UNSW School of Computer Science and Engineering

COMP6441

Security Engineering and Cyber Security

Something Awesome Report

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

Connor Hale (3414917)

Semester 1, 2017

Introduction

This report outlines the design, implementation, testing, and verification of a decision tree based machine learning system for the task of credit card fraud detection through behavioural analysis of cardholders. This project was completed to meet outcomes of the UNSW Computer Science and Engineering Course COMP6441 – Security Engineering and Cyber Security. Specifically, this project was designed to fulfil the 'Something Awesome' component of the course, where students were required to pursue a self-directed project related to cyber security. The aim of this report is to briefly discuss the processes, methodologies, and tools which were used to plan and complete the task itself, and to critically assess to what extend the project's aims were met. Through such analysis, the report also aims to make recommendations regarding how the resultant system may be improved, and lay out plans for proposed future design and development.

Problem Statement

The use of credit and debit cards as means of payment for regular consumers has been a prominent practice in the field of personal finance for many years. Credit and debit cards have become popular due to a number of unique benefits they offer, primarily regarding convenience and ease of use. As well as often offering a payment scheme through which purchases can be made prior to the actual payment of funds, all electronic payment methods offer cardholders the convenience of having to carry less physical cash on their person at any given time. Funds can simply be accessed through a single physical object with a relatively small and lightweight form factor – a card. Clearly this is more convenient than always having to carry bills and coins of multiple denominations at all times, to ensure that both large and small purchases are possible. However, it is important to note that whilst ever the tools we use to access our personal funds take a physical form, the risk of physical theft remains a very real possibility. Additionally, recent history shows that it is also possible for card theft to occur electronically, with multiple instances of this occurring on a large scale in recent years through vulnerabilities in the information security techniques employed by financial institutions to store customers' confidential account details.

With these risks in mind, it is therefore necessary to develop techniques through which credit card fraud can be prevented from occurring, and detected as early as possible to limit the potential financial impacts on rightful card holders, as well as the institutions themselves, who often bear the majority of financial burden when significant instances of fraud occur. One way fraud detection can be achieved is through continual analysis and monitoring of behavioural patterns with regards to spending and transactions. This notion has seen an advent in recent years, with significant technological developments in the field of machine learning and sharp increases in the volumes of data available for analysis prompting discussions on the potential for artificial intelligence to disrupt the financial industry in general, and more specifically allowing for deeper insights into how the behaviours of cardholders can be defined and monitored. The idea at the core of recent developments in this field is that if financial institutions are able to accurately establish notions of 'normal' behaviour for each of their cardholders, as well as common patterns correlated with fraudulent activity, it will be possible to detect when the behaviour of a cardholder becomes 'irregular', and therefore indicative of fraud. The more accurately these patterns can be defined and parameterised, the earlier fraud can be detected on average. This will allow financial institutions to maximise the extent to which they can limit their own exposure to fraud, as well as provide a service to their customers in limiting the potential effects of card theft. For these reasons, the applications of machine learning techniques for the task of behavioural analysis in credit card fraud detection has received significant attention in recent years, and continues to be a fertile ground for developments in artificial intelligence which have applications beyond finance.

Aims

The ideas presented in the previous section provide context for the work which was carried out for this project. Given the rise of machine learning and artificial intelligence techniques in the field of credit card fraud detection, the overarching aim of this project was to investigate the techniques, tools, and methods currently being used in the financial industry for this application. This was to be achieved both through the pursuit of background research, and through a practical, implementation based investigation into a specific subset of the techniques identified. Ultimately, the aims of this project can be distilled into three distinct steps:

1. Investigate the technologies, tools, and methodologies currently being employed by financial institutions for the task of credit card fraud detection through behavioural analysis
2. Choose a subset of these techniques based on the candidate's own technical capacity and time constraints of the project
3. Design, implement, and test a simulated credit card fraud detection system which utilises these techniques to classify incoming transactions as accurately as possible, in terms of their fraudulence

Ultimately, through the completion of this project, the candidate sought to design and implement a software system which simulated the use of machine learning techniques by a bank or similar financial institution to screen new transactions performed by it's customers for fraudulent activity. When a new transaction is performed by one of the bank's customers, the bank should make a judgement on whether or not this transaction is indicative of fraud, and alert relevant parties of fraudulent activity as required. As an additional goal, the candidate aimed to use the completion of this project as an opportunity to learn new technical skills, specifically the C++ programming language. This placed constraints on the implementation techniques used to achieve the project's general aims.

Project Planning and Structure

As the following sections will show, this project constituted a substantial commitment on behalf of the candidate in terms of both the time and resources which were dedicated to it's completion on a regular basis over the course of the semester. Therefore, it was necessary at the project's inception to define a structured planning framework which would be used to more efficiently monitor the candidate's progress, ensure that a finished product would be delivered by the due date, and also provide a division of the project into logical units. This would make it easier for the candidate to manage time more efficiently, and for more logical documentation of the tasks which were completed. As well as this, logically dividing the overall aims of the project into a set of subcomponents would make it easier for external parties to see the specific components which made up the final product. Ultimately, the project was divided into five distinct 'Modules', which would be completed sequentially (with explicit time constraints on the completion of each):

Module 1. Background research and project planning

Module 2. High level software system design

Module 3. Software system implementation

Module 4. Testing and verification

Module 5. Report writing

The proceeding sections will outline each of these modules in greater detail.

