

3-2019

# Application of an Artificial Neural Network as a Third-Party Database Auditing System

Amirali Daghighi  
adaghighi@stcloudstate.edu

Follow this and additional works at: [https://repository.stcloudstate.edu/msia\\_etds](https://repository.stcloudstate.edu/msia_etds)

---

## Recommended Citation

Daghighi, Amirali, "Application of an Artificial Neural Network as a Third-Party Database Auditing System" (2019). *Culminating Projects in Information Assurance*. 86.  
[https://repository.stcloudstate.edu/msia\\_etds/86](https://repository.stcloudstate.edu/msia_etds/86)

This Thesis is brought to you for free and open access by the Department of Information Systems at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Information Assurance by an authorized administrator of theRepository at St. Cloud State. For more information, please contact [rswexelbaum@stcloudstate.edu](mailto:rswexelbaum@stcloudstate.edu).

**Application Of An Artificial Neural Network As A Third-Party Database Auditing System**

by

Amirali Daghighi

A thesis

Submitted to the Graduate Faculty of

St. Cloud State University

In Partial Fulfilment of the Requirements

for the Degree

Master of Science in

Information Assurance

May, 2019

Committee Members:

Jim Chen

Jie Meichsner

Mark Schmidt

### **Abstract**

Data auditing is a fundamental challenge for organizations who deal with large databases. Databases are frequently targeted by attacks that grow in quantity and sophistication every day, and one-third of which are coming from users inside the organizations. Database auditing plays a vital role in protecting against these attacks. Native features in data base auditing systems monitor and capture activities and incidents that occur within a database and notify the database administrator. However, the cost of administration and performance overhead in the software must be considered. As opposed to using native auditing tools, the better solution for having a more secure database is to utilize third-party products. The primary goal of this thesis is to utilize an efficient and optimized deep learning approach to detect suspicious behaviors within a database by calculating the amount of risk that each user poses for the system. This will be accomplished by using an Artificial Neural Network as an enhanced feature of analyzer component of a database auditing system. This ANN will work as a third-party product for the database auditing system. The model has been validated in order to have a low bias and low variance. Moreover, parameter tuning technique has been utilized to find the best parameters that would result in the highest accuracy for the model.

### **Acknowledgments**

Foremost, I would like to express my sincere gratitude to my advisor Professor Jim Chen for his motivation, support, enthusiasm and continuous encouragement. He is an outstanding professor and a great advisor and I am grateful to have the opportunity to work under his supervision. I would like to thank Dr. Jie Meichsner and Dr. Mark Schmidt for their invaluable support, hard questions, and insightful comments.

I would like to thank my family who helped me find my way through hardship, make right decisions, and providing support for me throughout my life.

Last but not the least, I would like to thank Dr. Jim A. Rakhshani who has been my role model, and I am thankful for his continuous support, for his patience and the motivation that he provided throughout my study in United States.

## Table of Contents

	Page
List of Figures .....	5
List of Abbreviations .....	7
Chapter	
I: Introduction .....	8
II: Literature Review .....	11
III: Methodology and Dataset .....	23
Data Preprocessing.....	24
Building the ANN.....	30
Fitting the ANN and Validation.....	34
Improving the Model .....	41
Dropout Regularization .....	41
Parameter Tuning .....	42
IV: Results.. .....	46
Data Preprocessing.....	46
Building the ANN.....	52
Fitting the ANN and Validation.....	53
V: Discussions and Conclusions .....	60
Future Work .....	61
References .....	62

## List of Figures

Figure	Page
1.1: Main components of an auditing system .....	9
2.1: Single unified auditing architecture .....	12
2.2 : Demonstration of an ANN with one hidden layer .....	16
3.1 : Installation of Theano and Tensorflow libraries.....	25
3.2 : Installation of Keras library .....	26
3.3 : Setting console working directory .....	27
3.4 : Definition of Dense function in Spyder 3 .....	32
3.5 : Sigmoid function.....	34
3.6 : Fit method in Spyder 3.....	35
3.7 : Bias and Variance trade off .....	37
3.8 : Inspecting K-fold Cross Validation method in Spyder 3 (part 1).....	39
3.9 : Inspecting K-fold Cross Validation method in Spyder 3 (part 2).....	40
3.10 : Inspecting Dropout in Spyder 3 .....	42
3.11 : Inspecting GridSearchCV in Spyder 3 (part 1).....	43
3.12 : Inspecting GridSearchCV in Spyder 3 (part 2).....	45
4.1: Imported dataset in variable explorer in Spyder 3 .....	46
4.2: Values of the Dataset .....	47
4.3: Variable explorer after splitting the dataset into X and Y .....	48
4.4: Dependent variable vector (Y).....	48
4.5: IPython console.....	49

Figure	Page
4.6: Encoding the matrix of features.....	50
4.7: Demonstration of test set and training set in variable explorer .....	50
4.8: X_train after feature scaling .....	51
4.9: Building the ANN.....	52
4.10: Fitting the training set (part 1) .....	53
4.11: Fitting the training set (part 2) .....	54
4.12: The predictions of the model on test set .....	55
4.13: Results of the prediction in Boolean.....	55
4.14: The confusion matrix .....	56
4.15: Vector of accuracies after performing K-fold Cross Validation.....	57
4.16: Mean and Variance of the accuracies .....	58
4.17: Best parameters after parameter tuning .....	58
4.18: Best accuracy of the model after parameter tuning .....	59

### **List of Abbreviations**

ANN: Artificial Neural Network

NN: Neural Network

AI: Artificial Intelligence

SGD: Stochastic Gradient Descent

MLP: Multilayer Perceptron

SVM: Support Vector Machine

BPN: Back Propagation Neural Network

SMOTE: Synthetic Minority Over-sampling Technique

RNN: Recurrent Neural Network



## Chapter I: Introduction

Database Management Systems have traditionally been used to store and secure data in computer systems. Databases are frequently targeted by attacks that grow in quantity and sophistication every day. Arming (2018) from CSO online, identified a list of 18 biggest data breaches of 21<sup>st</sup> century. This list contains data breaches that happen to companies such as: Yahoo, JP Morgan Chase, Marriott International, eBay, Target, Facebook to name just a few. The list is based upon damage or risk that the breach caused for users and account holders. As Conway et al. (2005) point out, according to the US Secret Service and the Computer Emergency Readiness Team (CERT), one-third of threats to an organization's databases come from the individuals inside those organizations with data access privileges. These insider individuals cause 29% of intrusions to database systems. According to Wagner et al (2017), in order to achieve security in such systems, detecting security breaches and implementing preventive mechanisms such as access control policies must be performed. Knowing when these security breaches and security policy violations have taken place is the primary purpose of an auditing system. Due to its value in maintaining this security, database auditing has had grown exponentially in the past decade. The enormous presence of data in social media and other networks such as LinkedIn (LinkedIn, n.d), Facebook (Facebook, n.d), Twitter (Twitter, n.d), the health-care industry for example in Hosseini et al (2017), and scientific research has caused these organizations' database administrators to reevaluate the importance of database auditing. To maximize database security, administrators must create and implement new policies that go beyond what is required by government regulations (such as the Sarbanes-Oxley act or HIPAA).

According to Bishop (2003), a database auditing system has three major components: a “Logger”, an “Analyzer”, and a “Notifier”, as illustrated in Figure 1.1.

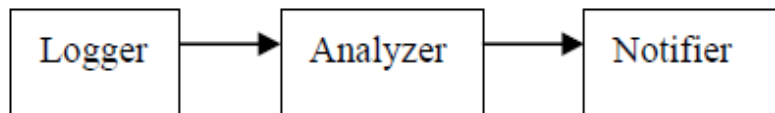


Figure 1.1: Main components of an auditing system (Bishop, 2003)

The logger records the information through binary or textual content. Pertinent events occurring within the database will be logged in the vault. What constitutes a pertinent event is dictated by the system security policy. The analyzer analyzes the content of the log file to detect anomalies, and to identify the need for policy modification, and to take over responsibility of the users' actions. Performing frequent analyzing operations in a database management system can lead to an increased number of violations detected. The output of the analyzer may result in changes to the data or error discovery in events. After the analyzer performs its duties, the output is passed to the notifier. The notifier's job is to inform the database administrator about the outcome of the auditing process.

Most database management systems have built-in auditing features to capture predefined auditing data and events. Some preliminary automatic alert functionality can be found in some DBMSs. However, there is very limited data auditing analysis capabilities built in these systems. The common practice is to extract auditing data and use embedded SQL tools to conduct manual data analysis to detect any breaches and policy violations. Clearly, there is a need for third party auditing tools to provide robust data analysis capabilities Woo et al (2010).

The primary goal of this thesis is to utilize an efficient and optimized deep learning approach to analyze user activities within a database and calculate the amount of risk that each user poses for the system. According to Najafabadi et al (2015), deep learning is a subset of machine learning which is used in big data analytics which extracts complex patterns in data by utilizing a hierarchical learning process. A deep neural network has more than one hidden layer. In previous works that will be mentioned Chapter II, neural networks were used to process audit data using machine learning techniques. However, by utilizing new technologies such as deep learning, more optimized SGD algorithms, and also utilizing new libraries which has been mentioned in Chapter III, a more precise implementation of previous works is available. This thesis is organized as follows. Chapter II, discusses previous research on audit data and demonstrates similar applications of Artificial Neural Networks. Chapter III, discusses the design and implementation of the deep learning model. Chapter IV, illustrates the results. Finally, Chapter V includes the discussion and future work.

## Chapter II: Literature Review

Many studies have tackled different applications of Artificial Neural Network (ANN). In this chapter, the previous and similar applications of ANN to this work such as improving the accuracy of ANN by adding extra hidden layers will be mentioned briefly. As mentioned in Chapter I, third-party database auditing products specifically tackle the lack of comprehensive audit analysis by providing useful recovery and auditing tools to cover the flaws of native database. Third-party auditing tools have much more flexibility when compared to the tools that come with the products. In case of a potential problem, a real-time third-party database auditing system would work faster and be more reliable than database administrators. Based on the users pattern activities, the auditing system will provide valuable reports for database administrators and organization's executives. Therefore, organizations are able to be more prepared for any possible threat. Some of the third-party products are designed to work proactively and alert the database administrators about possible threats, which could happen from either insiders or other users who have access to databases.

According to Oracle database security guide Oracle (2017), audit trails were separated in previous releases of Oracle products, but Oracle 12C release came with a new approach regarding audit trails. In the previous versions, the audit trails were separated for each component. SYS.AUD\$ was used for audit trail, SYS.FGA\_LOG\$ was used for FGA auditing, and DVSYS.AUDIT\_TRAIL\$ was used for database vault. However, in the latest release, all the audit trails were unified as one audit trail. The unified audit trails can be viewed from data dictionary view, UNIFIED\_AUDIT\_TRAIL. Unified audit has provided an easier way and a better opportunity for running analysis on the audits of a database. The problem in the previous

versions was that the analyzer had to gather all the audit files together and then perform the desired analytics tasks. By using the unified audit, users can access a single formatted audit of the database, which will provide a wide variety of services and can be allied with the audit vault as well as the database firewall. Unified auditing reduced the overhead in the performance of the system. Moreover, unified auditing makes it easy to audit conditionally and introduces new roles such as AUDIT\_ADMIN or AUDIT\_VIEWER, which increases the integrity of logs and policies.

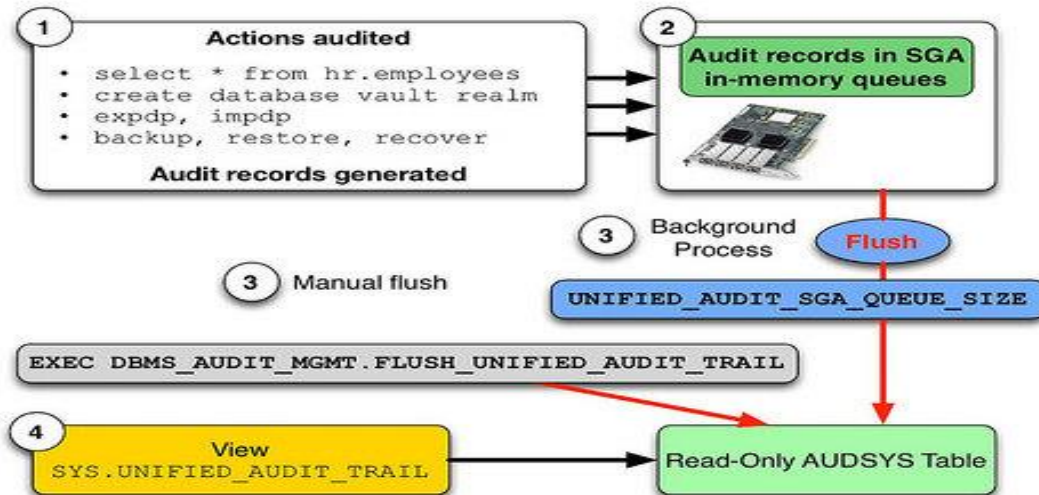


Figure 2.1: Unified auditing architecture

Figure 2.1 Salama (2014), demonstrates the unified auditing architecture. Unified auditing makes the analysis of audit logs more effective. In order to store audit trail, AUDSYS table is used. It is worth mentioning that if there is already any audit log in the FGA\_LOG\$ or AUD\$ tables, then that data would still remain in the SYS schema. It is highly valuable to have a good analysis of the database system activities to identify suspicious activity as soon as possible. Audit record is a place to save auditing activities within Oracle Database. Audit trails can be saved as a data dictionary or an operating system file. According to Yang (2009), audits saved in

a data dictionary are called database audit trail, while audits saved as an operating system file are called operating system audit trail. Oracle 12C has the unified audit trail option. The unified audit trail provides for more effective context-dependent logging and produces a superior single formatted audit log. For the majority of organizations who deal with personal information of their customers securing their data is extremely important, and there is no place for mistakes. Even one data breach within such organization would cost a fortune. On the other hand, the price of audit maintenance and analysis is relatively high and many of the auditing products, which come with purchasing the databases, will not guarantee a high state of protection for the organizations.

Third-party database auditing products specifically tackle the lack of comprehensive audit analysis by providing useful recovery and auditing tools. Third party auditing tools have much more flexibility when compared to the tools that come with the products. In case of a potential problem, a real-time third party database auditing system would work faster and be more reliable for database administrators. Based on the users pattern activities, the third-party auditing system will provide valuable reports for database administrators and organizations executives. Therefore, organizations are able to be prepared for any possible threats by updating their current policies and implementing new policies. Some of the third-party products are designed to work proactively and alert the database administrators about possible threats, which could happen from either insiders or other users who have access to database. Improvements suggested by monitoring the system using the third party systems decrease the risks down the road. Improvements suggested by monitoring the system using the third party systems decrease the risks down the road. The followings are some of the applications of ANN in many different

fields such as auditing and accounting, finance, tax cases, business, forecasting inflation, and bankruptcy prediction.

Another reason for choosing a deep learning approach is that, as the amount of training data increases, the model would have more data points to learn from, and the performance of the system would increase over time. The number of false alarms of the system will decrease and the outcome would then be more precise. A deep learning model with low bias and low variance would be beneficial to improve the database auditing process and implement more accurate policies. To carry this out, the ANN would be implemented by using the Python programming language in Spyder which is a distribution in Anaconda platform. The dataset that has been used for this project is a simulated dataset that contains information about 10000 employees of an organization. The ANN will be built in the Spyder distribution of Anaconda. The next steps are building, fitting, and then checking the validation of the ANN which has been explained in details in Chapter III. Having a well-trained deep neural network will significantly increase the chances of detecting unauthorized access by monitoring the user's activities. Using a third-party auditor that utilized an ANN would be beneficial for organizations because they would be able to create more efficient policies and policy modifications that would result in a more secure database system.

Machine learning, and Artificial Intelligence in general, can have a big impact in the future of database auditing. Finding certain patterns and analyzing data in great volume is nearly impossible for humans whereas the machine is able to perform millions of calculations in a short period of time. On the other hand, having such a large amount of data available will result in a better training for the ANN, due to a better weight adjustment in the ANN. Weights are crucial

for ANN, since the system will learn how to function based on the values that are being updated on weights. Generally speaking, in order to train an ANN with SGD, the following steps should be taken. First, weights should be randomly initialized to small numbers close to (but not exactly) zero. The first data observation would then be given to ANN as input. Each of the nodes will have one feature. In a deep learning model there are one or more layers other than input layer and output layer. Next step is the forward propagation, or left to right propagation. After this step, the gradient error will be calculated. Based on the result of this step, the back propagation will take place. Back propagation will update the weights based on the error that happened according to the target weight. The above steps have to be followed and weights must be updated after each observation.

Kotsiantis et al. (2002) emphasize the fact that “unintentional nonfraudulent financial statement errors are static in that their incidence is unaffected by the anticipated audit. But fraud is intentional and strategic such that its incidence is affected by the anticipated audit.”

Khare and Gaibhiye (2013) stated that an ANN is a parallel-distributed-information-processing system that simulated neural network of human brain. The authors worked on application of an ANN on operation of reservoirs. The theory of this work was basically explaining the basic architecture of an ANN and did not provide any detail explanation on how to achieve the best model. They have presented the previous applications of an ANN on water resources engineering with a focus on reservoir field.



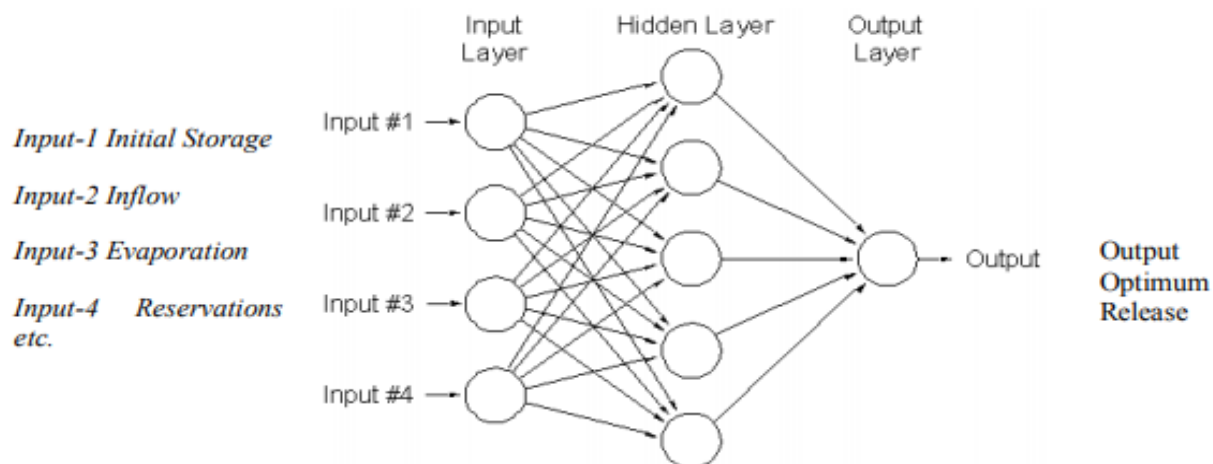


Figure 2.2 : Demonstration of an ANN with one hidden layer (Khare and Gaibhiye, 2013)

In auditing and accounting, Etheridge et al. (2000) and Gaganis et al. (2007) showed that ANN could decrease the misclassification in auditor's works. Another brilliant aspect of ANN has been shown in this work on how ANN could transfer categorical variables to binary values. They had two types of errors in their work. Errors in these works have been called "Type I" and "Type II"; Type I error is in correlation of failed banks and Type II error represented the non-failed banks. In this work the goal was to focus on Type I errors since misclassification of a true negative outcome would be more harmful than misclassification of a false positive outcome.

