University College London

Department of Computer Science

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computational Finance
University College London

Operational Risk VAR

Author
Parth Parakh

Academic Supervisor
Dr. Giacomo Livan
Department of Computer Science
University College London

Industrial Supervisor
Dr. Peter Mitic
Head of Operational Risk Methodology
Santander UK

August 11, 2021

Declaration

This dissertation is submitted as part requirement for the MSc Computational Finance degree at University College London (UCL). It is substantially the result of my own work except where explicitly indicated in the text.

The dissertation will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except with permission from the author.

Acknowledgments

Throughout the writing of this dissertation, I have received a great deal of support and assistance. I would first like to thank my industrial supervisor, Dr. Peter Mitic for his encouragement, patient guidance and constructive advice throughout the process. His expertise was invaluable while formulating the research topic and methodology. Besides my industrial supervisor, I would also like to thank Dr Giacomo Livan for his constant mentorship and guidance in thesis writing.

I would like to extend my gratitude towards my colleagues from Santander for their collaboration and help. It was a wonderful experience working and learning alongside you all.

In addition, I would also like to thank my parents and sister for their wise counsel and sympathetic ear. Thank you for always pushing me to give my best and trusting me with my decisions. Finally, there are my friends who were of great support in deliberating over the problem and findings, as well as providing happy distraction outside of my research.

Abstract

This thesis aims to develop a model which helps us better understand the factors which affect the VAR in the operational risk context. It could be some combination of an increase or decrease in a particular range of losses. For example, a 20% increase in the 50-75% quartile. But other combinations of increases and quartiles could have resulted in the same or similar VaR. We would like to locate the most likely combination.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

	Loss Events (Thousand £)
Count	500
Mean	24167.02
Standard Deviation	139545
Minimum	1.328105
25% quantile	893.1871
50% quantile	3098.362
75% quantile	12043.93
Maximum	2915654

Table 1.1: Basic Statistics of the data provided

Example of a figure:

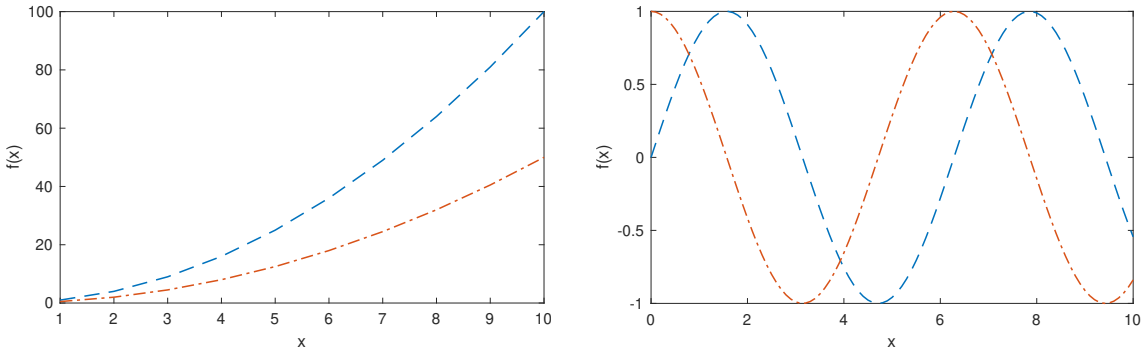

Figure 1.1: It is good practice to put a long descriptive title in your caption so that readers can immediately understand the significance of the figure without having to search through the accompanying text. The short title in square brackets before the long caption is for the more concise description included in the List of figures.

The use of "\FloatBarrier" allows you to specify that floats (e.g. Figures and Tables) should not appear past a certain point in a document but you must import the "placeins" package in order to use it.

In order to include formulas, you can use in line equations by using $ at the beginning and end, for example $y = f(x)$ and single line equations using the "equation" environment, for example

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{-ix\xi} \widehat{f}(\xi) d\xi. \tag{1.1}$$

You can also write equations on several lines using the "align" environment, notice the use of & at the point where the lines align and the double backslash to move to the next line, for example

$$\begin{aligned} f(x) &= \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{-ix\xi} \widehat{f}(\xi) d\xi \\ &= \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{-ix\eta} \widehat{f}(\eta) d\eta. \end{aligned} \tag{1.2}$$

Notice the use of "\nonumber" above to suppress the equation numbering on the first line.

You can also use the "\label{}" "\ref{}" commands to reference items in the dissertation such as Section **??**, Figure 3.5, Table **??** and Eq. (1.2).

Write your bibliography as a .bib file (an example is included) and then import using the "\bibliography{}" command.

