Predicting Future Hosts of Mutated Virus Strains through Machine Learning

PROJECT STATEMENT:

The objective of this project is to explore various machine learning models and deep learning models to propose a naive model that can accurately predict the host of viruses based on their DNA sequences. The data used in this project consists of viral DNA sequences obtained from databases such as expassy and NCBI, which encode the genetic information necessary for virus replication and interactions with hosts. Our aim is to create a dataset of viral DNA sequences and develop robust machine learning and deep learning models capable of classifying viruses into eight distinct host classes: humans, plants, vertebrates, invertebrates, bacteria, eukaryotic microorganisms, archaea, and fungi. By accurately predicting viral host specificity, we can gain valuable insights into virus-host interactions, facilitate disease surveillance, and enhance our understanding of viral transmission patterns. The project's outcome can significantly contribute to scientific research, public health, and environmental protection, making it a valuable asset to the academic community and society as a whole.

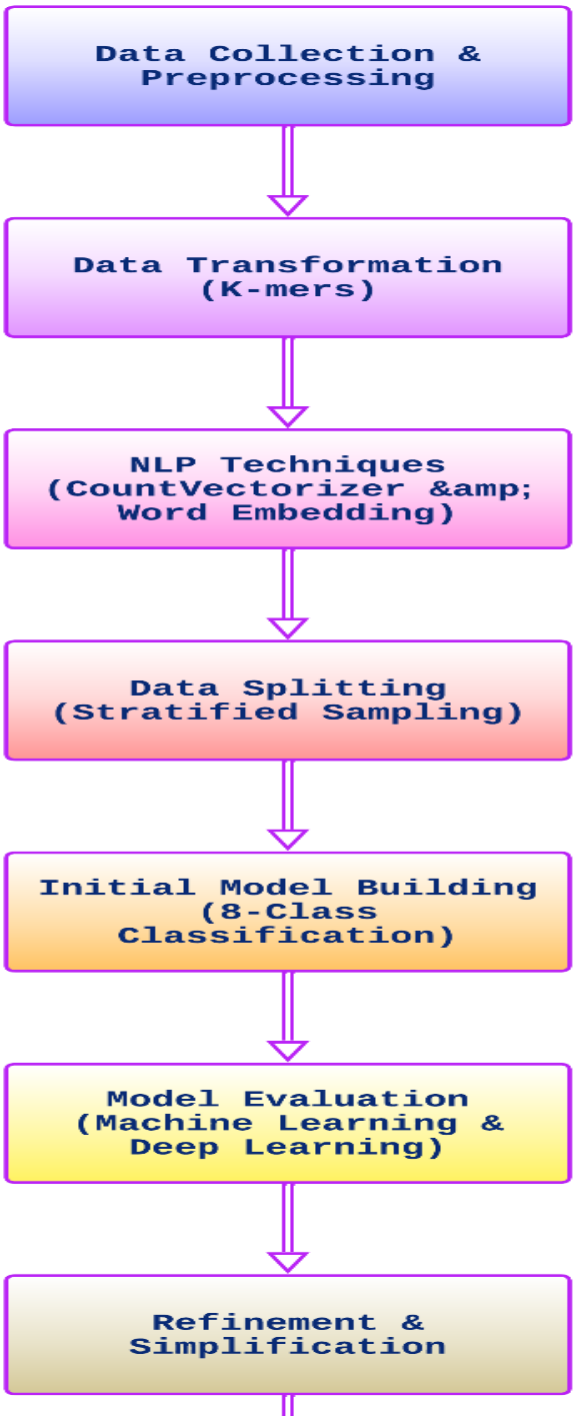
ABSTRACT

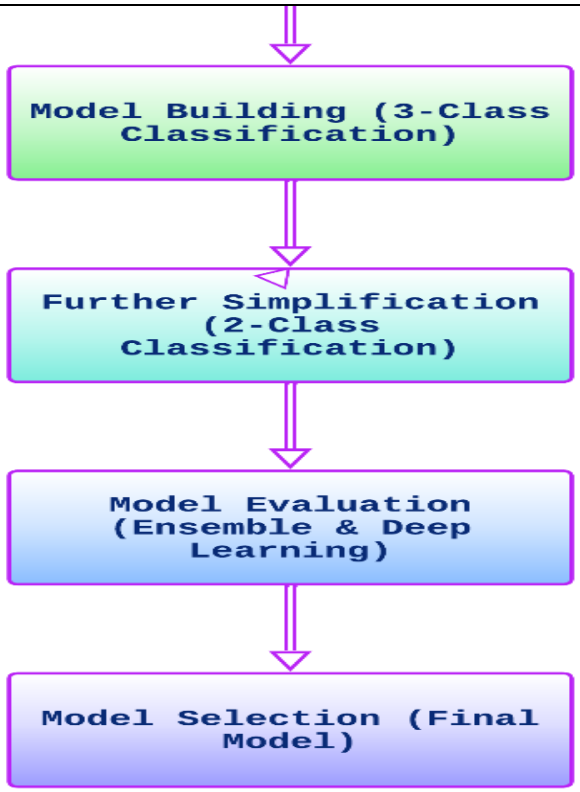
This research project embarks on a journey to harness the power of machine learning techniques for a crucial task: predicting the next host of newly discovered or mutated virus strains. By diving deep into virus-host interactions, our aim is to develop a predictive model that incorporates a blend of genomic and host-related data. Through the application of advanced machine learning algorithms, we delve into existing datasets to uncover the intricate patterns and features of virus DNA that determine successful host adaptation.

Our project holds the potential to revolutionize our understanding of viral evolution, zoonotic transmission, and the strategies we employ to combat emerging infectious diseases. We explore a variety of machine learning models, ranging from simpler classification algorithms like kneighbours, Naive Bayes, Random Forest and ensemble techniques to more complex deep learning models such as CNN and LSTM. These models leverage DNA fasta sequences of viruses to predict hosts, categorizing them into eight classes: humans, vertebrates, invertebrates, plants, fungi, protists, bacteria, and archaea. We kickstart the process with preprocessing the DNA fasta sequences to extract relevant features that encapsulate essential genetic characteristics.Notably, data balancing was meticulously executed to ensure equal representation across these classes. In addition to the original eight-class classification, the project explores reduced classifications of three classes and two classes.The project explored the classification of viral DNA sequences into host classes, including an initial eight-class classification. Among these models, Multinomial Naive Bayes, Random Forest, Gradient Boosting, and XGBoost exhibited superior accuracy and precision, while AdaBoost lagged behind. However, further optimization was suggested to enhance overall performance. Stratified K-Folds were applied to mitigate data imbalance, with varying model performance observed. The Random Forest and K-Nearest Neighbors models showed accuracy around 40%, while Multinomial Naive Bayes achieved a balanced performance. In contrast, AdaBoost performed poorly, while Gradient Boosting showed promise. After reducing the classes to three and two, Random Forest emerged as the top-performing model. Deep learning models, including CNN and LSTM, demonstrated varying degrees of success, with CNN achieving an

accuracy of approximately 51.85%, while LSTM struggled with generalization, achieving 29.63% accuracy. For three-class classification, ensemble methods like RandomForest, Gradient Boosting, and XGBClassifier yielded the best results. Reducing the classification to two classes simplified the problem and improved generalization. RandomForest, AdaBoostClassifier, GradientBoosting, and XGBClassifier excelled in classifying viruses into "Single-Celled" and "Multi-Celled" categories. The CNN model showed promise in binary classification, achieving an accuracy of approximately 82.14%, while the LSTM model exhibited potential overfitting to the training data. These findings underscored the importance of data preprocessing, hyperparameter tuning, and model selection in achieving robust viral host classification.The integration of machine learning with biological and computational approaches holds significant potential for advancing our understanding of viruses, improving disease management, and supporting biomedical research and healthcare applications.

