:

$$MDL(B|D_t) = \frac{logN}{2}|B| - LL(B|D_t) \tag{4.9}$$

where $|B|$ is the number of parameters in the network and $LL(B|D_t)$ is the log likelihood of B given D_t:

$$LL(B|D_t) = \sum_{i=1}^{N} log(P_B(x_i)) \tag{4.10}$$

for x_i an element in the training set D_t.

In this setting the optimal Bayesian Network is the one that corresponds to the lowest MDL score. Further, up to the number of parameters in the network we can minimise the MDL by maximising the log likelihood of B given D_t.

Theorem 4.3. $\sum_{i=1}^{N} log(P_B(x_i))$ *is maximised when*

$$\theta_{x_i|\prod_{x_i}} = \hat{P}_{D_t}(x_i|\prod_{X_i}). \tag{4.11}$$

Proof. Using our new notation including $\prod_{X_i}$ we can write

$$P_B(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P_B(X_i|\prod_{X_i}) = \prod_{i=1}^{n} \theta_{X_i|\prod_{X_i}}. \tag{4.12}$$

Under this definition we can rewrite log likelihood as

$$LL(B|D) = \sum_{i=1}^{N} log(\prod_{i=1}^{n} \theta_{X_i|\prod_{X_i}}). \tag{4.13}$$

The meaning behind log likelihood becomes more intuitive if we temporarily convert it to likelihood by means of exponentiation

$$L(B|D) = \prod e^{log(\prod_{i=1}^{n} \theta_{X_i|\prod_{X_i}})} = \prod_{i=1}^{n} \theta^{n}_{X_i|\prod_{X_i}}. \tag{4.14}$$

The ‘likelihood‘ of ‘B‘ given ‘D‘ is just the likelihood of the network structure given the training data. Lets make a further simplification by taking $t = \prod_{i=1}^{n} \theta_{X_i|\prod_{X_i}}$. For observations X this gives us the form

$$L(B|D) = \prod_{X} p(X) \tag{4.15}$$

If these observations repeat with frequency $\hat{P}_D(t)$ then the likelihood is

$$L(B|D) = \prod_X P(t)^{n\hat{P}_D(t)} \tag{4.16}$$

which in log likelihood terms reduces to

$$LL(B|D_t) = n\sum_X \hat{P}_D(t)logP(t) = n\sum_{i=1}^{n}\sum_X \hat{P}_D(x_i, \prod_{x_i})log(\theta_{x_i|\prod_{x_i}}). \tag{4.17}$$

We must now show that this expression is maximised for $\theta_{x_i|\prod_{x_i}}$. We can do so by using Kullback-Leibler divergence[21] and its non negativity property:

$$D_{KL}(\hat{P}_D||\theta_{x_i|\prod_{x_i}}) = -\sum_X \hat{P}_D log\left(\frac{\theta_{x_i|\prod_{x_i}}}{\hat{P}_D}\right) \geq 0 \tag{4.18}$$

thus,

$$\sum_X \hat{P}_D log(\hat{P}_D) \geq \sum_X \hat{P}_D log(\theta_{x_i|\prod_{x_i}}). \tag{4.19}$$

This gives

$$\theta_{x_i|\prod_{x_i}} = \hat{P}_D, \tag{4.20}$$

as required. □

This theorem is very powerful. It implies that if we have two Bayesian Networks, $B =< G, \Theta >$ and $B' =< G, \Theta' >$ if $\Theta = \hat{P}_D(x_i|\prod_{x_i})$ then $LL(B|D) \geq LL(B'|D)$. This means that rather than having to search the space of all possible graphs and parameters, we need only search the space of all graphs and then the optimal parameters can be calculated given the training data.

4.2.3 Practical optimisation

The next important question to answer given our demonstration that the best choice of parameters depends entirely on the choice of network is how to generate the best network structure. The most obvious choice would be exhaustive search. In other words,

1. Calculate the MDL for every combination of covariates (candidate network structure).
2. Select the network structure that maximises the MDL.

It should be evident that this approach becomes infeasible when the number of covariates is large. A more efficient approach (which we shall adopt) relies upon a result from Chow

& Liu (1968)[22] and was implemented in the context of Bayesian Networks in the paper: Bayesian Network Classifiers[20]. I would encourage those who are intrigued to read this paper as explaining in detail how this approach works goes beyond the scope of this project.

4.3 Implementation

In our application Bayesian Networks will be employed to model dependencies between agent actions and interactions in order to gauge an agents trustworthiness. First, a network agents action history is calculated by looking back through a log of their historical actions. This is possible because interactions on the Credit Network are routed by a Smart Contract located on the Ethereum Blockchain[23]. Smart Contracts are simply a set agreements between parties whose execution is managed and enforced by the Blockchain. The underlying design of the Ethereum Blockchain means that any historical activity on a Smart Contract can be recovered at any point in the future. Once an agents action history has been computed, it can be grouped together with the action histories of other agents who have previously been interacted with by the agent in question. The groups of action histories can then be treated as covariates and labeled either 'bad' or 'good' according to a predefined set of rules about what constitutes undesireable behaviour on the network. These covariates and their labels are then used as data to construct a Bayesian Network. From this Bayesian Network, the probability of an agent being bad can be computed.and if it is high she will be considered less trustworthy. For example, if an agent Alice is deemed to have a high probability of performing an action considered bad then she can be assigned a low trustworthiness score.

Chapter 5

Building the Model

Building our model requires completion of two main tasks. The first, simpler task is to train a traditional credit scoring model by selecting between the better performer of Random Forests and Gradient Boosted Decision Trees. The second, more complex task is to determine which network actions constitute a bad actor vs a good actor. Then, along with a Bayesian Network that we construct use that knowledge to determine the probability that an agent will act in such a way. Once we have constructed both of these models we can pass their results as input to a scoring function which will ouput a credit score.