Chapter 2

Literature review

Losing data because of system failure, floods disrupting power supply to servers, hackers stealing precious customer data, employees embezzling company funds are all day-to-day activities which can adversely affect the bottom line of a company without directly affecting the money making activity. This kind of risk which arise due to various operations performed, to keep the company running is called operational risk (oprisk). These operational risk are ubiquitous throughout the world and are not just restricted to the business. Example, as seen from our current real world scenario where wrong labelling led to incorrect inoculation of the patients or incorrect storage of vaccination led to wastage of previous veils of vaccine doses. In 2001, Basel II set out a regulatory framework for operational risk in international banking and gave the, now most widely accepted definition for operational risk : risk of loss resulting from inadequate or failed internal processes, people, system, or from external events [1]. Basel committee was also the driving force that led the to the development of operational risk management practises, appropriate risk measurement techniques and proper disclosures by companies for transparency. The basic structure for risk management is simple, identify what are the different hazardous causes that cab create issues within the bank/company. Once these causes are identified, what are the possible outcomes of the causes and how they threaten the bank. From there, we try to find the frequency of the losses and the impact or the severity of the losses caused by threats. Post that, it's about reconciling the frequency and severity to determine a possible framework to tackle the threat.

The following are the tools that can be leveraged to better manage risk in particular with respect to operations :

1. Risk and Control Self-Assessment : Companies are pro-active in managing their risks by continuously evaluating and checking their risk exposure, along with constant monitoring and recording of the key results and performance indicators. They also utilise external methodologies available to aid their assessment.

2. Loss Event Monitoring : This involves consistent invigilation and tracking of the loss events and near-miss loss events. There are two ways these loss events could be defined by either a regulatory or by a management body. Or the management body expands on the guidelines set by the regulatory body especially in the case of near miss events since they are indicative of future potential losses.

3. Stress and Scenario Testing : This again reflects good proactive nature of the company by employing contingency planning to handle tough and unforeseeable circumstances. It checks how robust are our current mechanisms to handle unexpected risks.

4. Key Risk Indicators : This means keeping constant vigilance over KRIs to help understand any early signs of impending high risk warnings. Going one step further would be by creating task force or framework to handle these stress signs effectively thereby minimising damage and loss.

Now, when we look at operational risk within the context of large multinational banks, few of the key areas which are high potential for loss events as specified by the Basel 2 committee and adopted in Solvency II, the EU regulations are as follows :

1. Internal Fraud (IF).

2. External Fraud (EF).

3. Employment Practices and Workplace Safety (EP and WS).

4. Clients, Products, and Business Practices (CP and BP).

5. Damage to Physical Assets (DPA).

6. Business Disruption and System Failures (BD and SF).

7. Execution, Delivery, and Process Management (ED and PM).

In practise, risk management depends on a lot of other factors. You could have a robust system in place, constant vigilance monitoring and frequent assessments but if these others steps are not carried out well, it can lead to inadequate preparation. Some of these factors include quality and quantity of the data collected, appropriateness of the model used, modeling/handling of the future potential risk adversities, assessing loss from a major loss event or near miss event, risk management decisions using historical data which might not be relevant to the current exposure and other subjective approach to quantitative management of operational risk. Incorrect use of these could potentially lead to aggravation of the risk. On the other hand, good loss modelling can be used to make forecasts, give insights into patterns and repetitions, help identify correlations

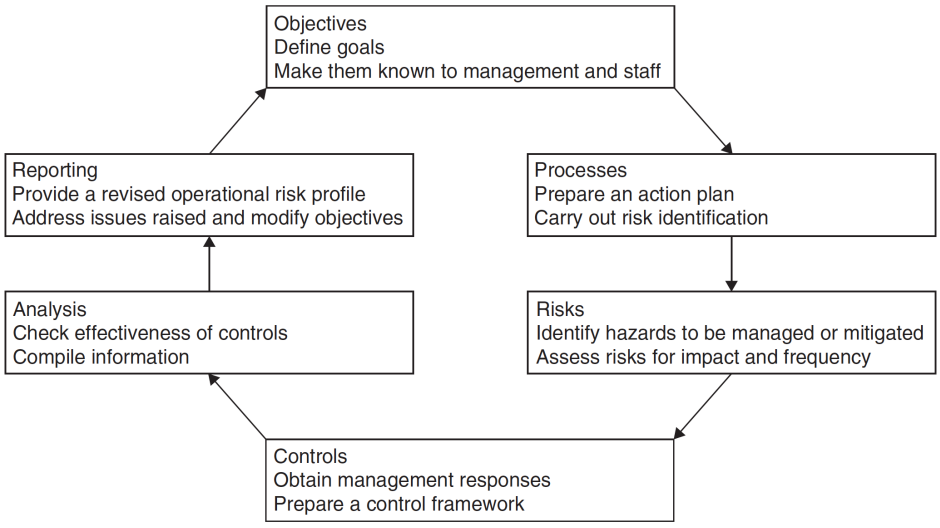

Figure 2.1: The operational risk management cycle.(Source: Taken from Risk Books [6].)

between business lines and event types (severity), benchmarking, backtesting etc. Further, it provides us with heuristics and tools which help us go beyond the limitations of the traditional statistical analysis. For these loss modelling, we'll be going to leverage the latest advancements made in the field of machine learning especially neural networks and supplement that with Bayesian analysis. The biggest drawback, however, of these models is their black box nature and difficult to interpret non linear mapping relationship between input features and output classes.

