Error Analysis on Machine Learning Techniques

Sid Diamond (CID 02227730)

Henry Hodges (CID 02221628

Aurelius Caesar (CID 02043518)

TABLE OF CONTENTS

ABSTRACT

Transformer-based machine learning models are ubiquitous in modern artificial intelligence, yet their interpretability and reliability are inherently limited by the adequacy of the uncertainty quantification used. This paper introduces four transformer-compatible distinct frameworks to address this challenge: two focusing on fine-tuning transformers for supervised text classification and two on unsupervised topic clustering, including dimensionality reduction and information loss.

First, we propose an approximated diagonal Fisher Information Matrix (ADFIM) to estimate per-parameter variance in fine-tuned transformers. We find empirical results revealing a negative correlation between ADFIM variance and the importance of parameters for text classification quality. However, the assumptions underlying ADFIM variance restrict its interpretation as a physical variance capable of representing meaningful uncertainty the classification output.

Second, we employ Monte Carlo dropout to simulate Bayesian neural networks, estimating output uncertainty through statistical analysis of the output of repeated stochastic forward passes. This method also highlights a negative correlation between model confidence and measured uncertainty for classification tasks.

The third section demonstrates use of bootstrap resampling to quantify stability and uncertainty in clustering outputs from unsupervised models, highlighting the variability in cluster assignments under data perturbations.

Finally, discuss how we can evaluate the performance of the Uniform Manifold Approximation and Projection (UMAP) technique for dimensionality reduction. By analysing cross-entropy, mutual information, and Fisher information, we identify trade-offs between information stability and the retention of local and global data structures. Results show that UMAP distributes information loss unevenly across data points, with regions of high distortion correlating with increased sensitivity to perturbations.

1. INTRODUCTION

The widespread adoption of transformer-based machine learning models necessitates methods for quantifying uncertainty metrics in their parameters and outputs. Traditional statistical techniques often falter with transformers due to their large-scale, non-linear architectures, prompting the need for alternative approaches. This paper introduces four frameworks to address uncertainty in transformers applied to text classification tasks, specifically categorizing political ideology as conservative and applying topic clustering to it.

The first framework employs a Fisher information-based method to approximate per-parameter variance during transformer training. Using an approximated diagonal Fisher Information Matrix (ADFIM), we estimate variance as the reciprocal of specific matrix elements under key assumptions. While ADFIM-derived variance correlates negatively with parameter importance for classification quality, its non-physical nature limits its application for overall output uncertainty estimation.

In the second section we provide a method to measure overall output uncertainty on supervised ML models. By passing through a standard neural network multiple times, setting random weights and parameters to zero (a process called Monte Carlo Dropout) we can simulate a probabilistic neural network (a 'Bayesian neural network'). This allowed the calculation of the variance of the output probability density function to be modelled as the relative statistical uncertainty of the output. The third section of the report describes how for unsupervised ML models, bootstrap resampling was utilized to assess the stability of the model's outputs. This stability is analogous to the uncertainty of the model, offering insights into the reliability of predictions produced by unsupervised systems. The fourth section aims to quantify errors and uncertainty in UMAP, a dimensionality reduction method used in unsupervised learning. By examining information loss and sensitivity using statistical methods, it identifies areas of high uncertainty. Balancing structural preservation with information loss is crucial for reliable tasks like clustering and classification.

2. Approximating Transformer Parameter Variance with the Diagonal Fisher Information

2.1. Transformers and Fine-Tuning

A transformer is a neural network architecture for processing sequential data, such as text, using self-attention. A full mathematical description is notoriously complex. The figure below (fig 2.0), from Anthropic's *A Mathematical Framework for Transformer Circuits* [1], outlines its high-level architecture, providing key terminology.

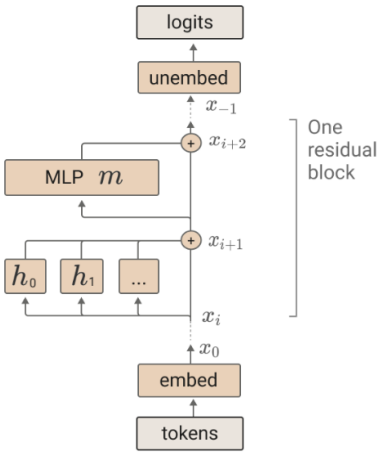

Fig 2.0 - Diagram forming the basis for the high-level understanding of a transformer we present.

The transformer begins with tokenization, dividing text into tokens (e.g., words or sub-words). An embedding matrix W_E maps tokens into a high-dimensional vector space, where semantically similar tokens are closer together. These embedded vectors, x, pass through an Attention Block with $h \in H$ parallel attention heads. Each head, with its own weight matrices W_Q, W_K, and W_V, performs matrix multiplications to compute attention scores, assessing word relationships. The outputs are concatenated and linearly transformed by an output weight matrix W_O, a learned parameter tensor capturing contextual token dependencies. Following the attention heads, a multi-layer perceptron (MLP) updates vectors independently within the residual block. The MLP details are beyond this paper's scope. The model concludes by transforming the final vector into logits via an unembedding matrix W_U. For text classification, these logits represent a probability distribution interpretable as the result. Given the transformer size, we used a pre-trained transformer, trained on masses of general data with millions of parameters. This pre-training encodes knowledge that greatly improves performance on downstream tasks like text categorization [2].

Whilst the commonly used BERT-Base-Uncased model achieves strong performance with 66 million parameters, its size made it impractical for computations on an 8GB laptop [3]. We used the pre-trained ALBERT Base v2 with 11 million parameters and a 768-dimensional embedding vector, using parameter sharing between layers to deliver performance comparable to BERT while being more size-efficient [4]. Introducing,

Assumption 1: *Parameter sharing in ALBERT does not invalidate the proposed variance calculation.*

So, given the use of a pre-trained model, how can we fine-tune it for our specific text classification task and how can we quantify the uncertainty attached?

The transformer was fine-tuned for political alignment analysis using the *political_ideologies* dataset from Hugging Face's library, categorizing statements as conservative or liberal [5]. This approach is readily applicable to other labelled datasets.

The weight matrices, W, consist of adjustable parameters that dictate the relationships between embedded vectors. We also introduce a second set of adjustable parameters, biases, b, modulating certain layer impacts by assigning activation thresholds. Certain weight parameter tensors are paired with corresponding bias tensors. By iteratively adjusting these parameters to minimize the discrepancy between the transformer's predictions and known value labelled data, we optimize the parameters for performance on unlabeled data. This works if only if the labeled data reasonably represents the unlabeled data distribution and the discrepancy minimization function (cost function) appropriately guides parameter updates. The cost function used was the PyTorch cross-entropy,

$$CE_j = -\ln(P_j) = -\ln\left(\frac{e^{Z_j}}{\Sigma_{i=1}^{K} e^{Z_i}}\right) \qquad (1)$$

Here, P_j, represents the j^{th} element of the probability distribution returned by the transformer, which has K elements. Z_j is the logit corresponding to the true value specified by the labeled data. Due to the properties of the natural logarithm, the function is smoothly minimized as

the predicted probability distribution becomes more accurate. These logits are given by:

$$Z = Wx + b \quad (2)$$

For earlier layers, the output is expressed as:

$$Z^{(l)} = W^l a^{l-1} + b^l \quad (3)$$

where a^{l-1} is the activation applied to the previous layer's output, using the GeLU function in ALBERT [4]. To minimize CE_j, gradients are computed to iteratively to optimize model parameters. This process, using the chain rule to compute gradients layer by layer, is known as backpropagation.

Increasing the amount of labelled data and the number of parameter update iterations generally improves fine-tuning but increases computation time. In this study, we fine-tuned the model using 400 data points, divided into 16 equal-sized batches, B.

We used the FAdam gradient descent optimizer to minimize each $CE(logit, label) \in B$ [6]. The negative gradient represents a direction vector in parameter space, pointing from current parameter values toward the optimizer determined minimizing value. The batch gradient was computed as the arithmetic mean across the batch:

$$g \equiv \overline{\nabla} CE_{batch} \quad (4)$$

Parameter values were updated by subtracting this direction vector scaled by a scalar learning rate η, set to 10^{-5}.

In high-dimensional parameter space, gradient calculations often lead to convergence at local minima. To mitigate this, a penalty term proportional to the weights, λ, is added, encouraging smaller weight solutions, smoothing the loss surface, and reducing local minima convergence. However, excessively large gradients cannot always be controlled by weight decay. To address this, we normalize gradients using the Euclidean (L^2) norm. If the norm exceeds a predefined threshold (c, the clipping factor), gradients are rescaled:

$$g_{clip} = g \cdot \frac{c}{\|g\|^2} \quad (5)$$

This ensures stability by preventing any gradient element from becoming disproportionately large. A clipping factor of 1 was applied.

The arithmetic mean reduces gradient approximation variance across batches. However, since recent gradients reflect more optimization updates, an exponential moving average (EMA) is used:

$$m_{B_t} = \beta_1 m_{B_{t-1}} + (1 - \beta_1) g_{clip,B_t} \quad (6)$$

such that,

$$\widehat{m}_t \triangleq \frac{m_{B_t}}{1 - \beta_1^t} \quad (7)$$

Here, β_1 determines the influence of past gradients. The FAdam optimizer also uses β_2 to compute an EMA of squared gradients v_t, which stabilizes the diagonal Fisher Information Matrix (FIM) [6]. The gradient update then becomes:

$$g_{FAdam} = \frac{\widehat{m}_t}{\epsilon + \sqrt{\widehat{v}_t}} \quad (8)$$

Where the ϵ term prevents division by zero. This study uses $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$. Combining this gives,

$$W^{B_t} \leftarrow W^{B_{t-1}} - \eta\left[g_{FAdam,B_t} + 2\lambda W^{B_{t-1}}\right] \quad (9)$$

Representing our fine-tuning operation. B_{t-1} is the batch preceding B_t. Starting with W^{B_t}, the next batch is processed, and this repeats across all batches, completing a full dataset pass (an epoch). Our fine-tuning ran over 8 epochs.

After fine-tuning, the F1 score, the harmonic mean of precision and recall, was used to evaluate classification accuracy:

$$\text{F1} = 2(\text{precision} + \text{recall})^{-1} \quad (10)$$

Precision is the ratio of true positives to the sum of true and false positives, while recall is the ratio of true positives to the sum of true positives and false negatives for a chosen output class in our dataset. As precision and recall approach 1, the F1 score also approaches 1, indicating perfect classification.

We now outline a possible framework to argue that g_{fadam,B_t} can be used to estimate per-parameter variance post fine-tuning.

2.2. Diagonal Fisher Information Theory

Statistical theory frames experimental results as observable, independent, identically distributed random variables (IIDs) drawn from a shared continuum with consistent parameters. A "good" estimate of the true parameters from sampled data, is:

1. $Unbiased$: The expected value, $E[\hat{\theta}]$, of the observed data equals the true value, θ^*, of the parameter being estimated.
2. $Consistent$: Converges in probability to the true parameter value in the limit that the sample size tends to infinity.
3. $Efficient$: For all unbiased estimators, an estimator is efficient if it has the smallest variance.

