Variational Approximation & Markov Chain Monte Carlo for Posterior Distribution Approximation using BNNs

COM3023 – Continuous Assessment

720007925

March 21, 2025

Abstract

This report covers the key ideas of the Bayesian approach to Machine Learning. Bayesian Neural Networks (BNNs), treat the learning goal as to learn probability distributions; not to learn weights like traditional Artificial Neural Networks (ANNs). Given the challenge of deriving a closed-form solution for the posterior distribution in BNNs, this report outlines how to use Variational Approximation (VA) and Markov Chain Monte Carlo (MCMC) to approximate this distribution: with critical comparison being made between the two. The report makes reference to empirical results from testing the methods on a Wine Quality dataset. With BNNs trained which predict quality to a 0.95 Confidence Interval (CI) accuracy of 99.1%.

I certify that all material in this document which is not my own work has been identified.

I have used any GenAI tools for minor Latex formatting preparing this assessment.

1 Introduction

1.1 Problem Definition

This report aims to explore the differences between Variational Approximation (VA) and Markov Chain Monte Carlo (MCMC) for approximation of posterior distributions for training Bayesian Neural Networks (BNNs). First, some background motivation will be outlined, before moving on to discuss each of the two methods themselves. Comparisons will be made between the models with reference to empirical results from testing the methods on a real world Vinho Verde wine quality dataset.

1.2 Importance of Neural Networks

Neural Networks (NNs) in their simplest form are an interconnected networks of nodes and edges, through which data flows in order to make a prediction. The structure is modelled from the function of the human brain, where a web of interconnected neurons come together to make decisions [4].

When training traditional NNs, a set of weights is guessed at the start of the process and then these weights are incrementally modified via the process of back-propagation. Back-propagation adjusts weights according to a loss function, with the aim of minimising the loss of the model and maximising the accuracy of predictions. Over a series of iterations, the network is shown the full dataset and adjustments are made according to predictions. Each one of these iterations is called an Epoch.

It is important to note that in a traditional Artificial Neural Network (ANN), the task is to optimise the model weights in order to minimise overall loss. This differs from a Bayesian Neural Network (BNN) where the network's objective is to learn sets of parameters producing accurate distributions for data; more details of which are to be discussed in Section 1.3.

1.3 Bayesian Approach

The Bayesian approach to Machine Learning assumes there are underlying probabilistic distributions within data. This approach utilises Bayesian statistics and Bayes theorem to approximate these distributions. This Bayesian approach means on small datasets distribution approximations would eventually converge producing accurate results, unlike traditional ANNs which are sensitive to input data in regard to both quality and quantity.

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta) \cdot p(\theta)}{p(\mathcal{D})} \tag{1}$$

$$p(y_i|x_i, \mathcal{D}) = \int_{\Theta} p(y_i|x_i, \theta) p(\theta|\mathcal{D}) d\theta \tag{2}$$

Eq.1 outlines how a posterior distribution is calculated from the probability distributions of likelihood, weights, and observed data (evidence). Where θ is a set of model weights/assumptions and $\mathcal{D}$ is the data. $p(\theta)$ is the prior distribution of the weights, i.e. the assumed distribution of weights before any observations have been made. The posterior, $p(\theta|\mathcal{D})$, is the conditional probability distribution of the model weights given the observed data, differing from the likelihood $p(\mathcal{D}|\theta)$ which is the probability of data given the distribution weights [8]. To predict new data points we require Eq.2, which shows the conditional probability of a target variable y_i given an input feature set x_i and the observed dataset. This formula makes use of the posterior distribution in order to solve the predictive function. In this case we must have a method to obtain the posterior distribution, which is an intractable problem hence Approximation Inference is required. Once an acceptable posterior has been approximated, BNNs can be trained to estimate model parameters to minimise the loss of the predictive function.

1.4 Approximation Inference

Approximation Inference is a vital technique when training BNNs, due to the intractable nature of problems in higher dimensional space [8]. As dimensionality grows, distributions of parameters become more complex, meaning traditional estimation methods for distributions become less and less efficient, eventually becoming computationally infeasible. Whilst approximation inference addresses the issue of finding an exact form for a distribution, it is still susceptible to the curse of dimensionality. As the number of features grows, the methods become less efficient. This is why appropriate selection of inference method is crucial to the training and performance of models at scale.