PROCESS OVERVIEW:





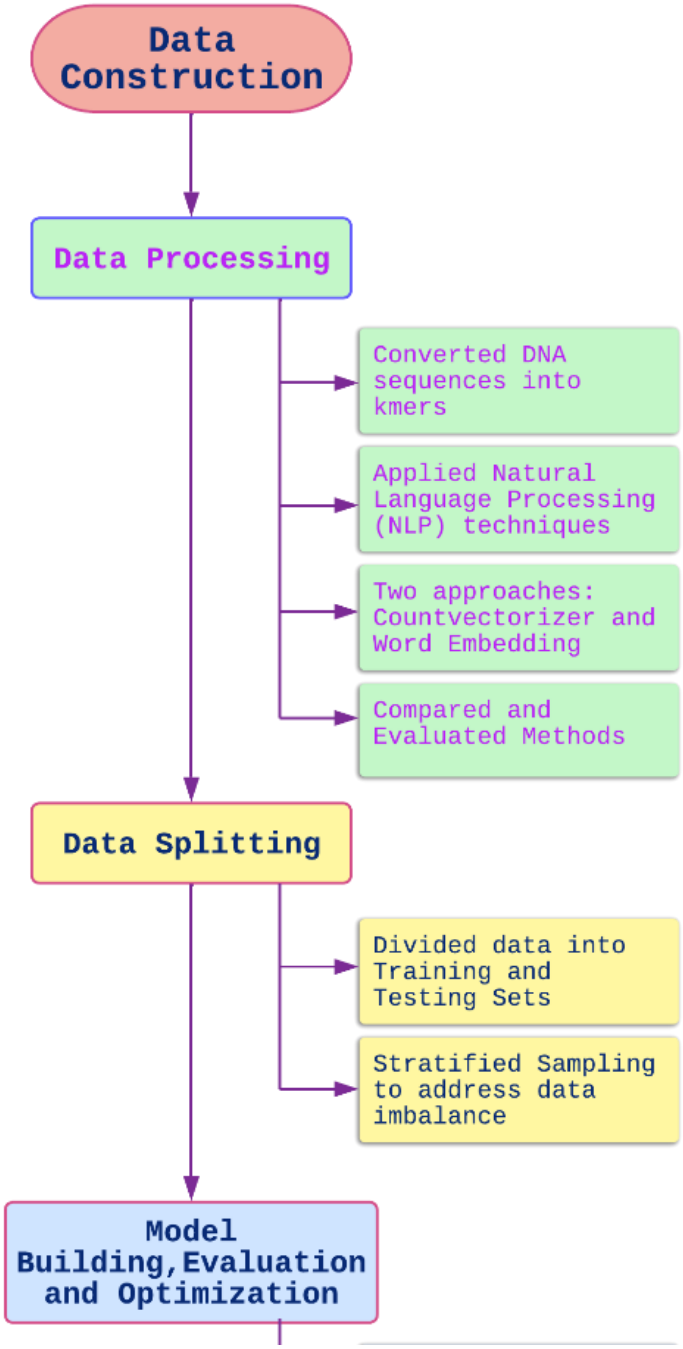
- Data construction: The dataset used for this project was meticulously collected from the ViralZone database, a comprehensive resource categorizing viruses based on their known hosts. Specifically, 35 viral DNA sequences were selected from each of the eight primary host classes listed on ViralZone, which include Archaea, Bacteria, Eukaryotic Microorganisms, Fungi, Plants, Humans, Vertebrates, and Invertebrates. These sequences were retrieved from the National Center for Biotechnology Information (NCBI) database via provided links, downloaded, and stored for subsequent processing. An essential step in data preparation was achieving a balanced dataset, ensuring an equal number of samples from each host class to prevent bias during model training.
- Data Features: Our dataset comprises diverse DNA sequences, each potentially associated with multiple organisms or "hosts." To ensure robust analysis, we carefully organized the data into distinct groups, with each group containing ample samples.
- Classification Challenges: Initially faced with an eight-class classification problem, we encountered complexities for both machine learning and deep learning models.
- Simplified Classifications: To enhance model performance, we pursued two key strategies:
    - 3-Class Classification: Aggregating the eight classes into three broader categories - animals, microorganisms, and plants - led to significant model performance improvements. Ensemble methods and LSTM networks showcased remarkable effectiveness.
    - 2-Class Classification: Further simplifying the problem into single-celled and multi-celled organisms yielded consistently high accuracy. CNN had initial challenges, while LSTM emerged as a robust choice.
- Data Transformation: Our journey began with meticulous data preprocessing, involving the transformation of DNA sequences into k-mers. We harnessed

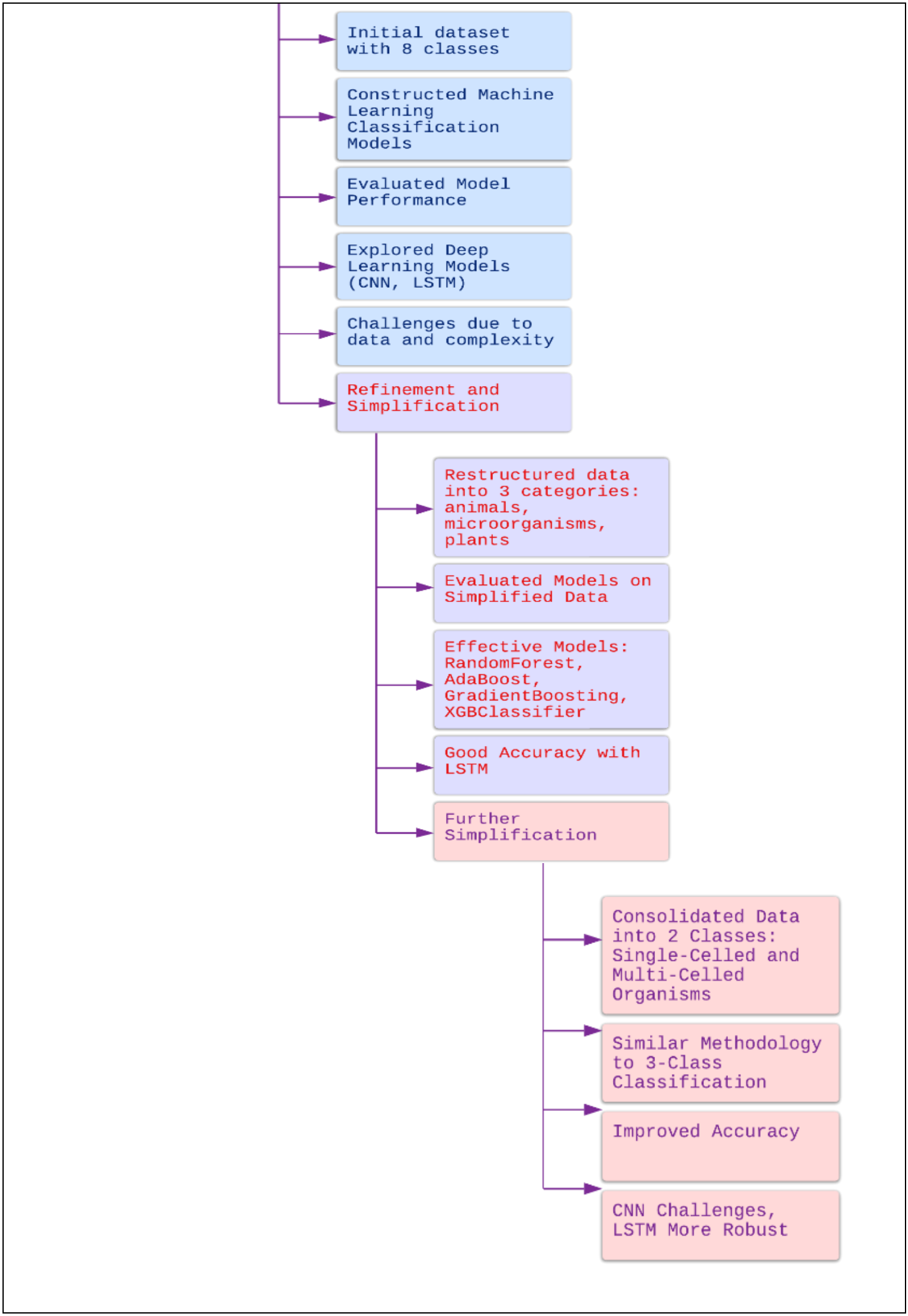
Natural Language Processing techniques, employing both CountVectorizer and Word Embedding for numerical representation, allowing for a thorough comparison.