Module 1. Background Research and Planning → Blog Post

The aims of this Module were twofold. First and foremost, through the completion of this Module the candidate aimed to gain an understanding of the techniques and methodologies currently being employed in the finance industry for the task of credit card fraud detection through behavioural analysis and monitoring of cardholders. This would primarily be achieved through a brief survey of available literature, and would be focussed on statistical and machine learning technologies, as well as the most commonly used indicators or 'red flags' of fraudulent behaviour which these technologies generally seek to identify. Second, once this literature review was complete, the candidate sought to use this information to choose a suitable subset of these techniques on which to base the rest of the project. The primary drivers of this decision would be the candidates own technical abilities, and the strict time constraints which had been place on the project. It was of utmost importance that the project be completed by the prescribed deadline, and this took precedence over the technical sophistication of the implemented system.

A survey of available literature highlighted the following techniques currently being employed by financial institutions for the task of credit card fraud detection:

- Decision trees – use information theory to construct recursive tree-based data structures which classify transactions by making a series of decisions based on their attributes. Constructed based on training data sets.
- Statistics based algorithms such as genetic algorithms and logistic regression – use a wide variety of statistical methods to classify transactions, usually by analysis of previously received data
- Clustering techniques – Seek to construct groups of accounts whose behavioural patterns are similar. Fraud is then identified when specific accounts begin to behave differently to the others in their clusters
- Neural networks – Currently subject of the most attention in terms of resources and research. Theories behind and descriptions of their specific method of operation are beyond the scope of this report. However, their noteworthy features are their ability to improve unsupervised over time, and the vast volumes of data which are often required to train them. This makes them the most effective candidate, but also the most time consuming to construct and train

The literature review also brought to light the most commonly used indicators or 'red flags' of fraudulent behaviour which these techniques seek to identify:

- Increases in transaction frequency
- Increases in percentage of transactions made online
- Dramatic or obvious changes in location
- Dramatic or obvious changes in transaction volume (often a large purchase following a series of smaller ones)
- Changes in purchasing style (e.g. frequently visited vendor types)
- Dramatic or obvious changes in purchase time
- Multiple large purchases in quick succession succession
- Dramatic or obvious changes in overall behaviour – financial institutions are increasingly utilising the practice of customer profiling, which aims to develop a broad and parameterised overview of customer behaviour. A departure from these parameters would then be a strong suggestion that fraud may be occurring
- Monotonically increasing fraud probability metric – while a single transaction may not be enough of an indicator of fraud on it's own, if the 'probability' that fraud is occurring has continued to increase over a number of transactions, this metric may be an indicator itself

Please refer to the link provided above for more information on the research conducted for this module.

As previously discussed, the second step in this module was to use the above information to decide which technique would form the basis of the rest of the project. Ultimately, a decision tree was chosen for a number of key reasons:

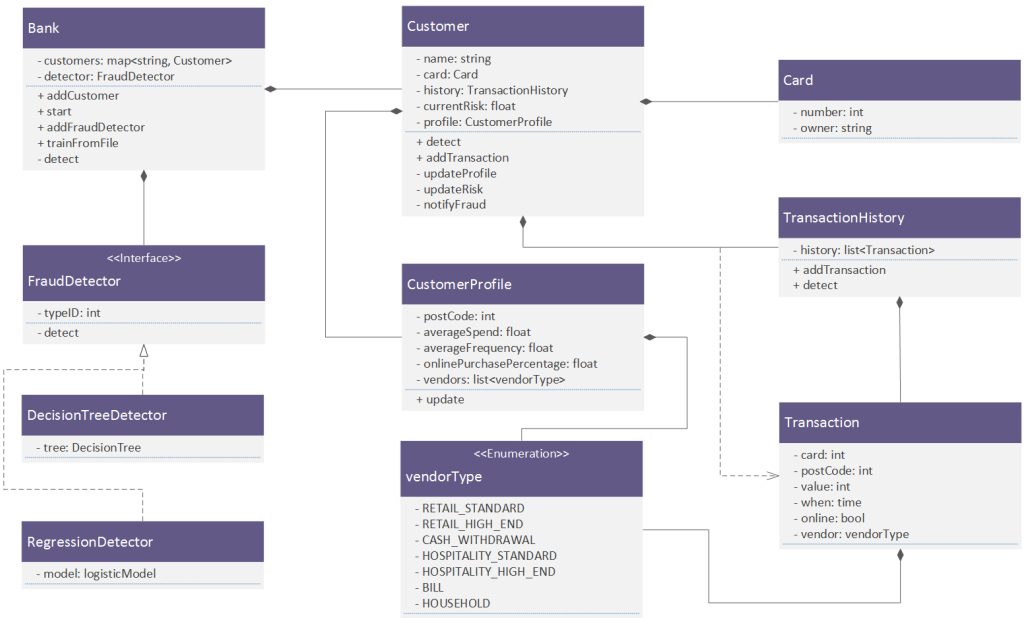
- The candidate had previous practical experience with the design of decision trees for data classification from study completed as part of a previous course. This would make the learning curve required to complete Module 3 far less steep, and would thus be beneficial with respect to the project's strict timeline
- Decision trees generally require a lower volume of training data to start working. While other methods are ultimately more accurate, they require vastly larger amounts of training data to reach peak performance. Again, this was considered favourable when thinking about time constraints
- Decision trees are easier to practically implement from first principals. Other techniques and methodologies would require a more complex implementation phase, and would therefore require a significantly larger time and resource commitment, which was already strained to begin with
- Other techniques would have required considerably more background research to fully grasp before they could be implemented and applied