In classification of tax cases, Wu (1994) used one hundred and eighty samples of tax audit case files to train and test an ANN to show if there is a need for further audit in a tax case. Information about income tax behavior were in each tax case. Wu also showed that the classification accuracy would increase by one percent by adding a new hidden layer to ANN. The total number of hidden layers after increasing the accuracy was three. The accuracy of this model has been increase from ninety four percent to ninety five percent by adding another hidden layer.

In a similar work to Wu (1994), Koshivaara (2000) demonstrated the application of an ANN to help planning the auditing monthly balances. The outcome of this ANN model was illustrated by telling the user whether further audit is required; (“no further audit is required” or “further audit is required”). The accounts that were used to train the ANN model in this work were chosen by a CPA-auditor. The researcher compared two different models to achieve better results. First model works inside the quartile and second model includes previous input variables. The results showed that the model with previous input variables had better results.

Another interesting application of ANN in financial statements to identify modified and unmodified audit options have been presented by Pourheydari et al. (2012). The authors used an ANN to different classes of audit options on financial statements. They achieved an accuracy of eighty seven percent on both modified and unmodified audit options. This work is another application of an ANN and shows how powerful and useful MLP and deep learning could be on financial statements.

Six different ANN designs have been studied by Busta and Weinberg (1998) to classify normal and manipulated financial data and find the best model. The limitation on this work is the size of the dataset which is relatively small in comparison to the other financial dataset and similar works that have been done in this field. This work utilized Benfors’s law that shows the predictable distributed pattern on digits of naturally occurring numbers. This work focused on digit distribution of numbers in financial data. The input layer for each of six classifiers consisted of thirty four nodes. This work had a big variance in the results section. The author claims that the model is reliable with the accuracy of seventy point eight percent. However, the variance of the results for this model is very big which will lead in inconsistency in the accuracy of the

model and would bring up the question of reliability. Accuracy of different models that have been utilized in this work, were between sixty seven percent to one hundred percent.

There are other approaches for training neural networks or other classification methods on unbalanced data sets, including over sampling, Synthetic Minority Over-sampling Technique (SMOTE), Borderline-SMOTE takes the critical data samples on the border of each class into account, and Hybrid methods. All the above methods have their own cons and pros, e.g. the pairwise distance of all data nodes should be measured in SMOTE based methods, which is expensive, or different weak learners should be trained in Hybrid based methods and Adaboost is used to combine the weak learners into a better classifier, which has expensive training overhead.

ANN has been used in audit problems to identify certain patterns to detect anomalies in different systems and organizations. This type of application of ANNs is very efficient in detecting frauds within organizations. Calderon and Cheh (2002) came to the conclusion that ANN can be superior than other methods after analyzing twelve different papers in categories of: risk assessment, issue of going concern opinion, fraud and error identification, identification of too much risk in financial field, and bankruptcy forecast.

Cerullo and Cerrullo (1999) stated that ANN is far more proficient for the purpose of analyzing large amounts of data and to recognizing complex patterns and relationships in financial statements. They analyzed the use of ANN in fraud prediction on financial statements using the coefficients and information of the accounting statements. They came to the conclusion that ANN can be more accurate in recognizing certain patterns which are not noticeable for other computational approaches. In a similar work, Taha (2012) compared the ANNs with statistical models in planning and managing audits. Taha stated that ANNs are could show which financial

statements are likely to have substantial errors which would help the auditor to have a better understanding of the depth of audit test and creating more accurate point of view on financial statements.

Shin et al. (2005) compared Back Propagation Neural Network (BPN) and Support Vector Machines (SVMs) in corporate bankruptcy prediction. They stated that SVMs has a better accuracy and generalization performance than BPN when the training size is smaller in the problem of bankruptcy prediction. This challenged the fact that training a neural network and updating weights takes a lot of training loads whereas SVM captures the geometric characteristics of the feature space and is capable of generating more accurate results than BPN.

Koshivaara (2000) compared four different ANN models to compare the information of a firm with similar information for the industry. The author used financial statements of 31 manufacturing companies over a four-year period of time. The pre-processing phase of data in this work was divided into four different sections: one a yearly basis, all together, on company basis, and finally combined yearly and company basis. In order to calculate quick ratio, sixteen income statement accounts were selected to demonstrate the financial statements and short-term liabilities. The average number of staffs were also included in the model. After comparison of the results of different models, the best results were achieved when the pre-processing stage was on a yearly basis or all together.

In Onural et al (2016), Musavi et al (2016), and also Musavi et al (2018), Musavi (2019), authors used another application of ANN in using the AI agents that trying to find an optimal approach to achieve a specific goal or improve performance on a specific task which is called Reinforcement learning. Reinforcement learning is a mathematical learning mechanism that is

based on human learning process. The objective of reinforcement learning is to come up with a policy, by which we mean a mapping from states to actions, that maximize a reward function. An agent observes the environment's state, chooses an action, moves to a new state of environment and receives a reward from environment. In the structure of reinforcement learning agent and environment interact with each other with a scalar signal named as reward. Reinforcement learning has applications in finding solutions for tasks in engineering problems where either the system model or the interaction model with the environment is complex. Reinforcement learning can be used for multi agents' systems as well. In these works, authors used the reinforcement learning to predict the outcomes of 2D and 3D scenarios where both manned and unmanned aircraft systems coexist. The classical and deep versions of reinforcement learning is combined with level-k reasoning method which is a game theoretical concept is used to model the human pilot reaction model. The multi-armed bandit problems are also applied to network settings including load balancing for data centers which is studied in Yekkehkhany and Nagi (2019), Yekkehkhany (2017), and Yekkehkhany et al. (2018). For more details regarding scheduling with multi-level data locality and load balancing see Xie et al (2016).

Coakley and Brown (1991) used ANN for predicting values of financial ratio and recognizing patterns. They used monthly account balances to train the ANN. They evaluated their model with a simulation to test whether the model is viable for this application or not. The results showed the efficiency and effectiveness of using an ANN in a financial data set to detect certain patterns. In another work, Coakley and Brown (1993) targeted the performance of material misstatements using an ANN. They test if the trend prediction ANN offers any performance improvement in detecting the material misstatements. Fifteen income statements

and balance sheet accounts were selected for training the ANN. However, due to the nature of their data set the process of training the ANN was very time-intensive because of the impact of number of neurons in the training process. In order to evaluate the performance of the model, the authors compared an assumed lack of actual errors and seeded material errors on the model.

After comparison of the results achieved from financial ratios and regression models, the results of this study indicated that ANN shows better performance to predict values with less variation. These results were based on: financial ratios, comparison of methods, error size effect, statistical level of confidence, and source of material error effect. The limitation in this work was the question of effectiveness of the results due to fluctuating nature of financial data. After this work, Coakley (1995) focused on patterns of related fluctuations in different financial accounts. The goal was to detect the probable source of a material monetary errors in mentioned financial accounts. The results indicate that by using ANN on top of traditional analytical procedures would improve performance in recognizing material misstatements within financial accounts.

Koskivaara (1996) also studied the fluctuations that could happen in monthly income statements based on the accounts that presented in Coakley and Brown (1993) work. Koskivaara implemented a new prediction model called “one-step-ahead” to monitor the fluctuations based on the non-linear dynamics of the statements and the relationships between the accounts based on income statements.

Finally, Wesner et al. (2014) and Sanders et al. (2006) both defended the importance of computational intelligence and use of ANNs in auditing while maintaining the continues auditing procedures. They mentioned that the new technologies such as cloud computing and data science techniques will increase and improve the effectiveness as well as the efficiency of the auditing

process. Sanders et al. also highlighted the applications of ANN that were utilized in Calderon and Cheh (2002) and Koskivaara (2003) works. Moreover, Byrnes et al discussed the applications of computational intelligence and ANNs would allow the auditors and researchers to have continuous and predictive audits and more efficient fraud detection.

### **Chapter III: Methodology and Dataset**

The main implementation that will be used for this project has been demonstrated in this chapter. There are other possible alternative implementations or modifications based on the nature of the dataset that is going to be used to train, test, and evaluate the model. De Sousa (2016) showed that the main advantage of this methodology is optimizations that “Keras” library has provided which resulted in a more accurate classification and more efficient NN. Keras, is one of the most popular neural network open source libraries in python. For the implementation of the ANN in this project, Spyder 3 distribution of the Anaconda platform has been chosen. Nowadays, Python is one of the best languages in machine learning and AI programming. Python provides powerful and comprehensive libraries. One of the most powerful libraries that has been invented by the Google Brain team is Tensorflow (Chollet, 2017). In this work Keras is running using the Tensorflow backend. Theano is another powerful library, which according to Bergstra et al. (2010) is a compiler for mathematical expressions in Python that combines the convenience of NumPy’s syntax with the speed of optimized native machine language . The second library is Tensorflow, and the next one is Keras. Keras library is built on top of Theano and/or Tensorflow in order to avoid thousands lines of codes and increase the proficiency and accuracy. Keras library uses the Tensorflow backend in this work which has been shown in Chapter IV. Features that are needed to initialize the nodes of NN will be extracted from the simulated dataset.

The dataset that has been chosen for this work consists of 10,000 records. Each row represents the information about employees of a company. The main features (independent variables) that have been utilized for this work including: location, gender, age, tenure, balance, number of devices, has admin warnings, less than three warnings, salary, and number of attempts.



## Data Preprocessing

The main goal in this stage is to prepare data as an input for ANN. Several steps have to be taken in the following order to achieve a well-preprocessed input. First and foremost, the right features from the dataset have to be selected. The selected independent variables will shape the matrix of features. Thanks to Spyder 3, the matrix of features can be seen in the variable explorer. After this, the dependent variables will be saved in a vector. As mentioned in Dowson et al (2004) and Gijarati (1970) in order to have a well-preprocessed data, multi-collinearity and dummy variable trap has to be eliminated. Multi-collinearity is a situation in which the effect of the independent variables on dependent variables is hard to distinguish which will result in an unstable estimation of the result. Therefore, some of the dummy variables have to be removed so that the independent variables can be free of multi-collinearity. Furthermore, as mentioned in Zhang et al. (2015) most of the datasets contain multivariate data, all of the categorical variables have to be encoded, meaning that all the independent variables that are in categorical form should be transformed into numerical values. Numerical numbers that have been assigned to independent categorical variables have been chosen totally random, because there should not be any relational order between the categories of the categorical variables. The model should not be able to predict one independent variable from the other one. For the installation of the libraries the following commands has been used in the Anaconda Prompt window.

Installing theano:

```
pip install theano
```

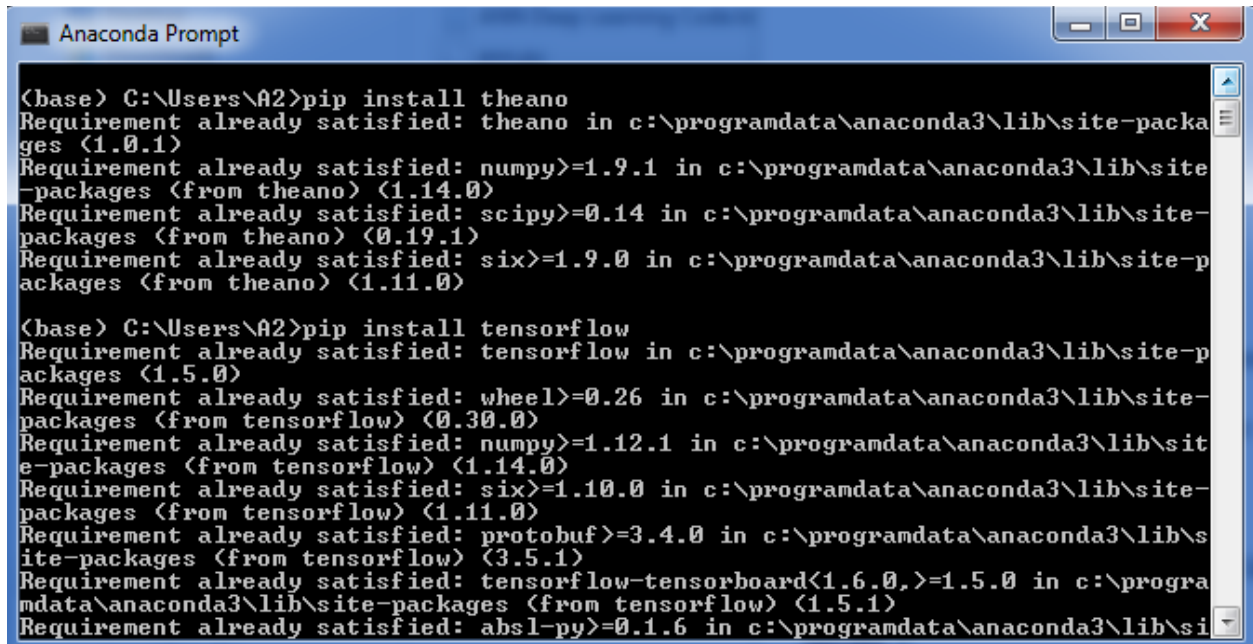
Installing tensorflow:

```
pip install tensorflow
```

Installing Keras:

```
pip install keras
```

If the libraries are already installed the following messages will appear on the Anaconda Prompt or the Terminal.

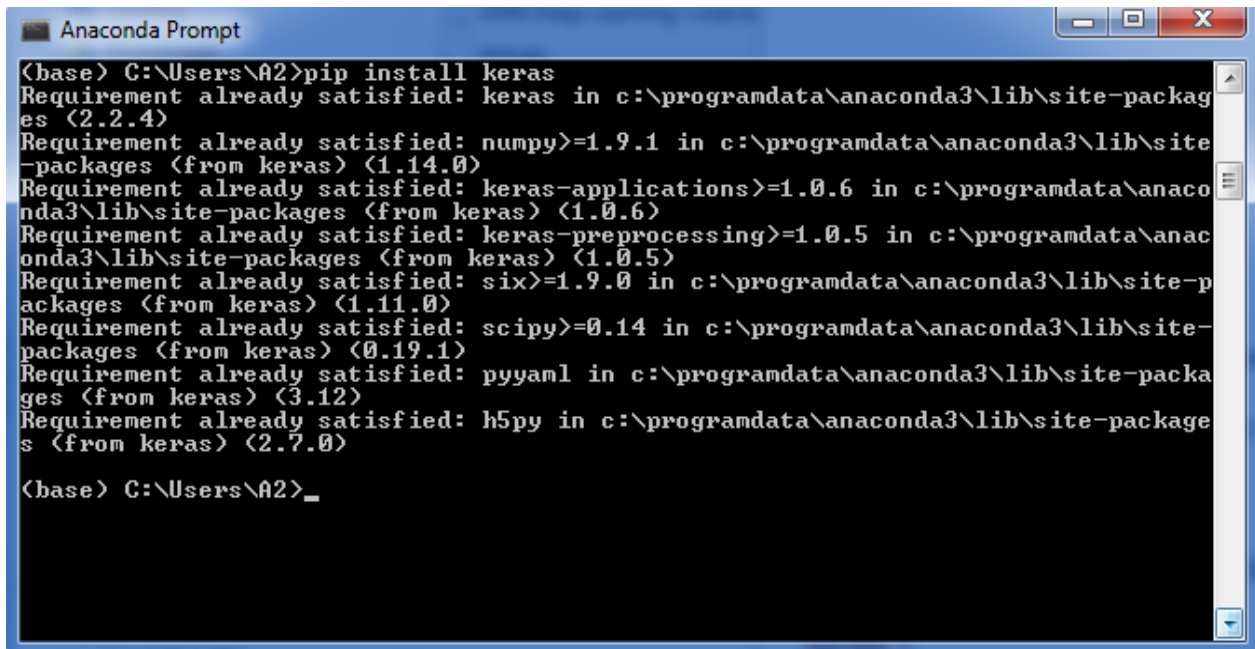


```
Anaconda Prompt

(base) C:\Users\A2>pip install theano
Requirement already satisfied: theano in c:\programdata\anaconda3\lib\site-packages (1.0.1)
Requirement already satisfied: numpy>=1.9.1 in c:\programdata\anaconda3\lib\site-packages (from theano) (1.14.0)
Requirement already satisfied: scipy>=0.14 in c:\programdata\anaconda3\lib\site-packages (from theano) (0.19.1)
Requirement already satisfied: six>=1.9.0 in c:\programdata\anaconda3\lib\site-packages (from theano) (1.11.0)

(base) C:\Users\A2>pip install tensorflow
Requirement already satisfied: tensorflow in c:\programdata\anaconda3\lib\site-packages (1.5.0)
Requirement already satisfied: wheel>=0.26 in c:\programdata\anaconda3\lib\site-packages (from tensorflow) (0.30.0)
Requirement already satisfied: numpy>=1.12.1 in c:\programdata\anaconda3\lib\site-packages (from tensorflow) (1.14.0)
Requirement already satisfied: six>=1.10.0 in c:\programdata\anaconda3\lib\site-packages (from tensorflow) (1.11.0)
Requirement already satisfied: protobuf>=3.4.0 in c:\programdata\anaconda3\lib\site-packages (from tensorflow) (3.5.1)
Requirement already satisfied: tensorflow-tensorboard<1.6.0,>=1.5.0 in c:\programdata\anaconda3\lib\site-packages (from tensorflow) (1.5.1)
Requirement already satisfied: absl-py>=0.1.6 in c:\programdata\anaconda3\lib\si
```

Figure 3.1 : Installation of Theano and Tensorflow libraries



```

Anaconda Prompt
(base) C:\Users\A2>pip install keras
Requirement already satisfied: keras in c:\programdata\anaconda3\lib\site-packages (2.2.4)
Requirement already satisfied: numpy>=1.9.1 in c:\programdata\anaconda3\lib\site-packages (from keras) (1.14.0)
Requirement already satisfied: keras-applications>=1.0.6 in c:\programdata\anaconda3\lib\site-packages (from keras) (1.0.6)
Requirement already satisfied: keras-preprocessing>=1.0.5 in c:\programdata\anaconda3\lib\site-packages (from keras) (1.0.5)
Requirement already satisfied: six>=1.9.0 in c:\programdata\anaconda3\lib\site-packages (from keras) (1.11.0)
Requirement already satisfied: scipy>=0.14 in c:\programdata\anaconda3\lib\site-packages (from keras) (0.19.1)
Requirement already satisfied: pyyaml in c:\programdata\anaconda3\lib\site-packages (from keras) (3.12)
Requirement already satisfied: h5py in c:\programdata\anaconda3\lib\site-packages (from keras) (2.7.0)

(base) C:\Users\A2>_

```

Figure 3.2 : Installation of Keras library

The installations could be done in several different ways based on the operating system. In order to install the libraries in windows operating systems, the Anaconda Prompt, and for Linux operating systems, the main terminal on the machine could be used. After installation, all the mentioned libraries should be updated to the last version using the following command.

```
conda update --all
```

The above command will update the libraries as well as the Anaconda platform which includes Spyder 3. The next step is to set up the working directory for in Spyder 3. The working directory is the location of the code file and the dataset. Active directory could be set by selecting the “File Explorer” icon on the top right side of the Spyder and then selecting the path.