The likelihood function $L(observed\ data, \theta)$ fits this definition. For textual data, since outcomes are IIDs, the Kolmogorov axioms

allow the likelihood of the observations to be expressed as the product of the individual probabilities of each observed outcome.

$$L(p_i) = \prod_{i=1}^{N} P_i(observed\ data, \hat{\theta}) \quad (11)$$

A larger likelihood sum indicates a more probable model. If the observed data proportionally represents the entire dataset, the model serves as a good estimate for global data. To create a model capable of global data inference, we optimize free parameters by maximizing the likelihood function. For simplicity, consider a model with one free parameter (extendable to multi-parameter models using the gradient operator). Since the natural logarithm is monotonic, maximizing $L(\hat{\theta})$ is equivalent to maximizing $\ln(L(\hat{\theta}))$, leading to the log-likelihood function.

$$\nabla_{\hat{\theta}} \ln\left(P\left(observed\ data\ , \hat{\theta}\right)\right) = 0 \quad (12)$$

The solution to (12) provides the optimized $\hat{\theta}$, assuming we identify a maximum. This parameter estimation process is known as maximum likelihood estimation (MLE).

MLE minimizes KL divergence, quantifying the difference between the empirical and expected distributions and constraining the function space that the MLE function belongs to [7]. However, even at minimum KL divergence infinitely many functions may satisfy this constraint. Consider two scenarios near the true parameter: one with steep geometry and one with shallow geometry. In the shallow case, small deviations in the MLE estimator's function are more likely to cause larger discrepancies between the true and MLE parameter values than in the steep case. Intuitively, the minimized KL divergence constraint implies equivalent overall function deviation between the two cases. Thus, steep geometry is more likely to produce estimates closer to the true parameter. This demonstrates that the loss surface curvature encodes information about the likelihood of an accurate MLE estimate—information not captured by the MLE value alone.

The second derivative of a function measures its curvature at a point. Thus, the robustness of an MLE-derived prediction relative to the true parameter depends on the negative expected value of the second derivative of the log-likelihood function over the sampled data. The negative arises because maximizing likelihood corresponds to consideration of the lowest point on a loss surface. This quantity defines the Fisher information:

$$I(\theta^*) \coloneqq -E\left[\nabla^2_{\theta^*} \ln p(observed\ data;\ \theta^*)\right] \quad (13)$$

The quality of an estimator depends on the Fisher information of the IID sampled points. Given that a good estimator satisfies the efficiency condition with constrained variance, we infer a relationship: variance increases as Fisher information decreases, and vice versa. The following theorem, from Rigollet's lecture series, establishes the conditions under which this relationship holds [7].

Let $\theta^* \in \boldsymbol{\theta}$, then under the following assumptions,

1. The model is identified.
2. $\forall \theta^* \in \boldsymbol{\theta}$, the support of $P_{\hat{\theta}}$ does not depend on θ^*.
3. θ^* is not on the boundary of $\boldsymbol{\theta}$.
4. $I(\theta^*)$ is invertible in a neighbourhood of θ^*.
5. Technical conditions.

Then as $n \to \infty$, $\hat{\theta}_n \to \theta^*$, we recover the definition of consistency and,

$$\lim_{n\to\infty} \sqrt{n}\left(\hat{\theta}_n - \theta^*\right) = \mathcal{N}(0, I(\theta^*)^{-1}) \quad (14)$$

Referred to as the asymptotic normality condition. In our study, the cross-entropy loss used in the transformer acts as a likelihood function, with labelled parameters representing a sample of the global dataset for inference. If we hypothetically label all global data points, the output would converge to the true model parameters. Thus, with a sufficiently large and representative sample, and assuming the above conditions hold, we can estimate per-parameter variance using the FAdam optimizer's gradient, as it corresponds to the first derivative of the cross-entropy likelihood function. For multivariate parameter models, the Fisher information becomes a Fisher information matrix (FIM). Its diagonal elements represent per-parameter Fisher information, and the off-diagonals represent covariance terms, and the referenced Normal distribution extends to its multivariate form [8].

***Condition 1*:** For sequences longer than the attention head dimension, attention weights become unidentifiable [9]. In ALBERT v2, with an attention head dimension of 64, this limits textual inputs to 64 tokens (around 48 words).

***Condition 2*:** For the likelihood function, all parameters must be able to take any value in$(-\infty, \infty)$, and the probability must remain non-zero. This ensures the KL divergence, which

includes the log of the ratio of two densities, avoiding undefined $\ln(0)$ cases. If this condition were not met, the probabilities in the log-likelihood function would multiply to trivial zero vectors, but this is not observed.

Condition 3: If θ^*were on the boundary of the parameter space, the computed gradient would be undefined, yielding results that are not real numbers. This does not occur, so the condition is met.

Condition 4: We will only consider the diagonal elements of the Fisher information matrix so inverting the matrix reduces to taking the reciprocal of each diagonal element. The ϵ factor in g_{FAdam} essentially ensures we never divide by zero meaning the condition is met.

Condition 5: We could not find the exact technical conditions Rigollet references, limiting the validity of Fisher information-derived variance. We speculate these involve existence and continuity of derivatives. If so, Kustner, Balles, and Hennig's findings [10] disputing empirical Fisher approximations suggest these conditions may not be met, leading to,

Assumption 2*: Rigollet's additional technical conditions are satisfied.*

To compute Fisher information practically and avoid the cost of directly calculating the second derivative, we use an alternative but equivalent formulation:

$$I(\hat{\theta}) = E\left[\nabla_{\hat{\theta}} \ln p(obsv\ d;\ \hat{\theta})^2\right] = E[(g_{FAdam})^2] \quad (15)$$

note that $\hat{\theta}$ represents sample-derived parameter values implying the introduction of an asymptotic normality approximation. It is, however, standard to treat this as an equality.

To further reduce computational cost, we consider only the diagonal terms of the FIM. Amari et al. show that off-diagonal elements are smaller by an order of $1/\sqrt{n}$, where n is the number of matrix elements [11]. This reduces computation time from $O(n^2)$ to $O(n)$. For our sample size, the neglected terms are approximately 5% the information size of the retained diagonal terms. Despite this, the FAdam optimizer has demonstrated superior performance compared to the conventional Adam optimizer on the LLM 1B text dataset [6]. This introduces,

Assumption 3*: Transformer model parameters are approximated as independent.*

This is not accurate in transformer architectures and neglecting covariance terms underestimates the variance. We also approximate the true expectation value with a discrete, computable form using the arithmetic mean. Although this assumption underpins FAdam's success, its validity was not explicitly evaluated in this study. Again, the Kustner, Balles, and Hennig study challenges this approach [10]. Now however, the diagonal FIM is now computable:

$$I_{diag}(\hat{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(g_{FAdam} \odot g_{FAdam}) \quad (16)$$

$$\Longrightarrow \quad Var(\hat{\theta}) \approx \frac{1}{I_{diag}(\hat{\theta})} \quad (17)$$

Giving us a computable per parameter variance approximation based on this approximate diagonal FIM (ADFIM).

2.3. ADFIM Variance Discussion and Results

Current research shows that transformer layers contribute unevenly to result quality [13]. While the F1 score has limitations, it suffices as a performance metric for this analysis. Using ADFIM-derived parameter variances, we analyse layer-wise differences and propose that parameters with higher ADFIM variance have less impact on the transformer's F1 score, consistent intuition derived from physical variance.

To test this hypothesis, we compute the mean variances of weight and bias parameter tensors for each layer, approximating computational characteristics across layers. While analysing the highest and lowest mean variances offers initial insight, further study is needed for more definitive conclusions. Results indicate that the highest variance occurs in the word embedding parameter tensor, while the lowest variance is in \the classifier weight parameter tensor.

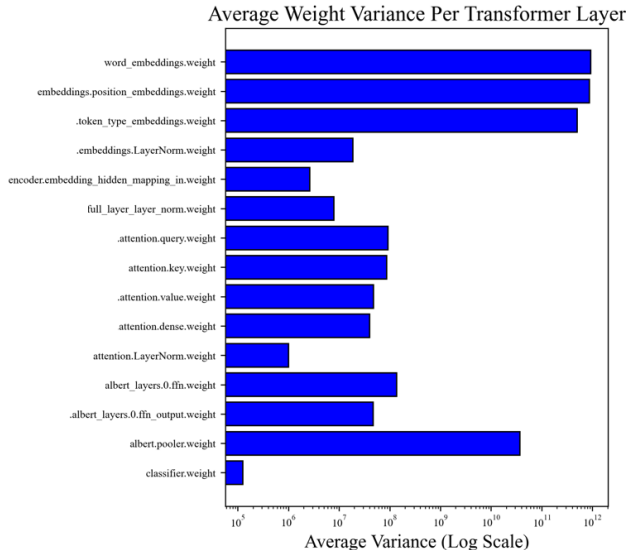

Fig 2.1 - Log-scale bar chart of the mean ADFIM variance per weight layer.

This result may support the lottery ticket hypothesis, which suggests that parameters near the output layer are more likely to form "winning tickets"—essential subnetworks for effective learning, especially in fine-tuning [13]. However, this relationship is not immediately clear from the overall layer distribution and requires further study.

Comparing per-parameter ADFIM variances shows that while most parameters in both the highest (embedding weights) and lowest (classifier weights) layers can be zeroed out, removing low-variance classifier weights significantly reduces the F1 score. Conversely, zeroing all embedding weights, the model still retained an F1 score above 0.34.

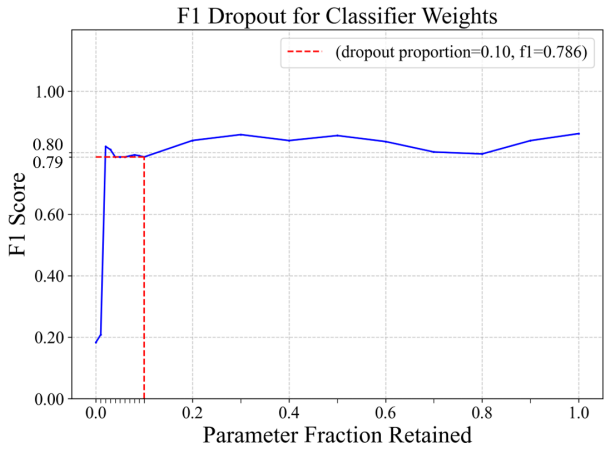

Fig 2.2 - plot of F1 score as parameters from the classifier weight layer (lowest variance layer) are zeroed out in 5% increments, arranged from low to high ADFIM variance. The red dashed line marks the F1 score when only the 10% lowest-variance weights remain.