Essentially, the choice to use a decision tree as the basis of the classification system was motivated primarily by the strict time constraints placed on the project completion. While possibly not as accurate as other methods, a decision tree was a technique which could be designed and implemented in a relatively short period of time, with limited additional background research required. The other techniques which were considered by the candidate were neural networks and logistic regression. As previously mentioned, neural networks require vastly larger volumes of training data to reach acceptable or peak performance levels, and this would have placed a strain on the project's proposed timeline. Similarly, a solution based on logistic regression would have required a significant amount of additional background research, as well as investigations into how such a technique would be practically implemented.

Module 2. High Level System Design → Blog Post

With background research for the project complete, and having made the choice to base the fraud detection and classification system on a decision tree, Module 2 was primarily focussed on conducting a high level design of the software system which would implement both the decision tree, and the overarching banking infrastructure. This took the form of a UML class diagram, a commonly used system design methodology:

Credit Card Fraud Detection System – UML Class Diagram



UML is a design language which is frequently used in industrial contexts for the purpose of object-oriented software design. The technique allows the developer to build a visual representation of the classes which constitute the target product, and the relationships which exist between each of them. The construction of this UML class diagram was motivated not only to act as a design aid to myself to ensure that I continued to follow sound design practices for the duration of the project, but also as a presentation tool to give others a visual representation of the components which make up the system which was built. It was also hoped that through the process of constructing the UML diagram itself, the candidate would be forced to slow down and think about how the system was being designed, and ensure that sensible design practices were being adhered to.

The explicit design shown in the figure above was motivated by a number of factors:

- Dividing the system into logical components found in the real world – where possible, classes were constructed to logically represent entities which are present in the real world. For example, the Bank should be distinct from it's customers, and should keep a list of customers which are currently active in the system. Customers should be distinct from transactions, but should similarly keep a list of transactions representing their transaction histories
- Scalability and modularisation – there was a high priority placed on the ability to add new components and features to the system in the future. For example, if we wanted to re-implement the classification algorithm to use logistic regression, minimal code changes should be required outside of the fraud detector itself
- Reusability – Following regular practices associated with code encapsulation, we wanted to maximise the extent to which code could be reused and repurposed for other potential applications
- Ease of use and debugging – we want to ensure that the system is as easy to use and as easy to debug as possible. Problems should be easy to isolate, and we wish to limit the extent to which any bugs which do exist break the entire system. Additionally, the system should be simple for the user to interact with, and it should be straightforward to design and run tests to measure performance

Ultimately, the UML diagram shown in the figure above provided a template for how the system was implemented in code as part of the following module. Upon completion, the finished product did not deviate significantly from this initial model, so the reader may consider it an accurate representation of the final solution, with a small number of notable additions.

Module 3. System Implementation → Blog Post (and onwards)

This Module formed the majority of work which was done to complete the project. Given the high level design and background research conducted as part of the previous two Modules, this phase of the project was primarily focussed on using these to implement a software system which would both simulate a basic banking infrastructure, and more importantly perform the task of classifying incoming transactions in terms of their fraudulence, using a decision tree. Given the breadth of work that this entailed, it proved useful to divide this Module further, into three sub-modules representing the distinct steps required to complete it:

Module 3 a. Implementation of the banking system's base infrastructure. The primary goal here was to implement all functionality of the software system external to the decision tree itself, but which was necessary for it to successfully operate, e.g. adding new customers and transaction, keeping records, etc.

Module 3 b. Customer profiling. It was necessary to train the system in a way which allowed it to establish 'normal' patterns of behaviour for each of it's customers.

Module 3 c. Design and implementation of the decision tree itself, and integration with existing software.

Module 3 a. Implementation of base banking system

As previously discussed, the first step in Module 3 was to implement the base of the software system, which would largely perform all functionality external to the decision tree itself. This included components such as the bank, customers, transactions, interaction with the user, transaction histories, etc. Which not critical to the decision tree or classification system itself, these components would complete the simulation as a whole, and ensure that testing and verifying the decision tree would be as efficient as possible. They are therefore highly critical components of the system which were necessary for the decision tree's successful operation, and hence great care needed to be taken to ensure that we adhered to the high level design defined in Module 2.

Implementation of the base banking system was done in C++, and all source code (as well as resources required to build the system) can be found by following the link provided below:

Credit card fraud detection system source code

Please note that this link can additionally be used to find all source code for the proceeding Modules of the project too.