A multitude of techniques are available for approximation inference. There are two main categories for it: Sampling methods and Variational Approximation. The most notable sampling method is Markov Chain Monte Carlo, which will be explored in Section 2.1. Variational Approximation is the other inference method and can be implemented using Mean Field Approximation or Variational Inference; these methods are to be discussed in Section 2.2.

2 Techniques

This report focusses on the methods of MCMC and VA, making comparison of the methods in reference to experimentation on a real world dataset. Other approximation inference methods such as Laplace Approximation (LA) do exist, however LA is most appropriate for unimodal distributions which can be well approximated with a Gaussian. This assumption does not hold for our context, therefore LA can be disregarded as an option.

2.1 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) aims to combine the statistical techniques of Monte Carlo simulations and Markov Chains in order to leverage both of their benefits. Monte Carlo simulation is a statistical method by which random samples can be taken. This allows a wide range of possibilities to be explored, estimating results for problems that may be deterministic or probabilistic [5]. Markov Chains are a special case stochastic process by which the state of the chain informs the transition probabilities for changing to a new state. In essence they are an implementation of conditional probability; importantly in a memoryless manner whereby the next transition depends only on the current state [9]. MCMC outlines a set of sampling methods to draw ordered samples from a predicted distribution, being able to handle complex distributions and in principle approximate the distribution itself [8]. As the Markov Chain progresses, some of the earlier samples may not be correctly approximating the distribution. As such, there is a "warm up" or "burn in" period in which the samples are disregarded until the point the chain reaches a stationary distribution. Given a sufficiently long burn-in period and a well-configured chain, MCMC will asymptotically generate samples from the true distribution, ensuring convergence in the limit.

In practice, an algorithm must be used to implement the MCMC, one of the most popular being the Hamiltonian Monte Carlo (HMC) algorithm [1]. HMC is an instance of the Metropolis Hastings algorithm which takes advantage of some efficiency techniques to improve computation. An extension again on HMC is the No U-Turn Sampler (NUTS) algorithm which removes the requirement for a step number parameter to be selected and adapts the step length dynamically to avoid inefficient sampling [7]. Overall NUTS provides a more efficient method for computing HMC without the requirement for manual parameter tuning. Gibbs sampling is another MCMC implementation method which instead updates one variable at a time, rather than making the jumps of HMC. The algorithm behind Gibbs sampling is far simpler to implement however it struggles to capture complex relationships within data like HMC [1]. Considering a variety of python libraries have pre-packaged implementations of NUTS, the complexity of hard coded implementation is not a concern. Due to the ability of HMC & NUTS to capture more complex distributions often associated with real world noisy data, NUTS will be selected for testing and comparing methods.

2.1.1 MCMC Advantages

The main advantage of MCMC is its ability to capture complex relationships within data and distributions [1]. This means MCMC is an essential technique for capturing relationships in data with many features and high noise levels. For scientific applications it is often the best approximation inference method due to its higher accuracy in scenarios involving risk or uncertainty. When a small amount of valuable data is concerned, using MCMC may be the most appropriate method since on small data the computational overhead is reduced, so that drawback has less effect. The convergence to an asymptotically exact sample series is precisely the benefit that ensures the capturing of complex distributions. Given enough of a burn in, and an appropriate set up, it will reach this convergence.

2.1.2 MCMC Disadvantages

The efficiency of MCMC comes at its detriment, with computation on large high feature datasets being extremely slow and inefficient, even after optimisation extensions such as NUTS. This effect means that very often with large datasets even though MCMC can produce asymptotically exact results, it may be an infeasible technique because of the computation time. If many trials and hyper-parameter tuning needs to be done, it becomes infeasible to consistently run more and more MCMC approximations. In these cases, Variational Approximation may be an appropriate alternative, offering faster approximations at the cost of some accuracy.