5.1 Traditional

The code written to determine the results of this section will be submitted as a part of this project. It is described in more detail in A. The dataset that I trained these models on is sourced at [24]. It is a large dataset of shape $(887000, 48)$ whose rows correspond to a loan that has been taken out by a lender. However, much of this is redundant information. Firstly, we are interested only in a subset of this data corresponding to loans taken out by individuals on behalf of small businesses. These specific data instances can be identified by a feature in the dataset corresponding to loan purpose ($loan\_purpose$). This alone cut the dataset size to $100,000$ data points. Another consideration is the fact that there is no explicitly stated binary label column corresponding to whether a loan defaulted or not. I have constructed my own by using the column $loan\_status$. This feature is categorical with 6 possible values. Below is a table corresponding to each category and its corresponding binary label.

	Default	Charged off	Late (31 – 120 days)	Late (16 – 30 days)	Fully Paid	Current
Label Assigned	1	1	1	1	0	0

TABLE 5.1: *Binary labels assigned according to loan_status*

loan_status was then removed as its inclusion leaks information about correct labels to the model if included. Additionally, 33 other columns were removed either for the same reason, because the information wasn't relevant/contained more than 90% NaNs or because they contained information that wouldn't be available about an applicant before they actually apply. For example, *last_pymnt_d* corresponds to the date of the last repayment which is not information that a model could know when considering an application. This left 15 features for consideration (16 columns including the labels).

loan_amnt	*term*	*installment*	*emp_length*	*home_ownership*	*annual_inc*	*pymnt_plan*	*dti*	*delinq_2yrs*	*inq_last_6mths*	*mths_since_last_delinq*	*mths_since_last_record*	*open_acc*	*total_acc*	*inq_fi*	*Default_Binary*

TABLE 5.2: *16 Features*

Precise meanings of these features can be sourced in *Data_ descriptions.txt* (B.1.3.2). The next step was to restructure how categorical variables were represented in the dataset. I opted to one hot encode since the total feature count was relatively low and thus doing so was unlikely to effect computational performance significantly. The result was a new dataset of shape $(10109, 21)$ including labels.

5.1.1 Performance metric

Upon inspection of the 'new' dataset there is notable class imbalance. This is an im-

	Non Defaults	Defaults
Proportion (%)	83.6%	16.4%

TABLE 5.3: *Relative class size*

portant observation as it has a significant implication on the metric we use to evaluate performance of our classifier. For example, assuming accuracy as our metric, a model would obtain an 83.6% accuracy on a stratified training set by simply always predicting that no default occured! One solution would be to calculate a score according to class ratio weighted predictions. A more elegant solution however, is to use the AUC metric discussed in chapter 2 instead. Using AUC will give a good overall depiction of the effectiveness of the classifier. Unfortunately, generalisation of the effectiveness in this way can come at the cost of detail and utilisation of domain specific knowledge. For example in our domain, lending, detecting when a loan application is likely to end in a default (class 1 Recall) is more important than minimising the number of non defaults classified incorrectly as defaults (explained by class 0 Precision). Put another way, whilst not ideal, one would rather tell someone without cancer that they have it, than not be able to detect whether a patient has it at all. Accordingly, we consider class 1 Recall as a secondary measure of classifier effectiveness.

5.1.2 Setup/Validation

At this point we can begin assessing performance of models on our data. In order to do so I have run 1000 'trials' in which the dataset is split into 3 random subsets in the ratio 70/15/15, Training/Testing/Validation respectively. The classifier is trained on the training set. The test set is used to construct an elbow plot used to find the threshold value for the classifier that maximises the AUC. Doing so helps combat the impact that class imbalance has on the models overall inference capabilities. The validation set is used to calculate a 3 fold cross validated AUC score and given the threshold value found by analysis of the test set, class 1 precision is calculated via 3 fold cross validation also. Results across the 1000 trials produce a set of values that can be used to form empirical distributions of both AUCs and class 1 Recalls. The distributions generated for each tested model can be compared with each other. Doing so provides a more flexible and informative approach to model comparison than one would get via simple comparison of AUC or class 1 Recall scores. The approach also gives us the ability to assess statistical significance of individual trials and perform other statistical validation methodologies such as variants of the permutation test [26]. Application of such methodologies within the machine learning workflow is rare which is unfortunate as detailed analysis of model effectiveness in certain domains can be highly valuable.

5.1.3 Out of the box classifiers

Figure 5.1 corresponds to the empirical distributions of AUC and class 1 Recall of the validation set over 1000 trials. Distributions were generated using Kernel density estimation[27] with the Normal kernel.

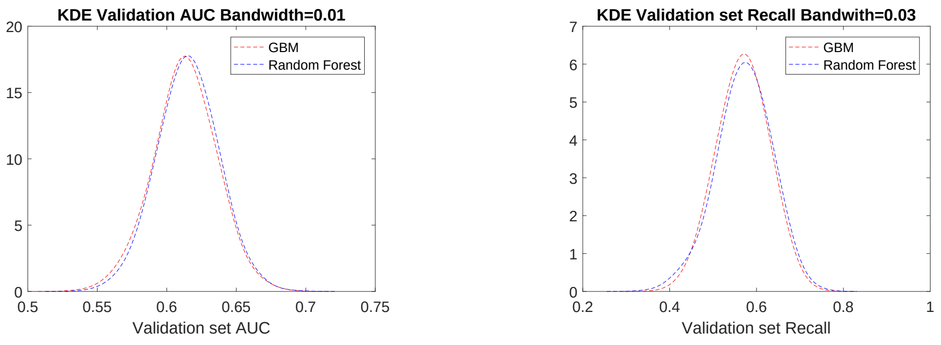


FIGURE 5.1: AUC and Recall for Random Forests and Gradient Boosted Trees

5.1.4 Feature scaling

Feature scaling is a data preprocessing step in which features are scaled so that they exist within the same range as each other. Doing so is particularly desirable for classifiers that are not scale invariant[28]. In cases where the classifier is scale invariant (such as Tree based models) it could be argued that employing the approach is redundant. However, an informal stylised fact of machine learning is that theory doesn't necessarily align with empiricism. Thus Figure 5.2 gives the empirical distributions generated by the models after standardisation using a Normal Quantile transformer[25].