Module 3 b. Customer Profiling

Detection of credit card fraud through behavioural analysis and monitoring first requires the system to establish a notion of what constitutes 'normal' behaviour for each of it's customers. The system must first determine how customers act on a regular basis to determine when their behaviour becomes 'irregular' and hence indicative of fraud. Most generally, this practice is known as 'customer profiling' – that is, using the transaction history of each customer to build a personal profile which defines and parameterises the ways in which the system expects them to behave. This practice can be achieved in a number of ways, each of varying complexity and difficulty. For example, the technique of clustering was mentioned earlier in this report. Clustering consists on grouping cardholders based on similar behavioural patterns. However, such techniques were far beyond the scope and time constraints of this project, and we therefore opted for a simplified yet effective quantitative approach to customer profiling.

To carry out customer profiling, the system was given a list of transactions which would act as the 'transaction histories' of each currently active customer. A method was then implemented in code by which the system would use these transaction histories to build a 'customer profile' for each customer which included the following parameters:

- Average volume per transaction
- Average interval between transactions
- Percentage of total transactions made online
- List of locations (postcodes) where transactions are frequently made
- List of vendor types where transactions are frequently made

These parameters would allow my system to monitor transactions for the following red flags:

- Changes in transaction volume
- Changes in transaction frequency
- Changes in online purchase behaviour
- Changes in location
- Changes in spending behaviour (in terms of frequently visited vendor classes)

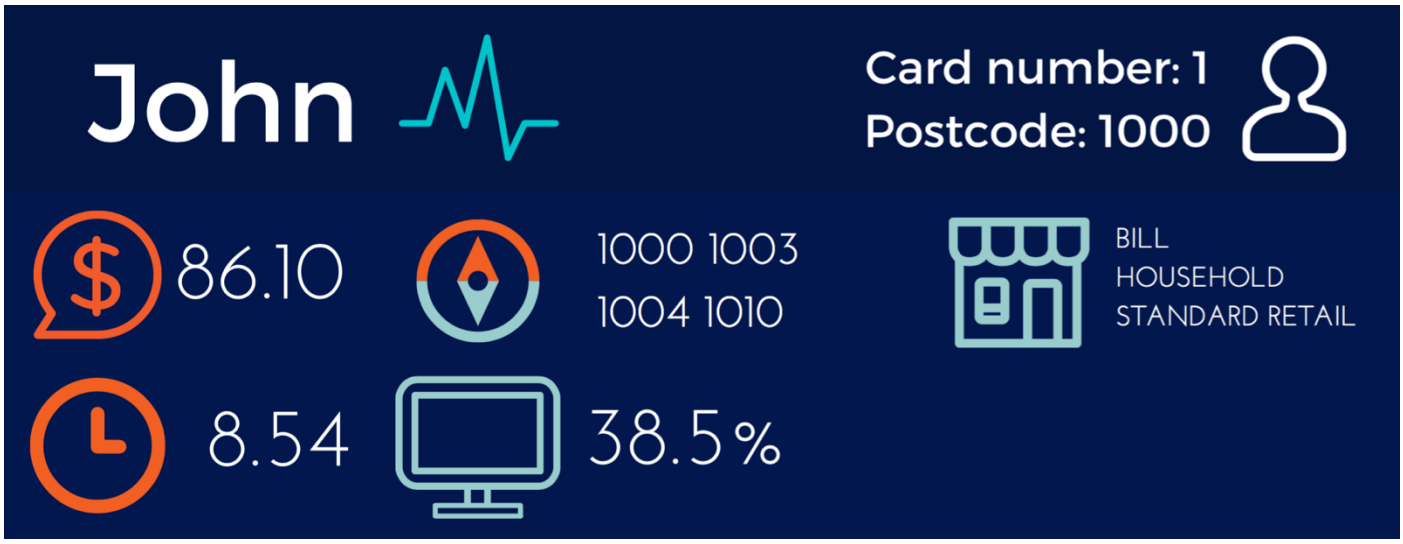
Essentially, while this method of customer profiling was not ideal in terms of accuracy or sophistication, the lists above show that it would successfully allow my system to monitor customer behaviour for all of the commonly used red flags I identified in Module 1. In the interests of time management, I therefore deemed this customer profiling method as sufficiently accurate, as it would allow me to develop a decision tree for all major indicators of fraudulent activity if necessary. The list of transactions which was passed to the system to carry out the process of customer profiling (effectively using the transactions in this list to calculate each of these parameters for every customer in the system) can be found by following the link below:

Customer profiling transaction list

The results of this process for each of the five customers I added to my system are summarised in the following infographic, which I created as a reference/visual aid for myself and anyone interesting in the project. It can be accessed by following the link below:

Customer profiling infographic

For convenience, a snippet from this infographic is shown here:



Note that as the regular blog posts which were written as part of this project discuss, the remainder of the project, including design and implementation of the decision tree, as well as the testing and verification phase were completed based on the fictional customer John, whose personal customer profile is shown in the infographic snippet above. This decision was primarily motivated by time constraints, as it was not feasible to carry out decision tree design and implementation for all customers in the system in time.

Module 3 c. Decision Tree Design and Implementation

Following the completion of the customer profiling phase, high-level structural design of the system's decision tree was carried out. Of course, the system's decision tree would form the core of the transaction classification functionality, and was therefore arguably the most important stage of this project. To design the system's decision tree, the ID3 algorithm was used. This first required us to generate a set of data points (transactions for our specific use case) which would act as 'training data' for the system. Training data for a classification system is essentially a set of data points whose correct classifications are already known. The system can then use these data points as a template from which it can 'learn' the patterns and features of transactions which are most strongly indicative of fraud. Therefore, the first step in the decision tree design process was the generation of a set of transactions which would act as training data for the system.