2.2 Variational Approximation

Variational approximation (VA) treats approximating the posterior distribution as an optimisation method. With VA, a family $\mathcal{Q}$ of approximated densities is proposed, and the Kullback-Leibler (KL) Divergence measure is used to find the optimal option from the set [2]. KL Divergence measures the difference between two distributions, with a KL measure of 0 meaning the distributions are identical. By using KL divergence as an optimisation goal, the model moves towards having as close to equivalent distributions as possible by finding the *argmin* over the set $\mathcal{Q}$, as shown in Eq.3, where $q^*(\theta)$ is the best approximation of the posterior $p(\theta|x)$. One of the main challenges is to select the set $\mathcal{Q}$ such that there is enough complexity whilst remaining efficient.

$$q^*(\theta) = \underset{q(\theta)\in\mathcal{Q}}{\arg\min}\ KL(q(\theta)||p(\theta|x)) \qquad (3)$$

$$\text{ELBO} = \mathbb{E}_{q(\theta)}\left[\log\frac{p(\theta|x)}{q(\theta)}\right] \qquad (4)$$

The minimisation of KL divergence is the general idea behind VA, however it is not computable since it would require knowing the posterior which is what we are aiming to approximate in the first place. Instead we use the Evidence Lower Bound (ELBO) shown in Eq.4; a lower bound for the log-likelihood of observed data [2]. By instead maximising ELBO, we indirectly minimise the KL divergence between the approximated distribution and the true posterior. This allows us to approximate the posterior without needing to compute it explicitly.

In practice VA is derived by performing Variational Inference (VI), the algorithmic process computing approximate complex posteriors. A common approach to VI is Mean-Field Approximation (MFA), which assumes the distribution can be factorised into independent distributions for each latent variable [2]. This approach reduces the task of approximating a high dimensional posterior to that of approximating simpler distributions. Because of this MFA is reasonably efficient to compute, even if the approximate is less accurate than that of MCMC.

Stochastic Variational Inference (SVI) is an extension of VI that allows more scalability of approximations, especially for large datasets [6]. Since MFA assumes variables can be factorised independently, it misses out on the complex relationships between features. SVI addresses this limitation by incorporating stochastic optimisation into the VI process by processing the dataset in random mini-batches. The batching of data means the method is much more scalable for large datasets whilst maintaining the flexibility to capture feature dependencies [2]. As a result, the implementation of VA we will use is SVI.

2.2.1 VA Advantages

One of the main advantages of VA over MCMC is the ability to efficiently handle high sample, high feature datasets [6]. By treating posterior approximation as an optimisation problem, VA is able to reach an approximate form far faster than MCMC. When working with real world data, especially in performance critical applications such as real time computation, this efficiency becomes crucial. As datasets expand, as is expected in real world applications, improved efficiency becomes paramount when approximating posteriors. The efficiency benefit of VA means it is extremely useful for quickly testing many models to find which one has the best architecture, providing effective scalability. There is also no burn in period which means results can be drawn faster. Additionally, since VA relies on optimisation, it can make use of stochastic optimisation for quicker convergence, as seen with SVI, which again improves the performance.

2.2.2 VA Disadvantages

In contrast to MCMC, VA does not produce an asymptotically exact posterior, meaning the accuracy of approximated distributions is not as strong as is with MCMC. VA will produce somewhat accurate approximations but unlike MCMC there is no guarantee these approximations eventually reach the true distribution. VA also is less likely to appropriately express uncertainty in a model. When training BNNs, one of the advantages over ANNs is the ability to express uncertainty in predictions. With VA, this uncertainty is not expressed as accurately as MCMC, meaning some advantage of BNNs is negated.

3 Experimental Design

3.1 Dataset

The dataset being used to test the BNN methods is made up of two sets of data regarding wine quality [11]. The dataset was chosen due to it containing reasonable noise, meaning it will provide a good foundation for real world application of methods. One dataset contains data for the Red Vinho Verde wine quality (1599 samples), and a second contains details for the White Vinho Verde wine (4898 samples). These datasets must be kept separate due to the natural differences between red and white wine which affect features and would naturally disrupt the BNN. Table 1 outlines the features in the dataset, as well as a short description and the nature of the feature. It is important to note that due to privacy reasons there is no data provided regarding the grape variety, vineyard, origin, or producer. By inspecting the covariance matrix of the normalised data, no features seem necessarily more or less useful, hence all will be used for training.

