The Approach

* I employed a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN)
* The motivation was to circumvent the Bayesian model's limitations, notably its reliance on manually extracted features such as word stemming and sentiment analysis.
* I envisage that a deep learning model, capable of autonomously extracting features from raw text, would be a more robust model
* LSTMs are capable of selectively remembering and forgetting information through their architecture, which includes memory cells and gates (input, forget, and output gates).
* These gates regulate the flow of information. Such an arrangement allows LSTMs not only to maintain crucial contextual information across long pasages of text but also to dynamically adapt to new inputs, effectively capturing the intricacies and nuances of natural language.
* The network effectively learns which information might be needed later on in a sequence and when that information is no longer needed. For instance, in the context of natural language processing, the network can learn grammatical dependencies.
* This ability allows them to preserve essential contextual data over long text sequences and adapt dynamically to new information, thereby by theory, able to capture the subtleties and complexities of natural language.
* Configuring the LSTM RNN model involved making thoughtful decisions regarding its architecture, such as selecting the appropriate number of layers, setting dropout rates, and choosing the type of word embeddings.
* I predict to gain a deeper comprehension of textual context and sentiment, achieving a level of feature learning sophistication unattainable with the Bayesian methodology.
* Bi-directional LSTM model captures patterns in past and future contexts, at the cost of adding comlexity to double the parameters.
* Implemented a deep learning model with an input layer, embedding layer, bi-directional LSTM layer and output layer.
* To help reduce overfitting, I implemented dropout to the LSTM layer and regularsation to the output layer.

The architecture of a deep learning model for NLP sentiment analysis can indeed seem complex relative to the dataset size. However, the complexity of the model should be appropriate for the task's requirements and the data's complexity, not just the number of entries.

For a dataset with 1,200 entries, here are a few points to consider:

1. \*\*Overfitting\*\*: A model with a large number of parameters is more prone to overfitting, especially when the dataset is small. Regularization techniques like dropout are essential to prevent this.

2. \*\*Embedding Layer\*\*: The embedding layer's size (1,280,000 parameters) indicates a large vocabulary. For smaller datasets, you might not need such a large vocabulary. It could be more efficient to use pre-trained embeddings like Word2Vec or GloVe or reduce the `input\_dim` if you have many rare words that do not contribute much to the sentiment analysis.

3. \*\*Bidirectional LSTM\*\*: While a Bidirectional LSTM captures patterns from both past and future data context, it also doubles the number of parameters of a regular LSTM. Consider whether this complexity is warranted by the predictive power it provides.

4. \*\*Model Simplification\*\*: For smaller datasets, a simpler model may be sufficient. You could experiment with reducing the embedding dimension, using a regular (unidirectional) LSTM, or even trying simpler models like Convolutional Neural Networks (CNNs) for text, which can be quite effective with fewer parameters.

5. \*\*Transfer Learning\*\*: Leveraging transfer learning by using a pre-trained model as a starting point can allow you to benefit from knowledge gained from larger datasets while fine-tuning on your specific small dataset.

6. \*\*Data Augmentation\*\*: Techniques to artificially expand the dataset can be beneficial, especially if the model is complex. For text, this might involve synonym replacement, back-translation, or other NLP-specific augmentations.

7. \*\*Parameter Tuning\*\*: Given the relatively small dataset size, extensive hyperparameter tuning (e.g., grid search, random search, Bayesian optimization) is important. It's often more valuable than increasing model complexity.

In summary, it might be worthwhile to simplify the model or apply other strategies to fit the small dataset better while still capturing the nuances necessary for accurate sentiment analysis. You can also monitor validation performance to decide if the model's complexity is justified. If the simpler models perform similarly or better, they are likely more appropriate for your dataset

References

*https://en.wikipedia.org/wiki/Long\_short-term\_memory*

*Russel . Norvig - Artifical Intelligence – A Modern Approach – 4th edition – Chapter 22.6.2*

*Russel . Norvig - Artifical Intelligence – A Modern Approach – 4th edition – Chapter 25.2.3*

[*https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17*](https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17)

[*https://www.embedded-robotics.com/sentiment-analysis-using-lstm/*](https://www.embedded-robotics.com/sentiment-analysis-using-lstm/)

[*https://www.analyticsvidhya.com/blog/2021/06/natural-language-processing-sentiment-analysis-using-lstm/*](https://www.analyticsvidhya.com/blog/2021/06/natural-language-processing-sentiment-analysis-using-lstm/)

I implemented a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) to address the limitations inherent in the Bayesian model, particularly its reliance on manually extracted features like word stemming and sentiment analysis. I anticipated that a deep learning model with the innate ability to autonomously extract features from raw text would prove to be more effective. The LSTM architecture, equipped with memory cells and various gates such as input, forget, and output gates, is designed to regulate information flow, allowing the network to retain essential contextual details over lengthy text passages and to adapt dynamically to new inputs. This architectural framework enables the LSTM to learn which information is relevant over a sequence and when it can be discarded, a capability particularly useful in understanding grammatical dependencies in natural language processing.

Configuring the LSTM RNN model required deliberate architectural choices, including the number of layers, dropout rates, and types of word embeddings used, to optimise the model's performance for our specific NLP task. I aimed to achieve an enhanced understanding of textual context and sentiment and a level of feature learning sophistication that was not possible with the Bayesian approach.

I opted for a bi-directional LSTM model, which can capture patterns from both past and future contexts, despite doubling the number of parameters and adding complexity. The deep learning model I implemented consisted of an input layer, an embedding layer, a bi-directional LSTM layer, and an output layer. To combat overfitting, dropout was applied to the LSTM layer, and regularization was added to the output layer, measures intended to ensure the model's ability to generalise well to new, unseen data.