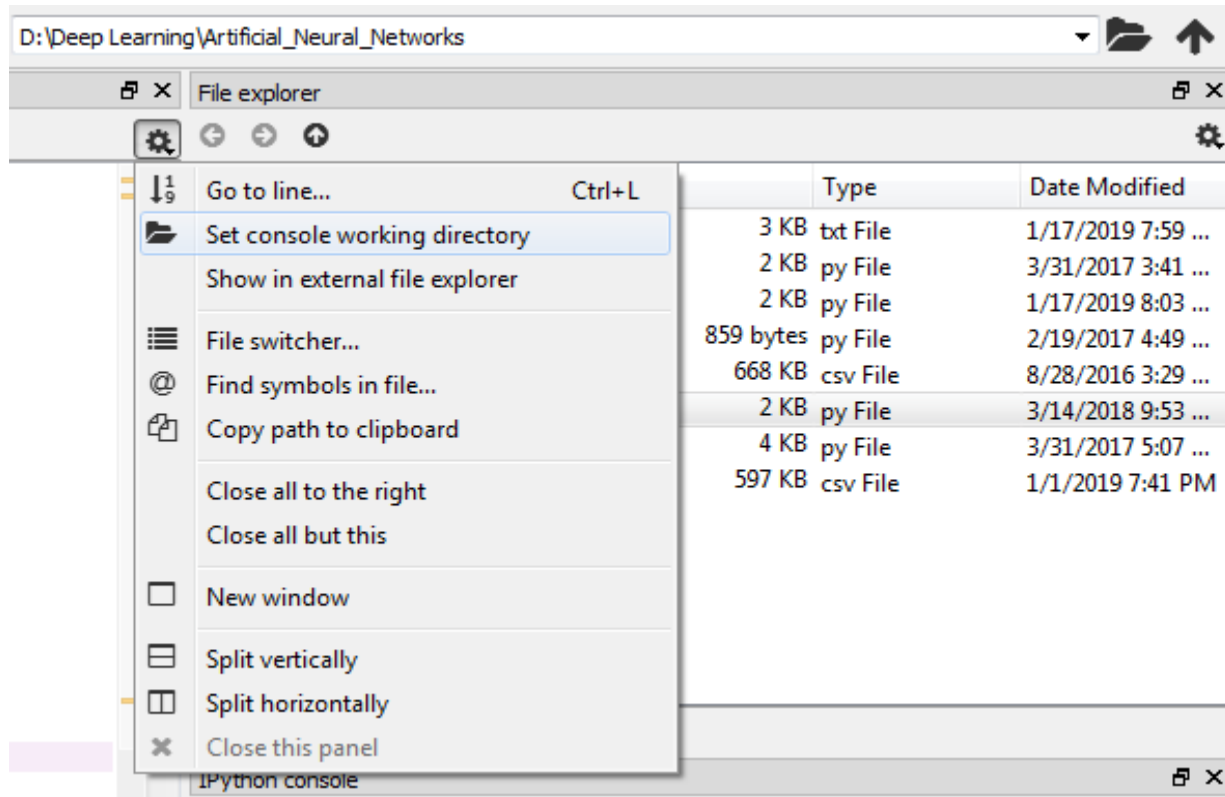


Figure 3.3 : Setting console working directory

After installing Theano, Tensorflow, and Keras, and setting the console working directory the following libraries (numpy, matplotlib, and pandas) need to be imported:

Import numpy as np

Import matplotlib.pyplot as plt

Import pandas as pd

The dataset for the project is imported by running the next line of code.

```
dataset = pd.read_csv('Thesis Dataset.csv')
```

CVS extension in general, is any tabular data such as spreadsheet or database type of data. In the next line the matrix of features has been created. The index in Python starts from zero. In the dataset there are some independent variables that have no impact on the outcome of the deep

learning model, for instance last name or employee ID has no impact of whether an employee behavior would lead to a data breach or not. Therefore, some of these irrelevant independent variables can be eliminated in this step. Based on the dataset, matrix of features (X) has been selected as follow.

```
X = dataset.iloc[ : , [ 1 , 12 ] ].values
```

Here, “X” is the matrix of features. “1” and “12” indicate the indexes of the independent variables inside the dataset that could have impact on the outcome of the model. Basically, they are indicating the indexes of the independent variables which have been selected as features.

```
Y = dataset.iloc[:, 12].values
```

“12” indicates the number of index in the dataset which contains the dependent variable. “Y” is called the dependent variable vector. This vector is usually the outcome of the previous observations. In the implementation of this project, the dependent variable indicates whether the user committed an act that lead to data breach or not which is a binary outcome.

All of the categorical variables within the dataset need to be encoded. Therefore, before splitting the dataset into training and test sets, all of the categorical variables should turn into numerical variables. There is no need to encode the dependent variables since vector “Y” has already have all the dependent variables as numerical values (zeros and ones).

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

```
labelencoder_A = LabelEncoder()
```

```
X[ : , 1 ] = labelencoder_A.fit_transform( X [ : , 1 ] )
```

The “fit\_transform” method converts the categorical variable into numerical variable (encoding). These numbers are randomly assigned to the categorical variables. “1” indicates the

index of the categorical variable. Based on the number of the categorical variables that exist in the chosen index. Random non-negative integer numbers will be assigned to these categorical variables. If there are more than one categorical variables in the dataset, for each of them, the `fit_transform` method should be used. Since some of the categorical variables have more than two elements, and in order to avoid the dummy variable trap, the following code can be utilized. “1” is the index of the variable that needs to be eliminated. It is worth mentioning that the following step happens after the `fit_transform` method has been compiled. Therefore, matrix of features has been changed.

```
onehotencoder = OneHotEncoder( categorical_features = [ 1 ] )
```

```
X = onehotencoder.fit_transform(X).toarray()
```

```
X = X[:, 1 : ]
```

“1” indicates the index of the first variable that is going to be part of the new matrix. In the other word, the new matrix of features will have all the columns of “X” except the first one.

The next step is splitting the dataset into training set and test set. In the most recent update of Spyder 3 (Spyder 3.6.3), the “`cross_validation`” has been modified to “`model_selection`”.

However, both of the methods will be complied by Spyder 3.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
```

After running the above code, the dataset will be divided into a train set and a test set. “`test_size`” indicates that twenty percent of the dataset will be considered as data for the test and eighty percent will be used to train the ANN.

Now we need to apply feature scaling that was mentioned in Zhou et al (2017) to scale the independent variables. Due to highly intensive calculations and parallel computations there a lot of computational overhead in the program, it is necessary to use feature scaling to ease these computations. Additionally, there should not be any independent variable dominating the other one.

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)
```

In order to create the ANN the Keras library will be imported as follow. Notice that keras library will be built on top of the Tnsorflow. The other alternative solution is to use Theano back-end.

```
Import keras
```

The “Dense” module is required for building the layers and the “Sequential” module is used for the initialization of the NN.

```
from keras.models import Sequential

from keras.layers import Dense
```

### **Building the ANN**

In order to start initializing the ANN, there are two possible approaches. The first one is defining the NN as a sequence of layers (deep learning) which is the selected approach in this project. The second approach is to create a graph. An object of the sequential model (a classifier) needs to be created in this step. Dense function will randomly initialize the weights.

Classifier = sequential()

The “Classifier” object is an object of the sequential class. Since the problem that has been tackled in this project is a classification problem. This classifier will be the ANN that will be built in the next steps.

Now the first layer of the neurons which is the input layer is going to be created. This is the second step in training an ANN. The number of inputs is equal to the number of independent variables in the matrix of features. Based on the previous variable that has been selected in the beginning of the code this number is equal to “6”. In this implementation the number of nodes in the hidden layer has been chosen as the average of the number of nodes in the input layer and the number of nodes in the output layer. As mentioned in Smit and Eiben (2009) Parameter Tuning Technique could be used in order to test different models with different parameters to achieve the best performance. Bengion and Grandvalet (2004) studied K-fold cross validation technique. This technique could be used to create a separate set in the dataset (other than test set and training set), that is called the “cross validation set”. Cross validation set can be used to measure the performance of different models with different parameters on the dataset. Using such techniques will help a lot in choosing the optimal parameters for a model. The K-fold Cross Validation technique has been used to evaluate the accuracy of the model at the end of this section. The “uniform” function will randomly initialize the weights. It is mandatory to specify the “input\_dim” argument. Input\_dim shows the number of independent variables. In the other word, the features that are being used as inputs of the NN. Since the NN doesn’t know this number yet, it has to be specified for the first hidden layer at least. If the activation function value of a neuron is higher, then it means that node is going have a higher impact on the NN.



Simply it means that node will more pass on the signal. The sigmoid function allows us to have the probabilities for each observation. Therefore, the sigmoid function has been chosen for the activation function of the output layer. The activation function of the hidden layers will be the rectifier activation function. By pressing “Ctrl + I” the variables of the Dense function will be shown in the help section.

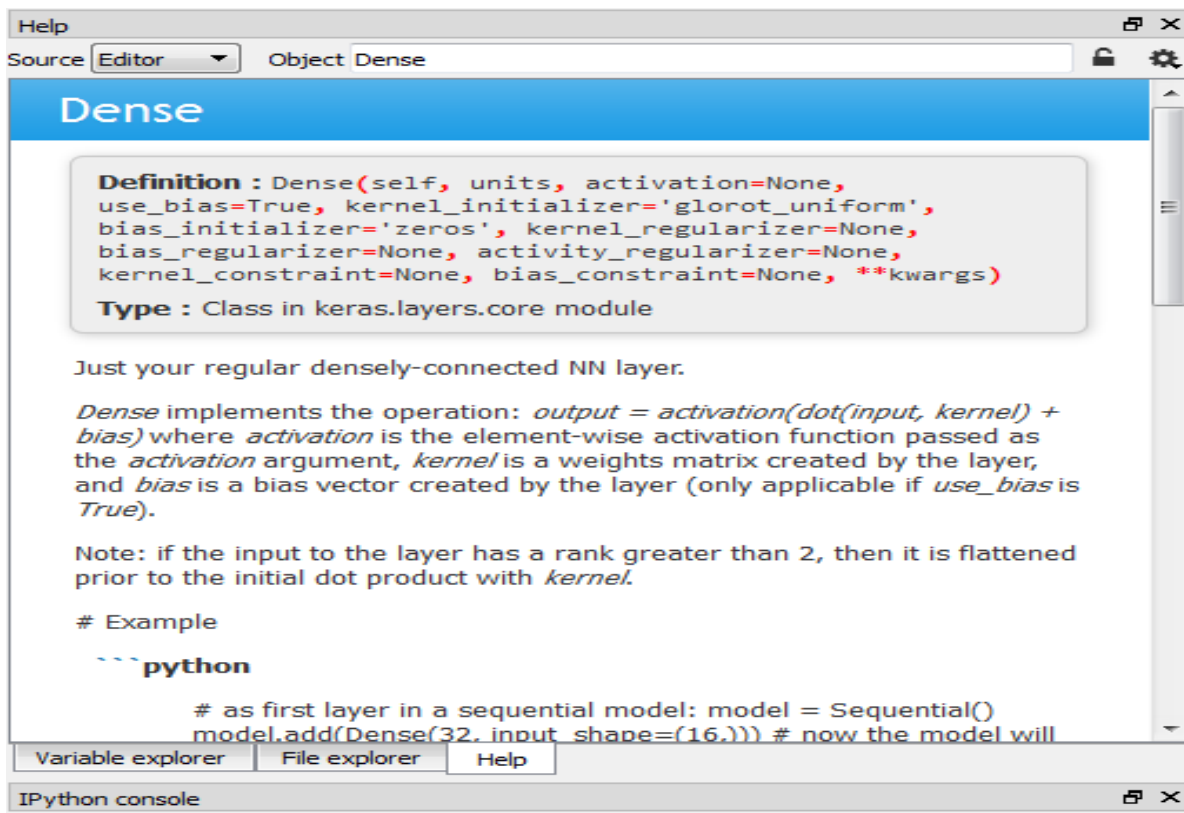


Figure 3.4 : Definition of Dense function in Spyder 3

The rectifier function will be shown as “relu” in the code. The following line adds the first hidden layer to the NN. Relu ( $F(x) = \max(0, x)$ ) is zero for all negative values and linear for all positive values.

`classifier.add( Dense ( units = 6, kernel_initializer = 'uniform', activation = 'relu' , input_dim = ( 12 )))`. The second hidden layer will be created in the following line. There is no need for `input_dim` in this layer.

```
classifier.add( Dense ( units = 6, kernel_initializer = 'uniform', activation = 'relu'))
```

The best accuracy for this deep learning model was a result of having three hidden layers in the ANN. The learning curve would be slightly better. The third and the last hidden layer is just like the second hidden layer.

```
classifier.add( Dense ( units = 6, kernel_initializer = 'uniform' , activation = 'relu' ))
```

Next layer is creating the output layer. Notice that there is only 1 node as output for this ANN. The goal is to have probabilities of different records of the dataset to see whether a user activities will lead in a data breach. If there is only one node as output, then the activation function would be “sigmoid” (see Figure 3.5) . If there are more than one output variable then, the “softmax” function has to be a better choice as activation function.

```
classifier.add ( Dense (units = 1, kernel_initializer = 'uniform', activation = ' sigmoid' ) )
```

Next, There are several types of SGD algorithms. In this step a SGD algorithm needs to be applied on the ANN (as the optimizer). Optimizer is simply the algorithm that would be used to optimize the weights on the neurons of the ANN. According to Kingma and Lei Ba (2015) Adam algorithm is a very efficient optimizer and it has been chosen as the optimizer in this project after parameter tuning. SGD is based on a loss function that needs to be optimized in order to optimizing the parameters(weights). As Weissman et al (2015) mentioned, loss function in here is not sum of the square errors like linear regression. Loss function for the NN can be a logarithmic loss. If the output is binary, then the “binary\_crossentropy” would be selected, and if

the loss function is going to be used in an output with multiple variables, then

“categorical\_crossentropy” would be selected. The metric will define the accuracy and the accuracy of the model will increase because of this method (only accuracy metric has been

chosen). Metric has to be entered as a list. Therefore, the brackets are necessary to define a list.

`classifier.compile( optimizer = 'adam', loss = 'binary_crossentropy', metrics = [ 'accuracy' ] )`

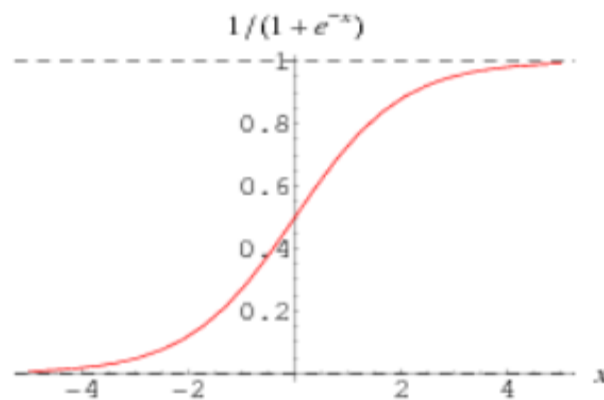


Figure 3.5 : Sigmoid function (Weisstein, n.d.)

### Fitting the ANN and Validation

In the next step, the ANN is going to be fit to the training set. Weights can be updated after each observation, or a batch of observations. Batch size 10, and 100 number of epochs has been chosen. Batch size is the number of observations that after which the weights are going to be updated.