The quality class distribution roughly follows a Gaussian distribution which is a useful observation for building prediction models. These classes could either be treated as a regression task or a classification task, due to the nature of the numerical class labels. The class imbalance present may indicate some over or under sampling methods may need to be explored in order to ensure any models produced are not biased. These techniques will be outlined in Section 3.2.

Feature Name	**Feature Description**	**Type**	**NaNs**
fixed acidity	Stable acidity content from grapes or additives not lost in fermentation	Continuous	0
volatile acidity	Gaseous acid content – often contributing to vinegary smell	Continuous	0
citric acid	Citric acid content, often added to increase acidity	Continuous	0
residual sugar	Residual sugar content after fermentation	Continuous	0
chlorines	Chlorine content of the wine – a major indicator of mold in the cork	Continuous	0
free sulfur dioxide	The portion of sulfur dioxide (SO_2) that is not bound to other compounds	Continuous	0
total sulfur dioxide	The total Sulfur Dioxide (SO_2) content of the wine	Continuous	0
density	Density of the wine – often related to sweetness (dry/sweet)	Continuous	0
pH	pH of the wine	Continuous	0
sulphates	Sulphate content of the wine – used as a preservative	Continuous	0
alcohol	Alcohol percentage content	Continuous	0
quality	Quality rating of the wine (1-10)	Categorical	0

Table 1: Description of features in the Wine Quality Dataset

3.2 Data Pipeline

Figure 1 outlines the pipeline for the flow of data for training the models. The following sections outline and justify the techniques being used at each stage of the pipeline. The preprocessing steps are a crucial technique for building ML models, ensuring the data is standardised and consistent in order to build precise models. In the model training section, first either VA or MCMC must be applied to approximate the posterior and then a BNN can be trained. The BNN outputs a trained model which can be shown new, unseen data and evaluation metrics can be produced based on this.

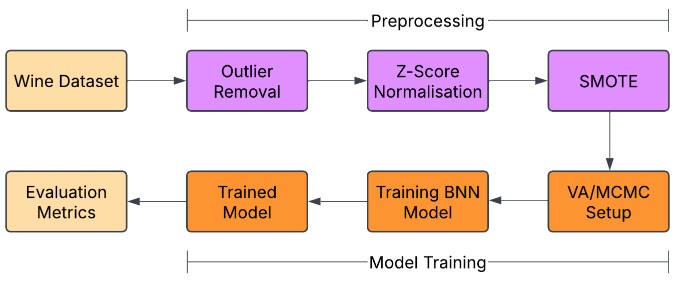


Figure 1: Data Processing Pipeline

3.2.1 Preprocessing

To prepare data for passing through a BNN, first any 'Not a Number' (NaN) values need to be dealt with. Within the context of the wine quality dataset, there are no NaN values present in the dataset; as shown by Table 1. If there was, they would have to be dealt with by either estimating them, removing them, or imputing them. At this point it is also important to remove duplicated data as this may disrupt the NN. The red wine dataset contains 140 duplicated rows, and the white dataset contains 937.

$$Z = \frac{(X - \mu)}{\sigma} \tag{5}$$

Eq.5 describes how data can be normalised using Z-Score normalisation [10], where X is a data point, μ is the feature mean and σ is the standard deviation of the feature. The normalisation method reduces μ and σ to 0 and 1 respectively. This ensures that the model doesn't penalise features that have smaller minimum and maximum values since all features have a similar range. In the context of our dataset, this ensures that "total sulfur dioxide", with a mean of 46.47, has the same feature importance as "chlorides", with an average of 0.09. If raw data needs to be rederived after scaling we can use the inverse can be intuitively derived. In Python, `scikit-learn` has the `StandardScaler` object which provides simple functionality for scaling.