In deep learning, choosing the right hyperparameters can greatly influence model performance. For the LSTM\_Classifier being utilized for a binary classification task (such as sentiment analysis), the selection of `optimisation\_model = 'adam'`, `dense\_output\_activation\_function = 'sigmoid'`, and `model\_loss\_function = 'binary\_crossentropy'` is strategic and grounded in the model's goals and the nature of the task at hand. Here's why these choices are considered good:

### 1. Optimisation Model: Adam

- \*\*Why a Good Choice\*\*: Adam (Adaptive Moment Estimation) is an optimization algorithm that combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. It's well-suited for problems that are large in terms of data and/or parameters. Adam is known for its effectiveness in both convergence speed and performance on problems with large datasets and high-dimensional spaces, making it a solid choice for training deep learning models.

- \*\*Advantages\*\*: Adam automatically adjusts the learning rate during training, which helps in converging faster. It stores an exponentially decaying average of past gradients and squared gradients, which helps in navigating the complex landscapes of high-dimensional data. This makes it particularly effective for tasks like text classification using LSTM models, where the data dimensionality can be quite high.

### 2. Dense Output Activation Function: Sigmoid

- \*\*Why a Good Choice\*\*: For binary classification tasks, the sigmoid function is an appropriate choice for the activation function in the output layer. This is because the sigmoid function outputs a probability score between 0 and 1, which can be interpreted as the probability of belonging to the positive class (in a binary classification problem).

- \*\*Advantages\*\*: The sigmoid activation function is smooth and differentiable at every point, which helps during the backpropagation process to calculate gradients more effectively. It clearly maps predictions to either 0 or 1 with a well-defined threshold (usually 0.5), which is useful for making a decision in binary classification problems.

### 3. Model Loss Function: Binary Crossentropy

- \*\*Why a Good Choice\*\*: Binary crossentropy, also known as log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. It is suitable for binary classification tasks. Binary crossentropy is a loss function that quantifies the difference between the predicted probabilities and the actual binary labels in the dataset, making it a natural fit for tasks predicting a binary outcome.

- \*\*Advantages\*\*: It penalizes incorrect predictions more heavily than those that are close to the true label, which drives the model to make more confident and accurate predictions. The use of binary crossentropy as the loss function complements the sigmoid activation in the output layer by directly optimizing the model based on the probability outputs, making the training process more coherent and efficient.

In summary, the choice of Adam as the optimizer provides robust and efficient gradient descent, the sigmoid activation function maps the output to a probability score suitable for binary classification, and binary crossentropy effectively measures the difference between the predicted probabilities and the actual labels, guiding the model towards better performance on binary classification tasks.

**Evaluation**

The LSTM (Long Short-Term Memory) model might be less accurate than the Bayesian model for several reasons, especially in specific contexts such as sentiment analysis on a relatively small dataset like car reviews. Here are some potential factors:

1. \*\*Complexity vs. Data Size\*\*: LSTMs are more complex models that have a larger number of parameters to learn. If the dataset is not large enough, this complexity can lead to overfitting, where the model learns noise in the training data instead of the underlying pattern. In contrast, simpler models like Naive Bayes might perform better because they have fewer parameters and are less likely to overfit on small datasets.

2. \*\*Feature Representation\*\*: Naive Bayes models, especially when coupled with bag-of-words or TF-IDF feature representations, can be quite effective at capturing the significance of specific words or phrases indicative of sentiment. LSTMs, despite their ability to understand sequence and context, may require more data to achieve a similar level of understanding and effectiveness in utilizing contextual information.

3. \*\*Training and Regularization\*\*: The effectiveness of LSTM models can significantly depend on the choice of hyperparameters, the regularization techniques applied, and the training procedure. Insufficient training, improper regularization (to prevent overfitting), or suboptimal choice of hyperparameters can lead to poorer performance compared to a well-tuned simpler model.

4. \*\*Vanishing Gradient Problem\*\*: Despite LSTMs being designed to mitigate the vanishing gradient problem, they can still suffer from it in practice, especially with very long sequences or improperly configured models. This can hinder the model's ability to learn effectively.

5. \*\*Initialization and Optimization\*\*: The way the LSTM model is initialized and optimized (e.g., choice of optimizer, learning rate) can have a significant impact on its final performance. Poor choices here can lead to suboptimal convergence, affecting accuracy.

6. \*\*Data Preprocessing and Feature Engineering\*\*: Naive Bayes models might benefit more from certain types of feature engineering and preprocessing, such as stemming, stop word removal, and sentiment-specific feature selection. If similar careful preprocessing and feature engineering are not applied when training the LSTM model, its performance might not be optimal.

7. \*\*Model Bias\*\*: Every model has its own biases based on the assumptions it makes about the data. The assumptions inherent in Naive Bayes (e.g., feature independence) might, by coincidence or design, work better for the specific characteristics of the dataset being used.

In summary, while LSTM models have the theoretical capability to outperform simpler models like Naive Bayes by leveraging sequence information and context, practical considerations such as data size, model complexity, training procedures, and feature representation can lead to scenarios where simpler models perform better on specific tasks.

3. \*\*Training and Regularization\*\*: The effectiveness of LSTM models can significantly depend on the choice of hyperparameters, the regularization techniques applied, and the training procedure. Insufficient training, improper regularization (to prevent overfitting), or suboptimal choice of hyperparameters can lead to poorer performance compared to a well-tuned simpler model.

5. \*\*Initialization and Optimization\*\*: The way the LSTM model is initialized and optimized (e.g., choice of optimizer, learning rate) can have a significant impact on its final performance. Poor choices here can lead to suboptimal convergence, affecting accuracy.