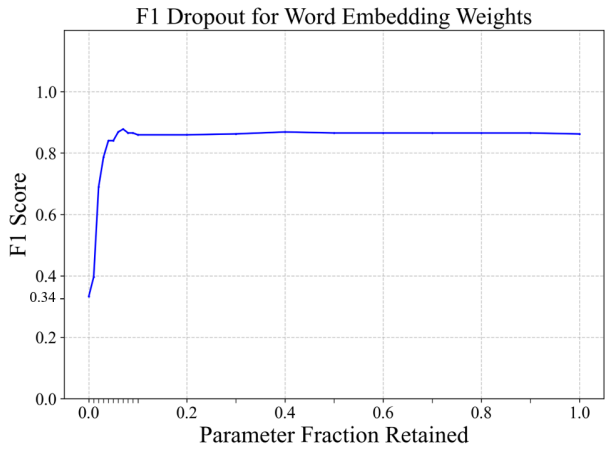

Fig 2.3 - plot showing the F1 score as parameters from the word embedding weight layer (highest variance layer) are zeroed out in 5% increments, arranged form low to high ADFIM variance

Removing the lowest 10% variance parameters in fig 2.2 reduced the F1 score to 0, showing that a small subset of classifier layer parameters is crucial for performance. Further refinement is left for future research. Retaining only the lowest 10% Fisher variance parameters in the classifier layer outperformed the non-fine-tuned pre-trained model by 3.67 standard deviations. This was determined by calculation of base-weight F1 scores across 10 data subsets. Even with 90% of high-variance weights removed, the fine-tuned model significantly outperforms its pre-trained counterpart.

The word embedding weight layer shows that zeroing out the entire layer results in a smaller F1 score drop compared to removing low-variance classifier weights, supporting our interpretation of ADFIM variance. At high-variance points within the layer, most parameters still have minimal impact on the F1 score, and even with all parameters removed, the F1 score remains above 0.34. This underscores the disparity in parameter importance between high and low-variance layers. While more rigorous testing and less speculative analysis are needed for definitive conclusions, these initial findings provide valuable insights into the varying significance of parameters in transformer architectures.

Finally, for the bias parameters, we observed that both the attention key bias (the largest ADFIM variance bias layer) and the classifier bias (the smallest ADFIM variance) had values on the order of 10^{-6}, indicating that their threshold activation impact was negligible.

To conclude, the following assumptions remain unproven:

Assumption 1**:** *Parameter sharing in ALBERT does not affect the proposed variance calculation.*
Assumption 2**:** *Rigollet's additional technical conditions are satisfied.*
Assumption 3**:** *Transformer model parameters can be approximated as independent.*

These limitations prevent us from asserting that the derived ADFIM variance is mathematically equivalent to a physical variance. Nevertheless, initial empirical findings suggest that ADFIM-derived variance is a promising, albeit not yet definitive, tool for assessing parameter importance. Specifically, removing low-variance ADFIM parameters has a greater impact on the classification F1 score compared to removing high-variance parameters. Moving forward, a non-physical variance complicates efforts to define the output uncertainty on these transformer text classification tasks, emphasizing the need for a different approach. The following section proposes such an approach.

3. Monte Carlo Dropout as a Measurement of Supervised ML Uncertainty

3.1. Theory

3.1.1. Bayesian Neural Networks

Bayesian neural networks (BNNs) resemble traditional neural networks but treat model weights and biases as probability density functions instead of discrete values. These follow Bayes' theorem given the training data as the posterior information required to form the probability distributions (Eq. 18).

$$\mathrm{p}(\mathbf{y}^*|\mathbf{x}^*,\mathbf{X},\mathbf{Y}) = \int \mathrm{p}(\mathbf{y}^*|\mathbf{x}^*,\mathbf{w})\mathrm{p}(\mathbf{w}|\mathbf{X},\mathbf{Y})\,\mathrm{d}\mathbf{w} \quad (18)$$

The weight PDFs are adjusted during backpropagation, with the output of the system producing continuous PDF outputs. This allows variance in the output to represent the algorithm's uncertainty, which makes BNN's conceptually extremely useful. However, they are rarely used within machine learning problems due to one major drawback: they are extremely computationally expensive to run for large neural networks. In practice, calculating the integral of the posterior distribution for all the weights in a large neural network is too complex for a system with millions of parameters. To work around this issue, we can propose a simpler distribution to approximate the posterior distribution that can be easily calculated for deep neural networks.

3.1.2. MC Dropout: Key Concepts

Monte Carlo (MC) dropout is a technique commonly used in neural networks (NNs), where random parameters are set to 0 with a probability p. It is primarily used during training to prevent overfitting (the model becoming too specialized to the training data) and to ensure the NN generalizes well to test data. Building on the idea of using simpler distributions for a BNN, MC dropout can be used to approximate the true posterior by repeatedly running a standard neural network, applying MC dropout each time. *Gal and Ghahramani (2016)* [14] showed that dropout can be viewed as variational inference with a Bernoulli prior over the weights. In practice, each forward pass applies a dropout mask, effectively sampling from an approximate posterior distribution of weights. Consequently, this approach enables a traditional neural network to behave like a Bayesian model.

To address BNNs' high computational cost, we can introduce a simpler proposal distribution, q(w), which is a valid approximation for the posterior if it minimizes the Kullback-Leibler divergence, a statistical method that compares the similarity of two PDFs. This Kullback-Leibler function can be extended to form the Evidence Lower Bound function (ELBO), which is maximised if the posterior distribution and proposed weight distribution are similar (Eq. 19).

$$\mathcal{L} = \int \mathrm{q}(\mathbf{w})\log\left(\mathrm{p}(\mathbf{Y}|\mathbf{X},\mathbf{W})\mathrm{d}\mathbf{w} - \mathrm{KL}(\mathrm{q}(\mathbf{w})||\mathrm{p}(\mathbf{w})\right) \quad (19)$$

3.1.3. Simulating Bayesian Neural Networks with Dropout

The first step in the process of using Monte Carlo (MC) Dropout is to approximate the BNN to be a gaussian process (GP), which assumes that the output of the system follows a gaussian distribution as its output probability density function. This is a valid approximation if we assume the system to be sufficiently large, (specifically, the NN needs to be sufficiently wide) meaning the product of the distribution functions of the weights will follow a gaussian distribution due to the central limit theorem. We

can mathematically model this approximation by first defining the training process of a BNN as $\boldsymbol{Y} = \boldsymbol{f}(\boldsymbol{X})$, for a vector of functions $\boldsymbol{f}$ and define F as a matrix of the output functions for the input values of $\boldsymbol{X}$. By modelling the BNN as a GP, we can see that the distributions of $\boldsymbol{Y}$ and $\boldsymbol{F}$ will follow (Eq. 20), (Eq. 21).

$$\mathrm{p}(\mathbf{F}|\mathbf{X}) \sim \mathcal{N}\big(0, \mathbf{K}(\mathbf{X}, \mathbf{X})\big) \tag{20}$$

$$\mathrm{p}(\mathbf{Y}|\mathbf{F}) \sim \mathcal{N}(\mathbf{F}, \tau^{-1}\mathbf{I}_{\mathrm{n}}) \tag{21}$$

τ^{-1} is a 'precision' hyperparameter that can be adjusted and $\mathbf{K}(\mathbf{X}, \mathbf{X})$ is the covariance matrix, measuring the similarity between two values within the $\mathbf{X}$ matrix, which will be derived in greater depth later. Combining these GP equations with Bayesian statistics gives us a new expression for the posterior distribution, described by (Eq. 22).

$$\mathrm{p}(\mathbf{Y}|\mathbf{X}, \mathbf{W_1}) = \int \mathcal{N}(\mathbf{F}, \tau^{-1}\mathbf{I}_{\mathrm{n}})\, \mathcal{N}\big(0, \mathbf{K}(\mathbf{X}, \mathbf{X})\big) \mathrm{p}(\mathbf{W_1}) \mathrm{p}(\mathbf{b}) \mathrm{d}\mathbf{W_1} \mathrm{d}\mathbf{b} \tag{22}$$

Here, we define $\mathbf{W_1}$ to be a matrix of the weights between the input layer and the second to last player of the hidden layer of the neural network and $\mathbf{b}$ to be a vector representing the biases. By defining our covariance matrix $\mathbf{K}$ with (Eq. 23), which includes a non-linear ReLU function σ, and integrating using MC sampling (Eq. 24), we can expand this posterior distribution using the results from *Pattern Recognition and Machine Learning (Bishop, 2006)* [15] to form (Eq. 25).

$$\mathbf{K}(\mathbf{x}, \mathbf{y}) = \int \mathrm{p}(\mathbf{w}) \mathrm{p}(\mathbf{b}) \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x} + \mathrm{b}) \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{y} + \mathrm{b}) \mathrm{d}\mathbf{w}\, \mathrm{db} \tag{23}$$

$$\widehat{\mathbf{K}}(\mathbf{x}, \mathbf{y}) = \frac{1}{\mathrm{K}} \sum_{\mathrm{k}=1}^{\mathrm{K}} \sigma\big(\mathbf{w}_{\mathrm{k}}^{\mathrm{T}}\mathbf{x} + \mathrm{b}\big) \sigma\big(\mathbf{w}_{\mathrm{k}}^{\mathrm{T}}\mathbf{y} + \mathrm{b}\big) \tag{24}$$

$$p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{W}) = \int \mathcal{N}(\boldsymbol{Y}|\boldsymbol{\Phi}\boldsymbol{W_2}, \tau^{-1}\boldsymbol{I}_n) p(\boldsymbol{W_1}) p(\boldsymbol{W_2}) p(\boldsymbol{b}) d\boldsymbol{W_1} d\boldsymbol{W_2} d\boldsymbol{b} \tag{25}$$

Additionally, we define a new matrix $\boldsymbol{\Phi}$, (Eq. 26) which is called the 'design matrix' and is constructed from the activations of the network, mapping the input data $\mathbf{X}$ through the neural network's structure (up to the final hidden layer) to represent the transformed feature space. $\mathbf{W_2}$ is also present in this equation, which represents the weight layers connecting the final hidden layer to the output layer, which is equivalent to the function matrix $\mathbf{F}$ when multiplied with the design matrix, $\boldsymbol{\Phi}$.

$$\boldsymbol{\Phi} = \left[\sqrt{\frac{1}{\mathrm{K}}} \sigma\big(\mathbf{W}_1^{\mathrm{T}}\mathbf{x}_{\mathrm{n}} + \mathbf{b}\big) \right]_{\mathrm{n}=1}^{\mathrm{N}} \tag{26}$$

The next stage in MC dropout, after calculating the true posterior involves proposing posterior functions that maximize the ELBO function. The proposal distributions for $\mathbf{W_1}$ and $\mathbf{W_2}$ are described by (Eq. 27) and (Eq. 28), representing Gaussian distributions with a probability p attached. This probability determines whether the centre of the Gaussian is set to a new variable $\mathbf{m_q}$ for $\mathbf{W_1}$ and $\mathbf{m_k}$ for $\mathbf{W_2}$, or to zero, analogous to dropout with Gaussian distributions. The proposed distribution for the biases is simply a gaussian centred at a vector $\mathbf{m}$ (Eq. 29). The vectors $\mathbf{m_q}$, $\mathbf{m_k}$ and $\mathbf{m}$ are variable parameters adjusted during the ELBO optimization, corresponding to the centres of the Gaussian-dropout distributions, where $\boldsymbol{\alpha}^2\mathbf{I_k}$ is the covariance matrix.