To address the class imbalance in both of the datasets, some rebalancing techniques are required. With the red and white wine datasets having a relatively low number of samples (1599 and 4898 respectively), undersampling may not be appropriate since it would remove samples from the majority class. This would leave less data in the set for training, which would likely yield a model with low complexity. This indicates oversampling is likely more appropriate for our context. Synthetic Minority Oversampling Technique (SMOTE) [3] creates synthetic data for the minority classes in order to resample dataset to restore some class balance. New data points are imputed by using linear combinations of existing datapoints to create data that is new, but logically similar. This ensures some data integrity is kept whilst improving the imbalance. Random Oversampling (ROS) [12] is another option, however it instead duplicates datapoints which comes with an increased risk of overfitting – where models learn the data as opposed to the underlying pattern.

In order to address outliers in the data, a threshold is set for outliers and samples where the Z-score value is higher than the threshold will be removed. By inspection of the data, it can be seen that outliers are present in the "total sulfur dioxide", where the standard deviation for the column is extremely high for both red and white wine. Removing these outliers ensures minimal noise makes its way into the training dataset and makes sure the BNN learns patterns and not the noise and outliers.

3.3 Evaluation Methods

In order to fairly test models, we must split our dataset into two sets, one for training and one for testing. This ensures the models are evaluated on unseen data and prevents the models learning the data rather than the patterns, i.e. overfitting. A standard test size of 0.3 will be used, which will provide enough training data but also prevent model overfitting.

To evaluate the models a range of evaluation metrics will be chosen. Mean Square Error (MSE) shown in Eq.6 is the measure of the average deviance between observations and their predicted value. Each difference is squared to penalise models more for higher differences. Similarly, Mean Absolute Error (MAE) calculates the average absolute error over the predicted values, this provides the real absolute mean of the error.

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - y_i')^2 \tag{6}$$

$$ACC = \frac{Correct\ Predictions}{Total\ Samples} \tag{7}$$

Accuracy is also another useful error metric, shown in Eq.7. Accuracy is generally only useful for classification tasks, measuring the number of correct classifications for test inputs over a test set. In our context, when testing a regression model we can round our outputs to the nearest whole number and treat that as its classification, meaning accuracy can be used across the board. Accuracy is most insightful for binary classification since there are just two class labels. When the model is a multi-class predictor, accuracy may be notably lower due to the higher range of possible labels. To mitigate this, two other metrics can be used, one measuring accuracy where the predicted label is within a ±1 tolerance, and another related to Confidence Intervals (CI). A major benefit of working with BNNs is that models output not only predictions, but also a range noting how confident the prediction is. For evaluation of our model we will use CI Accuracy to express if the true label sits within a 95% CI of the predicted label; i.e. is the correct classification in the range the model is 95% certain the true label sits in.

4 Results & Discussion

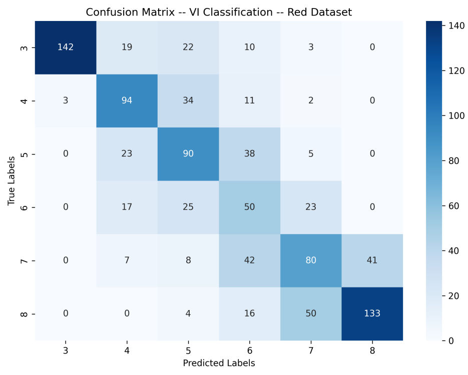
Table 2 displays the results obtained from a series of training trials of BNNs. Each row represents a different combination of dataset, training model, and posterior approximation method. Overall from the table it can be seen that models running on the Red wine dataset performed significantly better, this is likely due to the smaller size of dataset meaning less overall noise in the set. The confusion matrices in Figure 2 show the classification report for two particular models implementing VI and MCMC. It can be seen that class 3 and 8 have the highest correct classification rate. This is likely due to the use of SMOTE to rebalance the datasets, since classes 3 and 8 were the classes with the least raw data, hence the most over-sampled classes.