- Strategic Data Splitting: To facilitate model development, we strategically split the data into training and testing sets, addressing data imbalance issues with stratified sampling.
- Iterative Model Development: We iteratively explored an array of machine learning and deep learning techniques, assessing their performance and continually optimizing our models.

Our project navigates the complexities of DNA sequence classification through strategic data preprocessing, thoughtful data splitting, and an iterative model development approach. By simplifying the classification task, we leveraged a diverse set of techniques, unravelling the intriguing world of DNA sequence classification.

STEP BY STEP WALKTHROUGH OF THE SOLUTION:




Initial dataset with 8 classes
Constructed Machine Learning Classification Models
Evaluated Model Performance
Explored Deep Learning Models (CNN, LSTM)
Challenges due to data and complexity
Refinement and Simplification
Restructured data into 3 categories: animals, microorganisms, plants
Evaluated Models on Simplified Data
Effective Models: RandomForest, AdaBoost, GradientBoosting, XGBClassifier
Good Accuracy with LSTM
Further Simplification
Consolidated Data into 2 Classes: Single-Celled and Multi-Celled Organisms
Similar Methodology to 3-Class Classification
Improved Accuracy
CNN Challenges, LSTM More Robust

➢ DATA CONSTRUCTION:

Our dataset was obtained from the ViralZone database (viralzone.expasy.org). ViralZone provides a comprehensive classification of viruses based on their known hosts. We meticulously selected 35 viral DNA sequences from each of the eight primary host classes available on the database, Archaea, Bacteria, Eukaryotic Microorganisms, Fungi, Plants, Humans, Vertebrates, Invertebrates

- Sequence Retrieval:

After identifying the host classes, we accessed the respective viral DNA sequences through the provided links on ViralZone, which redirected us to the National Center for Biotechnology Information (NCBI) database. These sequences were subsequently downloaded and stored for further processing.

- Data Balancing:

One of the crucial aspects of our data preparation was ensuring that our dataset was balanced across all host classes. This balance prevents any particular class from dominating the training process and potentially leading to biased results. To achieve this, we selected 35 sequences from each host class, resulting in a balanced dataset containing an equal number of samples from each class

➢ DATA PREPROCESSING:

1) Data Cleaning:

The overall strategy applied for data cleaning is to remove characters from the DNA sequences that are either non-standard or do not convey specific nucleotide information.

The newline character '\n' is removed to ensure that sequences are in a clean, continuous format.

Ambiguous characters like 'Y', 'M', 'R', 'N', 's', 'S', 'K', and 'W' are removed to standardize the representation of nucleotides (A, T, G, C).

The result is that the 'DNA_sequences' column contains cleaned and standardized DNA sequences, making them suitable for further analysis or modeling.

2) Encoding the hosts columns:

The label encoding process is applied to the "hosts" column from the data, which presumably contains categorical host labels (e.g., "human," "vertebrates," etc.).

After the label encoding is complete, the encoded labels are stored in the variable y.The y variable now contains numerical representations of the original categorical host labels. These numerical labels can be used as target variables in machine learning models.

3) Converting Sequences Into Kmers

In bioinformatics and genomics, a k-mer is a sequence of 'k' nucleotide bases (e.g., A, T, G, C) taken from a longer DNA sequence. K-mers are widely used in tasks like sequence analysis, genome assembly, and feature extraction because they capture local patterns within DNA sequences.

With Kmer counting, one can divide the string into substrings according to its ‘k’ value. When we have huge list of DNA sequences then its analysis using fixed size ‘k vector’ is easy and prompt. In DNA sequence, divide the nucleotide sequence into part of nucleotide with ‘k value’ (k>0). Dividing the k-mers into tiny sizes also assist to remove the difficulty of variable read lengths. All our machine learning model can work on fixed length input but by using kmer we can take input of variable length. Because in kmer we can divide our input data on the basis of k value. k-mers can also be utilized to discover genome mis-assembly by determining k-mers.

Generating k-mers from DNA sequences is a common preprocessing step in genomics and bioinformatics tasks. K-mers serve as a compact and informative representation of DNA sequences.

- They capture local sequence patterns, which can be crucial for various tasks, including:
- DNA sequence classification (predicting host labels, as in your case).
- Sequence alignment.
- Identifying sequence motifs (recurring patterns).
- Detecting mutations or variations.
- K-mers are often used as input features for machine learning models to predict or classify biological properties or behaviors based on DNA sequences.

➢ SPLITTING THE DATA INTO TRAIN AND TEST SPLIT

The data was split into train and test split with a test size of 0.3

➢ DATA TRANSFORMATION:

Using a Count Vectorizer for data transformation, which involves classifying DNA sequences into different host categories, was found to be a suitable choice.

- Simple and Interpretable: Count Vectorization is a straightforward and interpretable method for converting text data into a numerical format. Each feature (k-mer) is represented by a count, which is easy to understand.
- Preservation of Sequence Information: Count Vectorization retains information about the frequency of k-mers in DNA sequences. This can be crucial for certain types of classification tasks where specific k-mer patterns are indicative of host categories.

- No Loss of Information: Unlike techniques like TF-IDF (Term Frequency-Inverse Document Frequency), Count Vectorization doesn't downweight terms that appear frequently across all sequences. This can be an advantage when all k-mers are potentially important.
- Compatibility with Various Models: The count-based representation can work well with a variety of machine learning models, including traditional models like Naive Bayes, decision trees, and ensemble methods, as well as more complex models like neural networks.

➢ MODEL BUILDING:

For a project involving the classification of viral hosts based on DNA sequences, The choice of the model depends on factors like the size of your dataset, the complexity of the problem, and the computational resources available. These are the commonly used models in bioinformatics and genomics which are used in project.