$$q(\mathbf{W_1}) = \prod_{\mathrm{q}=1}^{\mathrm{Q}} \left[p_1 \mathcal{N}(\mathbf{m_q}, \boldsymbol{\alpha}^2\mathbf{I_k}) + (1 - p_1) \mathcal{N}(\mathbf{0}, \boldsymbol{\alpha}^2\mathbf{I_K}) \right] \tag{27}$$

$$q(\boldsymbol{W}_2) = \prod_{k=1}^{K} \left[p_2 \mathcal{N}(\boldsymbol{m}_k, \boldsymbol{\alpha}^2 \boldsymbol{I}_D) + \big(1 - p_2\big) \mathcal{N}(\mathbf{0}, \boldsymbol{\alpha}^2 \boldsymbol{I}_D) \right] \tag{28}$$

$$\mathrm{q}(\mathbf{b}) = \mathcal{N}(\mathbf{m}, \boldsymbol{\alpha}^2\mathbf{I_K}) \tag{29}$$

Now that we have defined the posterior function in its entirety, as well as the proposed distributions for the weights and biases, we can define the entire ELBO maximisation problem (Eq. 30),

$$\mathcal{L} = \int \mathrm{q}(\mathbf{W_1}, \mathbf{W_2}, \mathbf{b}) \log[\mathcal{N}(\mathbf{Y}|\boldsymbol{\Phi}\mathbf{W_2}, \tau^{-1}\mathbf{I}_{\mathrm{n}})] \mathrm{d}\mathbf{W_1} \mathrm{d}\mathbf{W_2} \mathrm{d}\mathbf{b} - \mathrm{KL}\big(\mathrm{q}(\mathbf{W_1}, \mathbf{W_2}, \mathbf{b}) \big\| \mathrm{q}(\mathbf{W_1}, \mathbf{W_2}, \mathbf{b})\big) \tag{30}$$

(Eq. 30) can be simplified to a much simpler form providing (Eq. 31) as the final maximisation problem.

$$\mathcal{L} \propto -\frac{\tau}{2} \sum_{\mathrm{n}=1}^{\mathrm{N}} \|\mathbf{y}_{\mathrm{n}} - \hat{\mathbf{y}}_{\mathrm{n}}\|^2 - \frac{\mathrm{p}_1}{2} \|\mathbf{M}_1\|^2 - \frac{\mathrm{p}_2}{2} \|\mathbf{M}_2\|^2 - \frac{\mathrm{p}_2}{2} \|\mathbf{m}\|^2 \tag{31}$$

This can be maximised for appropriate values of $\boldsymbol{M}_1$, $\boldsymbol{M}_2$ and $\boldsymbol{m}$, allowing the use of the proposed

gaussian-dropout distribution functions as the posterior of the BNN. This corresponds to choosing suitable means for our gaussian-dropout distribution functions for the weights of the system, and as these are simply numerical matrices, this maximising problem can be solved using standard maximisation techniques, such as multi-dimensional gradient descent. Therefore, to simulate a BNN with a standard NN and MC dropout, we run multiple forward passes, applying the dropout mask each time (Eq. 32).

$$w_i' = \frac{z_i \cdot w_i}{p} \qquad z_i = Bernoulli(p) \quad (32)$$

The Bernoulli nature of dropout on individual weights for large numbers of forward passes makes each pass a stochastic sample from the posterior, approximating the Gaussian-dropout distributions $q(\mathbf{W_1})$ and $q(\boldsymbol{W_2})$ without modifying the network's architecture. These repeated passes produce outputs following a Gaussian distribution, consistent with the GP approximation. We take the mean of this distribution as our prediction and interpret its variance as the epistemic uncertainty, allowing us to 'simulate' a BNN from a NN without changing the architecture of the model.

3.1.4. Limitations of MC dropout

Monte Carlo dropout offers a practical way to measure uncertainty with lower computational cost than a full BNN, but it has drawbacks. The primary disadvantage being the fact that the variance of the output is governed by the dropout rate chosen by the user, as higher dropout rate leads to more variance in the final distributions and vice versa. Therefore, uncertainty estimates derived from MC dropout should be interpreted relative to the chosen dropout rate, while these estimates provide useful insights for comparative analysis (e.g., between data points or models), they are not absolute measures of confidence and are a subjective measurement of the uncertainty. To combat this, we have used the industry standard dropout rate of 0.1 [16] to compare the model used with others, yet this connection between the uncertainty measured and the dropout rate must be kept in mind. Another weakness is the assumption that the BNN is a GP, this arises from the central limit theorem, which is applicable to infinitely wide neural networks, but real neural networks may not be wide enough, causing this approximation to break down.

3.2. Methodology and Results

3.2.1. Implementation of MC Dropout

To integrate this method of uncertainty evaluation into the transformer model, the suitable locations to add the dropout had to be found. To approximate the posterior distribution of a BNN effectively, dropout masks were applied wherever possible within the model: immediately after the embedding vector output, throughout the transformer layers (affecting parameters in the attention heads and feed-forward layers), and in the final classification layer. To balance both computational time as well as how well the dropout mask approximates sampling from the dropout-gaussian proposal distribution, the dropout mask was applied for 50 forward passes, but given unlimited computational power this would ideally be set to a higher value to better approximate sampling from the proposed distribution.

3.2.2. Results and Analysis

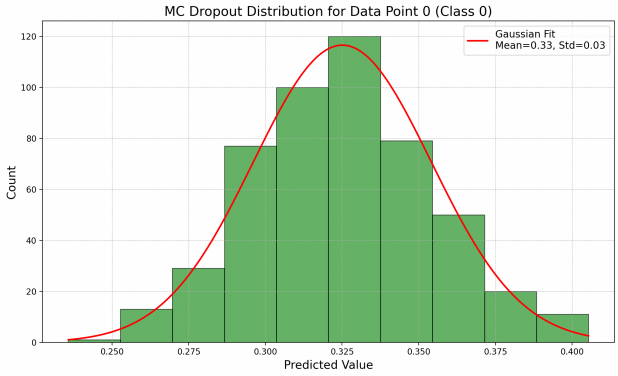

Fig 3.1 – A histogram showing the distribution of predicted values for a single data point using MC dropout, with a Gaussian fit overlay

By running the MC dropout process on a single data point, applying a continuous sentiment analysis, we can view the effects of multiple MC dropout passes on the output of the function. As we see in [Fig 3.1], the histogram of the 500 output results follows a gaussian distribution, which supports the approximation that the simulated BNN follows a gaussian process and shows that the transformer model used within the project is sufficiently large to support the central limit theorem. This plot also demonstrates how multiple passes with MC dropout approximate the Gaussian-dropout proposal distribution for the posterior—evidenced by the resulting Gaussian shape. As mentioned earlier, the choice of dropout rate and the number of MC passes

significantly influence both the variance (reflecting model uncertainty) and the computational cost. Balancing these factors is key to achieving efficient yet precise results. As discussed, this connection emphasizes that uncertainty estimates in MC dropout are not absolute but relative to the user-defined parameters, requiring careful interpretation.

To view the effects of MC uncertainty when dealing with classification tasks, the algorithm was fine tuned to classify a text to be labelled as 'conservative' or 'liberal', with a probability associated with each of the two classifications attached to the text. The model was applied to a large dataset, and the larger probability that a text lay in one bin was taken as the 'confidence' of the model on its prediction. This was then plotted against the MC dropout uncertainty associated with each of the data points, creating [Fig 3.2].

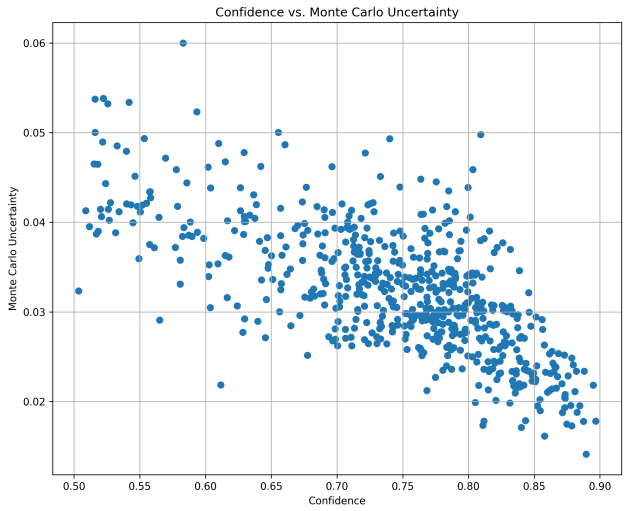

Fig 3.2 – A scatter plot showing the relationship between model confidence and MC dropout uncertainty for classification predictions

The graph demonstrates a negative correlation: as the model's confidence in classifying a data point increases, the associated MC dropout uncertainty decreases. This relationship arises because high-confidence predictions reflect a more stable neural network, where the neurons have clearly identified features strongly associated with a specific bin; consequently, the dropout of individual neurons or parameters has less impact on the final prediction. Conversely, lower confidence predictions indicate less stability, where the network is less certain about the features associated with a bin, making the output more sensitive to parameter dropout, and resulting in higher uncertainty. This graph highlights the strengths of MC dropout in providing relative uncertainty estimates, allowing for meaningful comparisons between data points. The noise in the graph arises due to the stochastic nature of the MC dropout process, introducing variability to the trend of the graph. As the number of data points and forward passes increases, this noise will reduce, making the correlation clearer.

4. Bootstrap Resampling as a Measurement of Unsupervised ML Models

4.1. Theoretical Background

4.1.1. Unsupervised Machine Learning Models

Unsupervised machine learning models work through iterative processes to uncover structures and patterns within data, without the use of training data or backpropagation to adjust parameters. These algorithms analyse raw, unlabelled data and are commonly applied to tasks such as clustering and anomaly detection. The processes often rely on dimensionality reduction to manage high-dimensional data, making the process integral to their performance. The primary unsupervised machine learning techniques in this analysis, K-means clustering and BERT topic modelling, rely on word-to-vector transformations—TF-IDF for K-means and BERT embeddings for BERT topic modelling—which produce high-dimensional vectors as data, requiring dimensionality reduction to facilitate clustering and reduce computational demands. K-means clustering iteratively assigns the closest data points in the reduced TF-IDF space to centroids, defined as the mean position of all points within a cluster, which gets updated with each iteration. In contrast, BERT topic modelling employs agglomerative clustering, constructing a hierarchy of clusters by calculating pairwise distances between clusters and data points. In order to evaluate uncertainties within the outputs of the unsupervised learning techniques used in the NLP, bootstrap resampling was employed to test the stability of data points and clusters.