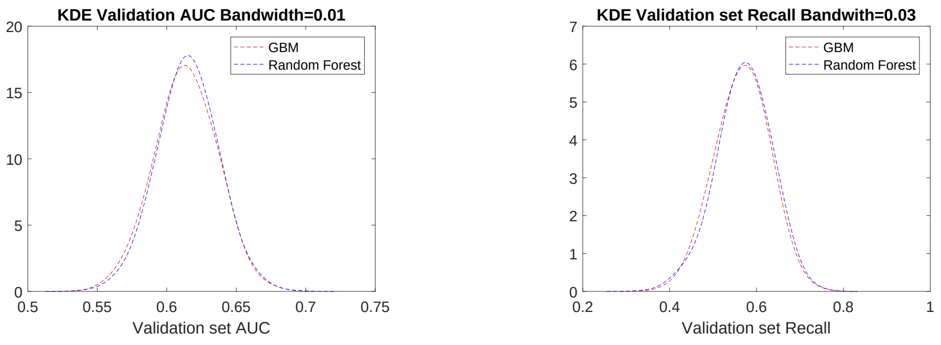


FIGURE 5.2: AUC and Recall for the models when standardised

In this case the results support support the theory of standardisation being redundant. Mean average AUC values for both Random Forests and Gradient Boosting are either higher for the unstandardised data or are negligible (see Table 5.4).

	RF AUC	GBDT AUC	RF Recall	GBDT Recall
Unstandardised	0.6151	0.6131	0.5714	0.5687
Standardised	0.6151	0.6137	0.5713	0.5670

TABLE 5.4: *Standardisation results*

5.1.5 Polynomial features

An important subtopic in machine learning is feature engineering. This discipline can be adequately summed up as "can we construct new features that improve performance of our classifiers?". The general consensus is yes, which is best embodied by the quote of famous data scientist Andrew Ng, "applied machine learning is basically just feature engineering"[29]. One approach to generate new features is to calculate polynomials using the existing features as primitives. Calculating polynomials up to second order using our original 21 features as primitives and adding these as new features produces the distributions in Figure 5.3.

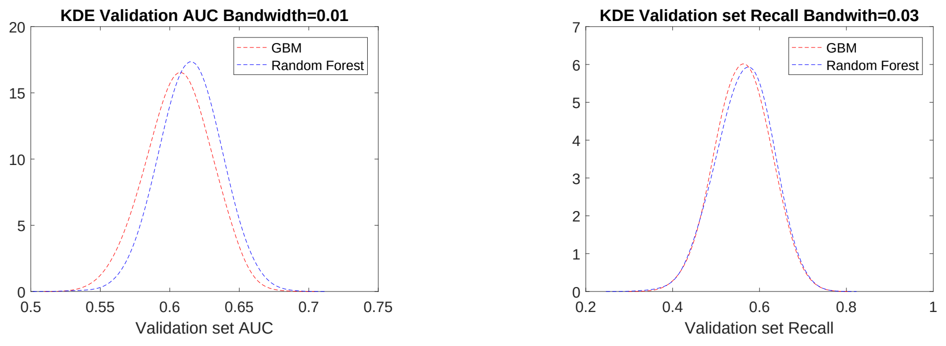


FIGURE 5.3: AUC and Recall with polynomial features

5.1.6 Feature importance and colinearity

Considering second order polynomials as features vastly increases the time taken to train and validate the model. In order to minimise computational overhead one can approximate feature importance using Random Forests and discard all features that do not meet some predefined importance threshold. Figure 5.4 details the most relevant features according to a Random Forests model of depth 3. This approach has no way of determining if the features are highly important for the same reasons. As a result there can still be some redundancy in the selected features even after removing the most irrelevant ones. One approach to circumvent this is to couple feature importance with colinearity analysis. Features are highly positively or negatively colinear when they share a linear relationship with each other. Colinearity scores exist normalised on the interval

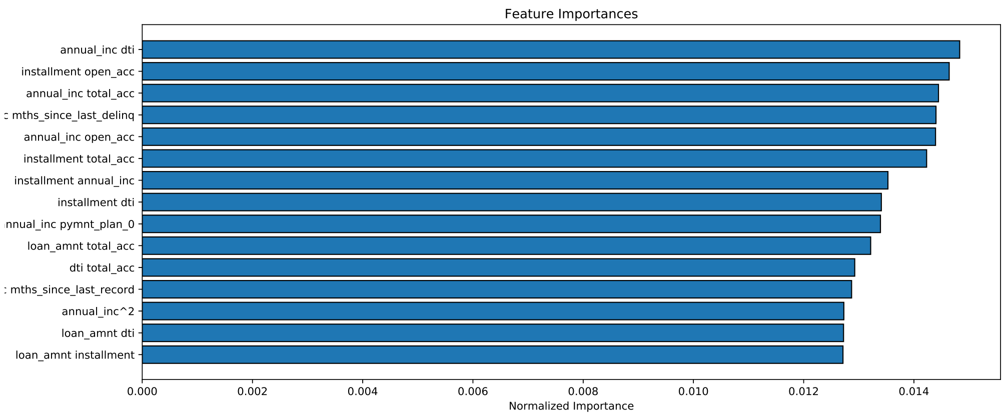


FIGURE 5.4: Feature importances

$[-1, 1]$. In Figure 5.5 we have selected only features in the 10th percentile for importance and less and with colinearity scores, $x < |0.9|$.