Random Forest: Random Forest is an ensemble learning method that is robust, handles high-dimensional data well, and is resistant to overfitting. It's a good choice for both small and large datasets.

K-Nearest Neighbors (K-NN): K-NN is a simple yet effective classification algorithm. It's particularly useful when much about the data distribution is unknown and can work with small to moderately sized datasets.

Gradient Boosting Models (e.g., XGBoost): These are ensemble methods that often perform very well in classification tasks. They are known for their speed and accuracy and can handle both small and large datasets.

Neural Networks: Deep learning models, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), can be used for sequence classification tasks like this. They can capture complex patterns in DNA sequences.

Multinomial Naive Bayes: Naive Bayes classifiers are simple and work well with high-dimensional data. They are particularly useful when dealing with text-based or sequence data.

Decision Trees: Decision trees are interpretable and can be useful for understanding feature importance. They can be used as standalone models or within ensemble methods like Random Forest.

Evaluation metrics:

as correctly identifying the viral host is of utmost importance, we must prioritize metrics like recall and specificity. Considering a balance between precision and recall, the F1-Score could be appropriate. To gain a comprehensive understanding of your model's performance we have used multiple metrics.

- MODEL OPTIMIZATION

Since multinomial naive bayes,random forest,gradient boosting and xgbclassifier have given the highest accuracy,precision,recall and f1 score, model optimization was done by tuning the hyperparameters.

process of hyperparameter tuning for multinomial naive bayes,random forest,gradient boosting and xgbclassifier using a randomized search approach, identified the best hyperparameters, fine-tuned the model, and reported various performance metrics to assess the model's classification accuracy and effectiveness on the provided DNA sequence data.

The evaluation metrices did not show a significant rise which means that there was no significant improvement in the models after the hyperparameter tuning. This suggests that the default values of parameters in the model worked the best for these classification models.

- Using stratified k folds for the train test split:

With a relatively small dataset like 35 sequences per class, using a simple train-test split can indeed lead to imbalanced training and testing sets, which can impact the performance and generalization of your classification model. Imbalanced data can result in biased models that perform well on the majority class but struggle with the minority classes.

Instead of a single train-test split, we further used k-fold cross-validation. Cross-validation involves dividing your dataset into k subsets (folds) and training/testing the model k times, with each fold serving as the testing set once. This helps in obtaining more reliable performance estimates and better generalization.

The best accuracy and F1 score among all the base models was given by random forest classifier and gradient boosting model

- Statistical test to check whether there is significant difference in the model performance after using stratified k folds to split the train and test data:

To check whether the models with stratified k-fold cross-validation provide better results, statistical testing was performed to establish the significance of the differences in accuracy and F1-score between the two approaches (stratified k-fold vs. normal train-test split).

hypothesis test was performed to determine whether the differences in performance metrics (accuracy and F1-score) between the two approaches are statistically significant. We used a paired t- test depending on the distribution of data and sample size.

Null Hypothesis (H0): There is no significant difference in performance between the two approaches.

Alternative Hypothesis (H1): There is a significant difference in performance between the two approaches.

Since the p value was greater than 0.05, Hence there is no significant difference or improvement in the accuracy or F1 score after using stratified Kfolds for splitting the train and test data.

- Building base models for different values of k:

With Kmer counting, one can divide the string into substrings according to its 'k' value. When we have huge list of DNA sequences then its analysis using fixed size 'k vector' is easy and prompt. In DNA sequence, divide the nucleotide sequence into part of nucleotide with 'k value' (k>0). Dividing the k-mers into tiny sizes also assist to remove the difficulty of variable read lengths. All our machine learning model can work on fixed length input but by using kmer we can take input of variable length. Because in kmer we can divide our input data on the basis of k value. k-mers can also be utilized to discover genome mis-assembly by determining k-mers.

the k-mer size has various effects on the series assembly. These effects differ between small sized and higher sized k-mers. The aim of various k-mer sizes is to attained a suitable size. A lower k-mer size will store the DNA sequence with the less amount of space and if we choose smaller kmer that it also lose our Informat.Larger sized k-mers will take more memory to store the DNA sequence. Larger k-mer sizes assist to solve the problem of small recurrent regions.So we choose that type of k value that give optimum accuracy.

Accuracy of models for different values of k:

the highest accuracy is achieved with k value 3 and 5 with models xgboost and random forest.A lower k-mer size will store the DNA sequence with the less amount of space and if we choose smaller kmer that it also lose our Informat.Larger sized k-mers will take more memory to store the DNA sequence. Larger k-mer sizes assist to solve the problem of small recurrent regions.hence we can go for kmer size as 5.

We used hyper parameter tuning to improve the model but no increase in the accuracy of models after the hyperparameter tuning, suggest that the algorithm's default parameters work best with the data.

- DEEP LEARNING MODELS:

1)CNN

CNNs are well-suited for tasks involving image and sequence data, making them a natural choice for analysing DNA sequences. They can capture local patterns and motifs within the DNA sequences, which can be informative for predicting viral host specificity. CNNs have shown impressive performance in various bioinformatics tasks, including DNA sequence classification, and can automatically learn hierarchical features from the data.

Data transformation:

For data transformation, word embedding was used. Word embeddings are typically considered a better method than count vectorization for deep learning models when working with text data.

For deep learning models such as recurrent neural networks (RNNs, using word embeddings as input data is often the preferred choice. It allows model to understand the semantic and contextual information within the DNA sequence data, potentially leading to better results compared to count vectorization, which lacks these advantages.

Model building:

- Model Architecture:
  - Input Layer: The model takes input sequences of length 200.
  - Hidden Layer 1: A dense layer with 100 units and ReLU activation function, which introduces non-linearity into the model.
  - Hidden Layer 2: Another dense layer with 50 units and ReLU activation function.
  - Output Layer: The final layer with 8 units (equal to the number of classes) and a softmax activation function, which provides probability distribution over the classes.
- Loss Function: The model uses the "sparse_categorical_crossentropy" loss function, which is commonly used for multi-class classification tasks where the target values are integers (class labels).
- Optimizer: The "adam" optimizer is used for gradient-based optimization during training. Adam is an efficient and commonly used optimizer for deep learning.
- Metrics: The model is evaluated using two metrics during training:
  - "accuracy": This metric calculates the accuracy of the model's predictions on the training and validation datasets.
  - "sparse_categorical_accuracy": This is a specific accuracy metric for the "sparse_categorical_crossentropy" loss, ensuring that it handles integer target values correctly.
- Training Configuration:
  - Number of Epochs: The model is trained for 35 epochs, indicating the number of times the entire training dataset is processed.
  - Batch Size: The training data is divided into batches of 32 samples for each update of the model's weights.
  - Verbose: It specifies whether to display training progress during each epoch (1 for display, 0 for no display).
  - Validation Data: The validation data (X_test and y_test) is used to evaluate the model's performance after each epoch.

2) LSTM:

LSTM (Long Short-Term Memory) can be used for this project of predicting the host of viruses based on DNA sequences.LSTM is specifically designed for sequential data, which is a natural fit for DNA sequences. The ability of LSTM to process sequences of

variable lengths and capture long-range dependencies between DNA bases can be crucial for predicting viral host specificity.