4.1.2. Bootstrap Resampling & Stability

Bootstrap resampling generates datasets by repeatedly sampling the original corpus, allowing data points to be repeated. This results

in perturbed datasets with repeated and omitted points. To apply this statistical method to evaluate the performance of the clustering algorithms, a range of bootstrap samples were created from a corpus of data, the clustering algorithms were applied to these perturbed datasets, and their outputs were recorded [17]. This enabled the calculation of the stability of an individual datapoint (Eq. 33) which shows the frequency of the point being assigned to the same cluster label.

$$s(x_i) = \frac{\#(\text{same cluster label for } x_i)}{\#(\text{appearences of } x_i \text{in the bootstrap})} \quad (33)$$

This stability value can then be subtracted from 1 to give the uncertainty of the singular data point, which can be averaged over all values within a cluster to obtain a cluster uncertainty. High cluster uncertainty reflects algorithmic instability or sensitivity to data changes. In contrast, low cluster uncertainty suggests the algorithm consistently assigns similar points to the same cluster, even with slight perturbations.

It is important to note that the term "uncertainty" used here is not equivalent to classical aleatoric or epistemic uncertainty as understood in supervised machine learning or probabilistic modelling. Instead, it serves as a mathematical tool to quantify the consistency of cluster assignments under perturbations. This distinction arises because traditional notions of uncertainty rely on labelled data or probabilistic frameworks, both of which are absent in most unsupervised machine learning algorithms. The absence of labelled data and explicit training-validation splits means we cannot directly evaluate uncertainties in the traditional sense. The "uncertainty" defined here thus reflects variability in outcomes due to the algorithm's sensitivity to data perturbations, rather than a true probabilistic measure of confidence. This lack of true uncertainty measurements is frustrating yet perturbing the datasets and viewing the changes in output, using metrics such as stability to review the performance of the model, is as advanced as error analysis can get for unsupervised machine learning with unlabelled datasets, due to the nature of the algorithms.

4.2. Methodology and Results

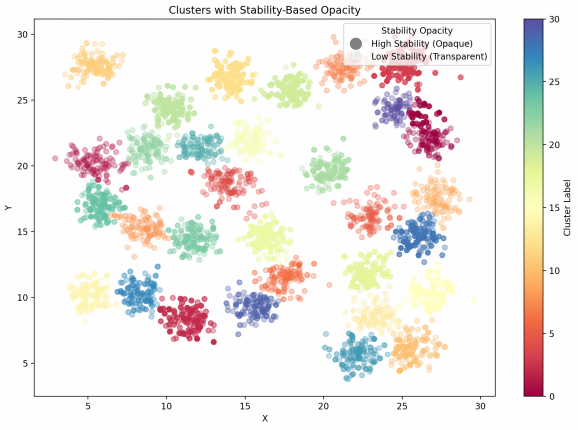

Fig 4.1 – A scatter plot showing 31 clusters from a benchmark dataset, with cluster stability visualized through opacity. Cluster labels are represented by colour.

To show the effects of bootstrap resampling, the BERT topic clustering algorithm used within the project was applied to a test dataset specifically designed to test clustering algorithms from *2002 Index IEEE Transactions on Pattern Analysis and Machine Intelligence Vol. 24* [18]. This dataset was chosen because the high-dimensional NLP data used in the project does not exhibit clear clustering correlations when projected into 2D space using dimensionality reduction, making this benchmark dataset more suitable for demonstrating bootstrap resampling. As shown in [FIG], the clustering algorithm correctly divided the dataset into 31 clusters, denoted on the graph by colour, which is the ideal outcome designed by the dataset. The stability of the data points, calculated from 100 bootstrap samples, was overlaid onto the graph and visualized through the opacity of the datapoints, with darker opacity indicating higher stability. As we can see in [FIG], the stability analysis allowed the uncovering of many unstable clusters, usually located near other clusters with high stability. The figure also shows that unstable data points are typically located on the boundaries in between clusters, as they are often assigned to other clusters when the dataset is perturbed. High-stability points appear near cluster centres or isolated from other clusters, consistently assigned under perturbations. This figure demonstrates how bootstrap resampling quantifies cluster stability, identifies algorithmic flaws, and highlights weaker clusters that may benefit from merging, which shows that even though the measurement of stability is not an uncertainty on the model in a traditional sense, it still provides valuable insights into the behaviour of unsupervised learning models.

The bootstrap-based stability analysis highlights how clustering assignments can fluctuate in response to data perturbations, drawing attention to regions or boundaries where individual data points exhibit high instability. Unsupervised methods lack labelled data and thus cannot rely on conventional uncertainty measures, techniques like bootstrap resampling provide an alternative means of measuring error - assigning a "cluster uncertainty" to each datapoint. Similarly, dimensionality reduction introduces its own uncertainty into unsupervised models: by examining the specific techniques employed, we can apply statistical methods that estimate error or instability for individual datapoints.

5. UMAP Error Analysis

5.1. Theory

5.1.1. Dimensionality Reduction: Key Concepts

Machine learning models such as large language models often possess a high number of dimensions - each representing distinct features or attributes. High-dimensional data spaces are difficult to process due to the curse of dimensionality [19], a phenomenon where the volume of the space rapidly increases with dimensionality making the data increasingly sparse and dramatically increasing computing times. Consequently, tasks such as clustering, classification and visualization become increasingly strenuous, requiring us to use techniques to manage and interpret high-dimensional data effectively.

Dimensionality reduction mitigates these challenges by transforming data from a high-dimensional space to a lower-dimensional one. Its primary objective is to preserve the structure and relationships within the data, while eliminating redundancies and noise. Reducing dimensions increases computational efficiency, facilitates visualisation, and helps uncover clusters or anomalies that may be obscured in higher dimensions.

Dimensionality reduction techniques can be categorized into linear and non-linear methods. Linear methods, such as Principal Component Analysis (PCA) and Multidimensional Scaling (MDS) [20], assume that the underlying structure of the data can be captured through combination of linear features. PCA, for instance, identifies the principal components that maximize the variance within the data, effectively projecting the data onto a new set of orthogonal axes. This approach is computationally efficient and works well when the data exhibits linear relationships. However, its linear nature limits its ability to capture more complex, non-linear structures inherent in many real-world datasets.

Non-linear methods like t-Distributed Stochastic Neighbour Embedding (t-SNE) and Uniform Manifold Approximation and Projection (UMAP) [21] are designed to preserve more complex relationships by considering how points group together at a chosen scale (for the 'k' nearest neighbouring points). For example, t-SNE, focuses on maintaining the similarity of local neighbourhoods making it highly effective for visualizing clusters in two or three dimensions. Despite its strengths, t-SNE can be computationally intensive and may require extensive tuning of hyperparameters to avoid problems such as false clusters.

UMAP builds upon the foundations of t-SNE, but uses 'fuzzy topology', which essentially uses continuous values for k-nearest neighbours instead of t-SNE's discrete values (McInnes, Healy, Melville 2018) [22]. UMAP constructs a high-dimensional graph (a series of points connected by 'edges') capturing the data's shape (topology), then optimizing a low-dimensional graph to approximate its higher-dimensional counterpart. Through obtaining a lower-dimensional representation, calculations for training models are greatly reduced increasing computational efficiency and the magnitude of the datasets we can process.

5.1.2. UMAP: Background

Uniform Manifold Approximation and Projection (UMAP) is a powerful non-linear dimensionality reduction technique, which balances preservation of both local and some global data structures while maintaining computational efficiency. Developed by McInnes, Healy, and Melville in 2018, UMAP uses a variety of techniques from Riemannian

manifolds[1] and topological data analysis to achieve high-quality embeddings for visualization and optimization of machine learning tasks.

At its core, UMAP is grounded in topological data analysis, it operates under the assumption that the high-dimensional data lies on a low-dimensional manifold embedded within the higher-dimensional space. To formalize this, UMAP uses fuzzy simplicial sets - which assign a probabilistic basis for capturing the relationship between points [23]. A simplicial set is a collection of simplices (points, lines, triangles, etc.) that describe how data points are interconnected. By making this set "fuzzy," UMAP assigns membership probabilities to these simplices, reflecting the strength or confidence of the connections between points. Unlike discrete neighbour graphs, fuzzy simplicial sets allow continuous membership values, permitting partial assignment of a point to multiple local neighbourhoods. The membership functions determine these relationships, as they quantify the likelihood that a given pair of points belongs to the same local neighbourhood on the manifold, with two functions representing the higher and lower dimensions.

The transformation process in UMAP involves constructing a high-dimensional fuzzy simplicial set that captures the topological structure of the data. This high-dimensional representation is then optimized to form a corresponding low-dimensional 'fuzzy' simplicial set (simplices connected by weighted edges). The alignment of these fuzzy simplicial sets is achieved through minimization of the fuzzy set cross-entropy, which quantifies the divergence between the high-dimensional and low-dimensional membership functions. For the manifold to have a topologically meaningful shape, it is assumed that the data points on the lower-dimensional manifold are uniformly distributed. In practice, the distances between data points vary, resulting in the physical space between them being stretched or compressed to achieve a reasonable distribution on the manifold.

5.1.3. UMAP: Functions

UMAP utilizes the membership functions to assign weights to the connections between datapoints when creating the higher and lower-dimensional graph, these functions are respectively represented by equation 34 and 35.

$$\mu_{ij} = e^{-\frac{d(X_i,X_j)-\rho_i}{\sigma_i}} \quad (34)$$

$$v_{ij} = \left(1 + \alpha\left(d\left(Y_i, Y_j\right)\right)^{2\beta}\right)^{-1} \quad (35)$$

Where $d(Z_i, Z_j)$ represents the distance between two points in the respective dimension, ρ_i is the distance to the nearest neighbour of point i, σ_i is a scaling parameter, and α, β are hyperparameters which are by default set at 1.929 and 0.7915 [22] respectively from empirical fitting.

Membership functions are the cornerstone of UMAP's ability to project high-dimensional data onto a lower-dimensional manifold, simplifying the complex geometry of data manifolds. This allows us to solely focus on its applications for rigorous analysis. UMAP optimizes the embedding by minimizing the global cross-entropy between the membership functions (Eq. 36).

$$C(\mu_{ij}, v_{ij}) = \sum_i \sum_j \left[\mu_{ij} \log_2\left(\frac{\mu_{ij}}{v_{ij}}\right) + (1-\mu_{ij}) \log_2\left(\frac{1-\mu}{1-v}\right.\right.$$

Although any log base can be used, base 2 is chosen for interpretability in binary bits. For a more detailed explanation of the theoretical basis of UMAP, please see the original paper [22].

5.1.4. UMAP: Error Analysis

Cross-entropy can be used as an error metric for each individual datapoint and globally, indicating how well the low-dimensional embedding has preserved the local and global structure. While cross-entropy offers an error measure of the structure, mutual information offers an alternative information-based approach, which can be used to calculate the amount of information lost for each datapoint. Mutual information is defined as [24]:

$$MI(X;Y) = H(X) + H(Y) - H(X,Y) \quad (37)$$

Where $H(Z) = -\sum p(z) \log_2 p(z)$ is the Shannon entropy[2], and X, Y represent the data point in the higher and lower dimensions

[1] A Riemannian manifold is a curved space where concepts Euclidean geometry can be generalized from flat surfaces (Do Carmo, Francis 1992)

[2] Shannon entropy quantifies the uncertainty in a system by measuring how evenly possibilities of states are distributed (Shannon 1948)

respectively. In simple terms, the mutual information tells us how the high-dimensional neighbourhood relationships overlap with the low-dimensional ones.