`Classifier.fit( X_train, Y_train, batch_size = 10, epochs = 100 )`

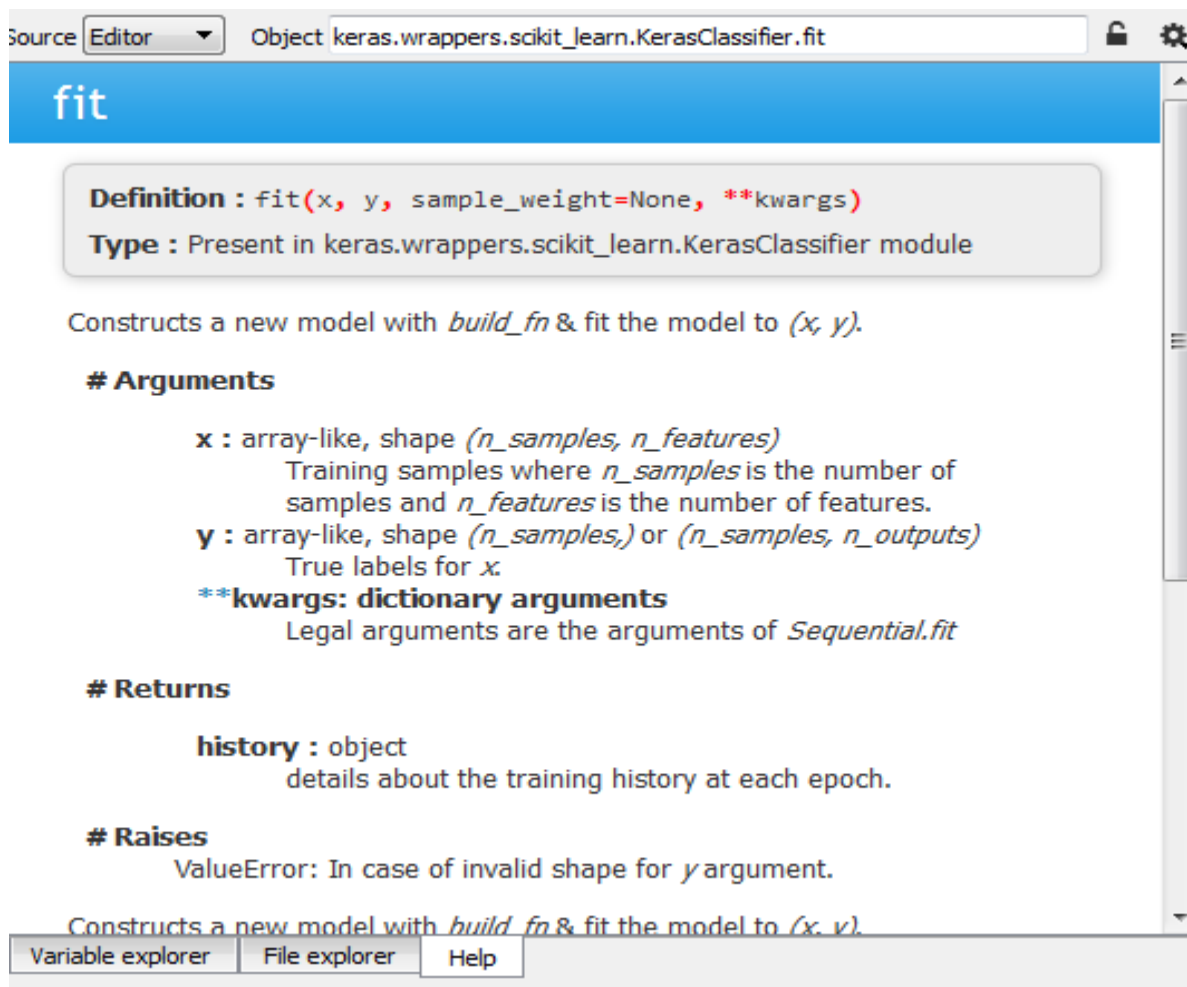


Figure 3.6 : Fit method in Spyder 3

As it has been demonstrated in Chapter IV, over different rounds of epochs the accuracy will be increased.

By utilizing the “predict” method, it is possible to make the prediction on the test set.

```
Y_pred = classifier.predict (X_test)
```

Making this prediction will provide a situation to compare the predicted test set with the accuracy of the model on the training set. Therefore, model could be validated on this particular test set and training set. The validated model can be applied to all the database users and based

on their behaviors, they can be sorted from, most suspicious to least. In order to use the confusion matrix, the prediction vector has to be in the form of true or false.

One of the most important parts in creating such a model is to create a threshold for the predicted vector. If the predicted result is over threshold, then the value would be one (1), and if the predicted result is below the threshold then, zero (0) will be assigned to that value. If the prediction is for very sensitive situations, then a higher threshold has to be chosen. 0.7 have been chosen for this work. Meaning that if one of the employees has more than 70 percent chance of committing a data breach then they will be shown as 1 in the results.

```
Y_predict = (Y_predict > 0.7)
```

After the execution of the above line, the prediction vector will only contain true or false results.

The next step is to making the confusion matrix. The goal here is to evaluate the model for the first time. Confusion matrix results will show the number of correct predictions and number of incorrect predictions on the test set in the format of true positive, false positive, true negative and finally false negative.

```
from sklearn.metrics import confusion_matrix
```

```
evaluate = confusion_matrix(Y_test, Y_pred)
```

The accuracy of the model could be simply calculated at this step by dividing the number of the correct predictions by 2000 which is the size of test set that has been chosen.

If the accuracy that the confusion matrix is showing on the test set was the same as the accuracy of the model on the train set, then the model has been validated. However, in order to make sure that the accuracy of the model is correct, K-fold Cross Validation technique has been

utilized to run the model on different random training sets and test sets to make sure that the accuracies don't fluctuate. If there was low bias and low variance between the outcome accuracies of K-fold Cross Validation technique then the model is reliable and is ready to use. A good model will have low bias and low variance.

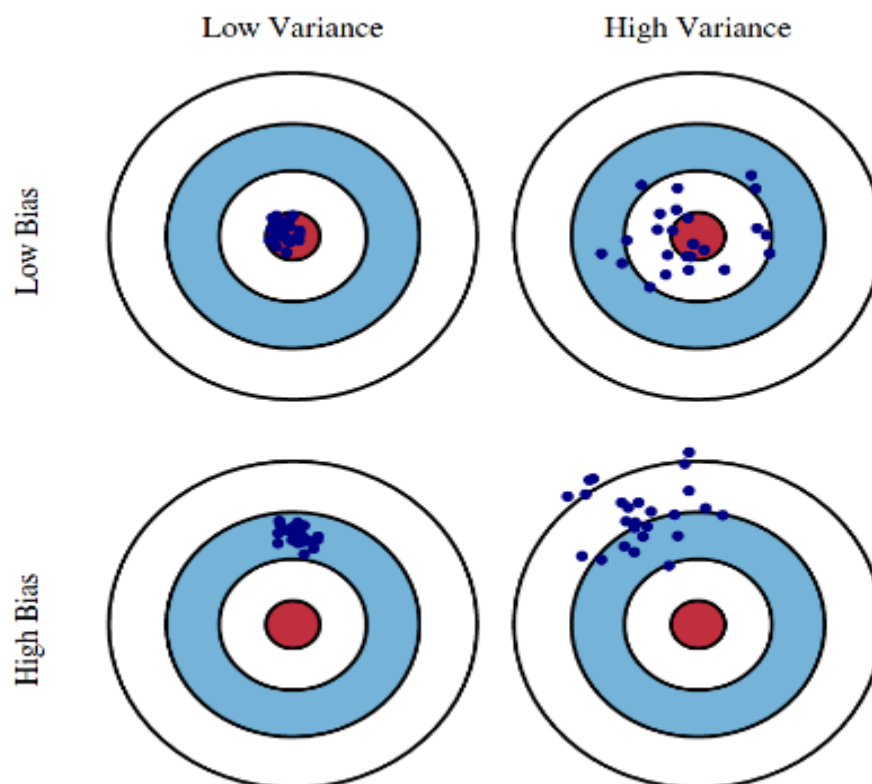


Figure 3.7 : Bias and Variance trade off (Fortmann-Roe, 2012)

As has been shown in the Figure 3.7 , the desired model has to have a low bias and low variance. Since the model accuracy could be different on different training session with different training and test sets it is very important to evaluate the model. K-fold Cross Validation will split the training set into “K” folds, and then the model would be validated based on “K” different combinations of training sets and test sets the model can be validated. “K” is usually equals to

10. The model performance would be much more relevant by validating it by using K-fold Cross Validation technique. K-fold Cross Validation technique belongs to “scikit\_learn” so it is needed to add this function to “KerasClassifier”.

```
from keras.wrappers.skicikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
```

The “KerasClassifier” expects a function for one of its arguments. Therefore, a function has to be written to meet the “KerasClassifier” requirement. The following function has the entire architecture of the deep learning model that has been created in this project. This function will return the classifier.

```
def build_classifier():
    classifier.add( Dense ( units = 6, kernel_initializer = 'uniform', activation = 'relu' ,
        input_dim = ( 12 )))
    classifier.add( Dense ( units = 6, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add( Dense ( units = 6, kernel_initializer = 'uniform' , activation = 'relu' ))
    classifier.add ( Dense (units = 1, kernel_initializer = 'uniform', activation = ' sigmoid' ))
    return classifier
```

After running the above function, the classifier object is an object that is local for the function. So a new classifier needs to be created to apply the K-fold Cross Validation technique. The training part for the new classifier is the same as what was implemented earlier with the “batch\_size” of 10 and 100 number of “epochs”.

```
Classifier = classifier = KerasClassifier(build_fn = build_classifier, batch_size = 10, epochs =
100)
```

K-fold Cross Validation is going to be used to return a relevant measure of accuracy in the ANN.

So a new variable is created to have the outcome of the K-fold Cross Validation which are the accuracies of 10 different training sets and test sets.

```
Accuracies = cross_val_score(estimator = classifier, X = X_train, Y = Y_train, cv = 10, n_jobs = -1)
```

In the figure 3. and figure 3. the definition of “cross\_val\_score” function has been demonstrated.

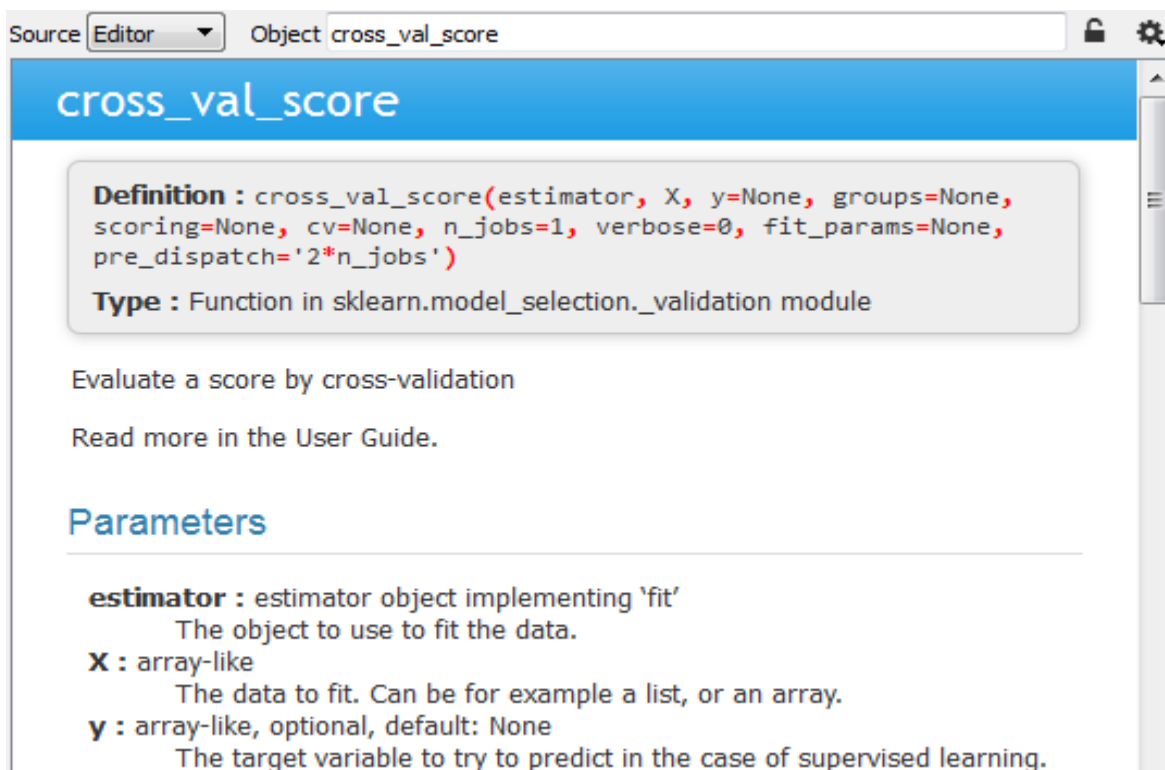


Figure 3.8 : Inspecting K-fold Cross Validation method in Spyder 3 (part 1)

The “estimator” is the classifier that was built to use to fit the data. The training sets have been added after the “estimator”. “cv” is the number of folds in the K-fold Cross Validation.



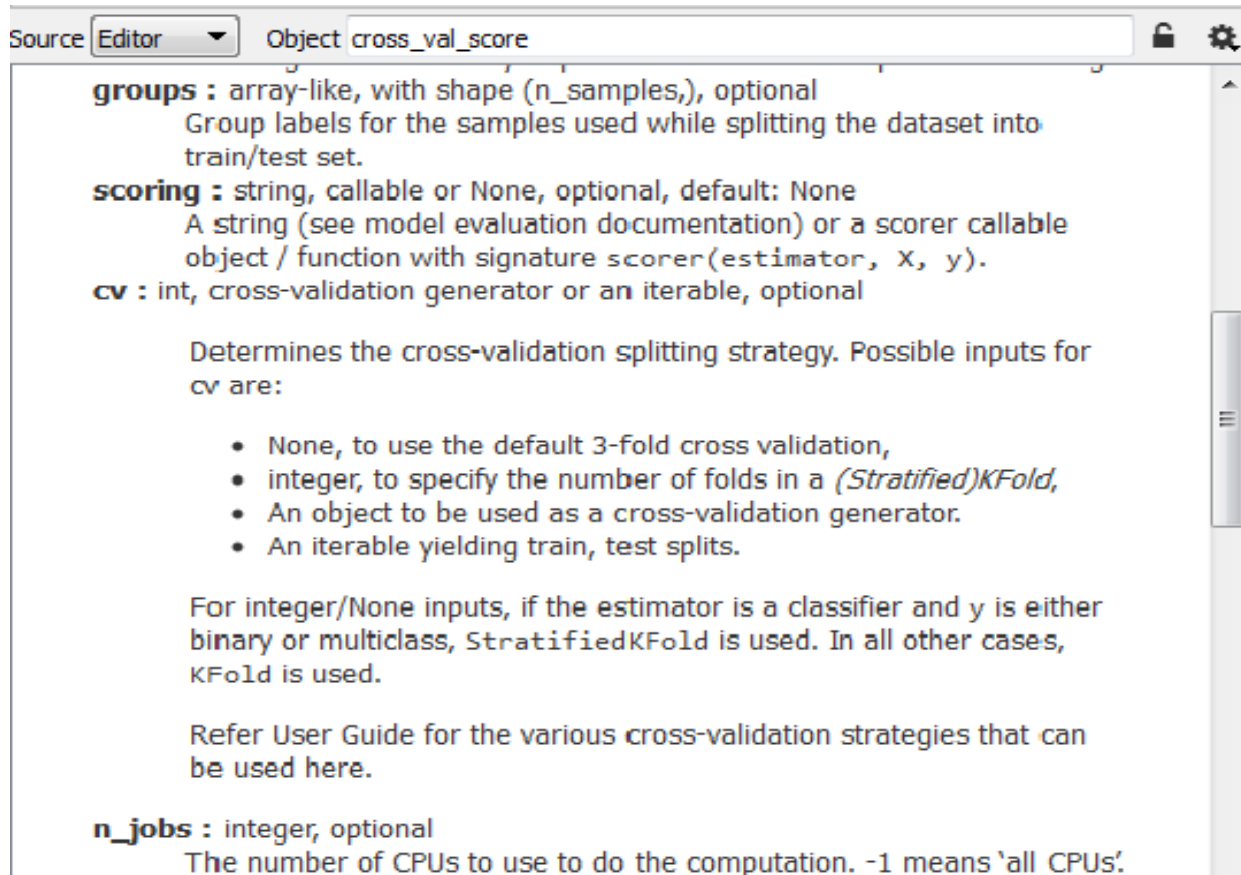


Figure 3.9 : Inspecting K-fold Cross Validation method in Spyder 3 (part 2)

Since the training and applying K-fold Cross Validation on 10 different training and test sets would take a long time, “`n_jobs = -1`” has been used to utilize parallel computation and using all the CPU’s of the system to accelerate the speed.

After this step the “mean” and the “variance” of different accuracies of the model will be calculated. The accuracies vector contains the accuracies of the model on 10 different training sets and datasets.

```
mean = accuracies.mean()
```

```
variance = accuracies.std()
```

## Improving the Model

**Dropout regularization.** Over fitting and under fitting are two common problems in an ANN. Over fitting usually happens when after training and fitting the ANN there is a large difference in accuracies between the training set and the test set. When over fitting happens there is a much higher accuracy between the training set and the test set and this indicates that the model learned too much on the training set. Moreover, if there is a high variance (some high accuracies and some low accuracies) after applying K-fold Cross validation then it means that the model is over fitted. High variance is one of the most conspicuous characteristics of an over fitted model. “Dropout” would be applied on one or more layers of the NN. This technique will prevent the over fitting problem by randomly disabling some of the neurons to prevent them to be too dependent on each other when they are learning the correlations of the data. In order to apply “Dropout” the following code has to be implemented in the ANN.

```
from keras.layers import Dropout
```

To eliminate the over fitting, the Dropout should be applied after creating each layer of the NN.

For example, for the first layer the Dropout could be implemented as follow.

```
classifier.add( Dense ( units = 6, kernel_initializer = 'uniform', activation = 'relu' , input_dim
= ( 12 )))
classifier.add ( Dropout ( p = 0.2 ) )
```

The argument “p” in Dropout indicates the fraction of the neurons that are needed to drop or disable. In the above example, “p = 0.2” means that 20 percent of the neurons will be disabled at each iteration. If the over fitting problem still exist after applying the Dropout technique to all the layers of the model then, the value of “p” should be increased until the over fitting has been

eliminated. If  $p = 1$ , then the weights on that layer of neurons will not be update. In the other word, that layer will not learn at all and it would cause under fitting problem.