LSTM models can be initialised with pre-trained embeddings, which can capture useful information about DNA bases from large-scale genomic datasets. Transfer learning from related tasks can be leveraged to improve model performance, even with limited viral genomic data.

Model building:

this model architecture leverages word embeddings to convert text data into numerical form, processes it through two LSTM layers to capture sequential patterns, and then uses a dense layer for classification. It's a common choice for text classification tasks and can be further tuned and optimized based on the specific dataset and problem requirements.

- Embedding Layer: The model starts with an Embedding layer. This layer is responsible for converting input text data (sequences of words) into dense numerical vectors. The input_dim parameter is set to the size of the vocabulary, which is determined by the number of unique words in your dataset plus one (to account for out-of-vocabulary words). The output_dim parameter specifies the dimensionality of the dense embedding vectors, in this case, 32. The input_length parameter defines the length of input sequences, which is set to max_sequence_length.
- LSTM Layers: Two LSTM (Long Short-Term Memory) layers are stacked on top of the embedding layer. LSTM is a type of recurrent neural network (RNN) that is well-suited for sequence data like text. The first LSTM layer has 128 units and is set to return sequences (return_sequences=True). This means it produces sequences of outputs for each time step in the input sequence. The second LSTM layer has 64 units and operates in the default mode, which returns only the final output for each sequence.
- Dense Layer: A Dense layer with 8 units and a softmax activation function is added as the output layer. This layer is responsible for classifying input text into one of eight possible classes (hosts).
- Compilation: The model is compiled with the categorical cross-entropy loss function, which is commonly used for multi-class classification tasks. The optimizer used is Adam, a popular choice for gradient-based optimization. The model also tracks the accuracy metric during training.
- Training: The model is trained using the training data (X_train and y_train) for 35 epochs with a batch size of 64. During training, the model learns to map input sequences to their corresponding host labels. The validation data (validation_split=0.2) is used to monitor the model's performance on unseen data and prevent overfitting.

BUILDING MODELS BY REDUCING THE NUMBER OF CLASSES TO 3 CLASSES:

Due to we challenges faced because of limited data and model performance, it can be considered to simplify the classes as needed.

Hence, we reduced the number of classes from 8 classes to 3 classes which are plants, animals and microorganisms.

Reasons to reduce the number of classes:

- Data Availability: If you had limited data for each of the 8 initial classes, combining them into broader categories can help balance the class distribution. Machine learning models often perform better when there is a roughly equal number of samples in each class. This step can help ensure that each class has a sufficient number of samples for training.
- Model Complexity: With 8 classes, the classification task can be more complex, especially if some classes have limited samples. Reducing the number of classes simplifies the problem for your models, making it potentially easier to train accurate classifiers.
- Scientific Relevance: The new class categories—humans, microorganisms, and plants—might align better with the scientific objectives of your research. If your research is primarily focused on understanding interactions or patterns among these three broad categories, this class reduction can be justified.
- Interpretability: Having fewer classes can make the results more interpretable and actionable. It might be easier to draw meaningful conclusions and insights when dealing with a smaller number of classes.
- Practicality: From a practical standpoint, working with fewer classes can simplify the implementation and deployment of machine learning models. It can also make it easier to communicate and explain the results to stakeholders.

DATA PREPARATION FOR 3-CLASS MODEL

Class Reduction: Our initial dataset encompassed eight host classes: Archaea, Bacteria, Eukaryotic Microorganisms, Fungi, Plants, Humans, Vertebrates, and Invertebrates. Recognizing the complexities associated with distinguishing between such a diverse range of hosts, we opted for a more streamlined approach.

New Class Categories:

The three new host class categories are as follows:

- Plants: This category comprises viruses that infect plant organisms.
- Animals: Animals encompass various host organisms, including humans, vertebrates, and invertebrates.
- Microorganisms: This category accommodates viruses that infect microorganisms, including Archaea, Bacteria, Eukaryotic Microorganisms, and Fungi.

- DATA CLEANING:

In the data cleaning phase, we applied several key transformations to ensure the quality and uniformity of our viral DNA sequence data. These steps were essential to eliminate noise and standardize the sequences for subsequent analysis.

- Removal of Special Characters:
  We removed extraneous characters such as '\n', 'Y', 'M', 'R', 'N', 's', 'S', 'K', and 'W' from the DNA sequences. These characters often represented ambiguities or sequencing artifacts and were not informative for our analysis.
- Whitespace Removal:
  To enhance uniformity, we removed all whitespace characters within the DNA sequences. This step streamlined the sequences, making them easier to process and analyze.

  By meticulously cleaning the data, we ensured that our DNA sequences were devoid of irrelevant characters and ready for feature extraction and subsequent modeling. This data refinement process laid the foundation for more accurate and meaningful results in our classification task

DATA TRANSFORMATION:

- Label Encoding: To facilitate machine learning, we encoded the host classes using LabelEncoder, which assigns numerical labels to each host category.
- K-mer Conversion: We segmented the DNA sequences into smaller fragments called k-mers, with each k-mer having a length of 6 nucleotides. This conversion allows us to represent complex DNA sequences in a structured manner.
- Count Vectorization: To convert text data into a numerical format suitable for machine learning, we employed CountVectorizer. This technique transformed our k-mer data into a Document-Term Matrix (DTM), where rows represent DNA sequences and columns represent the counts of specific k-mers.

Addressing Class Imbalance: Recognizing the class imbalance issue, we implemented the Synthetic Minority Over-sampling Technique (SMOTE) for oversampling. This technique balanced the distribution of minority classes (microorganisms and plants), mitigating potential biases in our models.

Train test split was used to create the training and test data with a test size of 0.3

MODELLING:

The following models were used for 3 class classification

K-Nearest Neighbors (KNN): KNN is chosen as a simple baseline model for classification. It operates on the principle that similar data points tend to belong to the same class.

Multinomial Naive Bayes (MNB): MNB is a probabilistic model suitable for text classification tasks like ours. It's chosen for its simplicity and efficiency in handling text data.

Decision Tree (DT): Decision Trees are interpretable and can capture complex relationships in the data. They are used for their ability to provide insights into feature importance.

Random Forest (RF): Random Forest is an ensemble of Decision Trees, offering improved generalization and reduced overfitting. It's chosen for its robustness.

AdaBoost (Ada): AdaBoost combines the outputs of multiple weak classifiers to create a strong classifier. It's chosen for its boosting ability, which can improve classification performance.

Gradient Boosting (GB): Gradient Boosting builds an ensemble of decision trees sequentially, optimizing for errors. It's known for its high accuracy.

XGBoost (XGB): XGBoost is an advanced gradient boosting algorithm known for its speed and performance. It's chosen for its ability to handle large datasets and complex tasks.

Evaluation metrics used:

as correctly identifying the viral host is of utmost importance, we must prioritize metrics like recall and specificity. Considering a balance between precision and recall, the F1-Score could be appropriate. To gain a comprehensive understanding of your model's performance we have used multiple metrics such as accuracy, precision, recall and F1 score.

MODEL OPTIMIZATION:

Hyperparameter tuning was used to optimize the model. We used gridsearchcv and randomized search methods to tune the hyperparameters for different models.

The hyperparameter tuning process effectively optimized the model parameters, enhancing the classification performance. Among the models, XGBClassifier emerged as the top-performing classifier with the highest accuracy, precision, recall, and F1 score after tuning, having 76 percent accuracy and 0.76 F1 score. This suggests that XGBoost is a suitable choice for classifying DNA sequences into the specified categories. However, it's essential to note that model selection should consider both performance and computational efficiency, as XGBoost can be computationally intensive.