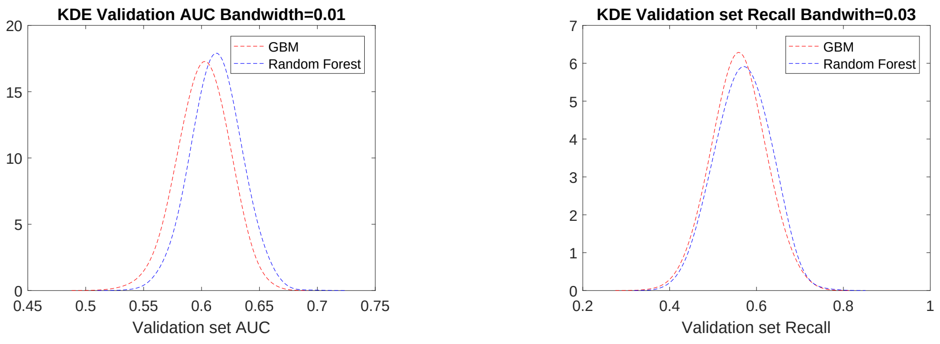
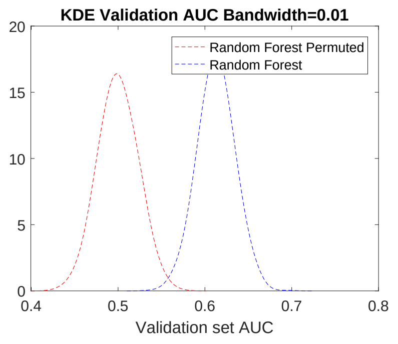


FIGURE 5.5: AUC and Recall without highly colinear features

5.1.6.1 Consolidation

Our final best performing model is Random Forests. To validate our results we generate Figure 5.6. It gives the empirical distributions of the models AUC and class 1 Recall compared to the results found by the model when all features are randomly permuted columnwise (i.e. shuffled such that the model should no longer catch any trends). The mean AUC of the best performing model is 5.56 standard deviations away from the mean of the permuted distribution. Let us make the assumption that the distribution of our model AUCs is Normal. Then one would expect that the mean AUC of the model trained on the randomised data would be observed in the non randomised models population is less than $3.8e - 8$ of the time (5.5 standard deviations away). This is good evidence to suggest that our models results are statistically significant.



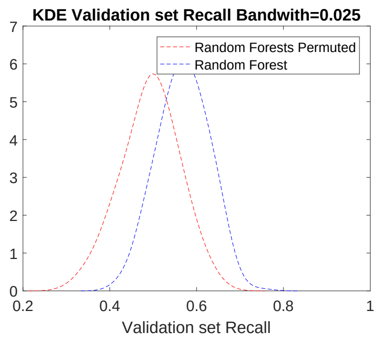


FIGURE 5.6: Permutation tested AUC and Recall for Random Forests

5.1.6.2 Future avenues

As stated before, feature engineering is a vital part of the ML workflow. The models results could potentially be optimised further by construction of more sophisticated features. The prototypical approach to doing so is application of domain specific knowledge by a data scientist who specialises in credit risk. A less conventional but equally interesting possibility is use of the Deep Feature Synthesis algorithm[30] (DFS). This algorithm constructs new features from old ones given a set of primitive operators such as averaging, multiplication etc. By combining DFS along with domain specific knowledge regarding effective primitives and an accurate measure for removal of redundant features many highly relevant features can be constructed with relative ease.

5.2 Bayesian Networks

We begin this section by defining what constitutes a bad actor.

5.2.1 Bad actor axioms

Definition 5.1. A bad actor is a network agent that frequently conforms to one or more of the following

1. Bad actors more frequently request scores than agents acting randomly.
2. Bad actors are more likely to perform less network actions than an agent acting randomly.
3. Bad actors actions are statistically dependent on other bad actors actions (bad actors act in tandem to game the system).

5.2.1.1 Criticisms

Clearly until the network has been run and real agents have been observed it is hard to quantify if these conditions sufficiently encapsulate the way in which a bad or good actor would behave. Additionally, one must be mindful that knowledge about how a bad actor is defined could allow an agent to manipulate their score by gaming the system.

5.2.2 Simulation

Given the above definition I wrote code that plugs into a high level API for the Credit Network and creates a network simulation. An arbitrary number of actors can be generated and assigned a good/bad alignment. Each are then given a random sub sample of the credit dataset we used for our Decision Tree model. Each agents goal is to maximise their credit score which they can ask to be provided with at any point they choose but after which they can no longer act. Bad actors are given the additional constraint that they must minimise their effort. I have provided all of my code for reference in the *code/* directory described in the appendix (A), however the actual Credit Network code is proprietary thus it has been omitted.