Membership functions determine the weight of the edge between two points, or the strength of their 'connection' in their local neighbourhood. This means we can normalize the membership functions to get them in the form of a probability distribution based on the distance between the points in their respective dimensions (Eq. 38).

$$p_\mu(ij) = \frac{\mu_{ij}}{\sum_k \mu_{ik}}, \quad p_v(ij) = \frac{v_{ij}}{\sum_k v_{ik}} \tag{38}$$

The joint probability used for $H(X,Y)$ can be calculated by taking the min of the probabilities $p_\mu(ij),\ p_v(ij)$ for each individual pair of points and summing them together and normalizing. The respective entropies quantify the uncertainty in high and low-dimensional neighbourhoods. High overlapping entropy, $H(X,Y)$, indicates a high level of shared information signifying effective information retention. We can then normalize the mutual information as an effective measure of information retention (Eq. 39) [24].

$$NMI = \frac{MI(X;Y)}{\sqrt{H(X)H(Y)}} \tag{39}$$

We now get a value between 0 and 1, with 0 signifying no shared information and 1 indicating perfect information retention. Though information loss measures error in the projected points it does not directly translate as uncertainty.

To determine the uncertainty associated with each datapoint on the projected manifold, we can use Fisher information. It measures how sensitive projected points are to perturbations, or changes in the local distance metrics associated between each point. As before, we can use the normalized membership functions as the probabilities of points being connected. By analysing variations in membership probabilities, we can assess the robustness of the UMAP embedding and identify regions of the data space that are particularly sensitive to alterations in the low-dimensional graph, indicating uncertainty in the accuracy of the projection. This is especially prevalent when working with complex datasets where subtle structural nuances can have significant implications for downstream analyses.

Fisher information quantifies the sensitivity of the projected points and is defined as [25]:

$$I(Y_i) = \mathbb{E}\left[\left(\frac{\partial \log L(X_i;Y_i)}{\partial Y_i}\right)^2\right] \tag{40}$$

Where $\mathbb{E}$ is the expected value and $L(X_i;Y_i)$ is the likelihood function. We can express the likelihood function by considering each connection of datapoints as a Bernoulli trial. Bernoulli trials are based on trials with binary outcomes, success or failure with probabilities p or $1-p$ respectively [26]. By using the normalized membership functions, we can consider each pair of points (i,j) and assume that for each pair the likelihood that an edge exists is independent. We can then say that the overall likelihood of observing a specific set of connections in the low-dimensional embedding is expressed as the product of the individual connections. Mathematically, this works out as the likelihood function being proportional to e^{-C}, where C is the cross-entropy (See appendix for proof). This means we can simplify the expression for the Fisher information giving us equation 41.

$$I(Y_i) = \sum_j \left(\frac{\mu_{ij} - v_{ij}}{v_{ij}(1-v_{ij})}\frac{\partial v_{ij}}{\partial Y_i}\right)^2 \tag{41}$$

Where $\partial v_{ij}/\partial Y_i$ will be in terms of the distance function between the projected points i and j (see appendix for full derivation).

5.2. Methodology

5.2.1. Overview of the Dataset and Embedding Space

For this analysis, the HuggingFace [5] 'political_idealogies' dataset is used, providing an assorted collection of text documents representing varying political ideologies. These documents span a wide range of topics - economics, social issues, climate change, and more - allowing us to explore how well UMAP preserves local and global structures in a semantically diverse dataset. Although any high-dimensional dataset could be used for our analysis, a text-based dataset with readily available embeddings offers a convenient platform to evaluate UMAP's manifold learning techniques.

Each document (datapoint) is transformed into a 768-dimensional vector via the Sentence-Transformer model "all-mpnet-base-

v2" [27]. This embedding space is pre-trained and fine-tuned to capture semantic similarities between words and short texts more effectively than simpler word-frequency methods. While the choice is somewhat arbitrary, as one could choose any other language model, the key requirement for us is to have a strong performing representative high-dimensional space on which to apply UMAP. To remove noise for subsequent analyses, word filtering and tokenization[3] were performed to remove extremely short or irregular documents, ensuring higher-quality embeddings. For completeness and clarity, the specifics of the modules used, additional filtering steps, and text processing routines can be found in the appendix.

5.2.2. Bayesian Optimization for Hyperparameter Tuning

Although our main goal is to study the reduced embedding's properties, it is beneficial to begin with an optimized set of hyperparameters for UMAP to avoid trivial or suboptimal results. To achieve this, Bayesian optimization was run for 10 trials before UMAP was fitted to the data. Although 10 trials may not exhaustively explore the parameter space, more trials provided limited returns given the time taken.

Bayesian optimization differs from grid or random searches by iteratively refining its model after each run based on the effect of varying the hyperparameters has on the coherence score [28]. The coherence score is a fast, computationally efficient way to determine the quality and interpretability of clusters produced by UMAP. It is calculated by counting the frequency of a words appearance in one cluster compared to another. In other words, if UMAP has clustered datapoints into topics correctly, we would expect similar words to appear more frequently in the same topics rather than be distributed across multiple clusters. This simple calculation allows the Bayesian Optimizer to quickly find a good configuration for UMAP's hyperparameters.

5.2.3. Key UMAP Hyperparameters

1. n_neighbors

Defines the scale of connectivity across the dataset. Smaller values emphasize preserving local structures, while larger values capture broader global structures. This directly influences the value of σ_i, and has the largest effect on performance (see fig 5.0)

2. n_components

Sets the desired dimension size of the reduced representation. For data visualisations we can set this to 2 or 3.

3. min_dist

Determines how tightly or loosely points are clustered in the reduced space, by setting a lower limit on distance between points in the low-dimensional embedding. Lower values produce denser clusters, whereas higher values spread points more evenly. This in turn determines the size of membership function hyperparameters α, β.

4. umap_metric

Defines the distance measure used to calculate distances between points in the dataset, $d(Z_i, Z_j)$. For this analysis, Euclidean metric was always found to be optimal, however as mentioned previously, space is often warped over the manifold.

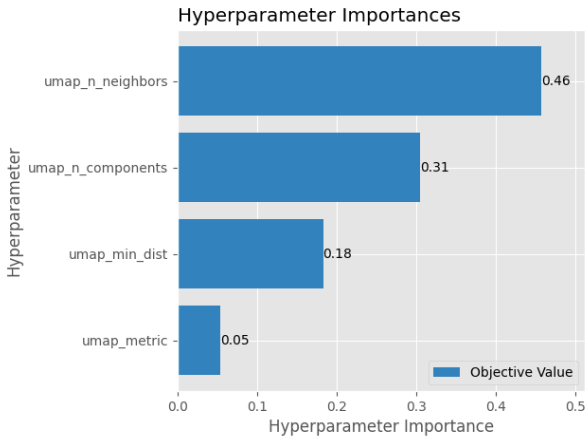

Fig 5.0 - A graph showing the relative importance of UMAP's hyperparameters based on the effect of their variation on the coherence score

5.3. Dataset Observations

5.3.1. Initial Compression and Errors

[3] Tokenization is the process of splitting text into smaller units

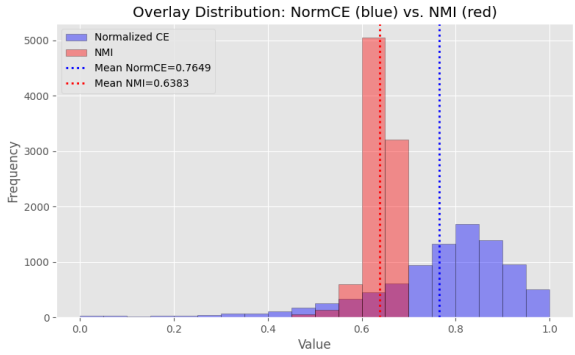

Fig 5.1 – A histogram showing the frequency of NMI (red) and CE (blue) values

Upon finding a suitable set of UMAP configurations from the Bayesian optimization, 10,000 data points were sampled from the dataset. In our first analysis, we compare the cross-entropy with the information retention across the dataset to see how UMAP balances the preservation of structure against retaining overall information. For a rough comparison, the cross-entropy values were normalized and subtracted from 1, such that for both information and retention, 1 signifies perfect preservation of structure or information, and 0 represents no preservation.

As shown in fig 5.1, the results indicate that mutual information loss is more pronounced than the cross-entropy divergence, as UMAP prioritizes structure rather than information this is consistent with what we would expect. The concentrated range of information loss suggests that it may be lost more uniformly across the dataset.

A defining characteristic of dimensionality reduction is that it compresses physical space. As UMAP is non-linear, datapoints are reduced unevenly; this leads to some datapoints moving closer together than others as seen in fig 5.2.

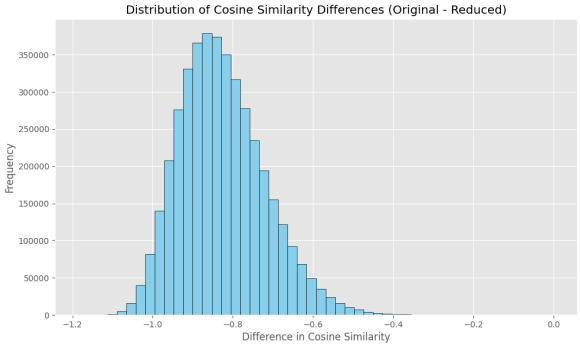

Fig 5.2 – A histogram showing the frequency of differences in cosine similarity between each datapoint pair

As expected, all the recorded values in fig 1.2 are negative indicating the dataset has been brought closer together, a median shift of ~ −0.85 indicates a consistent compression

Tasks relying on well-separated clusters (or large inter-point distances) could see degraded performance because dissimilar documents become more similar in the reduced space. However, as all points move similarly closer to each other, this by itself does not tell us if there has been any information loss. Hence, additional metrics (e.g., cross-entropy, mutual information) are still needed to evaluate the true "error" introduced by UMAP's compression.