DEEP LEARNING MODELS:

- Data Transformation:
- K-mers Generation: DNA sequences are initially converted into k-mers (short subsequences of length k).
- Feature Extraction with Word2Vec and TF-IDF: A DocToVec class is defined to convert k-mers into numerical vectors. It uses Word2Vec to represent individual words in the k-mers and TF-IDF (Term Frequency-Inverse Document Frequency) to assign weights to these word vectors.

- Embedding K-mers: The k-mers are embedded into numerical vectors using the DocToVec class, resulting in a vector representation for each DNA sequence.
- Label Encoding: The target labels (hosts) are encoded into numerical values using Label Encoding.

CNN MODEL FOR THREE CLASS CLASSIFICATION:

➢ Model Building:

- Data Splitting: The data is split into training and testing sets using a 80-20 split ratio.
- Reshaping Input: The training data (X_train) is expanded along the last axis to match the input shape expected by the convolutional neural network (CNN).
- CNN Architecture: A sequential CNN model is defined with the following layers:
    - Input Layer: Accepts input vectors of size 200 (the size of the embedded DNA sequences).
    - Dense Layer 1: A fully connected layer with 100 neurons and ReLU activation.
    - Dense Layer 2: Another fully connected layer with 50 neurons and ReLU activation.
    - Output Layer: A softmax activation layer with 3 neurons (corresponding to the 3 classes of hosts).
- Model Compilation: The model is compiled with the categorical cross-entropy loss function, the Adam optimizer, and metrics such as accuracy and sparse categorical accuracy.
- Model Training: The model is trained using the training data (X_train and Y_train) with 34 epochs and a batch size of 32.
- Model Evaluation: The model is evaluated on the test data (X_test and Y_test) to calculate loss and accuracy.

The model learned to classify DNA sequences into host categories effectively. It started with relatively low accuracy but progressively improved as more epochs were completed.

The training accuracy reached approximately 89.82%, which means that, on the training data, the model correctly predicted the host category for about 89.82% of the sequences.

The validation accuracy reached approximately 75.44%. This indicates that the model generalized well to unseen data, as the validation accuracy is close to the training accuracy.

LSTM FOR THREE CLASS CLASSIFICATION:

Model Architecture:

➢ Input Layer:

- Embedding Layer: Converts input DNA sequences into dense vectors.
- Input Dimension: Determined by the size of the vocabulary (number of unique words in the sequences), plus 1 for out-of-vocabulary words.
- Output Dimension: 32 (chosen hyperparameter).
- Input Length: Fixed to a maximum sequence length of 100 (chosen hyperparameter).

➢ LSTM Layers:
  - First LSTM Layer (128 units): This layer is designed to capture intricate sequential patterns in the DNA sequences. It returns sequences because return_sequences is set to True, allowing it to pass sequences to the next layer.
  - Second LSTM Layer (64 units): Building on the abstractions learned by the first LSTM layer, this layer further extracts features and patterns.

➢ Output Layer:
  - Dense Layer with 3 units: The final output layer, designed for multi-class classification, generates probability distributions over the three host categories using the "softmax" activation function. Each unit represents the likelihood of the sequence belonging to one of the three classes.

➢ Loss Function: Sparse Categorical Crossentropy
  - This loss function is suitable for multi-class classification tasks where labels are integer-encoded (as opposed to one-hot encoded).

➢ Optimizer: Adam Optimizer
  - The Adam optimizer is widely used for training neural networks and adapts the learning rate during training to improve convergence.

➢ Training Configuration:
  - Number of Epochs: 35
  - Batch Size: 64
  - Validation Split: 20%, During training, 20% of the training data is set aside for validation to monitor model performance and prevent overfitting.

➢ Data Preparation:
  - Input sequences (DNA sequences) are tokenized and converted into dense vectors using the Embedding layer.
  - Target labels (host categories) are encoded as integers using label encoding.
  - Padding is applied to ensure that all sequences have a uniform length of 100.

The model's primary purpose is to learn intricate patterns and relationships within DNA sequences that can distinguish between the three host categories. By training on labelled data, it aims to accurately classify unseen sequences into one of the specified host categories.

➢ Model Complexity: The model employs LSTM layers, which are effective in handling sequential data. The two LSTM layers allow the model to capture both short-term and long-term dependencies within the sequences.

- Performance Evaluation: To assess the model's performance, metrics such as accuracy and loss are tracked during training. After training, the model can be evaluated on a separate test dataset to gauge its real-world classification accuracy.

The LSTM model demonstrated strong learning capabilities during training, achieving nearly perfect accuracy on the training data.

However, the model's performance on the validation and test datasets suggests that it might be overfitting to some extent. It may have learned noise or specific details of the training data that do not generalize well.

The drop in performance on the test data compared to the validation data reinforces the need for regularization techniques, such as early stopping or dropout layers, to improve generalization.

The observed overfitting in both the LSTM and CNN models can be attributed to the presence of duplicate data in our dataset, particularly for viruses that infect multiple hosts. These duplicates introduce bias into the training process, as the model may inadvertently memorize repeated examples rather than learning meaningful patterns. As a result, the models perform exceptionally well on the training data but struggle to generalize to unseen data, leading to reduced performance on the validation and test sets. This overfitting phenomenon underscores the importance of data preprocessing techniques such as deduplication, stratified sampling, and data augmentation, which should be implemented to mitigate the impact of duplicates and build models that better capture the underlying genetic patterns across diverse viruses and hosts.

BUILDING MODELS BY REDUCING THE NUMBER OF CLASSES TO 2 CLASSES:

reducing the classes to "Single-Celled" and "Multi-Celled" organisms simplifies the problem, reduces data redundancy, and encourages the model to learn more generalizable patterns. This approach can enhance the model's ability to generalize to unseen data and mitigate overfitting, ultimately improving its predictive performance.

Reducing the classes in the data to "Single-Celled" and "Multi-Celled" organisms can help mitigate the overfitting problem caused by duplicate data in several ways:

- Simplified Classification: By categorizing viruses into broader classes based on whether they infect single-celled or multi-celled organisms, we reduce the complexity of the classification task. This simplification can make it easier for the model to learn and generalize patterns without being overwhelmed by the diversity of host species.
- Reduced Data Redundancy: Many viruses infect multiple hosts within the same broad category (e.g., multiple single-celled organisms). When we group these viruses together, we effectively reduce the redundancy in the data. This minimizes the chances of the model memorizing similar examples and instead

encourages it to focus on the distinctive genetic features of single-celled and multi-celled infecting viruses.

- Enhanced Generalization: With fewer classes, the model has a better chance of learning meaningful, transferable features that apply broadly to each category. This can lead to improved generalization when making predictions on unseen data. In essence, we're asking the model to recognize fundamental genetic traits associated with the infectivity of single-celled or multi-celled hosts, rather than trying to distinguish between a multitude of host species.
- Data Balance: It's likely that the distribution of viruses across the two broad categories will be more balanced than the distribution across numerous specific host species. A balanced dataset can help prevent the model from becoming biased towards the majority class, which is a common issue when dealing with imbalanced data.