Dataset	Method	Model Type	Time(s)	MSE	MAE	Acc.	Acc. (±1)	CI Acc.
Red	MCMC	Regression	161.6	**0.7912**	0.7179	0.4022	**0.9163**	0.9345
	VI	Regression	**82.54**	0.8266	0.74	0.375	0.9012	0.9607
Red	MCMC	Classification	203.78	0.9365	0.5635	**0.5806**	0.8942	0.9698
	VI	Classification	84.03	0.8508	**0.5504**	0.5756	0.8962	**0.9909**
Red	MCMC	Regression	241.56	1.9489	1.1258	0.2641	0.7293	0.9379
	VI	Regression	100.98	1.9861	1.1296	0.274	0.7125	0.9585
Red	MCMC	Classification	329.01	1.5667	0.7562	0.5151	0.8262	0.9791
	VI	Classification	111.96	1.5214	0.7312	0.5266	0.841	0.9846

Table 2: Bayesian Neural Network Model Performance Comparison

When running the models, MCMC consistently took far longer to make an approximation for the posterior distribution than VI. This is naturally understandable and one of the major criticisms of MCMC. VI with 30,000 iterations took significantly less time to train and as such could be a preferable method on especially large datasets. Referring to Table 2, the relative results of VI are reasonably comparable to that of MCMC, and very often outperform it.

The plot of Layer1 posterior distribution across parameter values shown in Figure 3a, shows an interesting result regarding the uncertainty of the posterior approximations. The flatter, wider distribution of MCMC shows that it has much better understanding of uncertainty in parameter values. In contrast, VI's sharp peak demonstrates high confidence of predictions, meaning VI is underestimating parameter uncertainty by assuming parameters are independent. The collapse of VI may imply the SVI algorithm has found a local optimum, not a global one, meaning the distribution may not represent the true nature of the posterior. This overall means samples taken from VI may be overconfident in predictions, unlike MCMC which may capture more of the true nature of parameter distribution. The lower prediction uncertainty of MCMC is supported by Figure 3b, which shows the uncertainty of MCMC is consistently lower than that of VI when smoothed and plotted.


(a) VI Results

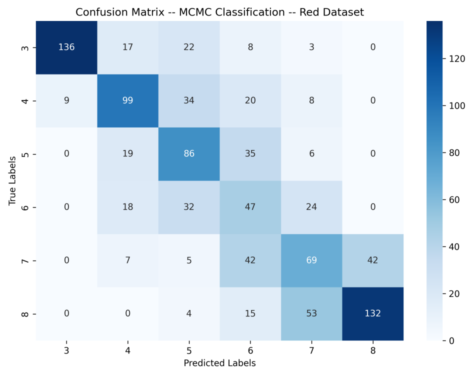
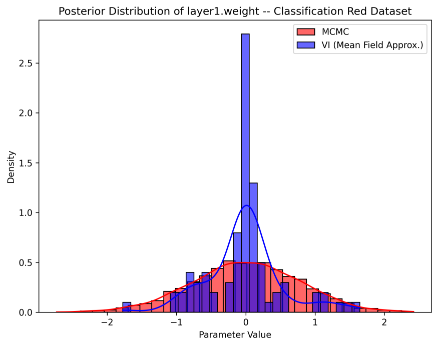

(b) MCMC Results

Figure 2: Confusion Matrices for Classification on Red Wine Dataset


(a) Layer1 Posterior Distribution

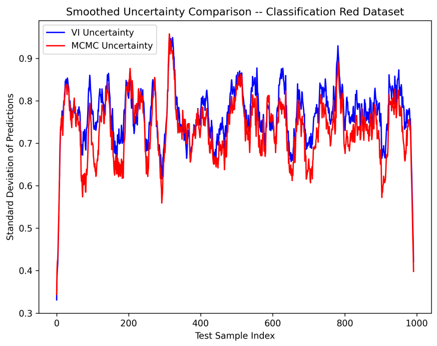

(b) Uncertainty Plot Comparing VI & MCMC

Figure 3: Distribution Plots for Comparison of VI & MCMC Red Classification

5 Conclusion

Overall, it can be seen that whilst MCMC may capture uncertainty of parameters better, and consistently produce lower prediction uncertainty over VA, VA may in some cases produce higher accuracy. In spite of this higher accuracy metric, VA is also likely to underestimate the uncertainty of parameters and be overconfident in predictions. This means when applying approximation inference to real world scenarios, if accurately capturing uncertainty is a priority, MCMC may be preferable over VA. However, if the dataset being worked with is high feature and high sample, VA may be a more useful method due to improved efficiency when compared to MCMC.