In the current literature for operational risk, machine learning has predominately been used for fraud detection and suspicious transaction detection. [2] proposes using a logistic regression a model to capture the fraud detection and dubious suspicious transactions. When dealing with financial transactions intended for illegal purposes, the money is routed and concealed amongst a web of transactions. There has been a lot of research done to detect money laundering and identify similar customer patterns using both traditional and state of the art machine learning algorithms. [3] and [4] are amongst them to come up with intelligent systems for this purpose. Credit card frauds are another big research area under the operational risk division. There are quite a few decision tree based algorithms proposed to detect credit card frauds. The biggest issue in this case is the lack of real datasets which make the such analysis hard to come by. There is some good work that has been done by [5], [6], [7], [8] in this area. Now in most of these areas machine learning techniques have performed better than it's traditional counterparts which encouraged us to pursue similar route with our operational risk analysis, at the same time, our research question seems to be an unexplored section of the operational risk which although can be challenging but ultimately is an opportunity.

Among the various risk management tools specified, we'll be looking into Loss Event

Monitoring (LEM). LEM practise involves collecting and tracking the loss events and in some cases (industries), expanding on these to include near-miss events which are indicative of potential future losses. We start our analysis with one such collection of loss events provided to us by Santander. It is a collection of 500 different events each represented by their own unique ID and the corresponding losses.

A common risk management approach is to list down all the major risks facing the company and it's corresponding impact where the impact is measured in terms of severity and frequency. Widely known statistical measure which can be thus be used to capture these major risk is Value at Risk (VAR). VAR is a measure of losses resulting from "normal" movement of the market; losses greater than VAR are suffered only with a specified small probability [9]. The reason VAR works well is because it allows us to succinctly represent the potential loss with just a single number. To compute VAR, we need to identify basic market factors that affect it's value.

[talk about factors affecting var, the factors we'll use and dive into how NN and Bayesian analysis will be used to determine]

Chapter 3

Methodology

3.1 Data Preparation

We start by creating a script in R for calculating basic routines for analysing operational risk data and fitting appropriate distributions. We were provided a basic test data by Santander with five hundred different losses. As the first step of data generation, we fit these points to different distributions and plot the results.

From the various distributions plotted, we see log normal curve fits the data best and hence proceed to use log normal as the distribution to fit our loss data. Log normal distribution is a probability distribution where the logarithm of it's values are normally distributed. After fitting the five hundred loss data points to log normal distribution we get the parameter values for the curve. These parameters which define the log normal distribution are ν which stands for expected value or mean and σ which stands for standard deviation of the variable's natural logarithm and not of the variable itself. Data with a low mean value, high variance and all positive values are best fitted to a log normal distribution. Since most of our losses are relatively low with few outlier high loss values and positive as usually seen in real world, it intuitively makes sense why log normal would be a good fit. So, once we have the parameter values from fitting it into the distribution, we use it to simulate a new distribution and use that to calculate the 99 percent value at risk. The way we do this is to generate a random variable from the log normal distribution created using the parameters from the operational loss. We repeat the entire simulating or generating the random variable a million times. Using these million points, we then calculate the value at risk for ten day period at ninety nine percentile. This is our base case. We choose the ninety ninth percentile and a ten day period because this is specified by the Basel Committee.

Further we divide our five hundred points losses into 5 different ranges. The first range is taking the twenty five lowermost quantile values, the second range is the values ranging from twenty five to fifty quantile range, the third range is the fifty to seventy five quantile

increase	low	low-mid	mid-high	high	max
	0-25	25-50	50-75	75-100	
10	1	2	3	4	5
20	6	7	8	9	10
30	6	7	8	9	10
40	11	12	13	14	15
50	11	12	13	14	15
-10	1	2	3	4	5
-20	16	17	18	19	20
-30	16	17	18	19	20
-40	21	22	23	24	25
-50	21	22	23	24	25

Figure 3.1: Different cases considered. For each of these 50 cases, loss values were calculated and therefore VAR was worked out.

range, fourth range is the seventy five to the hundred quantile range i.e the highest twenty five percentile values and finally the fifth range where we only consider the highest value point amongst the five hundred different values. Once we have segregated the data in these different ranges, we then perform ten different operations on them. These operations are:

1. Increase these values by 10 percent.
2. Increase these values by 20 percent.
3. Increase these values by 30 percent.
4. Increase these values by 40 percent.
5. Increase these values by 50 percent.
6. Decrease these values by 10 percent.
7. Decrease these values by 20 percent.
8. Decrease these values by 30 percent.
9. Decrease these values by 40 percent.
10. Decrease these values by 50 percent.

This in total gives us fifty different cases. For example, if we choose the lower most quantile and increase by thirty, it means we are increasing the smallest one twenty five values out of five hundred (lowest twenty five quantile) by thirty percent and the keeping the remaining values constant. So, we use various combinations of increase/decrease and quantiles to get different values of operational losses. Using these new losses, we repeat the series of operations, we performed for the base case i.e. fitting the losses to a log normal distribution to get the parameters of the losses, then using these parameters to simulate a million log normally distributed random variables and finally using these million points to get the requisite VAR.