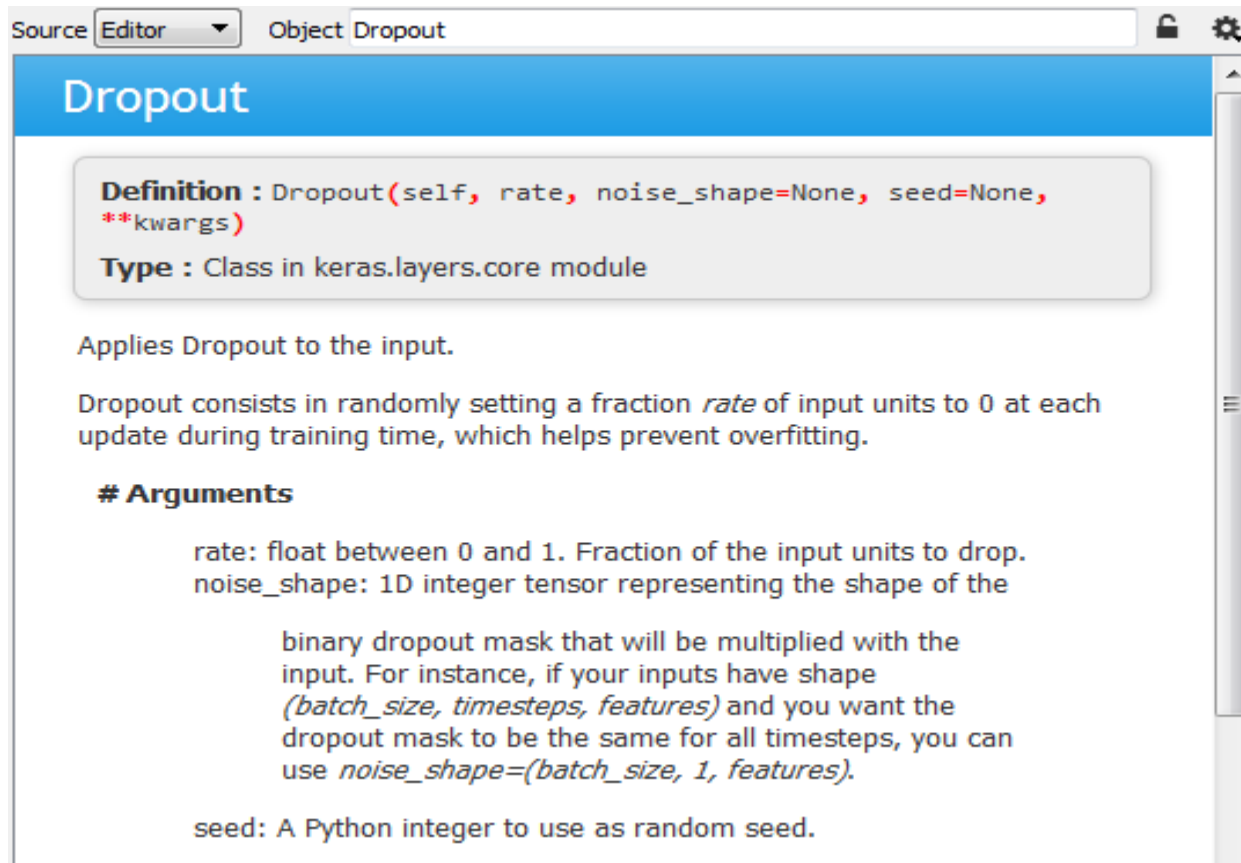


Figure 3.10 : Inspecting Dropout in Spyder 3

**Parameter tuning.** Parameter tuning can be used to find the best optimal values for the model such as number of neurons in hidden layers, type of optimizer, and the number of epochs as well as the batch size for each iteration. This phase is a critical phase in developing the model since there could be better values to improve the performance and accuracy of the model. The “GridSearchCV” class can be used to implement parameter tuning.

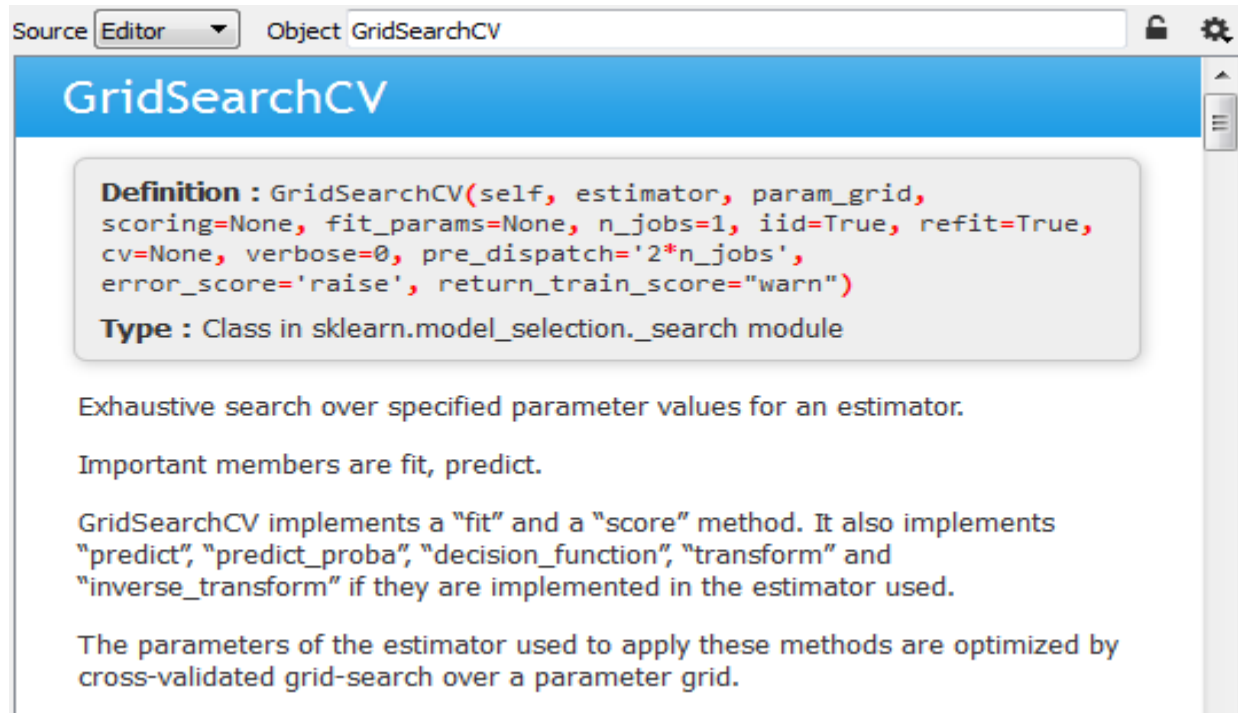


Figure 3.11 : Inspecting GridSearchCV in Spyder 3 (part 1)

“GridSearch” will test several combination of different values for the model and will return the best values. In this step the parameters of the model will be tuned to reach a better model.

Parameter tuning technique for this project has been implemented as follow.

```

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSeachCV
from keras.models import Sequential
from keras.layers import Dense

def build_classifier(optimizer):

    classifier.add( Dense ( units = 6, kernel_initializer = 'uniform', activation = 'relu',
input_dim = ( 12 )))

    classifier.add( Dense ( units = 6, kernel_initializer = 'uniform', activation = 'relu'))

```

```

classifier.add( Dense ( units = 6, kernel_initializer = 'uniform' , activation = 'relu '))
classifier.add ( Dense (units = 1, kernel_initializer = 'uniform', activation = ' sigmoid' ))
classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics =
['accuracy'])
return classifier

classifier = KerasClassifier ( build_fn = build_classifier)

parameters = { 'batch_size': [25, 32], 'epochs': [100, 500], 'optimizer': ['adam', 'rmsprop']}
grid_search = GridSearchCV(estimator = classifier, param_grid = parameters, scoring =
'accuracy', cv = 10)

grid_search.fit(X_train, Y_train)

better_parameters = grid_search.best_params_
better_accuracy = grid_search.best_score_

```

Instead of having the name of the optimizer in compile method the argument “optimizer” has been given as the argument of build\_classifier function. Since, the goal here is also test the type of optimizer that should be used in the model as one of the parameters. “parameters” is a dictionary type and contains hyper parameters that have been used in the model. In order to see which optimizer is a better fit for the model the optimizer feature has been used in this dictionary. The model will be tested with both “adam” and “rmsprop” optimizers who are SGD optimizing algorithms to compare which one would be a better fit for the model.

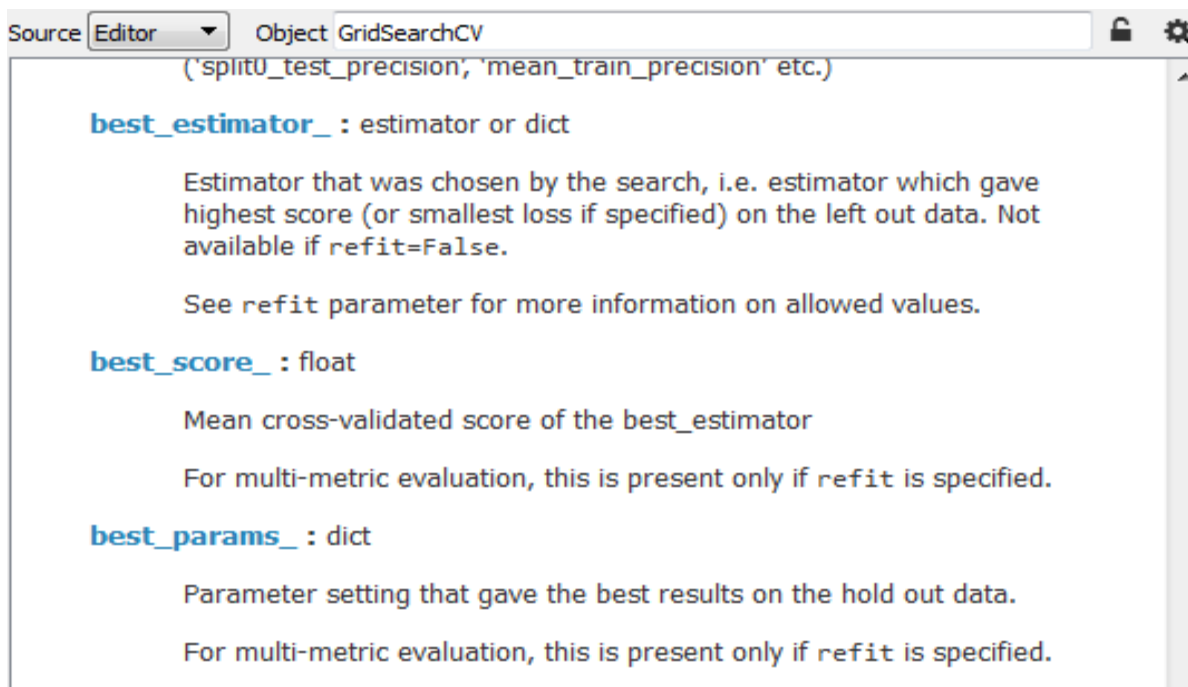


Figure 3.12 : Inspecting GridSearchCV in Spyder 3 (part 2)

Running the above parameter tuning technique will take several hours since the model will be run and tested against different parameters to achieve a better accuracy. In the end, the `grid_search` method will return the best parameters and best accuracy for the model as “`grid_search.best_params_`” and “`grid_search.best_score_`”. These numbers would be demonstrated as “`better_parameters`” and “`better_accuracy`”.

## Chapter IV: Results

In this chapter, the results of the running the python code that have been explained in Chapter III has been demonstrated step by step. After importing the libraries, the dataset should be preprocessed to use in the ANN. Therefore, the dataset has been imported to the Spyder 3 platform.

### Data Preprocessing

The variable explorer section in the Spyder 3 provides the option of seeing values of different vectors or data dictionaries and variables. Notice that the type of data in the dataset is “DataFrame” which contains both numerical and categorical values.

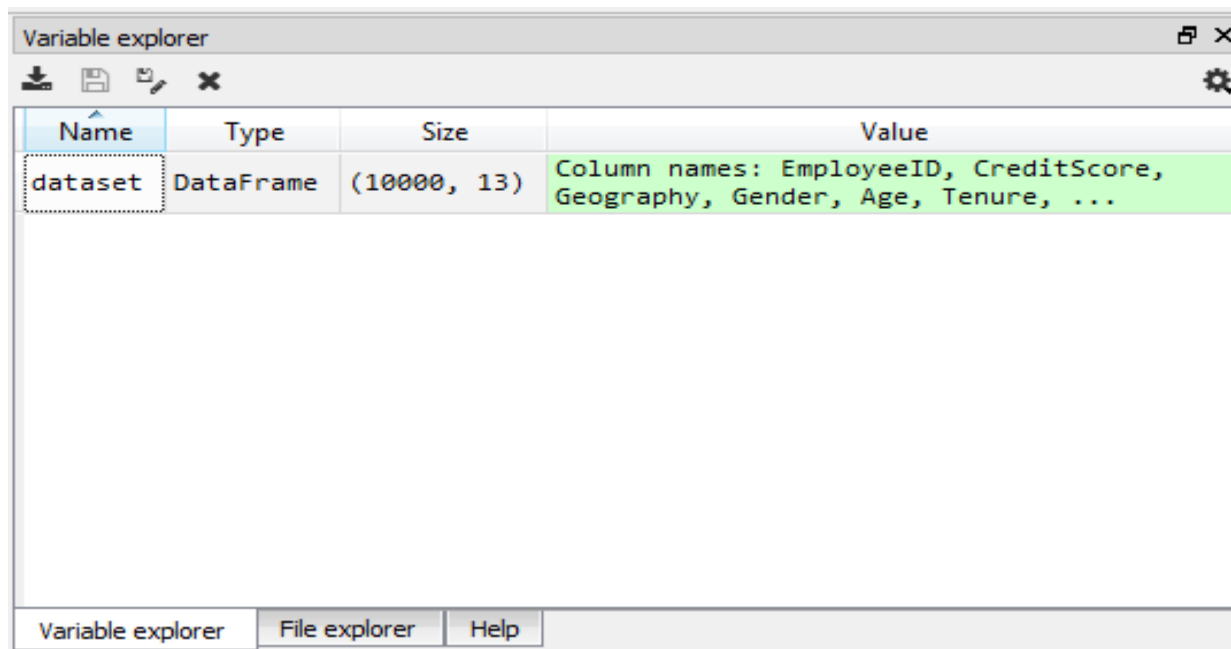
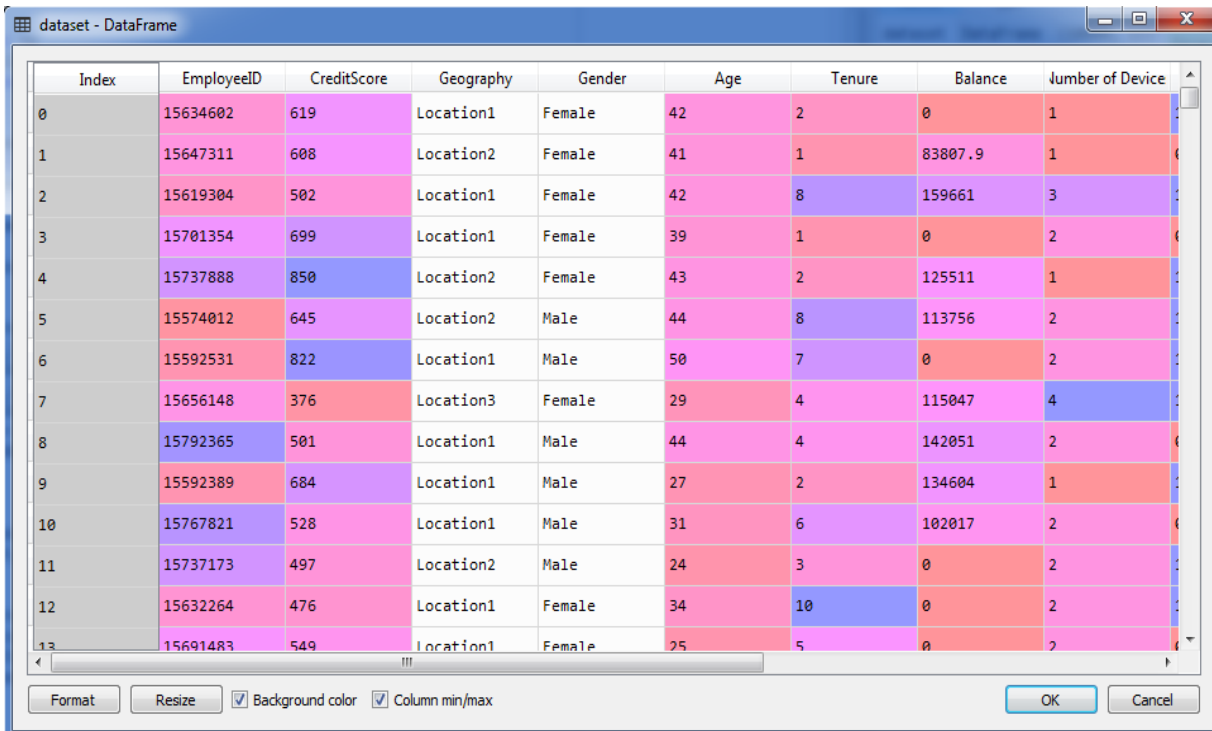


Figure 4.1: Imported dataset in variable explorer in Spyder 3

After opening the dataset that has been imported, in variable explorer section(as shown in Figure 4.1) the values of all the variables would be shown. As it has been shown in Figure 4.1, there are

10,000 rows and 13 columns in the dataset which represent the information about the 10,000 employees.



Index	EmployeeID	CreditScore	Geography	Gender	Age	Tenure	Balance	Lumber of Device
0	15634602	619	Location1	Female	42	2	0	1
1	15647311	608	Location2	Female	41	1	83807.9	1
2	15619304	502	Location1	Female	42	8	159661	3
3	15701354	699	Location1	Female	39	1	0	2
4	15737888	850	Location2	Female	43	2	125511	1
5	15574012	645	Location2	Male	44	8	113756	2
6	15592531	822	Location1	Male	50	7	0	2
7	15656148	376	Location3	Female	29	4	115047	4
8	15792365	501	Location1	Male	44	4	142051	2
9	15592389	684	Location1	Male	27	2	134604	1
10	15767821	528	Location1	Male	31	6	102017	2
11	15737173	497	Location2	Male	24	3	0	2
12	15632264	476	Location1	Female	34	10	0	2
13	15691483	549	Location1	Female	25	5	0	2

Figure 4.2: Values of the Dataset

In the next step of the data preprocessing stage, the dataset will be splitted into two different parts. The independent variable matrix or the matrix of features (X), and the dependent variable vector (Y). As mentioned in Chapter III, the indexes in python start from zero. Therefore, the indexes for matrix of features(X) would be from 1 to 12 and the index of the dependent variable vector (Y) is 12. The DataFrame type will be converted to object and the object type will not be shown in Spyder 3. Therefore, in order to see the data again, X should be converted into the DataFrame type. This could be done by running the below line of code.

```
df = pd.DataFrame(X)
```



Variable explorer

Name	Type	Size	Value
X	object	(10000, 11)	ndarray object of numpy module
dataset	DataFrame	(10000, 13)	Column names: EmployeeID, CreditScore, Geography, Gender, Age, Tenure, ...
df	DataFrame	(10000, 11)	Column names: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
y	int64	(10000,)	array([1, 0, 1, ..., 1, 1, 0], dtype=int64)