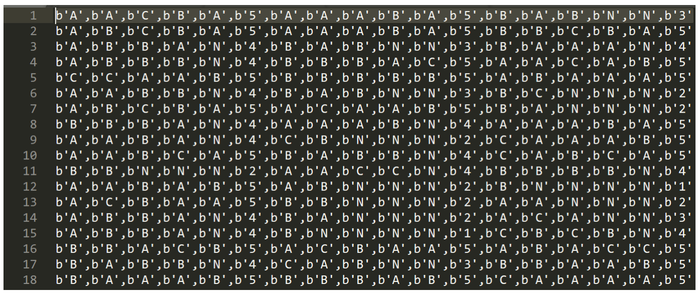
5.2.2.1 Setup

An example set of 100 simulated bad actor actions are provided. The file is named: *badActor.csv*. Similarly a sample set of good actor actions are provided named *goodActor.csv*. More information about these files is available in B.1.1.2. Each row of the dataset represents an actors actions as well as the actions of 2 other actors who we assume their actions are statistically dependent on. In the real world this could be determined by looking at groups of actors who have attested each others data. For the sake of demonstration the number of historical actions considered for each actor is capped at 5. Additionally a number between 1 and 5 representing how many actions each actor in the group has taken is also given. Of course this value could be increased but this adds computational complexity to the process of generating a network. Elements in each row have the following meaning:

1. **A**: Added data
2. **B**: Attested data
3. **C**: Requested credit score
4. **N**: No action taken

5. 1-5: The number of actions taken by the agent

In summary, each row contains 18 elements with the final 6 corresponding to the actor in question and all other multiples of 6 corresponding to the actions of other agents that are assumed to be statistically related.



```
b'A',b'A',b'C',b'B',b'A',b'5',b'A',b'A',b'A',b'B',b'A',b'5',b'B',b'A',b'B',b'N',b'N',b'3'
b'A',b'B',b'C',b'B',b'A',b'5',b'A',b'A',b'A',b'B',b'A',b'5',b'B',b'B',b'C',b'B',b'A',b'5'
b'A',b'B',b'B',b'A',b'N',b'4',b'B',b'A',b'B',b'N',b'N',b'3',b'B',b'A',b'A',b'A',b'N',b'4'
b'A',b'B',b'B',b'B',b'N',b'4',b'B',b'B',b'B',b'A',b'C',b'5',b'A',b'A',b'C',b'A',b'B',b'5'
b'C',b'C',b'A',b'A',b'B',b'5',b'B',b'B',b'B',b'B',b'B',b'5',b'A',b'B',b'A',b'A',b'A',b'5'
b'A',b'A',b'B',b'B',b'N',b'4',b'B',b'A',b'B',b'N',b'N',b'3',b'B',b'C',b'N',b'N',b'N',b'2'
b'A',b'B',b'C',b'B',b'A',b'5',b'A',b'C',b'A',b'A',b'B',b'5',b'B',b'A',b'N',b'N',b'N',b'2'
b'B',b'B',b'B',b'A',b'N',b'4',b'A',b'A',b'A',b'B',b'N',b'4',b'A',b'A',b'A',b'B',b'A',b'5'
b'A',b'A',b'B',b'A',b'N',b'4',b'C',b'B',b'N',b'N',b'N',b'2',b'C',b'A',b'A',b'A',b'B',b'5'
b'A',b'A',b'B',b'C',b'A',b'5',b'B',b'A',b'B',b'B',b'N',b'4',b'C',b'A',b'B',b'C',b'A',b'5'
b'B',b'B',b'N',b'N',b'N',b'2',b'A',b'A',b'C',b'C',b'N',b'4',b'B',b'B',b'B',b'B',b'N',b'4'
b'A',b'A',b'B',b'A',b'B',b'5',b'A',b'B',b'N',b'N',b'N',b'2',b'B',b'N',b'N',b'N',b'N',b'1'
b'A',b'C',b'B',b'A',b'A',b'5',b'B',b'B',b'N',b'N',b'N',b'2',b'A',b'A',b'N',b'N',b'N',b'2'
b'A',b'B',b'B',b'A',b'N',b'4',b'B',b'A',b'N',b'N',b'N',b'2',b'A',b'C',b'A',b'N',b'N',b'3'
b'A',b'B',b'B',b'A',b'N',b'4',b'B',b'N',b'N',b'N',b'N',b'1',b'C',b'B',b'C',b'B',b'N',b'4'
b'B',b'B',b'A',b'C',b'B',b'5',b'A',b'C',b'B',b'A',b'A',b'5',b'A',b'B',b'A',b'C',b'C',b'5'
b'B',b'A',b'B',b'B',b'N',b'4',b'C',b'A',b'B',b'N',b'N',b'3',b'B',b'B',b'A',b'A',b'B',b'5'
b'B',b'A',b'A',b'A',b'B',b'5',b'B',b'B',b'B',b'A',b'B',b'5',b'C',b'A',b'A',b'A',b'A',b'5'
```

FIGURE 5.7: Data format
Note: b's correspond to the data being stored by Python as binary

5.2.3 Network learning/findings

A Bayesian Network is constructed given both the good actor and bad actor data. What we wish to determine from this network is $P(Bad|A_h, I_h)$ as well as $P(Good|A_h, I_h)$ where A_h, I_h correspond to an actor requesting a credit scores action and interaction history respectively. In order to do so we simply add either **'G'** for good or **'B'** for bad to the end of each row in the data accordingly. When an actor submits a request for their credit score to be calculated A_h and I_h are determined. Inferences are then made by the network automatically by calculating the two probabilities $P(Good|A_h, I_h)$ and $P(Bad|A_h, I_h)$ and returning either good or bad dependent on which probability is largest. Because using both A_h and I_h potentially leads to many covariates the chow-liu based approach is used. Testing the predictive power of the network via making inferences on a subsample of the first 100 rows of *badActor.csv* given a network constructed via the last 100 rows of *badActor.csv* and 200 rows of *goodActor.csv* yields an accuracy of 96%. In other words 96 out of 100 bad actors were identified by the Bayesian Network model.