This above described operation is performed for all the fifty cases, each done two hundred times giving us a total of ten thousand data points to work with (50*200). Thus, our 10,000 data points had been obtained by considering a combination of different quantiles

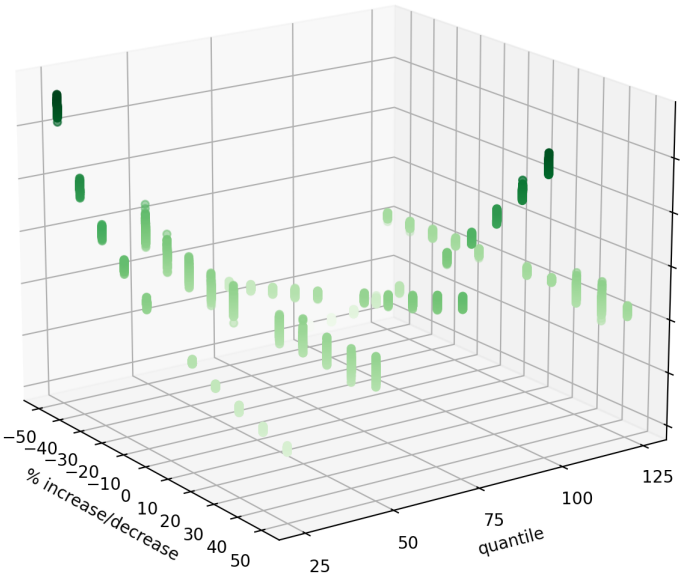

Figure 3.2: All the 10,00 different points plotted. Here each of these "clouds" represent the 200 different values we obtained for the 50 cases. Since the MC simulations were 1 million points, clouds are low variance and are centered around the mean.

and increase/decrease of data in that particular quantile. Along with quantile, we were also considering the case where we only increase the largest loss.

Next, we create a 3D plot cloud distribution and a 3D surface plot to visualise our data. These plots are seen below.

The plots obtained so far are as follows :

3.2 Models

We build our first model using neural networks to model value at risk. Neural networks is used to determine the underlying non linear relationship between the input and the output. We represented the smallest quantile as 25, the next one as 50, the one after that as 75, the largest quantile as 100 and finally, increasing the largest loss as 125. For the increase or decrease, each case was represented by the amount the loss data was changed, meaning the ten percent increase was represented by 10, fifty percent decrease represented by -50 and so on for all the cases. These different combinations of quantiles and increase/decrease of operational losses form the input to our model whereas the output is the corresponding value at risk. From the ten thousand data points available, we divide eighty percent points for training and the remaining twenty percent points for testing. Now, that we have segregated the data, it's time to build the neural network.

We use grid search to build our model. Grid search is the process of performing hyper

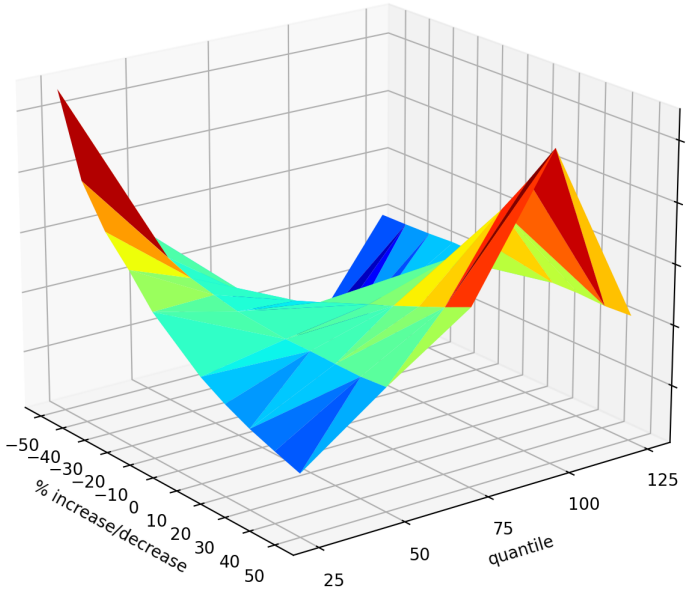

Figure 3.3: For each of these 50 different clouds, we took the mean and used it to represent those clouds, these points were then connected to it's adjacent points to create the surface plot.

parameter tuning in order to determine the optimal values for our given models. Selecting the optimal hyperparameters is essential since that helps us get the best model. Initial hyperparameter tuning for these models is performed through extensive grid search. We start by importing GridSearchCV from the sklearn library [10], a machine learning library for python. The estimator parameter of GridSearchCV is used for hyper parameter tuning process by performing cross-validation on grid search. This is done by either shifting through the parameter range in the grid search or by setting a specific value for the hyperparameter.