Variable explorer | File explorer | Help

Figure 4.3: Variable explorer after splitting the dataset into X and Y

y - NumPy array

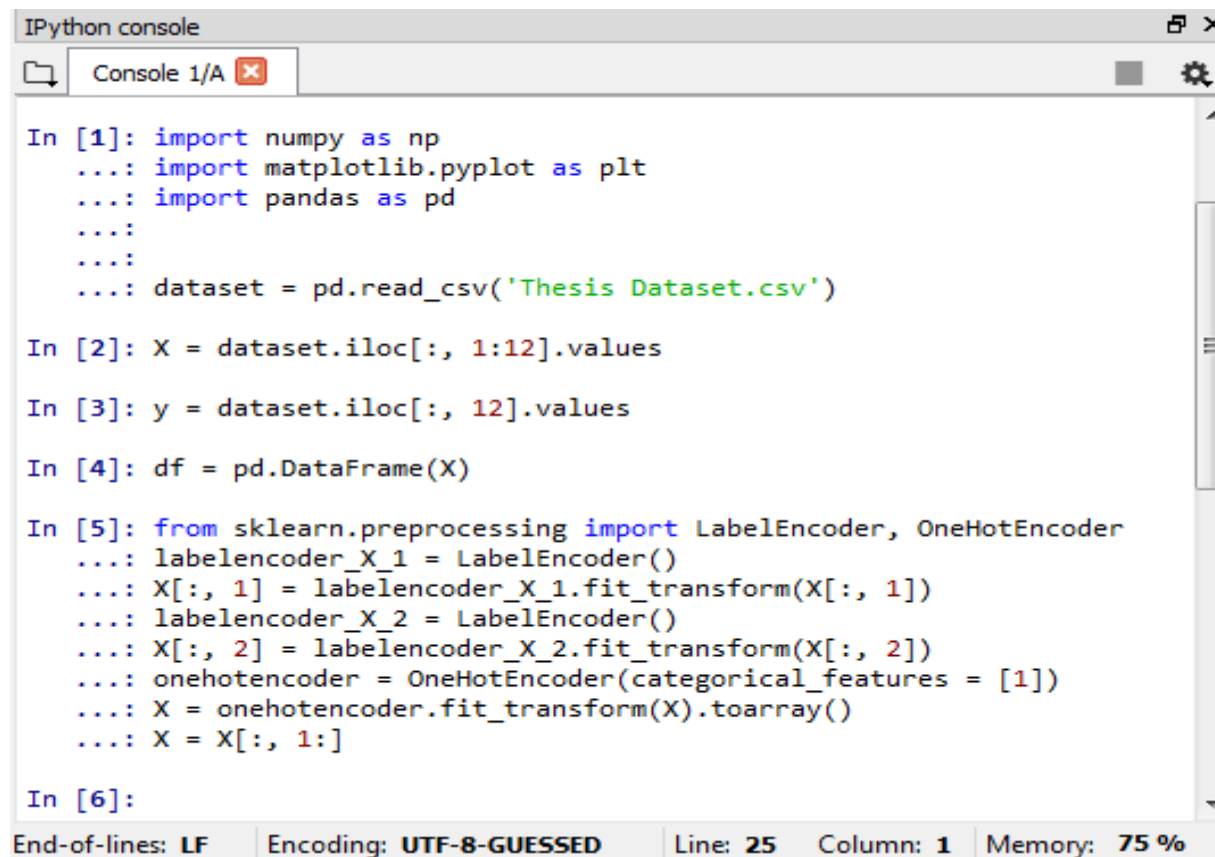
	0
0	1
1	0
2	1
3	0
4	0
5	1
6	0
7	1
8	0
9	0
10	0
11	0
12	0

Format | Resize |  Background color

OK | Cancel

Figure 4.4 : Dependent variable vector (Y)

As it has been shown in Figure 4.3, the dependent variable vector (Y) has only 1 column and matrix of features (X) has 11 columns. Y contains the binary outcome of the Data Breach process.



```
IPython console
Console 1/A

In [1]: import numpy as np
...: import matplotlib.pyplot as plt
...: import pandas as pd
...:
...:
...: dataset = pd.read_csv('Thesis Dataset.csv')

In [2]: X = dataset.iloc[:, 1:12].values

In [3]: y = dataset.iloc[:, 12].values

In [4]: df = pd.DataFrame(X)

In [5]: from sklearn.preprocessing import LabelEncoder, OneHotEncoder
...: labelencoder_X_1 = LabelEncoder()
...: X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
...: labelencoder_X_2 = LabelEncoder()
...: X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
...: onehotencoder = OneHotEncoder(categorical_features = [1])
...: X = onehotencoder.fit_transform(X).toarray()
...: X = X[:, 1:]

In [6]:

End-of-lines: LF | Encoding: UTF-8-GUESSED | Line: 25 | Column: 1 | Memory: 75 %
```

Figure 4.5: IPython console

Figure 4.5, demonstrates the results of the encoding and fixing the dummy variable trap that has been explained in Chapter III. Matrix of features has been encoded in this section. Since Y already has numerical (binary in this case) values, there is no need for encoding the dependent variable vector. All the categorical values such as location and gender in the matrix of features have been transformed into numerical values.

	0	1	2	3	4	5	6	7
0	0	0	619	0	42	2	0	1
1	1	0	608	0	41	1	83807.9	1
2	0	0	502	0	42	8	159661	3
3	0	0	699	0	39	1	0	2
4	1	0	850	0	43	2	125511	1
5	1	0	645	1	44	8	113756	2
6	0	0	822	1	50	7	0	2
7	0	1	376	0	29	4	115047	4
8	0	0	501	1	44	4	142051	2
9	0	0	684	1	27	2	134604	1
10	0	0	528	1	31	6	102017	2
11	1	0	497	1	24	3	0	2

Figure 4.6: Encoding the matrix of features

Figure 4.6 shows the matrix of features after being encoded. After this step, matrix of features and the dependent variable vector has been splitted into training set and test sets. Eighty percent of data has been considered as the training set and twenty percent for the test set.

Name	Type	Size	Value
X	float64	(10000, 12)	array([[0.0000000e+00, 0.0000000e+00, 6.1900000e+02, ..., 1.0000000e+0 ...
X_test	float64	(2000, 12)	array([[0.0000000e+00, 1.0000000e+00, 5.9700000e+02, ..., 1.0000000e+0 ...
X_train	float64	(8000, 12)	array([[1.0000000e+00, 0.0000000e+00, 6.6700000e+02, ..., 0.0000000e+0 ...
dataset	DataFrame	(10000, 13)	Column names: EmployeeID, CreditScore, Geography, Gender, Age, Tenure, ...
df	DataFrame	(10000, 11)	Column names: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
y	int64	(10000,)	array([1, 0, 1, ..., 1, 1, 0], dtype=int64)
y_test	int64	(2000,)	array([0, 1, 0, ..., 0, 0, 0], dtype=int64)
y_train	int64	(8000,)	array([0, 0, 0, ..., 0, 0, 1], dtype=int64)

Figure 4.7 : Demonstration of test set and training set in variable explorer

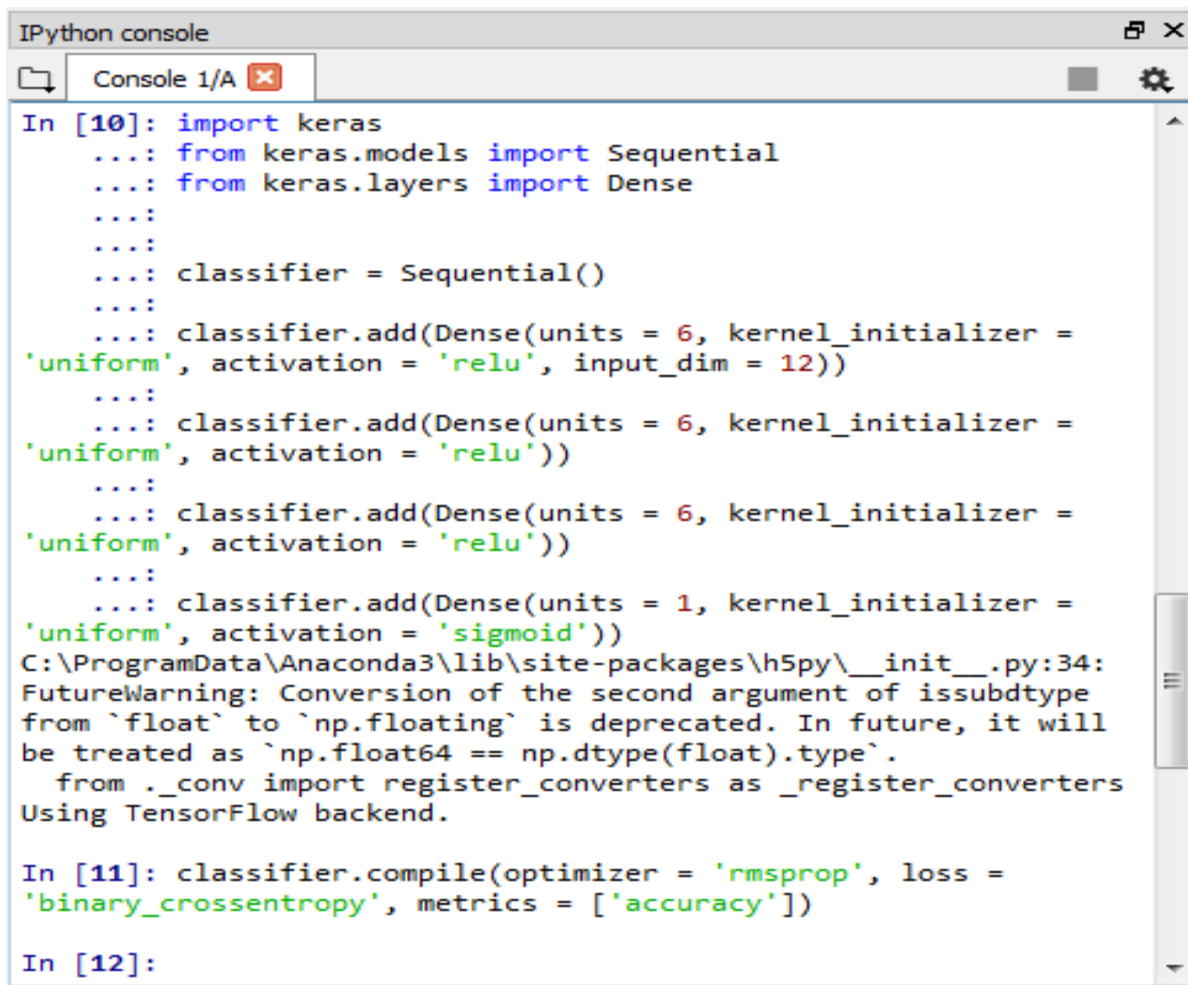
As it has shown in Figure 4.7, X\_test contains two thousand, X\_train eight thousand, y\_test two thousand and eventually y\_train eight thousand records. In order to ease the computation process, feature scaling has been applied to both training set and the test set (X\_train and X\_test), and the results of applying feature scaling on X\_train has been demonstrated in Figure 4.8.

	0	1	2	3	4
0	1.74309	-0.569844	0.169582	-1.09169	-0.464608
1	-0.573694	1.75487	-2.30456	0.916013	0.301026
2	-0.573694	-0.569844	-1.1912	-1.09169	-0.943129
3	1.74309	-0.569844	0.0355658	0.916013	0.109617
4	1.74309	-0.569844	2.05611	-1.09169	1.73659
5	-0.573694	1.75487	1.29325	-1.09169	-0.177495
6	-0.573694	-0.569844	1.61283	0.916013	0.779547
7	1.74309	-0.569844	-0.541734	0.916013	0.205321
8	1.74309	-0.569844	-0.149995	0.916013	3.55497
9	-0.573694	-0.569844	-0.29432	-1.09169	-0.656016
10	-0.573694	-0.569844	0.324216	-1.09169	-0.560312
11	-0.573694	-0.569844	0.612865	0.916013	1.44948

Figure 4.8 :X\_train after feature scaling

## Building the ANN

In this step the classifier will be built based on what was discussed in Chapter III. There are twelve different features in the matrix of features. So, the input of the first layer of ANN has twelve neurons. The model has three hidden layers. Each hidden layer has six neurons and the output layer contains only one neuron which is the desired probability and the main goal of this project. Based on this probability, the amount of risk that a certain employee could possess for the organization can be determined. Note that the activation function for all the layers except the output layer is rectifier function. Sigmoid function is the activation function for the output layer.



```

IPython console
Console 1/A

In [10]: import keras
...: from keras.models import Sequential
...: from keras.layers import Dense
...:
...:
...: classifier = Sequential()
...:
...: classifier.add(Dense(units = 6, kernel_initializer =
'uniform', activation = 'relu', input_dim = 12))
...:
...: classifier.add(Dense(units = 6, kernel_initializer =
'uniform', activation = 'relu'))
...:
...: classifier.add(Dense(units = 6, kernel_initializer =
'uniform', activation = 'relu'))
...:
...: classifier.add(Dense(units = 1, kernel_initializer =
'uniform', activation = 'sigmoid'))
C:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:34:
FutureWarning: Conversion of the second argument of issubdtype
from `float` to `np.floating` is deprecated. In future, it will
be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.

In [11]: classifier.compile(optimizer = 'rmsprop', loss =
'binary_crossentropy', metrics = ['accuracy'])

In [12]:

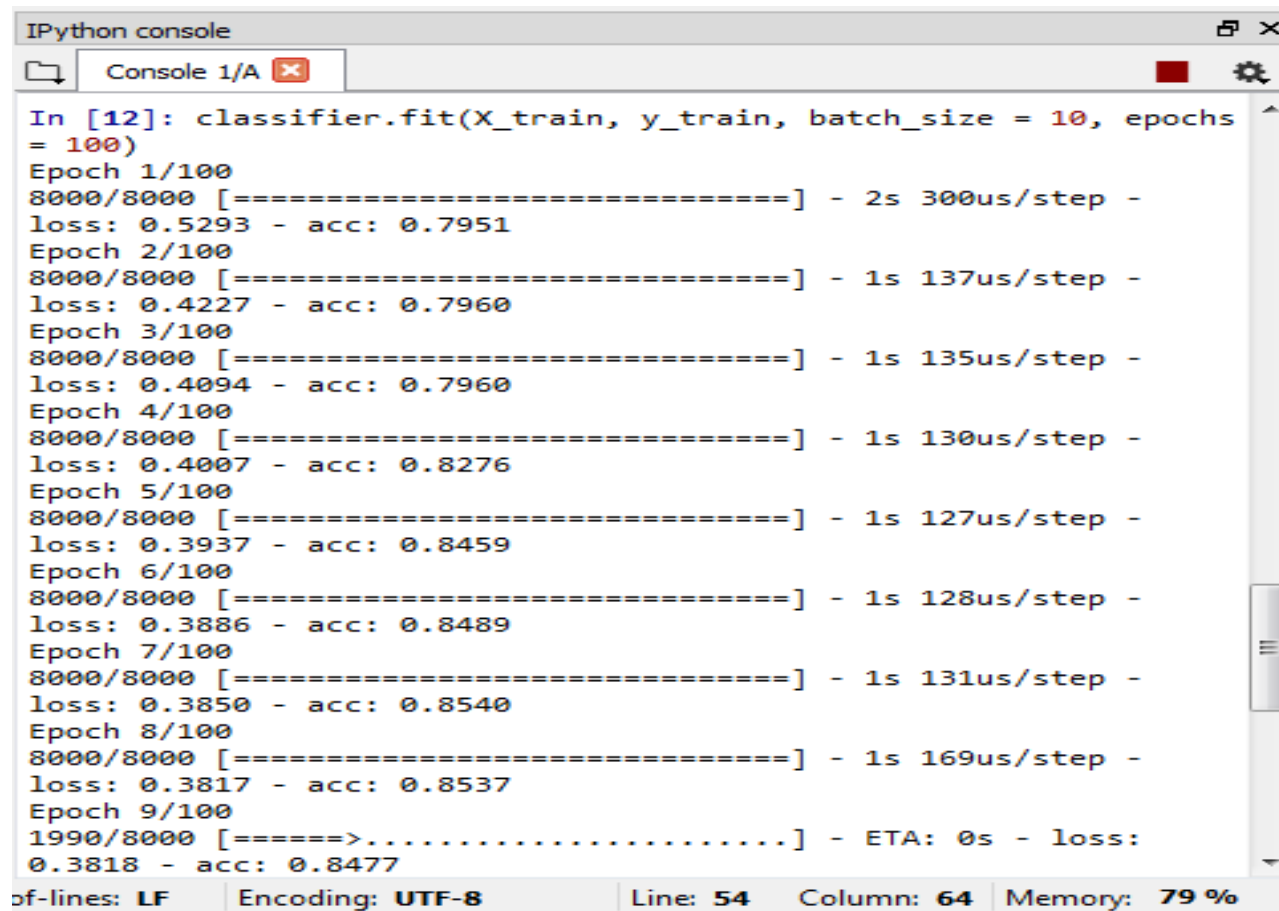
```

Figure 4.9: Building the ANN

Figure 4.9 shows the results of the code for building the NN. At this point, the dataset can be fitted to the model.

### Fitting the ANN and Validation

As it has been shown below in Figure 4.10 and Figure 4.11, after each epoch, the accuracy is increasing and the weights are being updated.