➢ DATA CONSTRUCTION FOR 2-CLASS MODEL:

In our dataset, we initially encountered challenges related to data redundancy, where many viruses infected multiple host species. To address this issue and enhance the model's performance, we simplified the classification task. Instead of classifying viruses based on specific host species, we categorized them into two broad classes:

- Single-Celled Organisms: This class includes viruses that primarily infect single-celled organisms, such as bacteria and archaea. By grouping these viruses together, we create a more generalized category that captures common genetic traits associated with infectivity in single-celled hosts.
- Multi-Cellular Organisms: In contrast, this class comprises viruses that primarily infect multi-cellular organisms, including plants, animals, and humans. Similar to the first category, this grouping simplifies the classification problem, focusing on features relevant to infecting multi-cellular hosts.

By implementing this two-class approach, we streamlined the data, reduced redundancy, and allowed the model to concentrate on essential genetic characteristics associated with infectivity in either single-celled or multi-cellular organisms. This data preparation step aims to improve the model's generalization, mitigating overfitting issues and enhancing predictive accuracy.

➢ Data Description and Cleaning:
  - The dataset contains 279 samples with two columns: 'DNA_sequences' and 'hosts'.
  - 'DNA_sequences' contains DNA sequence data as strings, and 'hosts' represents the target labels indicating whether the virus infects single-celled or multi-cellular hosts.
  - Data cleaning involved removing unnecessary characters ('Y', 'M', 'R', 'N', 's', 'S', 'K', 'W') from the DNA sequences to ensure consistent and clean data.

➢ DATA TRANSFORMATION:

- DNA sequences were transformed into k-mers (subsequences of length 6) to represent the data as features for modeling.
- Label encoding was applied to the 'hosts' column, converting 'single_celled' to 1 and 'multicellular' to 0.
- Count Vectorization was used to convert k-mer sequences into a Document-Term Matrix (DTM).

➢ MODELLING:

- The imbalanced dataset issue was addressed using Synthetic Minority Over-sampling Technique (SMOTE) to balance the class distribution.
- The dataset was split into training and testing sets.
- Two classes classification models were trained and evaluated using, KNeighbors Classifier, Multinomial Naive Bayes, Naive Bayes (Gaussian), Decision Tree Classifier, Random Forest Classifier, AdaBoost Classifier, Gradient Boosting Classifier, XGBoost Classifier

➢ MODEL EVALUATION:

- For each model, accuracy, precision, recall, and F1-score were calculated.
- The Random Forest, AdaBoost, Gradient Boosting, and XGBoost models showed the highest accuracy and F1-scores, indicating strong predictive performance.
- It's noted that the models achieved almost perfect accuracy on the training data but showed some variation in performance on the test data. This could indicate potential overfitting, and further hyperparameter tuning or cross-validation may be necessary to enhance model generalization.

DEEP LEARNING MODELS FOR 2 CLASS CLASSIFICATION:

➢ Data Transformation:

- K-mers Generation: DNA sequences are initially converted into k-mers (short subsequences of length k).
- Feature Extraction with Word2Vec and TF-IDF: A DocToVec class is defined to convert k-mers into numerical vectors. It uses Word2Vec to represent individual words in the k-mers and TF-IDF (Term Frequency-Inverse Document Frequency) to assign weights to these word vectors.
- Embedding K-mers: The k-mers are embedded into numerical vectors using the DocToVec class, resulting in a vector representation for each DNA sequence.
- Label Encoding: The target labels (hosts) are encoded into numerical values using Label Encoding.

CNN MODEL:

➢ Model architecture: Our CNN-based binary classification model is designed to classify DNA sequences into two distinct categories. It comprises three key layers:

- Input Layer: This layer accepts input sequences with a length of 200.
- Hidden Layers: The model has two hidden layers for feature extraction and representation:
    - The first hidden layer contains 100 neurons with the ReLU activation function.
    - The second hidden layer consists of 50 neurons, also using the ReLU activation function.
- Output Layer: The output layer is composed of two neurons using the sigmoid activation function, making it suitable for binary classification tasks.
- Model Training: We trained the model using the 'sparse_categorical_crossentropy' loss function and the 'Adam' optimizer. The model underwent 34 training epochs, with a batch size of 32. During training, we utilized metrics such as accuracy and sparse categorical accuracy to assess model performance.
- Evaluation: Upon completing training, we evaluated the model on our test dataset. The evaluation yielded two essential metrics: the test loss and test accuracy.

The CNN model demonstrated strong learning capabilities during training, achieving high accuracy on both the training and validation datasets.

The test results suggest that the model can generalize well to new data, with a test accuracy of approximately 82.14%.

This CNN model shows promise for binary classification tasks and can be considered a reliable tool for distinguishing between the two classes in your DNA sequence dataset. Further analysis and fine-tuning can help optimize its performance if necessary.

LSTM MODEL:

- LSTM Model Architecture:
    - The LSTM (Long Short-Term Memory) model is structured as follows:
    - Embedding Layer: This layer translates tokenized input sequences into dense vectors, each comprising 32 dimensions, for compatibility with LSTM layers.
        - LSTM Layer 1: A recurrent layer consisting of 128 units, designed to return sequences.
        - LSTM Layer 2: A subsequent LSTM layer featuring 64 units.
        - Dense Output Layer: Comprised of two neurons, activated by the sigmoid function, representing the binary output classes.
- Model Compilation:
    - The model is compiled using the sparse categorical cross-entropy loss function, a fitting choice for multi-class classification scenarios.

- To facilitate gradient descent, the Adam optimizer is employed, with accuracy serving as the model's primary evaluation metric.

➢ Model Training:

- Over a course of 35 epochs, the model is trained, with each epoch processing a batch of 64 samples.
- Throughout training, a subsection (20%) of the training data acts as a validation set, enabling continuous assessment of the model's performance.

➢ Model Assessment:

- Subsequently, the trained model is subjected to evaluation using the designated test dataset.
- The evaluation process computes the test loss and accuracy, providing key insights into the model's classification capability.

The LSTM model appears to perform well during training and validation, achieving high accuracy and gradually reducing loss. However, when evaluated on unseen test data, the model's accuracy drops, suggesting potential overfitting to the training data.