5.3.2. 2D UMAP Projection

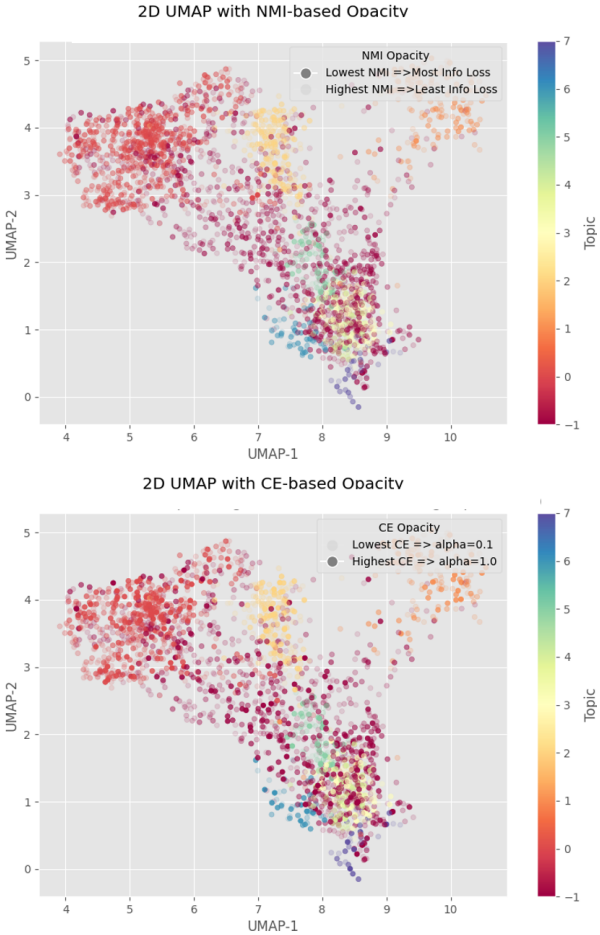

Fig 5.3 – 2D projections of the dataset sorted into topics with opacity representing the cross-entropy and mutual information loss respectively

Fig 5.3 appears to illustrate some subsets of points with larger cross-entropy or greater information loss. However, no simple correlation is observed with either original dimensional distances or reduced distances.

Since UMAP attempts to minimize global cross-entropy (rather than optimize each point's embedding individually), the distribution of local errors may appear semi-random. Small perturbations in the data or changes in the hyperparameters, can alter which regions of the manifold become

compressed or stretched, shifting the location of high-divergence or high-information-loss points.

5.3.3. Dependence of Information Retention on Distance

UMAP's membership functions depend on pairwise distances in the projected space (Eq. 35), so it follows that mutual information should correlate with the average distance a point has to its neighbours in the manifold.

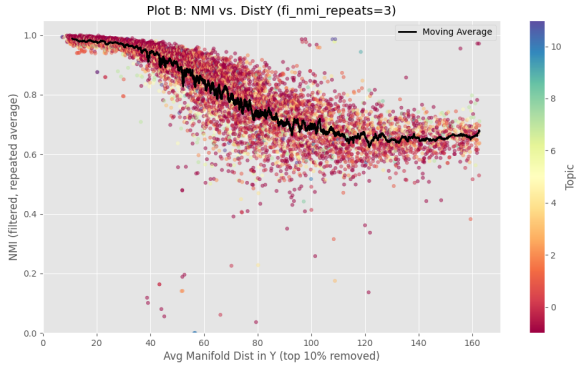

Fig 5.4 – A scatter plot showing the average distance to a points nearest neighbour over the manifold

Fig 5.4 confirms that points with longer distances to their neighbours exhibit higher information loss. However, as mentioned previously however, UMAP assumes a uniform distribution of datapoints on the manifold, leading to the physical space being warped. While distances between points on the manifold provides a meaningful way of discerning information loss, the Euclidean distance between the resultant lower dimensions displays no such correlation. Intuitively, if UMAP pushes a point away from its immediate neighbours, the membership probabilities in the reduced space will diverge from the high-dimensional membership probabilities, inflating the point's mutual information loss.

Predicting exactly where UMAP stretches, or compresses distances is difficult. Slight alterations in data or hyperparameters can dramatically change how cross-entropy is minimized, causing unpredictable shifts in the global geometry.

5.3.4. Fisher Information

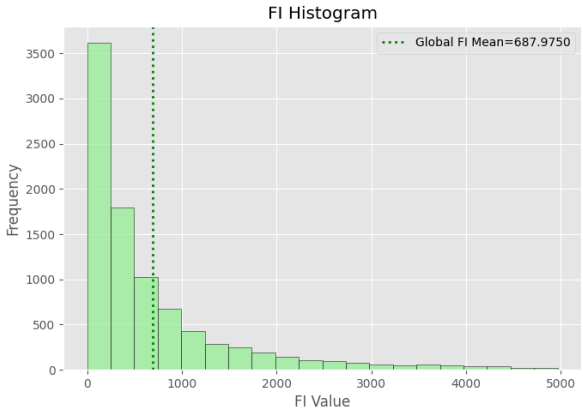

Fig 5.5 – Histogram showing the frequency of Fisher information values

Fig 5.5 highlights that most points have relatively low Fisher information, suggesting that minor adjustments in the manifold do not drastically change their local memberships. A smaller fraction of points have higher Fisher information, indicating they reside in complex regions of the manifold.

In the context of our manifold, Fisher information depends primarily on the distance between points (see appendix for proof), which only varies if a point changes location or the local distance metric changes. Geometrically, we can interpret this as meaning stable points on the manifold have simpler geometries, and sensitive points having complex geometries. For instance, if we have two points connected by an edge, moving a point generates two possible outcomes; either it remains an edge or becomes two separate points. Whereas if we have three points connected by two edges, moving a point can lead to three separate points, two permutations of a point and an edge or a triangle. This suggests that complex geometries have many more possible states, and as mutual information is calculated based on the entropy of these possible states (Eq. 37), we can infer that these complex regions contain more information. As information dense regions are more complicated to preserve, we may hypothesize that these structures may be harder to reduce effectively leading to higher information loss.

5.3.5. Unified View: Cross-Entropy, Mutual Information, and Fisher Information

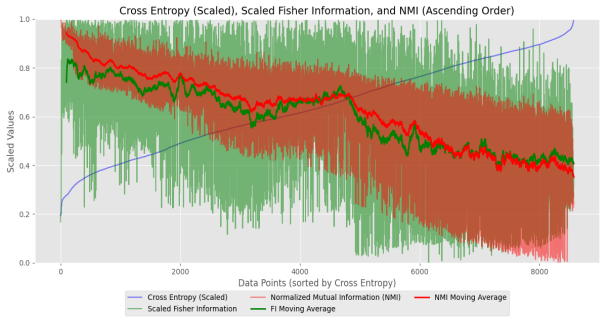

Fig 5.6 – A scatter plot with datapoints ordered by ascending cross-entropy (blue) with the normalized mutual information (red) and normalized Fisher information (red, with the highest top 5% of points removed to reduce noise)

Fig 5.6 confirms the link between structural preservation and informational preservation. Despite the noisy results, we can see by the moving average that information retention steadily decreases as cross-entropy increases. Notably, this also coincides with our Fisher information decreasing very similarly to the information retention, despite its range fluctuating a great deal more. This confirms that structure, information and uncertainty are all linked together. Intriguingly, it was consistently found that on average, the first few datapoints (~10) with the lowest cross-entropies seemed to have much lower Fisher information than the following datapoints. This implies that points with very good structural preservation are in some way more stably projected on the manifold.

The analysis was run 5 times applying UMAP to the same datapoints (each time re-tuning hyperparameters) which resulted in varying distributions each time. However, the same generalized trends were observed each time. The high degree of noise in the results show the technique's sensitivity to small changes in data and parameter settings, as well as underlying inconsistencies and randomness in the optimization processes involved in UMAP.

5.4. Implications and Future Directions

Collectively, these findings demonstrate that dimensionality reduction via UMAP inevitably loses some relational data, distributing that loss unevenly across the dataset. UMAP optimizes global cross-entropy to prioritize local structure, but certain points or regions may pay a higher price in terms of information loss. While such trade-offs favour computational speed and convenience, other manifold learning methods might be better suited to prioritize information retention - albeit at the cost of longer runtime or added complexity.

Additionally, UMAP relies on approximate nearest neighbours (ANN), in which distances between points are estimated rather than calculated directly. While this accelerates UMAP's construction of lower-dimensional manifold, it introduces small distortions and inconsistencies. Future research might investigate more direct methods for minimizing global information loss or leverage advanced graph-theoretic analyses to pinpoint exactly how local distance metrics build the manifold. More experiments are needed to see how high-information regions affect downstream tasks, clarifying whether these "sensitive" zones are also more crucial for classification, topic modelling, or other predictive goals.

While UMAP successfully yields coherent embeddings for large-scale data, many of its internal techniques prioritise computational efficiency at the cost of information retention. Its combination of non-linear compression, global cross-entropy minimization, and internal approximation strategies leads to intricate, sometimes unpredictable patterns in local distortion and information retention. Though cross-entropy, mutual information, and Fisher information provide a valuable insight into the errors introduced by the manifold's structure, predicting exactly how UMAP will distort certain points or regions from the initial higher-dimensional space remains a difficult problem.

6. Conclusion

To conclude, we find in analysis of ADFIM variance its potential as a tool for assessing parameter importance in transformer models when thought of like a physical variance while assumptions like parameter independence limit its exact equivalence to physical variance, in turn restrict its usefulness to calculate overall output uncertainty. We then demonstrated how Monte Carlo (MC) dropout offers an alternative to probabilistic neural networks, enabling the simulation of a BNN to extract statistical error analysis as a measure of system uncertainty. The primary weakness of this process is the uncertainty produced is dependent on the hyperparameters of the

dropout algorithm, particularly the dropout rate, making the estimates relative rather than absolute.

Bootstrap resampling was used to create perturbed datasets, enabling analysis of data point and cluster stability as a measure of uncertainties. This provided insight into the relative strength of clusters and highlighted the model's confidence in clustering individual data points. Moving forward, this may benefit from more advanced methods of perturbing datasets, such as weighted bootstrap resampling to better evaluate the data point stability.

Finally, the work on UMAP dimensionality reduction error analysis, used cross-entropy, mutual information, and Fisher information to quantify how well neighbourhood structure is maintained, the stability of datapoints and how information is lost in the dataset. The uncertainty shows evidence that although UMAP preserves local structure effectively for many points, information loss is distributed unevenly, with certain regions showing disproportionately high error. Additionally, no single observable factor (e.g., original data distances) consistently predicted which points will experience greater distortion, highlighting some inherent randomness in how UMAP compresses the data.

Collectively, these methods provide groundwork for quantifying forms of uncertainty within transforms, offering research pathways to enhance the statistical techniques and interpretability of transformer model results.

Appendix

UMAP Analysis Code Summary

The analysis is carried out using multiple Python scripts and custom modules:

1. main.py
   • Loads the chosen dataset (e.g., 20 Newsgroups) and manages runtime configurations.
   • Performs a Bayesian search for hyperparameter optimization using Optuna, including UMAP settings.
   • Calls helper modules for text preprocessing, embedding generation, topic modelling, and UMAP analyses.

2. bertopic_module.py
   • Implements a TextProcessing class for basic cleaning (e.g. removing short or empty documents), tokenization, lemmatization (reducing words to base components i.e. walking → walk) and stopword filtering.
   • Manages BERTopic components (UMAP + HDBSCAN) for topic extraction from the preprocessed text.
   • Uses **SentenceTransformers** (e.g., "all-mpnet-base-v2") for generating the 768-dimensional embeddings.

3. hyperparameters.py
   • Handles the Bayesian optimization of BERTopic and UMAP hyperparameters based on coherence scores.
   • Iteratively refines parameters such as n_neighbors, n_components, and min_dist.