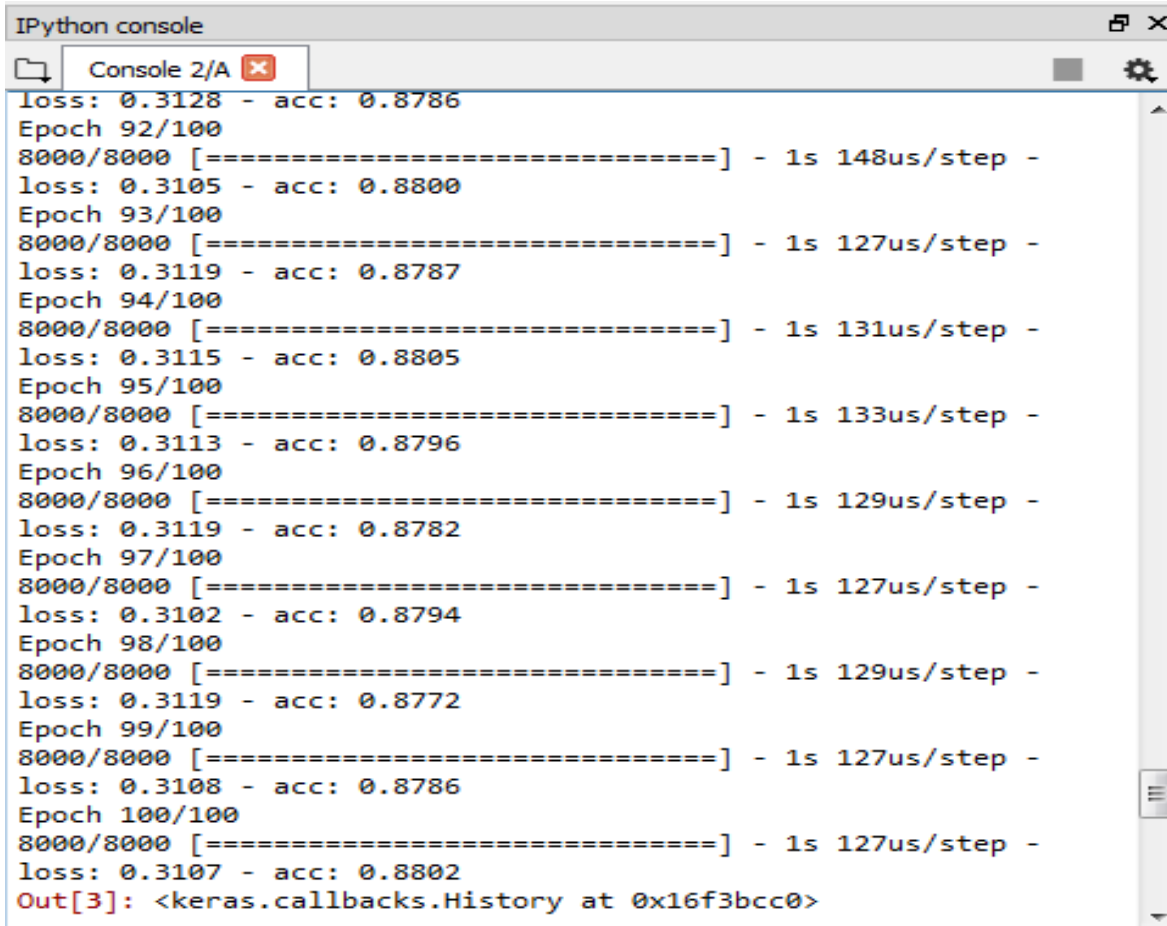


```

IPython console
Console 1/A x
In [12]: classifier.fit(X_train, y_train, batch_size = 10, epochs
= 100)
Epoch 1/100
8000/8000 [=====] - 2s 300us/step -
loss: 0.5293 - acc: 0.7951
Epoch 2/100
8000/8000 [=====] - 1s 137us/step -
loss: 0.4227 - acc: 0.7960
Epoch 3/100
8000/8000 [=====] - 1s 135us/step -
loss: 0.4094 - acc: 0.7960
Epoch 4/100
8000/8000 [=====] - 1s 130us/step -
loss: 0.4007 - acc: 0.8276
Epoch 5/100
8000/8000 [=====] - 1s 127us/step -
loss: 0.3937 - acc: 0.8459
Epoch 6/100
8000/8000 [=====] - 1s 128us/step -
loss: 0.3886 - acc: 0.8489
Epoch 7/100
8000/8000 [=====] - 1s 131us/step -
loss: 0.3850 - acc: 0.8540
Epoch 8/100
8000/8000 [=====] - 1s 169us/step -
loss: 0.3817 - acc: 0.8537
Epoch 9/100
1990/8000 [=====>.....] - ETA: 0s - loss:
0.3818 - acc: 0.8477
of-lines: LF | Encoding: UTF-8 | Line: 54 | Column: 64 | Memory: 79 %

```

Figure 4.10 : Fitting the training set (part 1)



```

IPython console
Console 2/A
loss: 0.3128 - acc: 0.8786
Epoch 92/100
8000/8000 [=====] - 1s 148us/step -
loss: 0.3105 - acc: 0.8800
Epoch 93/100
8000/8000 [=====] - 1s 127us/step -
loss: 0.3119 - acc: 0.8787
Epoch 94/100
8000/8000 [=====] - 1s 131us/step -
loss: 0.3115 - acc: 0.8805
Epoch 95/100
8000/8000 [=====] - 1s 133us/step -
loss: 0.3113 - acc: 0.8796
Epoch 96/100
8000/8000 [=====] - 1s 129us/step -
loss: 0.3119 - acc: 0.8782
Epoch 97/100
8000/8000 [=====] - 1s 127us/step -
loss: 0.3102 - acc: 0.8794
Epoch 98/100
8000/8000 [=====] - 1s 129us/step -
loss: 0.3119 - acc: 0.8772
Epoch 99/100
8000/8000 [=====] - 1s 127us/step -
loss: 0.3108 - acc: 0.8786
Epoch 100/100
8000/8000 [=====] - 1s 127us/step -
loss: 0.3107 - acc: 0.8802
Out[3]: <keras.callbacks.History at 0x16f3bcc0>

```

Figure 4.11 : Fitting the training set (part 2)

The accuracy of the model will increase as the number of training data increases. The accuracy of the model for this specific training set and the test set is at 88.02 percent. In the Figure 4.12, the results of the prediction of the model on the test set has been demonstrated. Since the threshold for this model has been set on 70 percent, any outcome which has the probability of higher than the threshold will be considered as red flag.

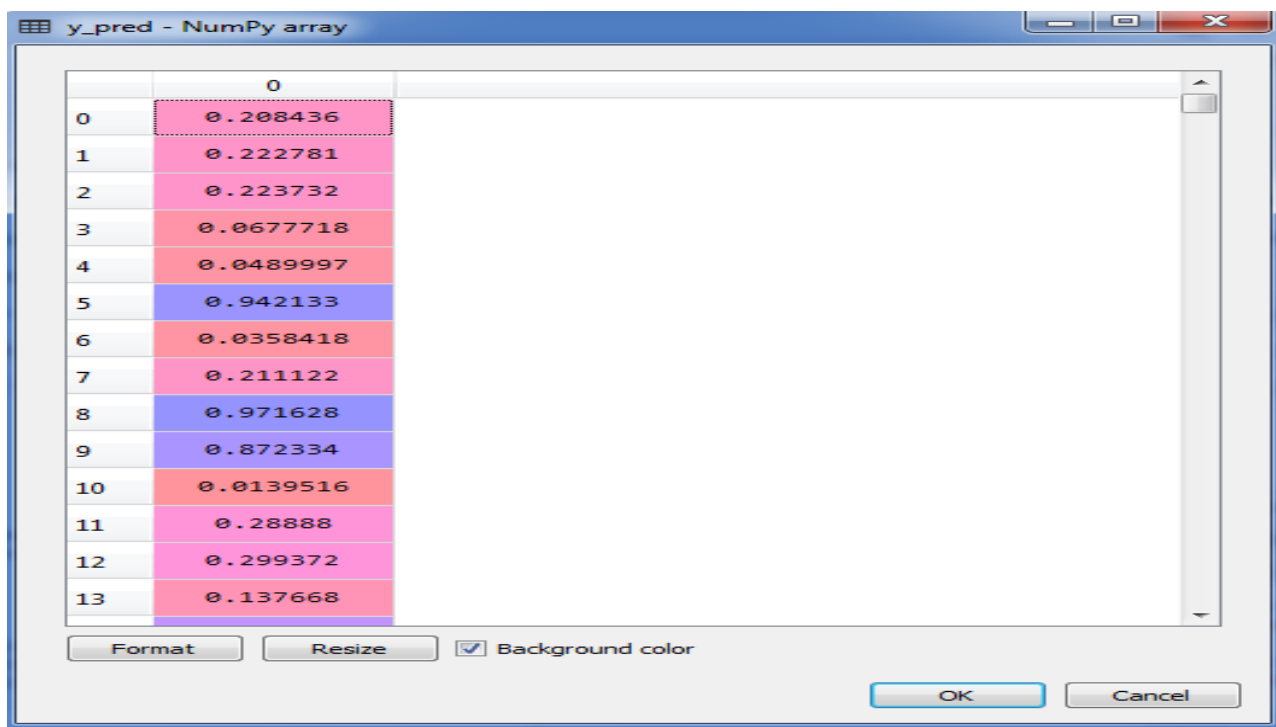


Figure 4.12 : The predictions of the model on test set

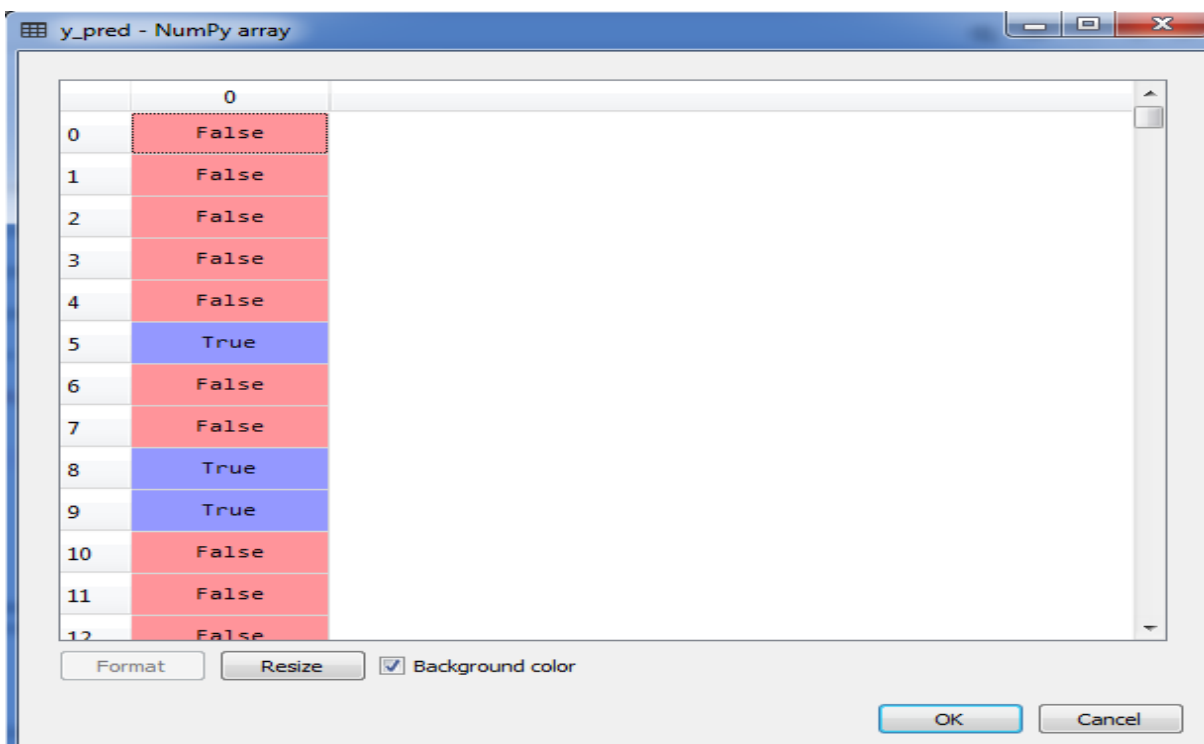
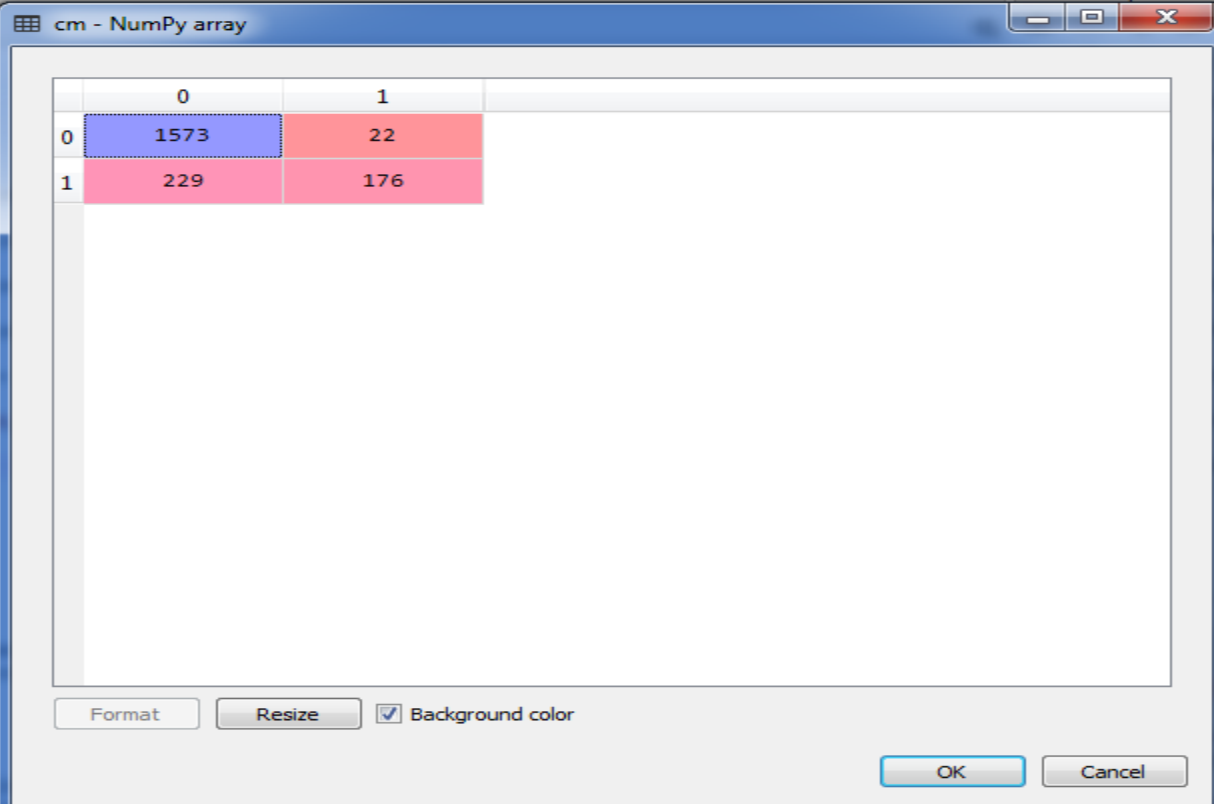


Figure 4.13 : Results of the prediction in Boolean



In order to demonstrate a better graphical outcome of the model, the results of prediction on test set have been transformed to Boolean. Figure 4.13 demonstrates the outcome of the model on `y_pred` vector. Any row that has probability of more than 70 percent would be count as a suspicious behavior and a potential threat to the system. Having the results in Boolean will also help us to have the confusion matrix. The next step is to validate the model on this particular training and test set using the confusion matrix.



	0	1
0	1573	22
1	229	176

Figure 4.14 : The confusion matrix

Figure 4.14 demonstrates the confusion matrix on this particular training and test set. Out of 2000 new observations there are 1573 true negative, 22 false negative, 229 false positive, and eventually 176 true positive predictions. The accuracy of the model on this training and test set

can be calculated by adding the number of true positive and true negative results from the confusion matrix and then dividing the results by 2000.

$$(1573 + 176) / 2000 = 0.8745$$

In order to validate the model in terms of bias and variance the K-fold Cross validation has been implemented in the next step.

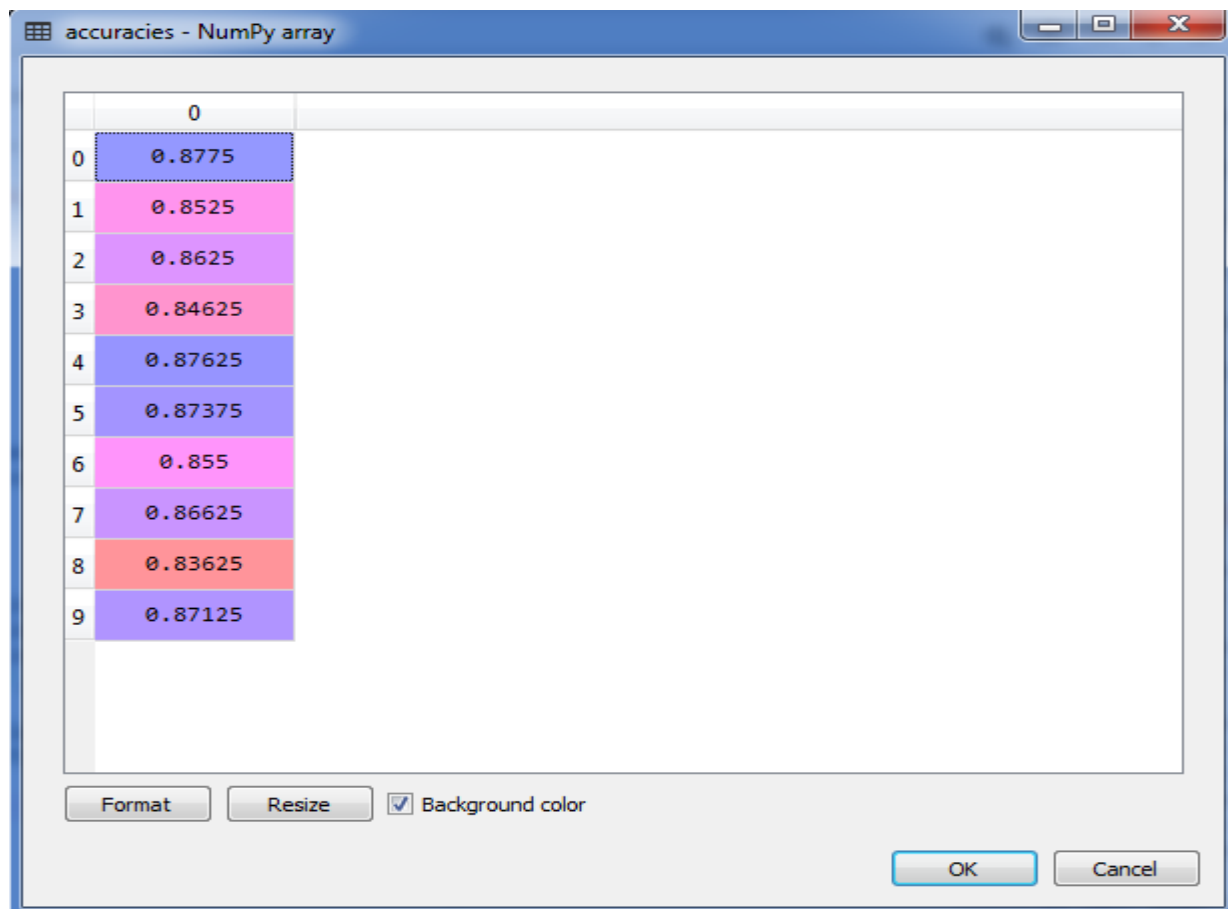
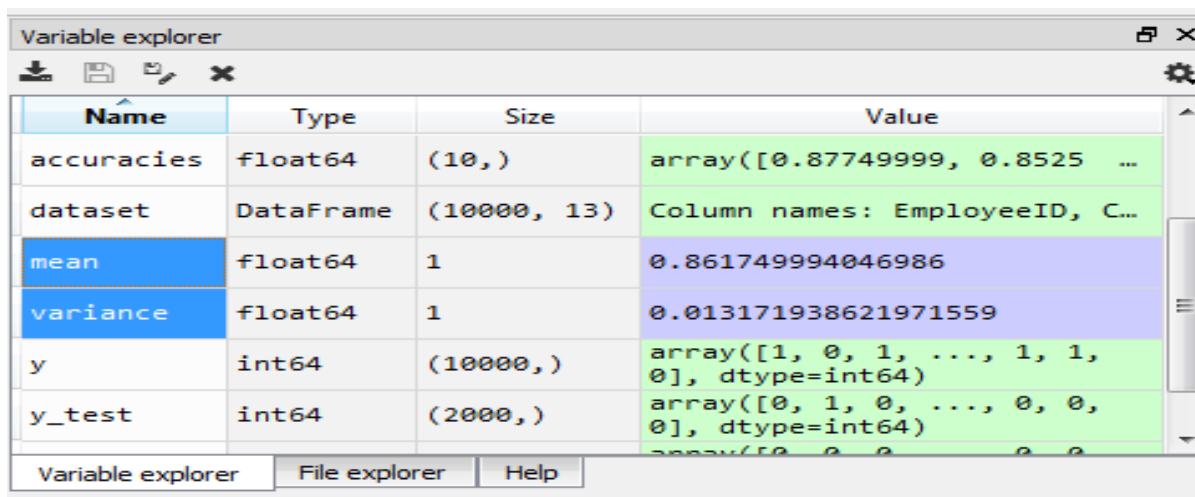


Figure 4.15 : Vector of accuracies after performing K-fold Cross Validation

As mentioned in Chapter III, in order to validate the model, K-fold Cross Validation technique (K = 10) has been implemented. Figure 4.15 demonstrates the results of this technique which will test and train the model on 10 different combinations of test sets and training sets. By taking the average of the accuracies vector, and calculating the standard deviation, it is possible

to have a look at the variance and have a much more relevant analysis. Moreover, the model can be validated by having the variance and the mean of these accuracies.