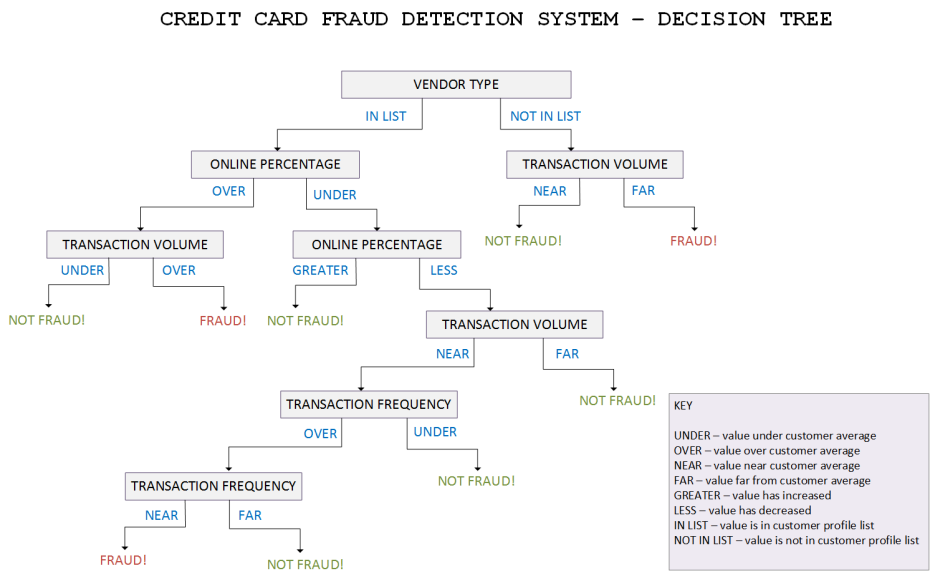
A link to this training data is provided below:

Decision tree training data transaction set

As previously discussed, the training data transaction set was based on the customer profile associated with John, and the spending patterns/behaviours reflected in the data were created in accordance with this. More information regarding each customer's personal profile and personality traits can be found in the Something Awesome blog posts detailing the Customer Profiling phase of this project. As well as a set of training data, the ID3 algorithm for decision tree design also requires us to define a set of attributes on which data points can be classified. As per the discussion carried out in the relevant blog posts, decision trees work best on categorical data – attributes which take on discrete and finite values. Therefore, as well as defining which transaction parameters or 'attributes' would be used by the decision tree to recursively classify new data, it was also necessary to convert some profiling parameters to categorical form, to accommodate for this requirement. A number of decision were made which allowed us to arrive at this list of possible attributes which the decision could use to group or 'split' incoming transactions. Note that they are listed with their names, as well as the values they can take:

- **Volume** under customer's average, over customer's average
- **Volume** near customer's average, far from customer's average
- **Vendor Type** in customer's list, not in customer's list
- **Location** in customer's list, not in customer's list
- **Frequency** under customer's average, over customer's average
- **Frequency** near customer's average, far from customer's average
- **Online Purchases** recent percentage over average, recent percentage under average
- **Online Purchase** percentage has decreased, percentage has increased

One may notice that all attributes have been defined as binary. This was a conscious design decision which was made to ensure that the tree did not become to complex to implement or design with in the project's time constraints. Given a set of training data, and a set of attributes on which to make decisions about data points, the ID3 algorithm works on the concepts of entropy and information gain. Essentially, a tree is constructed by starting with the list of all available training data points, and all available attributes, then at each iteration choosing to split the data into subgroups using the attribute which yields the greatest information gain (change in entropy) and has not yet been used. The algorithm stops when all groups are 'pure', that is, when all groups contain data points which share the same classification. The resultant network of subgroups and attribute splits resembles a tree data structure, hence the name 'decision tree'. Following ID3 gave us the following resultant decision tree:



To automate the majority of the ID3 algorithm, a number of MATLAB scripts were written. A link to the top level MATLAB script (which used each of the others) can be found by following the link provided below:

ID3 algorithm - top level MATLAB script

Please note that each of the other MATLAB scripts which were written for this purpose can be found in the same Git repository as the one provided above. With structural design of the system's decision tree complete, the final stage of Module 3 c. was implementing this design in code, and integrating it with the rest of the software which had been implemented previously. A detailed discussion of how this was done, and the motivations behind any critical implementation decisions which were made can be found in the Something Awesome blog posts for this phase of the project. For convenience, a link has been provided below:

Blog post discussing decision tree software implementation

Ultimately however, my implementation of the decision tree shown in the diagram above is simply a collection of nodes whose connections to each other have been explicitly defined. Each node carries with it a type identifier, which defines the explicit attribute of any new transaction it receives to use when determining which of it's children to pass the transaction on to next. There are explicit type identifiers for leaf nodes, which reside at the bottom of the tree. These nodes have no children, and simply return their classification to the parent node. Essentially, when the tree receives a new transaction for classification at it's root, the transaction will travel down the tree to a leaf, and the resultant classification will then travel back up to the root, which will alert the bank of the result. This happens in a recursive manner. In pseudocode, the procedure followed at each node of the tree looks like this:

```
1  bool classify_new_transaction(transaction t, customer c) {
2
3      work out what type of node this is;
4
5      if this is a leaf node:
6          return classification;
7      otherwise:
8          determine which decision function I need to call;
9
10     bool left_or_right = decision_function(t, c);
11
12     if left_or_right == left:
13         return left_child->classify_new_transaction(t, c);
14     otherwise:
15         return right_child->classify_new_transaction(t, c);
16 }
```

To interface the code which implements the decision tree with the rest of the banking system which was implemented as part of earlier Modules, a Fraud Detector class was created which abstracts the details of the decision tree itself from the bank, instead providing a simple interface between the two, through which the bank can provide the fraud detector with incoming transactions for classification, and the Fraud Detector can use the decision tree to provide the bank with a result. Source code for both the Decision Tree and the Fraud Detector can be found by following the link provided below:

Source code for Decision Tree and Fraud Detector

Module 4. Testing and Verification → Blog Post

With design and implementation of both the base banking infrastructure and the fraud detection system complete, the final stage of this project's practical components was Module 4, which was primarily focussed on the testing and verification of the final product. It is highly recommended that the reader refer to the corresponding Something Awesome blog post for this Module of the project, where one can find an in depth discussion of how the test environment was designed, and the motivations behind all critical design decisions. Broadly speaking, testing and verification is a critical stage of any project which aims to develop a new piece of technology. One may have successfully implemented a complex and sophisticated system, but this is irrelevant if such complexity does not yield strong performance. The importance of high performance is especially applicable to the problem this project aims to solve. By correctly classifying the fraudulence of credit card transactions, we aim to detect fraud as soon as it occurs, thereby limiting the financial impacts it can have both for the cardholder and their financial institution. Failure to do so results in tangible and measureable financial impacts, which may affect the livelihoods of large numbers of individuals. Additionally, this is a system whose performance is highly structured and of a quantitative nature – for a given transaction it returns a classification which is either correct or incorrect. Therefore, it is a system whose performance is both easy to measure, and whose performance has the potential to affect a large number of people's lives in a significant way (hypothetically).

For this reason, great care was put into both designing and carrying out the test procedure. Ultimately, the test procedure was an iterative process which involved the system being given a set of test inputs whose correct classifications where already known, it's performance being measured with respect to this test data, some adjustments being made to the product's implementation, and the process repeated until acceptable performance levels were reached. The first step to achieve this was to generate a set of test data points whose expected classifications were known in advance. These transactions were generated to reflect behavioural patterns consistent with the customer profile of John, and to mimic a number of the 'red flag' behaviours highlighted in Module 1. Knowing the correct classification of each transaction, the system's outputs could then be compared with the expected output of a 'perfect' system. The performance of the system for a given test was then defined as the percentage of transactions in the test set which it classified correctly. This was ultimately the metric we sought to optimise when making adjustments to the test during the testing phase.

The list of transactions which were used to test the system can be found by following the link provided below:

List of transactions used to test the fraud detection system

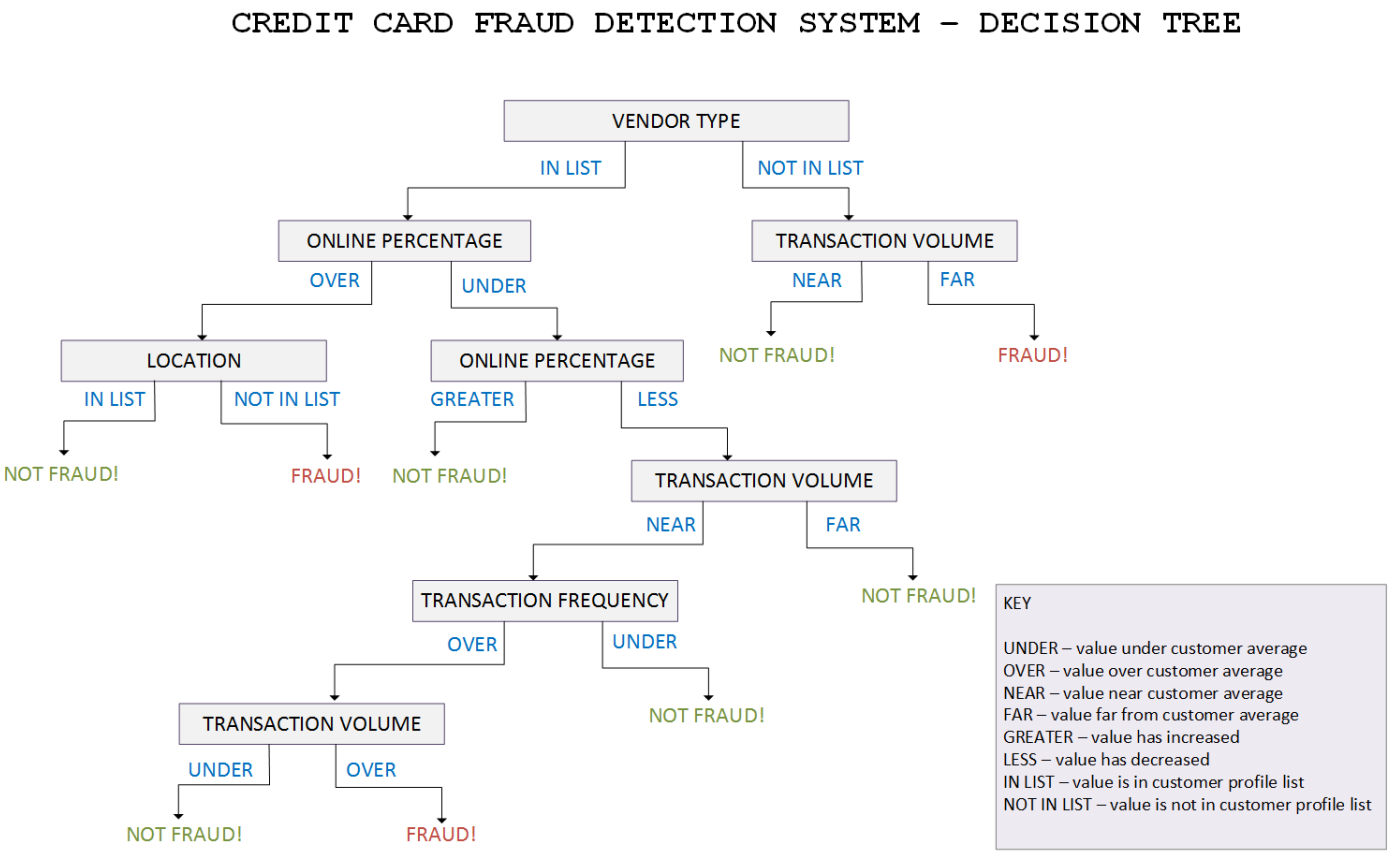
As the system stood prior to the testing a verification phase, the process of passing new transactions to the system for classification was based on mechanical input from the user one transaction at a time. To make the classification of a large number of transactions at once more efficient (so that tests and diagnostics could be run as quickly as possible), it was necessary to add a 'test' mode to the system. When the system is in test mode, the system's input is pointed at a file containing a list of pre-generated transactions which it processes as a batch. Additionally, all output except for the resultant classifications is supressed, and output is written to a file which can then be processed to calculate the system's performance for that test. The majority of changes which were made to the existing system to implement 'test mode' can be found in the main routine:

System main routine which implements test mode

Finally, a Python script was written which accepted the correct classifications for a set of test inputs and the system's actual response, and used these to calculate the percentage of transactions which were classified correctly:

Python script which calculates the percentage of test transactions classified correctly

Using the list of test transactions, the system's test mode which accepted inputs and directed outputs to and from files, and the Python script provided in the links above, an iterative procedure of testing and adjusting the system was followed until a final result was reached. Over the course of this testing and verification phase, a number of small yet critical changes were made to the fraud classification system, with ultimately yielded maximal performance results. First, the system was updated so that customers were re-profiled following every new transaction, so that the system continues to learn more about them as more transactions are added. By following this procedure, it is hoped that the system would improve even further as time goes on. This is a notable characteristic of all machine learning systems – their performance gets better as more data becomes available. It is important to note that the system only uses non-fraudulent transactions when re-profiling customers. Fraudulent transactions are not used when performing customer profiling so that they are not considered as part of a customer's 'normal' behaviour. As well as this, a number of nodes in the decision tree were changed in terms of their type identifier, and hence the attribute they used to make decisions. At the end of the testing and verification procedure, the structure of the system's decision tree was now as follows:



This is the tree's final structure, as implemented in the final product. Using this version of the decision tree, the following performance result was achieved, in terms of the percentage of total test transactions classified correctly:

87 %

Percentage of test transactions classified correctly

At this point, it is important to make a distinction between the types of errors which are possible when making classifications. Type 1 errors are also known as 'false positives'. This would be when the system believes that fraud has occurred when in fact, it has not. Type 2 errors are also known as 'false negatives'. This would be when the system believes that fraud has not occurred when it actually has. Clearly Type 2 errors carry with the, a significantly larger risk than Type 1 errors. Type 2 errors may result in fraud being allowed to continue, increasing liability for both the rightful cardholder and the institution. Meanwhile, Type 1 errors carry limited inherent risk aside from inconvenience. They can essentially be considered as the system being 'too sensitive' or 'too safe'. Type 1 errors in credit card fraud detection occur frequently, and can actually be used to help the system learn. Therefore, considering Type 1 errors as being of limited risk to the customer or the institution, and acknowledging their potential to help the system improve further if the customer notifies the institution of the error, the results of my tests show:

93 %

Percentage of test transactions classified correctly or Type 1 errors

These were considered favourable initial results, which we believe could be improved even further in future work, given the recommendations which will be discussed in the following sections.