The parameters we are looking to tune using hyperparameter tuning includes the number of neurons, number of layers in the network, activation function, optimiser, learning rate and epochs. The first two parameters are self explanatory, having less number of neurons or layers might prevent the model from learning properly while high number of neurons and layers will lead to overfitting of the data. Activation function is used to transform a set of inputs into the corresponding output from various nodes in various layers of a network. So activation function applied on different nodes of different layers come together to create the non linear relationship mapping between inputs or outputs. Now, the next question is what difference does it make ? Some like identity function are used to a one to one mapping while others like binary will change any negative input to zero and positive input to 1, others still like tanh will give output in the range of $-1, 1$ depending on a complex exponential operation. Another subtle point to keep in mind is that each

layer can have different activation function, which makes the task of finding the correct activation function tougher. Also, when we say correct function, we mean functions which work better for our model compared to others.

Optimizers are algorithms that are used to change the attributes of the network like learning rate and weights of the nodes. They are used to solve the optimisation problem(s) by minimising the loss. Gradient Descent is one such optimisation algorithm which is most widely known and used. In this, we take the first order derivative of the loss function and using back propagation this is transmitted backwards through the network to update to weights of the nodes such that the difference between the true value and calculated value can reduce. Some of the advantages of using Gradient Descent are that it is easier to calculate and implement and some of the disadvantages are that it calculates gradient over the entire dataset so if the dataset is too big, it will lead to huge calculation times. Moreover, it has the tendency to get stuck at local minimas thereby not giving the best possible results at times. There are some optimisations available on Gradient Descent like Stochastic Gradient Descent which updates the weights more frequently and the calculation does not depend on entire dataset or Mini-Batch Gradient Descent which divides the dataset in small batches of data and updates the weights or other parameter values on basis of this. One of the other drawbacks of Gradient Descent is that it can oscillate around the value before converging to it. One way around the problem is to use Momentum algorithm which reduces movement in irrelevant direction and accelerates towards the direction of convergence. The only issue with this that it adds a new hyperparameter in the mix called momentum which is used to tune this algorithm thereby adding unwanted complexity. Also, if this parameter, momentum, is too high, it can shoot past the local minimas which isn't ideal. An alternative to this is to use Nesterov Accelerated Gradient which slows down as it approaches the minimas thereby preventing overshooting but computationally it can get quite expensive and requires tuning of a hyperparameter of it's own, hence, is not widely used. All of the above mentioned algorithms are single derivative based, however, to be thorough with our research before choosing an optimal algorithm, we also need to consider second order derivative based algorithms. First such algorithm in our list is called Adagrad. This algorithm changes the learning rate at each time instant thereby changing the rate of convergence and not just convergence i.e we aren't just calculating the losses, we are also calculating the rate of change of losses to decrease it as we love closer to our minima. Implications of this is massive as we don't need to manually tune any hyperparameter and can work with sparse data as well. Downside ofcourse is that it can be computationally expensive and hence slower than first order derivatives. There are two improvements to this called AdaDelta and ADAM algorithms which have tried to negate some of the issues plaguing Adagrad. Of these, ADAM is quite frequently used and has become one of the go to algorithms due to its fast converging property.

Learning rate determines how much to change the model to minimise the error between the true value and calculated value. This calculated error is then back-propagated to change the weights of the neurons eventually training the network well inform to identify the pattern between the input(s) and output(s). It is, unarguably, one of the most important hyperparameters needed to be tuned well to get a well trained network. Large learning rates can lead to unstable training whereas small learning rates can too long to converge and make the model practically unviable. Using second order derivative based optimisation algorithms, we can leverage momentum parameter to tune learning rate better and accelerate training.

Batch size refers to collection of data points over which the network runs before calculating the error and updating the neuron weights. It's an important hyperparameter that's tuned when using Gradient Descent or one of it's variants as an optimising algorithm. The other parameter called epoch determines the number of times, the algorithm runs over the training dataset. A good way of visualising and understanding this is to imagine epochs as a for loop and batch as a nested for loop within it. For each iteration over the batch, the weights are changed as the model is learning and the entire process is repeated epoch number of times. We generally use the large number as epoch to let the model run as many times as possible to minimise the error and make the model more robust.

Clearly, training a neural network is not easy. We initially start with assigning random weights to each neuron and using loss function, we determine the difference between the true value and calculated value, finally we propagate this difference backwards to update the weights of each neuron to train them. This highlights the huge role loss function plays in determining the proper training of the network. There are a range of loss functions to choose from, each with it's set of advantages and disadvantages. To determine the appropriate loss function, we work under the framework of maximum likelihood estimation (MLE). Under MLE, the loss function tries to reduce the gap between the predicted variables and the target variables. But the reason MLE works well as a framework to determine loss function is because it upholds the property of "consistency". Property of consistency states that as we increase the data in the training set, the predicted variable should tend towards the true value. So, working under the MLE framework, the first loss we'll consider is mean square loss. As, the name suggests, it calculates the average of the squared difference between the predicted and true values. The next loss function is called cross entropy which penalises the difference between the predicted probability and actual class. Higher the difference, greater is the penalty. This is achieved using logarithm of the difference. Within cross entropy, there are few variations such as binary when we are using just two classes and hence have only one output node and categorical when we are using multiple output classes and working in a high dimensional space.

Now that we understand the different hyperparameters, let's see how we leverage these parameters to build our plethora of models.