In our empirical context, VI did perform better on aggregate although it did fail to accurately capture the uncertainty of it's predictions. This is not necessarily a concern for the context of predicting wine quality, however if our context were high risk and capturing uncertainty was paramount to safety, VI may not be appropriate. VI however did run on average 139.11 seconds quicker than MCMC. This supports the efficiency benefit of VI one may want to leverage for performance sensitive applications, or when testing models at scale.

The selection of approximation inference technique largely relies on context of the work being undertaken. If you have a lot of data with a lot of noise that you need to produce a posterior for quickly, or you want to experiment with a range of parameters and models through lots of tests: VA is likely the best choice. However, if the accuracy of the posterior approximation is critical, and there is a strong need to capture prediction uncertaincy: MCMC is likely the better option.

References

[1] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50:5–43, 2003.

[2] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[3] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[4] AD Dongare, RR Kharde, Amit D Kachare, et al. Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1):189–194, 2012.

[5] John H Halton. A retrospective and prospective survey of the monte carlo method. *Siam review*, 12(1):1–63, 1970.

[6] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *the Journal of machine Learning research*, 14(1):1303–1347, 2013.

[7] Matthew D. Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo, 2011.

[8] Martin Magris and Alexandros Iosifidis. Bayesian learning for neural networks: an algorithmic survey. *Artificial Intelligence Review*, 56(10):11773–11823, 2023.

[9] James R Norris. *Markov chains*. Number 2. Cambridge university press, 1998.

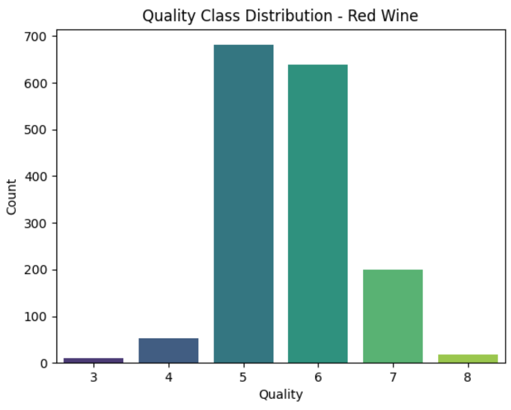
[10] SGOPAL Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.

[11] Cortez Paulo, Cerdeira A., Almeida F., Matos T., , and Reis J. Wine Quality. UCI Machine Learning Repository, 2009. DOI: https://doi.org/10.24432/C56S3T.

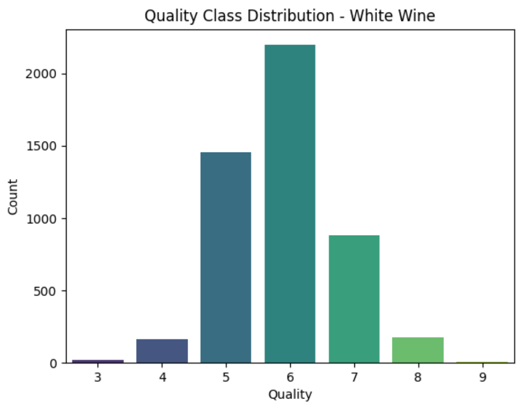
[12] Mayuri S Shelke, Prashant R Deshmukh, and Vijaya K Shandilya. A review on imbalanced data handling using undersampling and oversampling technique. *Int. J. Recent Trends Eng. Res*, 3(4):444–449, 2017.