Future Work

While the work carried out over the duration of this Something Awesome project has yielded positive initial results, we believe that there are a number of improvements which could be made to the existing system, and such improvements are left as future work. It is believed that future development would greatly improve both the performance of the system under the existing test environment, as well as it's performance under more comprehensive and realistic conditions. Additionally, we recommend that before extending the existing functionality of the system any further, more rigorous testing should be carried out using the current testing tools which were designed during Module 4. Ultimately, the steps which we believe should be taken as part of future development of this project are listed below, and it is recommended that these should be followed sequentially, in the order provided:

1. Using the existing test harness, generate a longer and more comprehensive list of test inputs for the customer John, to gain deeper insights into the performance of the system in it's current form, and to critically assess the ID3 algorithm's implementation which is currently being used. During these tests, observe the occurrences of both Type 1 and Type 2 errors, to better understanding the sources of error.
2. Use a more comprehensive set of training data, to optimise decision tree design. The larger the training data set, the more accurate the resultant decision tree will be.
3. Fully automate decision tree design via the ID3 algorithm. The MATLAB scripts which have currently been written only automate approximately 75% of the ID3 algorithm. The rest is still being done by hand. We recommend that the ID3 algorithm should be fully automated, to allow for more rapid iterations of the decision tree design, based on new training data. This will make optimisation far more efficient. Once this has been done, decision trees can be generated for ALL customers in the system with ease, and performance tests can be much more comprehensive.
4. Once ID3 has been automated, it should be more trivial to lift the binary restriction on values of attributes. Attributes can now be assigned more complex sets of attributes, rather than just a binary classification set. This will further improve the accuracy of the decision trees which can be designed.

5. Once ID3 has been automated, decision trees are more complex, and the system has been tested on a wide variety of customers with large data sets for both training and testing, the system can then be fed real-world training data, which is freely available online for the purpose of optimising fraud detection systems. This will give more definitive evidence as to how well the system will perform under real-world conditions. This step will likely require significant development in terms of how input is passed to the system, and how quickly it runs, since these training datasets are significantly larger than any we have used here. They usually contain tens of thousands to hundreds of thousands of transactions. Therefore, performance of the system will need to be optimised in terms of both speed and accuracy.
6. Ultimately, we do not wish to limit the implementation of this system to the exclusive use of decision trees only. Once the system has been extensively tested using decision trees, it is recommended that the same process is followed using other machine learning technologies available. Since they are currently the subject of most academic and industrial research, we recommend that any future development looks into the use of neural networks for this application, as such technology is showing great promise in applications such as this.

Were the objectives of this Something Awesome project met?

At the beginning of the semester, the first stage of this Something Awesome project was a project proposal, which defined both the aims of the project in terms of a number of distinct modules, and a project timeline, which placed explicit time constraints on the delivery of a finished product (in this case, that all modules be completed by the course's due date). The aim of this proposal was both to demonstrate to the markers that the candidate had carefully considered how the project would be completed in terms of the time and resources which would be required, and to act as a time management aid to the candidate themselves, to ensure that objectives were being met on a regular basis. This would give the candidate the greatest chance of achieving the project's objectives on time. With the project now complete, we can revisit the project proposal, and critically assess each component, making a final judgement as to whether or not the project has been a success:

Module 1. Background Research and Planning. Status: Complete
A survey of available resources and literature highlighted the techniques which are currently being employed by financial institutions to detect credit card fraud through behavioural analysis of cardholders, as well as the most commonly used 'red flags' which have been found to strongly indicate behavioural changes correlated to fraudulent activity. This information was used to determine which techniques and red flags were most suited to this project and application.

Module 2. High Level System Design. Status: Complete
The UML design language was used to generate a class diagram which would define the structure and components of the software system which would implement the project's fraud detection algorithm. This would be based on the technique of decision trees. The UML class diagram created as part of Module 2 remains and accurate representation of the current structure of the software system which was developed.

Module 3. System Implementation. Status: Complete
A software system was developed using the C++ programming language. This included the basic functionality of a bank, i.e. adding and storing new customers/transactions. Customer profiling was carried out to establish normal patterns of behaviour for all customers, followed by decision tree design using the ID3 algorithm, and MATLAB scripts to automate the associated operations. The resultant decision tree was then implemented again using the C++ language, and subsequently integrated with the base system.

Module 4. Testing and Verification. Status: Complete

The system was tested and verified using a number of automation techniques. A 'test mode' was added to the system to allow for more rapid testing and development, as well as a Python script to calculate and measure performance. Testing was based on a list of test transactions, and a number of small changes were ultimately made to the decisions tree's structure to achieve maximum performance. These changes yielded positive initial results, with 87% of test inputs being classified correctly, and 93% being either classified correctly, or as Type 1 errors, which carry little risk.

Module 5. Report Writing. Status: Complete

Self evident.

Conclusions

This report outlined the design and implementation of a simple machine learning system for the application of credit card fraud detection through behavioural analysis of cardholders. The project was completed in fulfilment of the Something Awesome component of the UNSW Computer Science and Engineering course COMP6441 – Security Engineering and Cyber Security. The project was divided into five distinct Modules, each dealing with a difference component of the final product's development. Ultimately, all Modules were successfully completed, with the system being based on the technique of decision trees, and having displayed positive initial results, classifying 87% of test input transactions correctly. While these results are favourable, the candidate believes that future development will be able to improve these results even further, and prove the system's validity in a wider variety of use cases and test environments. We conclude that the project has been a success, with all primary objectives having been met.