As mentioned earlier, for the first model, we use combination of various quantiles and increase/decrease as the inputs and try to determine the VAR as the output. We begin by normalising our data and rounding off the VAR values to 2 decimal points. We then divide our data into training and testing set in the ratio of 8:2. Our neural network had 3 layers excluding the input and output layer. The activation function used is 'relu' for all layers except the output layer. For output layer, we used 'linear' as the activation function. We add a dropout layer between the second and the third layer, keeping the dropout value at 0.05. We went ahead with 50 neurons for the first layer, 100 for the second and 50 again for the third. For the optimiser, we used Adam with learning rate as 0.01, beta1 as 0.9, beta2 as 0.999 and epsilon as 1e-08. To calculate the loss, we used mean squared error as our loss function. While training, we set the value of epochs as 50 and the batch size as 10. Again most of these values of the hyperparameters were obtained through rigorous grid searching. In some cases, the values were determined by considering the different advantages and disadvantages depending on which mattered to us. For example, since we aren't dealing with a huge dataset, we were willing to got for an algorithm which although was slow and computationally heavy was more accurate than it's counterparts.

Now, the main research question we are trying to answer here is, what are the different factors affecting value at risk. However, we see from our first model that we are looking at the mirror image of the problem i.e. we are trying to determine the var at risk when we change the loss instead of the other way around to get the answer for our research question. For example, we are changing losses in the second quantile by 10 percent and observing the change in the VAR. We should, on the other hand, be doing the opposite of that. And that's what we'll try and achieve with the other neural network models that we build.

For our second model, we just reverse our inputs and outputs. Similar to the first model, we separate training and testing data into a ratio of 8:2. We first reduce the precision to just 2 decimal points and feed into our network, then using it, we determine the output value of quantile and percentage increase/decrease. The values for the hyperparameters are similar to the ones we used for the first model : number of neurons and layers are same while the activation function is 'relu' for all layers except the final one. For the final layer, we again went for 'linear'. The other parameters are exactly the same as from the previous model but that's where we went wrong. We tried to use the same prediction methodology to solve a completely different problem. In the first case, the output was VAR which was more or less continuous is nature but here the outputs are different quantile and percentage change which are not continuous in nature so this doesn't work. Solution is change this to a classification problem and use a new network to classify into different classes based on Figure 3.1.

Using our learning from the previous model, we embark on the journey to build a classification model which does what it's supposed to i.e classify VAR into different classes.

	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0	21.0	22.0	23.0	24.0	25.0	26.0	27.0	28.0	29.0	30.0	31.0	32.0	33.0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9996	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
9997	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
9998	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9999	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.4: One hot encoding of our data for our NN model 3. We were only able to capture 33 classes here but there are 50 classes in total.

We start by reading the data that we have and creating numpy arrays. The inputs and outputs are segregated accordingly and randomly shuffled. This is to make sure all the data pertaining to a particular class is not grouped together but is rather spread across to be available as part of both training and testing data sets. Post this, for the output variable, we use something called one hot encoding. Which basically means, for a particular value of VAR, if it belongs to a class x, that class is marked as 1 and the other classes are marked as 0. As shown below. Although the table only shows 33 classes, we had 50 classes in total.

After getting the data in the right format, the next step is to segregate it into training and testing sets. We again proceed with using a ratio of 8:2 for dividing the data. We use 3 layers for our neural network excluding the input and the output layer. Our input layer has only 1 neuron because we have only the VAR as the input and for the output layer, we have 50 neurons to classify our data into 50 different classes. For the 3 inner layers, we used 100,150 and 100 neurons respectively as determined through gridsearch. We also include a dropout layer fixing the dropout value at 0.2 to prevent overfitting our model. For the other hyperparameters we went with 'relu' as the activation function for all the layers except the final. For the final layer, we chose 'softmax' due to it's ability to normalise the input into the probability distribution where the total sums up to one. Using this, during classification, we get probability distribution for various classes, which is what we are looking for. For the loss function, we use categorical cross entropy function. As mentioned earlier, the cross entropy is a good loss function to use as it penalises large difference between the two value and prediction more than small difference using logarithmic error calculating function. Since we have more than two classes, we use categorical and not binary function. We use 'ADAM' as the optimiser and batch size was 32 while the epoch was fixed at 100.

Adding further to our model 3, we try two more variations. For the first variation, we combine our different VAR values which lie within 0.5 values apart i.e. instead of using the exact values of VAR, we combine different VAR values on the above mentioned criteria and map it to the various quantile and increase/decrease class it belongs too. The idea was to reduce the precision of the values of VAR we were using since in practical scenarios,

the value we'll be using are usually not that precise and have tolerance of around 2 percent depending on how it's calculated. Other than changing the VAR values, we didn't change anything else from the previous method and let it be a classification problem. One of the major reasons for low accuracy for our models is that we are dealing with a very dimensional data. Accurately classifying into 50 different classes is a mammoth task and one of the ways of countering this is to reduce the number of classes. This is precisely what we we did for the second variation. We combined the different classes (as shown in figure 3.5). The idea was to reduce the dimensionality of the data and thereby making the problem a little easier to classify.