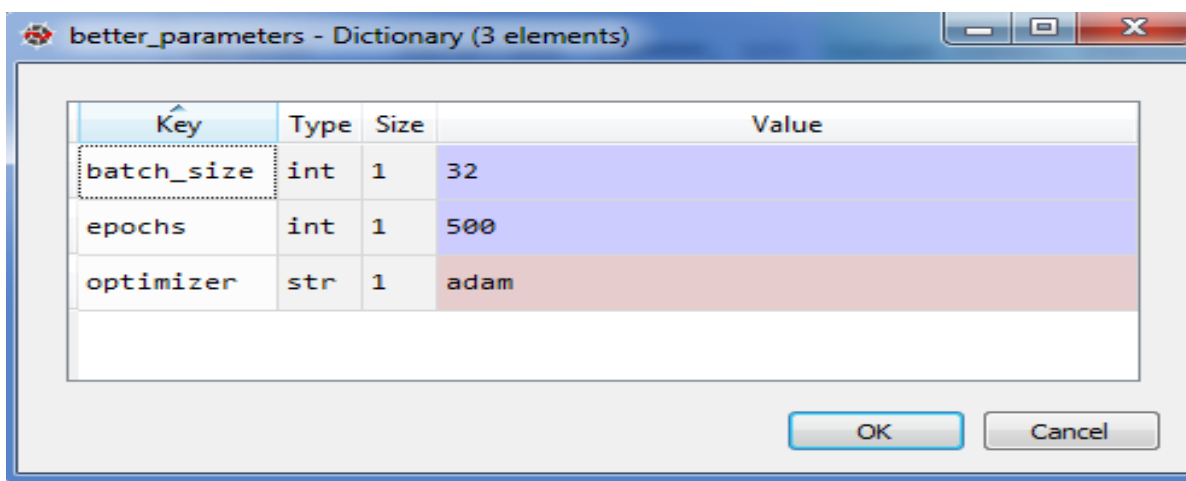


The screenshot shows a 'Variable explorer' window with a table of variables. The 'mean' and 'variance' rows are highlighted in blue. The 'mean' row shows a value of 0.861749994046986, and the 'variance' row shows a value of 0.013171938621971559. Other variables include 'accuracies', 'dataset', 'y', and 'y\_test'.

Name	Type	Size	Value
accuracies	float64	(10,)	array([0.87749999, 0.8525 ...
dataset	DataFrame	(10000, 13)	Column names: EmployeeID, C...
mean	float64	1	0.861749994046986
variance	float64	1	0.013171938621971559
y	int64	(10000,)	array([1, 0, 1, ..., 1, 1, 0], dtype=int64)
y_test	int64	(2000,)	array([0, 1, 0, ..., 0, 0, 0], dtype=int64)

Figure 4.16 : Mean and Variance of the accuracies

As it has been demonstrated in Figure 4.16, the model has the mean accuracy of 86.17 percent and the variance is relatively low which results in a stable and reliable model. The most important part of this work after achieving a low bias and low variance model is to analyze the parameters that has been used and find the best parameters to improve the model's accuracy.



The screenshot shows a 'better\_parameters - Dictionary (3 elements)' window with a table of parameters. The 'batch\_size' row is highlighted in blue, 'epochs' in purple, and 'optimizer' in red. The 'batch\_size' row shows a value of 32, 'epochs' shows 500, and 'optimizer' shows 'adam'.

Key	Type	Size	Value
batch_size	int	1	32
epochs	int	1	500
optimizer	str	1	adam

4.17 : Best parameters after parameter tuning

As it has been shown in Figure 4.17 best parameters for the model include of: batch\_size = 32, number of epochs = 500, and optimizer (SGD algorithm) = Adam.

Name	Type	Size	Value
X	float64	(10000, 12)	array([[0.0000000e+00, 0.0000000e+00, 6.1900000e+02, ..., 1.0000000e+0 ...
X_test	float64	(2000, 12)	array([[-0.57369368, 1.75486502, -0.55204276, ..., 0.9687384 , ...
X_train	float64	(8000, 12)	array([[ 1.74309049, -0.5698444 , 0.16958176, ..., -1.03227043, ...
better_accuracy	float64	1	0.86725
better_parameters	dict	3	{'batch_size':32, 'epochs':500, 'optimizer':'adam'}
dataset	DataFrame	(10000, 13)	Column names: EmployeeID, CreditScore, Geography, Gender, Age, Tenure, ...
parameters	dict	3	{'batch_size':[25, 32], 'epochs':[100, 500], 'optimizer':['adam', 'rms ...
y	int64	(10000,)	array([1, 0, 1, ..., 1, 1, 0], dtype=int64)
y_test	int64	(2000,)	array([0, 1, 0, ..., 0, 0, 0], dtype=int64)
y_train	int64	(8000,)	array([0, 0, 0, ..., 0, 0, 1], dtype=int64)

Figure 4.18 : Best accuracy of the model after parameter tuning

As it has been demonstrated in Figure 4.18, using the aforementioned best parameters for the model will result in the accuracy of 86.72 percent.

## Chapter V: Discussion and Conclusion

As mentioned earlier, a deep learning NN model has been selected to create a new, efficient, and optimal approach to detect suspicious behavior caused by insiders within a database system by utilizing an artificial neural network as a third-party database auditing system. Thanks to the deep learning model, the accuracy will increase as the size of the dataset increases. Independent variables which are the input of the ANN are the features that a user has within database. After training and fitting the ANN on dataset, the ANN has been validated by comparing the accuracy that the ANN is predicting on the training set with the calculated accuracy of the ANN on test set. After evaluating the performance of the deep learning model, the parameter tuning technique has been applied to the model to find the best hyper parameters for the model. It is worth mentioning that, in scenarios in which detecting users suspicious behavioral patterns are of high value, the threshold of the prediction vector could be set on a different value as per requirements. Overall the model has an accuracy of 86.72 percent. To interpret and understand the result of this model in simple terms, one can look at the accuracy of the model as the level of confidence; meaning that the percentage of risk that a certain employee poses based on the outcome of the model is 86.72 percent trustworthy.

In order to find the best number of hidden layers in the model, several different models with different number of hidden layers have been compared to each other. By increasing the number of hidden layers from two to three, the accuracy of the model will increase by one percent. Accuracy of the model will not significantly increase by adding more hidden layers and adding more layers will only cause the over fitting problem due to complexity of the architecture of the model. In the other words, adding more than three hidden layers to the model will cause

inconsistency (high variance) in predicting the results which is not in line with the purpose of an ideal model.

All in all, the outcome of the proposed model on top of data base auditing system would be beneficial for organizations to analyze their users behaviors. Based on the users behavioral patterns, the policies can be modified and new policies or restrictions could be applied to certain users which would result in a more secure database system.

### **Future Work**

Although a simulated dataset could be as good as real-world experiment and could have a very close outcome to a real-world scenario, a real-world scenario would be ideal. The NN model that has been presented in this project could be used on a real-world dataset. In order to tackle the problem of over fitting or under fitting the Dropout Regularization technique could be used. Parameter tuning approach that has been tackled in this thesis can be utilized to improve the accuracy of the model on the new dataset to find the best hyper parameters of the model such as best accuracy, best parameters and best optimizer. Moreover, such model could be also built into the existing database auditing systems to detect fraud patterns and suspicious behaviors. Additionally, the mutli-armed bandit approaches can be used to predict the percentage of risk that a certain employee poses. For more details, refer to different types of multi-armed bandit in Yekkehkhany et al., (2019), and the references therein.

## References

- Armeding, T. (2018, December). *The biggest data breaches of 21st century*. Retrieved from: <https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html>
- Bengio, Y., & Grandvalet, Y. (2004). No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*, 5, 1089-1105.
- Bishop, M. (2003) *COMPUTER SECURITY: Art and science*. Boston, MA: Addison-Wesley.
- Busta, B. & Weinberg, R. (1998). Using Benford's Law and neural networks as a review procedure. *Managerial Auditing Journal*, 13(6), 356-366.
- Calderon, T. G., & Cheh, J. J. (2002). A roadmap for future neural networks research in auditing and risk assessment. *International Journal of Accounting Information*, 3(4), 203-236.
- Cerullo M. J., & Cerullo, V. (1999). Using neural networks to predict financial reporting fraud. *Computer Fraud & Security*, 1999(5), 14-17.
- Coakley, J. K., & Brown, C. E. (1991). Neural Networks for financial ratio analysis. *The world Congress on Expert Systems*, pp. 132-139
- Coakley, J. K., & Brown, C. E. (1993). Artificial neural networks applied to ratio analysis in the analytical review process. *Intelligent Systems in Accounting, Finance and Management*, 2(1), 19-39.
- Coalkey, J. (1995). Using pattern analysis methods to supplement attention directing analytical procedures. *Expert Systems With Applications*, Elsevier, 9(4), 513-528.
- Conway, T., Keverline, S., Keeney, M., Kowalski, E. Williams, M., Cappelli, D., P. Moore, A., Rogers, S., J. & Shimeall, T. (2005). Insider threat study: Computer system sabotage in

critical infrastructure sectors. Retrieved from:

[https://www.researchgate.net/publication/244466601\\_Insider\\_Threat\\_Study\\_Computer\\_System\\_Sabotage\\_in\\_Critical\\_Infrastructure\\_Sectors](https://www.researchgate.net/publication/244466601_Insider_Threat_Study_Computer_System_Sabotage_in_Critical_Infrastructure_Sectors)

Citro, C., Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., ... Zheng, X. (2015).

TensorFlow : Large-scale machine learning on heterogeneous distributed systems, *1*(2).

Chollet, F. (2017). Deep learning with python. *Manning Publications*.

De Sousa, R. M. (2016). Computational intelligence applied to audits: Pattern classification using artificial neural networks.

Etheridge, H. L., Sriram, R. S. , Hsu, H. Y. (2000). A comparison of selected artificial neural networks that help auditors evaluate client financial viability. *Decis. Sci*, *31*, 531–550.

“Facebook.com.” Retrieved from: <https://facebook.com>.

Fortmann-Roe, S. (2012, June). *Understanding the Bias-Variance Tradeoff* . Retrieved from:

<http://scott.fortmann-roe.com/docs/BiasVariance.html>

Gaganis, C., Pasiouras, F., Doumpos, M. (2007). Probabilistic neural networks for the identification of qualified audit opinions. *Exp. Syst. Appl*, *32*, 114–124.

Garrity, E. J., O’Donnell, J. B., Sanders, G. L. (2006). *Continuous Auditing and Data Mining*.

Retrieved from: <https://www.igi-global.com/chapter/continuous-auditing-data-mining/10596>.

Gujarati, D. N. (1970). Use of dummy variables in testing for equality between sets of coefficients in two linear regressions. *American Statistician*, *24*(1), 50-52.



- Hosseini, M., Jiang, Y., Yekkehkhany, A., Berlin, R. R., Sha, L. (2017, June). A mobile geo-communication dataset for physiology-aware dash in rural ambulance transport. *In Proceedings of the 8th ACM on Multimedia Systems Conference*, pp. 158-163.
- James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, Yoshua Bengio Theano. (2010). A CPU and GPU math compiler in python. *The 9<sup>th</sup> Python in Science Conference*.
- Jiao, J., Courtade, T., Vendkat, K., Weissman, T. (2015, December). Justification of logarithmic loss via the benefit of side information. *IEEE Transactions on Information Theory*, 61(10), 5357.
- Khare, S., & Gajbhiye, A. R. (2013, June). Literature review on application of artificial neural networks (ANN) in operation of reservoirs. *International Journal of Computational Engineering Research*, 3(6).
- Kingma, D. P., & Lei Ba, J. (2015). ADAM: A method for stochastic optimization. *ICLR*.
- Koskivaara, E. (1996). Artificial neural network models for predicting patterns in auditing monthly balances. *51(9)*, 1060-1069.
- Koskivaara, E. (2000). Artificial neural network models for predicting patterns in auditing monthly balances. *Journal of the Operational Research Society*, 51(9), 1060-1069.
- Koskivaara, E. (2000). Different pre-processing models for financial accounts when using neural networks for auditing. *ECIS 2000 Proceedings*, p. 3.
- Koskivaara, E. (2003) . Artificial neural networks in auditing: state of the art.

Kotsiantis, S., Koumanakos, E., Tampakas, V., Tzelepis, D. Forecasting fraudulent financial statements using data mining. 2006. *International Journal of Computational Intelligence*, 3(2).

“Linkedin.com.” Retrieved from: <https://linkedin.com>.

Marsh, H. W., Dowson, M., Pietsch, J., Walker, R. (2004, September). Why multicollinearity matters: A reexamination of relations between self-efficacy, self-concept, and achievement. *Richard Journal of Educational Psychology*, 96(3), 518-522.

Musavi, N., Tekelioğlu, K. B., Yildiz, Y., Gunes, K., Onural, D. (2016). A game theoretical modeling and simulation framework for the integration of unmanned aircraft systems into the national airspace.

Musavi, N., Onural, D., Gunes, K., Yildiz, Y. (2016). Unmanned aircraft systems airspace integration: A game theoretical framework for concept evaluations. *Journal of Guidance, Control, and Dynamics*, pp. 96-109.

Musavi, N. (2019). A game theoretical framework for the evaluation of unmanned aircraft systems airspace integration concepts.

Najafabadi, M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1).

Oracle docs. (2017) *Oracle Database Security Guide*. Retrieved from :

[https://docs.oracle.com/database/121/DBSEG/audit\\_admin.htm#DBSEG1026](https://docs.oracle.com/database/121/DBSEG/audit_admin.htm#DBSEG1026)

Ostberg, P. O., Groenda, H, Wesner, S, Byrnes, J, and more. (2014). The CACTOS vision of context-aware cloud topology optimization and simulation. *In Proceedings of the 6<sup>th</sup>*

*IEEE International Conference on Cloud Computing Technology and Science. ISBN: 978-1-4799-4093-6.*

- Pourheydari, O., Nezamabadi-pour, H., Aazami, Z. (2012). Identifying qualified audit opinions by artificial neural networks. *African Journal of Business Management*, 6(44), 11077-11087.
- Salama, T. (2014, February 28). *Oracle Database 12c Security: New Unified Auditing*. Retrieved March 08, 2018, Retrieved from: <https://blogs.oracle.com/imc/oracle-database-12c-security:-new-unified-auditing>.
- Shin, K. S., Lee, T. S., Kim, H. J. (2005). An application of support vector machines in bankruptcy prediction model. *Syst. Appl*, 28, 127–135.
- Smit, S. K., & Eiben, A. E. (2009, May). Comparing parameter tuning methods for evolutionary algorithms . *Evolutionary Computation IEEE*, pp. 399-406.
- Taha, R. (2012). The possibility of using artificial neural networks in auditing: Theoretical analytical paper. *European Journal of Economics, Finance and Administrative Sciences*, (47). Retrieved from: [https://www.academia.edu/1634776/The\\_Possibility\\_of\\_using\\_Artificial\\_Neural\\_Networks\\_in\\_Auditing\\_-Theoretical\\_Analytical\\_Paper](https://www.academia.edu/1634776/The_Possibility_of_using_Artificial_Neural_Networks_in_Auditing_-Theoretical_Analytical_Paper)
- “Twitter.com.” Retrieved from: <https://twitter.com>.
- Wagner, J., Rasin, A., Glavic, B., Heart, k., Furst, J., Bressan, L., Grier, J. (2017) Carving database storage to detect and trace security breaches. *Proceedings of the Seventeenth Annual DFRWS US*, 22, pp. S127-S136.

Weisstein, E. W. *Sigmoid function*. Retrieved from MathWorld-A wolfram web resource:

<http://mathworld.wolfram.com/SigmoidFunction.html>

Woo, J., Lee, S., Zoltowski, C. (2010). *Database Auditing*, Retrieved from:

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.5107&rep=rep1&type=pdf>

Wu, R. C. (1994). Integrating neurocomputing and auditing expertise. *Managerial Auditing Journal*, 9(3), 20-26.

Xie, Q., Yekkehkhany, A., & Lu, Y. (2016, April). Scheduling with multi-level data locality:

Throughput and heavy-traffic optimality. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pp. 1-9.

Yang, L. (2009, March). Teaching database security and auditing, *ACM SIGCSE Bulletin*, pp. 241-245.

Yekkehkhany, A. (2017). Near-Data scheduling for data centers with multiple levels of data locality. *arXiv preprint arXiv:1702.07802*.

Yekkehkhany, A., Arian, E., Hajiesmaili, M., Nagi, R. (2019). Risk-Averse explore: then-commit algorithms for finite-time bandits. *arXiv preprint arXiv:1904.13387*.

Yekkehkhany, A., Hojjati, A., Hajiesmaili, M. H. (2018). GB-PANDAS: Throughput and heavy-traffic optimality analysis for affinity scheduling.

Yekkehkhany, A., & Nagi, R. (2019). Blind GB-PANDAS: A blind throughput-optimal load balancing algorithm for affinity scheduling. *arXiv preprint arXiv:1901.04047*.

Zhang, Z., T. McDonnell, K. Zadok, E., Mueller, K. (2015, February). Visual correlation analysis of numerical and categorical data on the correlation map. *IEEE Transactions On Visualization And Computer Graphics*. 21(2).

Zhou, L., Pan, S., Wang, J., Vasilakos, A. V. (2017). Machine learning on Big Data: Opportunities and challenges. *Neurocomputing*, 237, 350-361.