Good	8.607e-09	3.310e-07	7.316e-07	8.607e-08
Bad	0.00111	0.00105	0.00575	0.00415

TABLE 5.5: *Sample probabilities generated by the Bayesian Network from the actor action data.*

5.2.4 Credit scoring function

The only remaining question is how we can concatenate the outputs of the two models above to get a credit score. Given that we cannot confirm that our definition of a bad actor encapsulates all ways in which a bad actor would act in the real world our Bayesian Network component should initially only constitute to overall score minimally. When the network becomes public given some time we will be able to adjust its contribution in line with how accurate it turns out to be. This could be done using a Bayes type approach or Regression. For now we will set a static ratio of 1:4 Bayesian Network to Trees respectively. Additionally we will set the condition that the Bayesian Networks only penalise scores rather than improve them. We do so because the definition of what constitutes a good actor is more open ended than what constitutes a bad one. Scaling to the desired range using our empirical maximum and minimum values and imposing required conditions gives a credit scoring metric of:

$$score = \begin{cases} 5 * RF + 115 * BAYES & \text{if } P(Good|A_h, I_h) > P(Bad|A_h, I_h), \\ 5 * RF & \text{otherwise.} \end{cases} \tag{5.1}$$

$BAYES = P(Good|A_h, I_h) - P(Bad|A_h, I_h)$ and RF is the probability output of the random forests model. This gives a maximum (riskiest) score of 5 given our empirical data.

The average score given by the Tree component of those who do not default is 3.102 on an unseen validation set of size 1264 drawn from our original credit scoring dataset. To decide whether to issue a loan a lender could compare the value output by the entire scoring function to this baseline. Explicitly, as a rule of thumb any loan applicant scoring less than 3.102 can be interpreted as an applicant unlikely to default.

Chapter 6

Conclusion

The end result of what we have discussed throughout this project is a functioning prototype risk model that acts as a proof of concept for credit risk scoring on the Credit Network. Further improvements and beta testing will take place then ultimately what has been developed will be implemented in finality and used to determine issuance of real unsecured loans to small businesses in emerging economies. To my knowledge this is the first time that Bayesian Networks have been used in such a capacity. Previous papers that developed Bayesian Networks for the sake of credit scoring[8] focused on how they could be used as computationally efficient means to compute accurate loan default probabilities rather than any sort of network interaction related to credit risk (which is how we have used them). We finish this report by discussing some drawbacks of the approach that was adopted as well as potential avenues for future exploration.

6.1 Drawbacks

6.1.1 Actor definitions

There are issues with defining what constitutes a bad actor. Clearly until the network has been made public and real agents have been observed it is hard to quantify if the definition sufficiently encapsulates the way in which a bad actor would behave. Additionally, careful consideration must be made for how the effort function is defined and knowledge of how good and bad actors are defined could allow an agent to manipulate their score.

6.1.2 Validation of final credit score

At this time, the value that modeling with Bayesian Networks provides in the real world (as opposed to our simulation) is unclear. We will only be able to accurately determine its effect when the Credit Network is made public and real life network data is available.

6.1.3 Impact

The work conducted as part of this project is impactful in two ways.

6.1.3.1 Academia

This body of work provides a novel basis upon which further academic research can be conducted. Specifically, the work here implicitly proposes an in depth search of integrated approaches to scoring credit based not only on traditional indicators of risk. For example, there is no reason that instead of modeling actions and interactions on the Credit Network one couldn't perform similar modeling on network activity in general such as on a website like Facebook and use the results of this analysis as part of a traditional credit risk analysis in a similar manner to as is done here. Additionally, I believe adoption of a similar scheme to the validation we have performed, specifically, framing evaluation of performance metrics in terms of traditional statistical methodologies (generating empirical distributions of AUC and Recall scores) would help other Machine Learning practitioners validate and interpret their results more effectively.

6.1.3.2 Industry

As mentioned upon completion of the Credit Network this body of work will be used to assign credit scores to real small businesses in emerging economies. This means that in effect this project provides the basis for a system that is intended to be responsible for disbursement of millions of pounds in loans. My ambition is that this project will be an important part of closing or in the least significantly shrinking the global credit gap that arises as a result of small businesses typically having only minimal amounts of data to analyse.

6.2 Going forward

Further work must be done before the full impact of the work that has been done can be felt. This work can be split into three categories

6.2.0.1 Data

As it stands we have used open source data for the traditional credit modeling undertaken in this project. Clearly before this system can be used effectively in production actual network agent data must be captured and a new Tree based model trained accordingly. The Decision Tree based modeling performed in this project as it stands provides only a proof of concept rather than a production ready system.

6.2.0.2 Production network analysis

As touched on earlier, we cannot guarantee that our assumptions about how a bad actor might act on the network are accurate. A more accurate definition of a bad actor can be determined by analysis of Credit Network activity when the system is in production. Another possibility is the use of reinforcement learning[31] or another adaptive approach such as use of genetic algorithms[32] or even Monte Carlo Tree Search[33] to explore and generate potential actions that a bad actor may take given a rudimentary assumption of how an actor may make decisions. For example, we could assume that a bad actor will perform a set of actions A such that $|A|$ is minimal compared to the credit score the actor receives.

6.2.0.3 Further validation and optimisation of results

It is important going forward to continue to make iterative updates to the core model architecture and parameters. For example, the Decision Tree model could be improved further by hyperparameter tuning by means of grid search or Bayesian Optimisation. Similarly, further work could be done feature engineering as in this project we only explored use of polynomial features rather than opting to implement more general approaches (some of which could be automated via use of Deep Feature Synthesis[30]).