We also repeated to the above 5 models with replacing VAR with expected shortfall (ES). As with VAR, we calculated 10,000 ES points, 200 points for each of the 50 cases mentioned in figure 3.1. Although VAR as a measure of risk is recommended by the Basel committee, ES is a good measure of risk as well and used widely by the industry. So, it only makes sense to have models that work by using ES independently or alongside VAR.

So our first set of models were based entirely on neural networks leveraging its ability to map non linear relationship between inputs and outputs. Next we tried to use collection of classification algorithms based on Bayes' Theorem called Naive Bayes classifiers. One of the major reasons for wide adoption and use of this model is its closed form solution to maximum likelihood algorithm for loss generation and linear time complexity for it's calculation. One of the fundamental assumption of Naive Bayes is that each of the input variables should be independent and contribute equally to probability calculation. Although this is a limiting and often a 'naive' assumption, this collection of classification algorithms work quite well in real world complex scenarios. At it's core, it a conditional probability based model. Given n different independent features that need to be classified into k different classes, instance probabilities is given by :

$$p\left(C_k \mid x_1, \ldots, x_n\right) \tag{3.1}$$

where C represents the different classes and x represents the different features to be considered. Applying Bayes' theorem, this equation then becomes :

$$p\left(C_k \mid \mathbf{x}\right) = \frac{p\left(C_k\right) p\left(\mathbf{x} \mid C_k\right)}{p(\mathbf{x})} \tag{3.2}$$

or using Bayesian terminology :

$$\text{posterior } = \frac{\text{prior } \times \text{ likelihood}}{\text{evidence}} \tag{3.3}$$

Practically speaking, the numerator is what we care about since the denominator is independent of C_i. The numerator can then by application of chain rule and definition of conditional probability be written as :

$$
\begin{aligned}
p\left(C_k, x_1, \ldots, x_n\right) &= p\left(x_1, \ldots, x_n, C_k\right) \\
&= p\left(x_1 \mid x_2, \ldots, x_n, C_k\right) p\left(x_2, \ldots, x_n, C_k\right) \\
&= p\left(x_1 \mid x_2, \ldots, x_n, C_k\right) p\left(x_2 \mid x_3, \ldots, x_n, C_k\right) p\left(x_3, \ldots, x_n, C_k\right) \\
&= \cdots \\
&= p\left(x_1 \mid x_2, \ldots, x_n, C_k\right) p\left(x_2 \mid x_3, \ldots, x_n, C_k\right) \cdots p\left(x_{n-1} \mid x_n, C_k\right) p\left(x_n \mid C_k\right) p\left(C_k\right)
\end{aligned} \tag{3.4}
$$

Now using the 'naive' assumption, all features x_i are independent and thus, the joint model can be written as :

$$
\begin{aligned}
p\left(C_k \mid x_1, \ldots, x_n\right) &\propto p\left(C_k, x_1, \ldots, x_n\right) \\
&\propto p\left(C_k\right) p\left(x_1 \mid C_k\right) p\left(x_2 \mid C_k\right) p\left(x_3 \mid C_k\right) \cdots \\
&\propto p\left(C_k\right) \prod_{i=1}^{n} p\left(x_i \mid C_k\right)
\end{aligned} \tag{3.5}
$$

This above derived model is a naive Bayes probability model. This is combined with decision rule to get the a classifier. One of the simplest and most widely used decision is to select the most probable hypothesis or also called maximum a posteriori or MAP decision rule. In our case, we'll be modelling with VAR as input variable and we'll try to classify it into 50 classes created by using various combination of quantiles and increase/decrease. It's also important to remember that the distribution of the input variable is assumed to be Gaussian distributed.

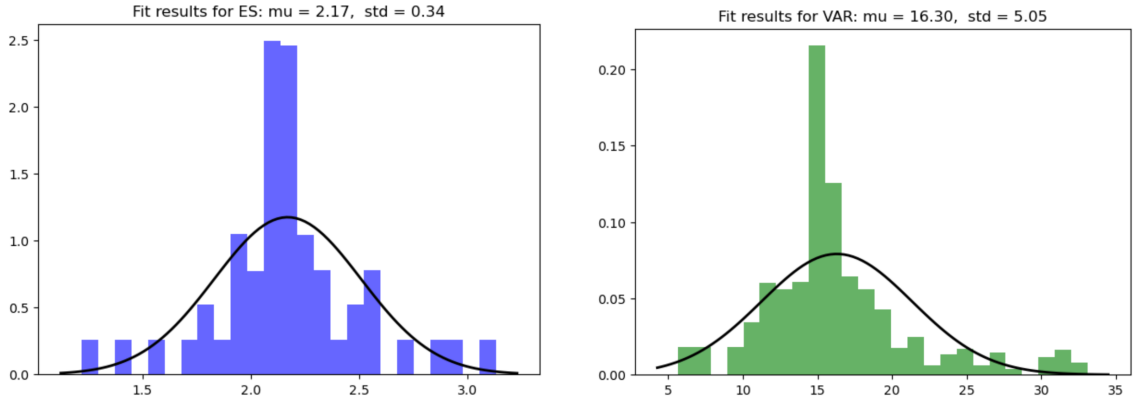

Figure 3.5: Histogram of the ES and VAR data and fitting it to a Gaussian distribution as required by the Naive Bayes classifier assumption.