4. UMAP_analysis.py
   • Provides functions for calculating Fisher information, cross-entropy, and mutual information.
   • Includes utilities for plotting data in 2D/3D, analysing data-size effects, and examining similarity changes before and after reduction.

Word Filtering Process

• Before embedding, texts are cleaned and tokenized.
• Documents with insufficient tokens (e.g., extremely short or empty) are removed.
• Stopword elimination is applied to reduce noise.

Topic Modeling

• **BERTopic** is used to cluster documents and compute topic-level coherence scores.
• The BERTopic pipeline integrates UMAP for dimensionality reduction and HDBSCAN for clustering, linking reduced embeddings to topic formation.

All modules interact to process text, generate embeddings, reduce dimensions, and compute metrics for the UMAP error analysis.

Bernoulli Trials Proof: Likelihood $\propto e^{-C}$

We define $y_{ij} \in \{0,1\}$ as a binary variable indicating if points i and j are connected. Let p_{ij} be the model's probability that $y_{ij} = 1$. Hence $P(y_{ij} = 1) = p_{ij}$, and $P(y_{ij} = 0) = 1 - p_{ij}$. Thus,

$$P(y_{ij}) = p_{ij}^{y_{ij}}(1 - p_{ij})^{1-y_{ij}}$$

Considering all pairs (i, j) in our dataset and assuming independence the likelihood of pairwise connections, L is:

$$L = \prod_{(i,j)} p_{ij}^{y_{ij}}(1 - p_{ij})^{1-y_{ij}}$$

Taking the log-likelihood:

$$\log L = \sum_{(i,j)} \log\left[p_{ij}^{y_{ij}}(1 - p_{ij})^{1-y_{ij}}\right]$$

$$= \sum_{(i,j)} \left[y_{ij} \log p_{ij} + (1 - y_{ij}) \log(1 - p_{ij})\right]$$

As the higher dimensional data remains constant, its membership function μ_{ij} will also be a constant. As we can relate v_{ij} to the probability of connection (through normalization), we get (in Einstein notation):

$$C \propto -\mu_{ij} \log v_{ij} + (\mu_{ij} - 1)\log(1 - \mathrm{v_{ij}})$$

Hence,

$$L \propto e^{-C}$$

Fisher Information Derivation

The formula for Fisher information is based on the expectation value and the likelihood function:

$$I(Y_i) = \mathbb{E}\left[\left(\frac{\partial \log L(X_i; Y_i)}{\partial Y_i}\right)^2\right]$$

As shown above, we can relate the log likelihood function to the cross entropy giving us:

$$\frac{\partial \log L(X_i; Y_i)}{\partial Y_i} = \frac{\partial L_{ij}}{\partial Y_i}\frac{1}{L_{ij}}$$

$$= \frac{1}{L_{ij}}\left[\mu_{ij} v_{ij}^{\mu_{ij}-1}\frac{\partial v_{ij}}{\partial Y_i}\left(1 - v_{ij}\right)^{1-\mu_{ij}} + v_{ij}^{\mu_{ij}}(1 - \mu_{ij})\left(1 - v_{ij}\right)^{-1}\left(-\frac{\partial v_{ij}}{\partial Y_i}\right)\right]$$

$$= \frac{\partial v_{ij}}{\partial Y_i}\left[\frac{\mu_{ij}}{v_{ij}} - \frac{1 - \mu_{ij}}{1 - v_{ij}}\right]$$

$$= \frac{\mu_{ij} - v_{ij}}{v_{ij}(1 - v_{ij})}\frac{\partial v_{ij}}{\partial Y_i}$$

As the expected value is the sum of the weighted outcomes, we sum over other points j, to get our final result:

$$I(Y_i) = \sum_j \left(\frac{\mu_{ij} - v_{ij}}{v_{ij}\left(1 - v_{ij}\right)}\frac{\partial v_{ij}}{\partial Y_i}\right)^2$$

We can simplify this further using the expression for v_{ij}:

$$v_{ij} = (1 + az^{2b})^{-1}$$

$$1 - v_{ij} = \frac{az^{2b}}{1 + az^{2b}}$$

$$\frac{1}{v_{ij}\left(1 - v_{ij}\right)} = \frac{(1 + az^{2b})^2}{az^{2b}}$$

Hence,

$$\frac{\mu_{ij} - v_{ij}}{v_{ij}\left(1 - v_{ij}\right)}\frac{\partial v_{ij}}{\partial Y_i} = \left(\mu_{ij} - v_{ij}\right)\frac{(1 + az^{2b})^2}{az^{2b}}\left(-\frac{a2bz^{2b-1}}{(1 + az^{2b})^2}\frac{\partial z}{\partial Y_i}\right)$$

Which all simplifies to:

$$= -\frac{2b\left(\mu_{ij} - v_{ij}\right)}{z}\frac{\partial z}{\partial Y_i}$$

Where $a = \alpha, b = \beta, z = d(Y_i, Y_j)$ Finally putting it all together we have:

$$I(Y_i) = \sum_j \left(\frac{2\beta\left(\mu_{ij} - v_{ij}\right)}{d(Y_i, Y_j)}\frac{\partial d(Y_i, Y_j)}{\partial Y_i}\right)^2$$

Hence Fisher information is solely in terms of the projected distance on the manifold, $d(Y_i, Y_j)$.

BIBLIOGRAPHY

[0] Chatgpt was used to assist in making passages of text more grammatically correct.

[1]Olsson C, Steinhardt J, Amodei D, et al. A Framework for Transformer Circuits. Transformer Circuits. 2021. Available from: https://transformer-circuits.pub/2021/framework/index.html

[2]Xu Han, Zhengyan Zhang, Ning Ding et al. Pre-trained models: Past, present and future. AI Open. Volume 2. 2021. Pages 225-250. ISSN 2666-6510. https://doi.org/10.1016/j.aiopen.2021.08.002.. (https://www.sciencedirect.com/science/article/pii/S2666651021000231)

[3]Sun C, Qiu X, Xu Y, Huang X. How to Fine-Tune BERT for Text Classification. arXiv. 2019; Available from: https://doi.org/10.48550/arXiv.1905.05583

[4]Lan Z, Chen M, Goodman S, Gimpel K, Sharma P, Soricut R. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. arXiv. 2019; Available from: https://doi.org/10.48550/arXiv.1909.11942

[5]Jyoti. Political Ideologies Dataset [Internet]. Hugging Face; [cited 2025 Jan 13]. Available from: https://huggingface.co/datasets/jyoti/political_ideologies

[6] Hwang D. FAdam: Adam is a natural gradient optimizer using diagonal empirical Fisher information [Internet]. arXiv; 2024 [cited 2025 Jan 13]. Available from: https://arxiv.org/abs/2405.12807

[7] Rigollet P. Lecture notes on maximum likelihood estimation and Fisher information [Internet]. [cited 2025 Jan 13]. Available from: https://math.mit.edu/~rigollet/

[8]Stanford University. Lecture notes on Fisher information [Internet]. [cited 2025 Jan 13]. Available from: https://statweb.stanford.edu/

[9]Brunner G, Liu Y, Pascual D, Richter O, Ciaramita M, Wattenhofer R. On Identifiability in Transformers. arXiv preprint arXiv:1908.04211. 2019 Aug 12

[10]Kunstner F, Balles L, Hennig P. Limitations of the Empirical Fisher Approximation for Natural Gradient Descent [Internet]. arXiv; 2020 [cited 2025 Jan 13]. Available from: https://arxiv.org/abs/1905.12558

[11]Amari, Shun ichi / Karakida, Ryo / Oizumi, Masafumi(2019): Fisher information and natural gradient learning in random deep networksIn: The 22nd International Conference on Artificial Intelligence and Statistics694–702

[12]Wang W, Tu Z. Rethinking the Value of Transformer Components [Internet]. arXiv; 2020 [cited 2025 Jan 13]. Available from: https://arxiv.org/abs/2011.03803

[13] Frankle J, Carbin M. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Internet]. arXiv; 2018 [cited 2025 Jan 13]. Available from: https://arxiv.org/abs/1803.03635 arXiv

[14] Gal Y, Ghahramani Z. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. Proceedings of the 33rd International Conference on Machine Learning. JMLR: W&CP; 2016. Available from: https://arxiv.org/abs/1506.02142.

[15] Bishop CM. Pattern Recognition and Machine Learning. 1st ed. New York: Springer; 2006.

[16] Sun T, Yin B, Bohte S. Efficient uncertainty estimation in spiking neural networks via MC-dropout. In: Artificial Neural Networks and Machine Learning – ICANN 2023. Lecture Notes in Computer Science. Springer; 2023. p. 393–406.

[17] Shin M, Cho H, Min H, Lim S. Neural Bootstrapper. Adv Neural Inf Process Syst. 2021;34:16596–609

[18] IEEE Transactions on Pattern Analysis and Machine Intelligence. 2002 Index. IEEE Trans Pattern Anal Mach Intell. 2002;24(12):1679-1695.

[22] McInnes, L., Healy, J. and Melville, J., 2018. Umap: Uniform manifold approximation and

projection for dimension reduction. *arXiv preprint arXiv:1802.03426*

[25] Fisher, R.A., 1922. On the mathematical foundations of theoretical statistics. *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, *222*(594-604), pp.309-368.

Cover, T.M. and Thomas, J.A., 1991. Entropy, relative entropy and mutual information. *Elements of information theory*, *2*(1), pp.12-13.

[31] Kullback, S. and Leibler, R.A., 1951. On information and sufficiency. *The annals of mathematical statistics*, *22*(1), pp.79-86.

[26] Bertsekas, D. and Tsitsiklis, J.N., 2008. *Introduction to probability* (Vol. 1). Athena Scientific.

[30] Do Carmo, M.P. and Flaherty Francis, J., 1992. *Riemannian geometry* (Vol. 2). Boston: Birkhäuser.

[23] Barth, L.S., Joharinad, P., Jost, J., Keck, J. and Mikhail, T.J., 2024. Fuzzy simplicial sets and their application to geometric data analysis. *arXiv preprint arXiv:2406.11154*.

[21] Van der Maaten, L. and Hinton, G., 2008. Visualizing data using t-SNE. *Journal of machine learning research*, *9*(11).

[20] Abdi, H. and Williams, L.J., 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, *2*(4), pp.433-459.

[29] Shannon, C.E., 1948. A mathematical theory of communication. *The Bell system technical journal*, *27*(3), pp.379-423.

[24] MacKay, D.J., 2003. *Information theory, inference and learning algorithms*. Cambridge university press.

[19] Bellman, R., 1966. Dynamic programming. *science*, *153*(3731), pp.34-37.

[27] Reimers, N., and Gurevych, I., 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. doi:10.18653/v1/D19-1410

[28] Wu, J., Chen, X.Y., Zhang, H., Xiong, L.D., Lei, H. and Deng, S.H., 2019. Hyperparameter optimization for machine learning models based on Bayesian optimization. *Journal of Electronic Science and Technology*, *17*(1), pp.26-40.