6.3 Summary

In this project we have explored how one can develop a credit scoring risk model for small businesses using an approach combining Decision Tree based methodologies and Bayesian Networks. We first defined all prerequisite terms needed for the unfamiliar reader to get to grips with the terminology heavy paradigm of Machine Learning. Next, we explored the mathematical and logical intuition behind Decision Tree based machine learning models. Following that we performed a similar exploration of the mathematics and reasoning behind Bayesian Statistics and how traditionally applied Naive Bayes classifiers can be generalised to perform less naive analysis of dependencies between covariates. At this point we moved from theory into implementation. We explained and justified our evaluation techniques and research methodology for training a Gradient Boosted Decision Trees models as well as Random Forests on an open source loan dataset[24]. We then defined what would constitute a bad actor on the Credit Network and using these assumptions and the network prototype simulated a dataset of network agent actions. This simulated data was then used to construct a Bayesian Network under the Chow-Liu[22] approach and probabilistic inferences were made to demonstrate how this approach will work when real data exists. Finally, using our empirical results we defined a credit scoring function giving a hypothetical maximum score of 5 corresponding to the riskiest possible loan applicant and determined an empirical baseline upon which to decide whether or not a loan application should be granted.

Appendix A

Code

A.1 Overview

The code for this project was written mostly in Python 3.x with a minority being in Matlab. I have aimed to structure my Python code as a rudimentary library, with the core empirical output being determined through a set of 'Experiments' each of which having its own file within the *"experiments"* subdirectory. Matlab code was mainly used to build simple helper scripts and graph plotters since I personally prefer the use of Matlab's inbuilt plotting functionality over that of the popular Python graphing framework MatPlotLib. For instructions on how to run the code see the ***README*** in the root *code/* directory. The following subsections will detail the purpose of each folder within the *code/* directory.

A.1.1 "*code/traditional_ model/src/experiments/*"

I have written my code so that the various experiments performed are segregated into their own files within their corresponding subdirectories of this folder. More details about the precise purpose of these experiments can be found within their corresponding ***README***s within the respective subdirectory. All of these experiments inherit from a base class experiment within the *lib/* directory (A.1.2) that acts in the same way that an abstract class would in other programming languages. Similarly all the preprocessing and model initialisation and other data handling functionality for each is drawn from the*lib* directory as well.

A.1.2 *"code/traditional_ model/src/lib/"*

This folder is a library containing all of the code I have written to assist with creating and performing experiments. The intention was to make the code as generalisable as possible so additional experiments could easily be added. I will provide an overview of the purpose of each subdirectory here. I strongly encourage the reader to view each constituent file's comments in order to gain a more detailed insight.

A.1.2.1 *"data/"*

Each dataset that is worked with has special requirements about how it must be preprocessed. For example, some datasets have columns with many NaN values which must be removed or imputed. I also believed it to be useful functionality if results of experiments were written in such a way that they wouldn't overwrite each other and would be time stamped for later reference. The purpose of this folder is to house files to perform dataset specific operations, create a common interface with the rest of the library and manage writing of results. The file *small_ business_ 16.py* is responsible for performing these dataset specific actions and the files *loader.py* and *results.py* allow for interfacing and writing of results respectively.

A.1.2.2 *"experiments/"*

This folder simply houses the experiment base class.

A.1.2.3 *"models/"*

The training and inferences of Gradient Boosted Decision Trees and Random Forests and all helper functions required to facilitate these tasks are stored within this folder. More details about the specific workings can be found inside comments in the constituent files.

A.1.2.4 *"plotting/"*

This folder contains functions that allow results to be plotted. The code is written to be compatible with our results specific data format.

A.1.2.5 "*preprocessing/*"

Before training the model their are some generic preprocessing steps that must be performed on the data such as one hot encoding categorical labels and potentially standardisation. This folder contains files containing functions that perform these operations on generic data.

A.1.3 "*code/network_analysis/*"

This folder contains code corresponding to the Bayesian Network portion of the project.

A.1.3.1 "*run_network.py*"

This file creates a Bayesian Network from our actor data (B.1.1.2) using the Chow-Liu based approach detailed in[20]. It then tests the network by making predictions on an unseen validation set of bad actors containing 100 entries.

A.1.4 "*code/helpers/*"

This folder contains Matlab code in the form of simple scripts.

A.1.4.1 "*2d_data.m*"

This script plots a simple scatterplot. It was used to generate a graphic to help explain CART.

A.1.4.2 "*2d_data_boundary.m*"

Similarly to *2d_data.m* this code was used to generate a graphic to help explain CART. Its purpose is to show a boundary drawn between the points generated by the first helper script in addition to a new, unseen point and how it would be classified.

A.1.4.3 "*kernel_density.m*"

This script reads a set of result file locations stored in *scores.csv* (B.1.4) and uses them to retrieve the recorded AUC and class 1 Recalls of each model from the *final_results/* directory (B.1.1). With these results it performs Kernel Density Estimation to estimate distributions as well as constructing an approximate cumulative distribution and plotting all generated points.

Appendix B

Data

B.1 Overview

This appendix details the data used as well as generated by the code detailed in A.

B.1.1 "*code/final_results/*"

This directory contains the final results obtained as a result of undertaking this project.

B.1.1.1 "*traditional_model/*"

This folder contains the data generated and analysed as a result of training Random Forest and Gradient Boosted Decision Tree models on the lending dataset (B.1.3.1).

B.1.1.2 "*network_analysis/*"

This folder contains the good actor and bad actor data observed on the Credit Network as seen in Figure 5.7.