As seen from the above figures, our data doesn't fit the Gaussian distribution very well and that's a major assumption of the model which doesn't hold too well. For VAR, we can see there is a slight skewness towards the right hand side, whereas both ES and VAR have excess positive kurtosis. Our implementation of Naive Bayes is quite simple. We

use the inbuilt Gaussian Naive Bayes classifier for modeling available through sklearn [10]. Our training and test set is split in the ratio 8:2. As part of modelling, the conditional probabilities are calculated for all the 50 different classes, conditioned on the given VAR value. We then repeat this using ES data. Just as with our first set of NN models, we combine VAR values lying within 0.5 range and create new VAR values to map to different classes. Idea is by reducing the precision of VAR values, we are preventing from overfitting the data. In practical scenarios, the data calculated often has some tolerance to the precision since it can vary by around $\pm$ 1 percent depending on how it's calculated. So, we thought it'll be a good idea to explore this variation on the model data.

The next set of models we looked at were logistic regression model. Again the idea is trying to map non-linear relationship between VAR which is our input and classify it into the 50 classes. It's imperative to keep in mind the task we are trying to accomplish here i.e. to determine what are the factors affecting VAR or in other words if we get a particular value as our VAR, what combination of quantile and percentage change in operational loss could have led to that value. Logistic regression is used to model the probability of a certain event occurring. This could be a binary classification or a multi-class classification. Each of these classes are assigned a probability on classification such that they sum up to one. At it's core, the logistic regression uses a logistic function to segregate data into different classes. A logistic function is given by :

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \tag{3.6}$$

where L stands for the maximum value of the curve, x_0 stands for the midpoint of x, and k defines the steepness of the curve. If the range of x is $-\infty$ to $+\infty$ then the curve we get from the logistic function looks like a slanted and flattened S. The second key principle underlying logistic model is that the logarithm of odds (log-odds) of the dependent variable representing 1 is a linear combination of independent variables. Now assuming we have a model with x_1 and x_2 as the two predictors then using the log-odd relation described we get

$$\ell = \log_b \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \tag{3.7}$$

by taking exponent on both the sides, we further get :

$$\frac{p}{1-p} = b^{\beta_0 + \beta_1 x_1 + \beta_2 x_2} \tag{3.8}$$

thus by simple algebraic manipulation and probability of classifying an event as 1 is given by :

$$p = \frac{b^{\beta_0 + \beta_1 x_1 + \beta_2 x_2}}{b^{\beta_0 + \beta_1 x_1 + \beta_2 x_2} + 1} = \frac{1}{1 + b^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}} = S_b\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2\right) \tag{3.9}$$

where S is the sigmoid function. For our model, instead of going with ES and VAR separately as in the past models, we took a different route and used a combination of ES and VAR as the inputs. For our first model, we just use VAR as the input to the model and classify it into different classes. For our second model, we use ES

Bibliography

1. Coleman R. Operational Risk. *Wiley Encyclopedia of Operations Research and Management Science*. 2011. DOI: 10.1002/9780470400531.eorms0591
2. Khrestina MP, Dorofeev DI, Kachurina PA, Usubaliev TR, and Dobrotvorskiy AS. Development of algorithms for searching, analyzing and detecting fraudulent activities in the financial sphere. European Research Studies Journal 2017; 20
3. Sudjianto A, Nair S, Yuan M, Zhang A, Kern D, and Cela-Díaz F. Statistical methods for fighting financial crimes. Technometrics 2010; 52. DOI: 10.1198/TECH.2010.07032
4. Kannan S and Somasundaram K. Autoregressive-based outlier algorithm to detect money laundering activities. Journal of Money Laundering Control 2017; 20. DOI: 10.1108/JMLC-07-2016-0031
5. Zareapoor M and Shamsolmoali P. Application of credit card fraud detection: Based on bagging ensemble classifier. *Procedia Computer Science*. Vol. 48. C. 2015. DOI: 10.1016/j.procs.2015.04.201
6. Pun J and Lawryshyn Y. Improving Credit Card Fraud Detection using a Meta-Classification Strategy. International Journal of Computer Applications 2012; 56. DOI: 10.5120/8930-3007
7. Dal Pozzolo A. Adaptive Machine Learning for Credit Card Fraud Detection Declaration of Authorship. PhD Thesis 2015
8. Mhamane SS and Lobo LM. Internet banking fraud detection using HMM. *2012 3rd International Conference on Computing, Communication and Networking Technologies, ICCCNT 2012*. 2012. DOI: 10.1109/ICCCNT.2012.6395910
9. Linsmeier TJ and Pearson ND. Risk Measurement : An Introduction to Value at Risk. Working Paper 1996; 6153
10. scikit-learn 0.24.1 Other versions. Sklearn. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Appendix A

Title of first appendix

A.1 Code listing

If you wish to list code in your appendices, one way is to use the "verbatim" environment.