Appendix

Supplementary Plots & Tables

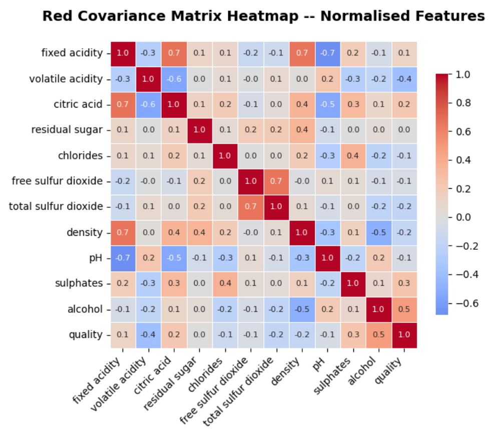


(a) Red Wine Quality Distribution

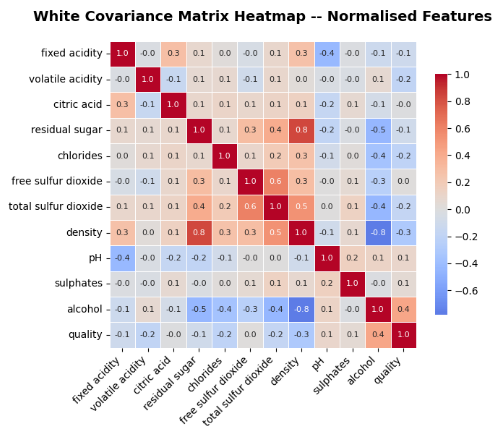


(b) White Wine Quality Distribution

Figure 4: Quality Class Distribution Plots



(a) Red Wine Covariance Matrix



(b) White Wine Covariance Matrix

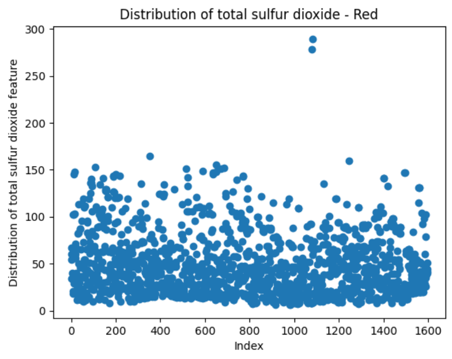
Figure 5: Covariance feature plots

Feature Name	**Min**	**Max**	**Mean**	**Std**
fixed acidity	4.6	15.9	8.32	1.74
volatile acidity	0.12	1.58	0.53	0.18
citric acid	0.0	1.0	0.27	0.19
residual sugar	0.9	15.5	2.54	1.41
chlorides	0.01	0.61	0.09	0.05
free sulfur dioxide	1.0	72.0	15.87	10.46
total sulfur dioxide	6.0	289.0	46.47	32.9
density	0.99	1.0	1.0	0.0
pH	2.74	4.01	3.31	0.15
sulphates	0.33	2.0	0.66	0.17
alcohol	8.4	14.9	10.42	1.07
quality	3	8	5.64	0.81

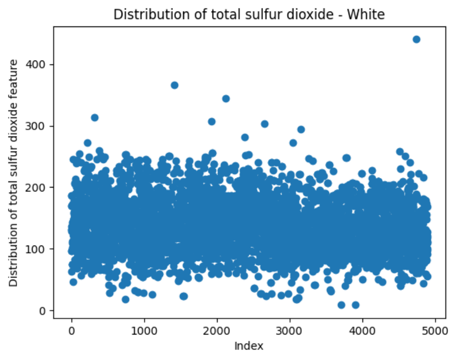
Table 3: Red Wine Feature Statistics

Feature Name	**Min**	**Max**	**Mean**	**Std**
fixed acidity	3.8	14.2	6.85	0.84
volatile acidity	0.08	1.1	0.28	0.1
citric acid	0.0	1.66	0.33	0.12
residual sugar	0.6	65.8	6.39	5.07
chlorides	0.01	0.35	0.05	0.02
free sulfur dioxide	2.0	289.0	35.31	17.01
total sulfur dioxide	9.0	440.0	138.36	42.5
density	0.99	1.04	0.99	0.0
pH	2.72	3.82	3.19	0.15
sulphates	0.22	1.08	0.49	0.11
alcohol	8.0	14.2	10.51	1.23
quality	3	9	5.88	0.89

Table 4: White Wine Feature Statistics



(a) Red Wine "total sulfur dioxide" distribution



(b) White Wine "total sulfur dioxide" distribution

Figure 6: Distributions of Total Sulfur Dioxide – Showing Potential Outliers