B.1.2 "*code/traditional_model/temp_results/*"

This folder contains the time stamped results generated by the code after running an experiment. Data stored in this directory is temporary and thus was regularly emptied to circumvent storage issues. Any desirable results were transferred over to *final_results/* (B.1.1) for permanent storage.

B.1.3 *"code/traditional_ model/data/"*

This folder contains the source lending dataset used to train the Random Forest/Gradient Boosted Decision Tree models in addition to descriptions of the feature meanings and input files used during preprocessing. See the folders ***README*** for more details about each individual file.

B.1.3.1 *"lending-club/small_ businesses_ subset/small_ business_ 16.csv"*

This file contains the data used to train the Random Forests and Gradient Boosted Decision Trees models.

B.1.3.2 *"lending-club/Data_ descriptions.txt"*

This file contains the meanings of the features in B.1.3.1.

B.1.3.3 *"input/"*

This folder contains files that are used to locate the data file (B.1.3.1) as well as aid with its preprocessing.

B.1.4 *"code/helpers/scores.csv"*

This file contains the file directory locations of AUC and class 1 Recall score data files. Its purpose is to make it easy to switch out which data files are being used by A.1.4.3.

Bibliography

[1] Benet, J., 2014. IPFS - content addressed, versioned, P2P file system. arXiv preprint arXiv:1407.3561.

[2] Hand, D. J., & Henley, W. E. (1997). Statistical classification methods in consumer credit scoring. Journal of the Royal Statistical Society Series A. Statistics in Society, 160, 523–541.

[3] Gurney, K., 2014. An introduction to neural networks. CRC press.

[4] McCulloch, W.S. and Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), pp.115-133.

[5] Han, Jun; Morag, Claudio (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning". In Mira, José; Sandoval, Francisco. From Natural to Artificial Neural Computation. pp. 195–201.

[6] Bowen Baker - Consumer Credit Risk Modeling
`https://bowenbaker.github.io/assets/credit-score.pdf`

[7] Khandani, A.E., Kim, A.J. and Lo, A.W., 2010. Consumer credit-risk models via machine-learning algorithms. Journal of Banking & Finance, 34(11), pp.2767-2787.

[8] Leong, C.K., 2016. Credit risk scoring with bayesian network models. Computational Economics, 47(3), pp.423-446.

[9] Pe'er, D., 2005. Bayesian network analysis of signaling networks: a primer. Sci. STKE, 2005(281), pp.pl4-pl4.

[10] Lending Club - Lending Club Loan Data
`https://www.kaggle.com/wendykan/lending-club-loan-data`

[11] Landgrebe, T.C. and Duin, R.P., 2007. Approximating the multiclass ROC by pairwise analysis. Pattern recognition letters, 28(13), pp.1747-1758.

[12] Laplace, P.S., 1820. Théorie analytique des probabilités. Courcier.

[13] Breiman, L., 2001. Random forests. Machine learning, 45(1), pp.5-32.

[14] Chen, T. and Guestrin, C., 2016, August. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794). ACM.

[15] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. and Liu, T.Y., 2017. Lightgbm: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems (pp. 3146-3154).

[16] Bradley, J.K., Kyrola, A., Bickson, D. and Guestrin, C., 2011. Parallel coordinate descent for l1-regularized loss minimization. arXiv preprint arXiv:1105.5379.

[17] Gelman, A., 2008. Objections to Bayesian statistics. Bayesian Analysis, 3(3), pp.445-449.

[18] Hand, D.J. and Yu, K., 2001. Idiot's Bayes—not so stupid after all?. International statistical review, 69(3), pp.385-398.

[19] Chris Piech - Maximum a Posteriori
`https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/pdfs/37%20MaximumAPosteriori.pdf`

[20] Friedman, N., Geiger, D. and Goldszmidt, M., 1997. Bayesian network classifiers. Machine learning, 29(2-3), pp.131-163.

[21] Joyce, J.M., 2011. Kullback-Leibler divergence. In International encyclopedia of statistical science (pp. 720-722). Springer, Berlin, Heidelberg.

[22] Chow, C. and Liu, C., 1968. Approximating discrete probability distributions with dependence trees. IEEE transactions on Information Theory, 14(3), pp.462-467.

[23] Wood, G., 2014. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, 151, pp.1-32.

[24] Lending Club - Lending Club Loan Data
`https://www.kaggle.com/wendykan/lending-club-loan-data`

[25] scikit-learn - Quantile Transformer
`http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.quantile_transform.html`

[26] Joe Felsenstein - Bootstraps, permutation tests, and cross-validation
`http://evolution.gs.washington.edu/gs560/2011/lecture8.pdf`

[27] Zambom, A.Z. and Dias, R., 2012. A review of kernel density estimation with applications to econometrics. arXiv preprint arXiv:1212.2812.

[28] Statistics HowTo - Scale Invariance
`http://www.statisticshowto.com/scale-invariance/`

[29] Towards Data Science - Understanding feature engineering
`https://towardsdatascience.com/understanding-feature-engineering-part-1-continuous-numeric-data-da4e47099a7b`

[30] Kanter, J.M. and Veeramachaneni, K., 2015, October. Deep feature synthesis: Towards automating data science endeavors. In Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on (pp. 1-10). IEEE.

[31] Sutton, R.S. and Barto, A.G., 1998. Introduction to reinforcement learning (Vol. 135). Cambridge: MIT press.

[32] Davis, L., 1991. Handbook of genetic algorithms.

[33] Coulom, R., 2006, May. Efficient selectivity and backup operators in Monte-Carlo tree search. In International conference on computers and games (pp. 72-83). Springer, Berlin, Heidelberg.