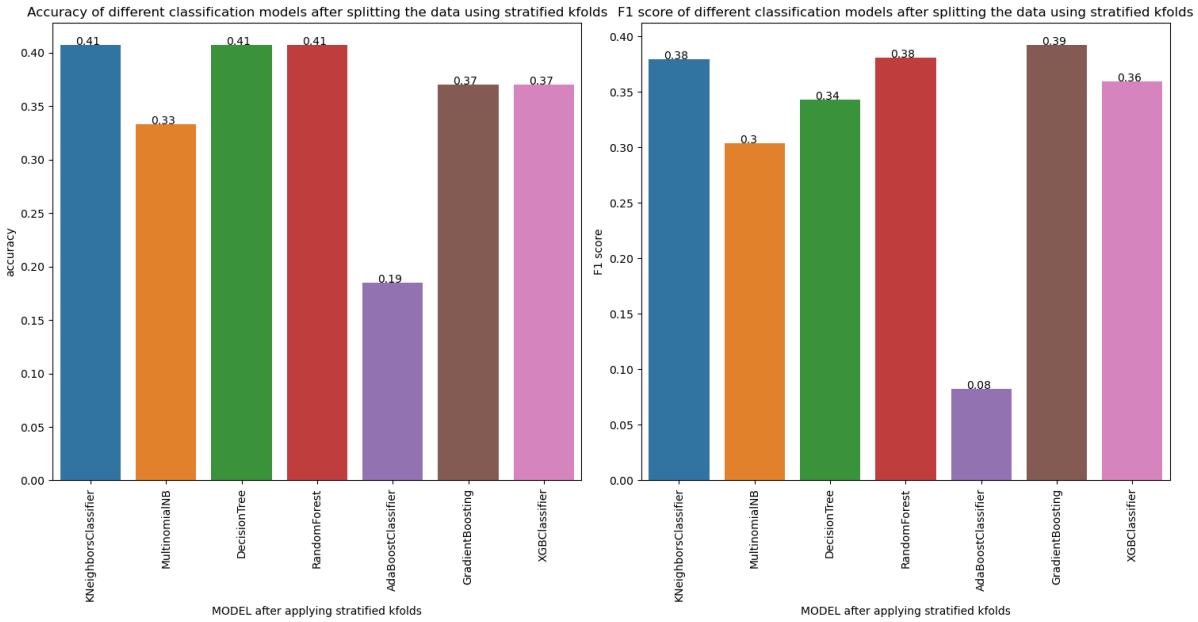
➢ MODEL EVALUATION:

as correctly identifying the viral host is of utmost importance, we must prioritize metrics like recall and specificity. Considering a balance between precision and recall, the F1-Score could be more appropriate. To gain a comprehensive understanding of your model's performance we have used multiple metrics such as accuracy, precision, recall and F1 score

Models of the three datasets are evaluated using classification metrics. These models are compared by their accuracy and f1 scores on the train data.

Evaluation metrics of 8 class classification models:

	MODEL	accuracy(k=6)	precision(k=6)	recall(k=6)	F1 score
0	KNeighborsClassifier	0.392857	0.430543	0.392857	0.363221
1	MultinomialNB	0.416667	0.466781	0.416667	0.410926
2	DecisionTree	0.297619	0.306430	0.297619	0.293781
3	RandomForest	0.416667	0.450086	0.416667	0.405165
4	AdaBoostClassifier	0.119048	0.024685	0.119048	0.037477
5	GradientBoosting	0.416667	0.448138	0.416667	0.407114
6	XGBClassifier	0.428571	0.456562	0.428571	0.409335



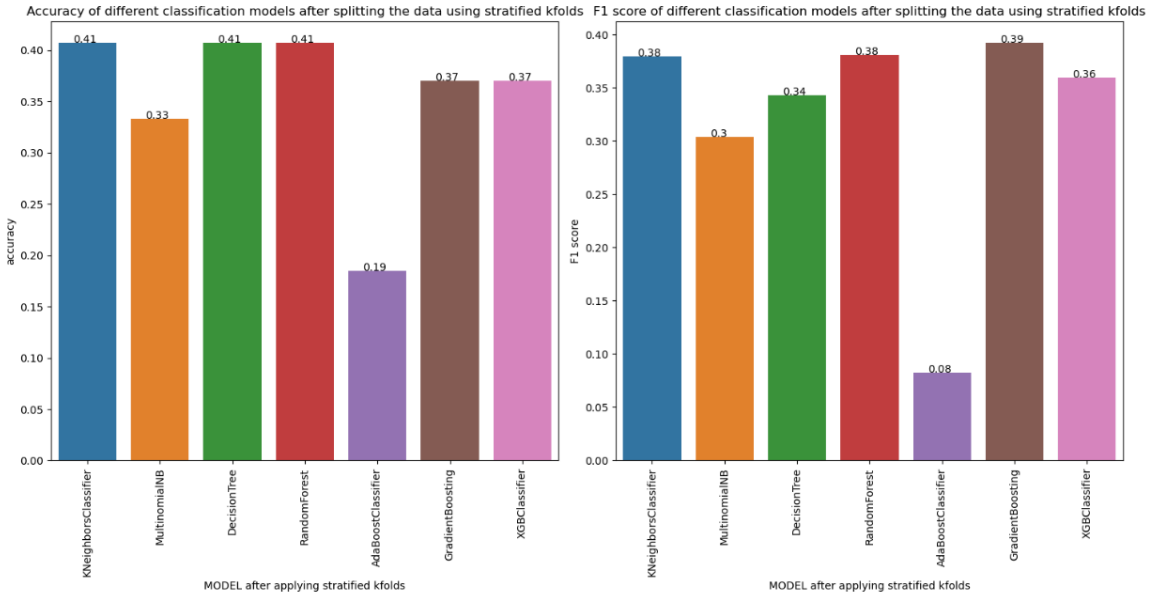
the 8-class classification models exhibited varying degrees of performance, with accuracies ranging from approximately 11.90% to 42.86%. Multinomial Naive Bayes, Random Forest, Gradient Boosting, and XGBoost showed better accuracy and precision compared to other models, while AdaBoost performed poorly. Further optimization and experimentation may be needed to improve the overall performance of these models, especially in the context of 8-class classification.

From the above plot, we can observe that all the models have not shown good results. Among all the models, KNearestNeighbors and RandomForest algorithms have done a better job at the classification.

- Evaluation of models after using stratified k folds for splitting the data:

Evaluation metrics of models after using stratified k fold for optimization:

	MODEL after applying stratified kfolds	accuracy	precision	recall	F1 score
0	KNeighborsClassifier	0.407407	0.395885	0.407407	0.379358
1	MultinomialNB	0.333333	0.391975	0.333333	0.304249
2	DecisionTree	0.407407	0.385185	0.407407	0.343210
3	RandomForest	0.407407	0.444444	0.407407	0.380792
4	AdaBoostClassifier	0.185185	0.059596	0.185185	0.082222
5	GradientBoosting	0.370370	0.466490	0.370370	0.392593
6	XGBClassifier	0.370370	0.428042	0.370370	0.359259



After applying the Stratified K-Folds technique to split the data for the 8-class classification models, we observed some interesting trends. While the overall performance of the models remained consistent in terms of accuracy and F1 score, there were variations in their capabilities.

Among the models, the Random Forest and K-Nearest Neighbors (KNN) classifiers displayed accuracy levels around 40%, while their F1 scores indicated a relatively lower capacity for precision and recall. Multinomial Naive Bayes exhibited a balanced but slightly lower accuracy and F1 score.

On the other hand, AdaBoost performed the least effectively with the lowest accuracy and F1 score among all models. In contrast, Gradient Boosting showed promise with a higher precision and F1 score, suggesting its potential for certain classes within the dataset.

Overall, these models have shown varying degrees of performance when considering both accuracy and F1 score. It is important to note that further model optimization and feature engineering may be required to enhance their capabilities, especially for certain classes that present more challenging classification tasks.

Statistical test to validate whether there is significant difference in the model performances after using stratified k folds to split the train and test data:

```
Paired t-test results for accuracy:
t-statistic: -0.03583669301487836
p-value: 0.9725750945121413

Paired t-test results for F1-score:
t-statistic: -0.5830998424158996
p-value: 0.581064705647061
The difference in accuracy is not statistically significant.
The difference in F1-score is not statistically significant.
```

The paired t-tests were conducted to compare the performance metrics (accuracy and F1-score) between models trained using the Stratified K-Folds technique and models trained with the normal data split. Here are the key findings:

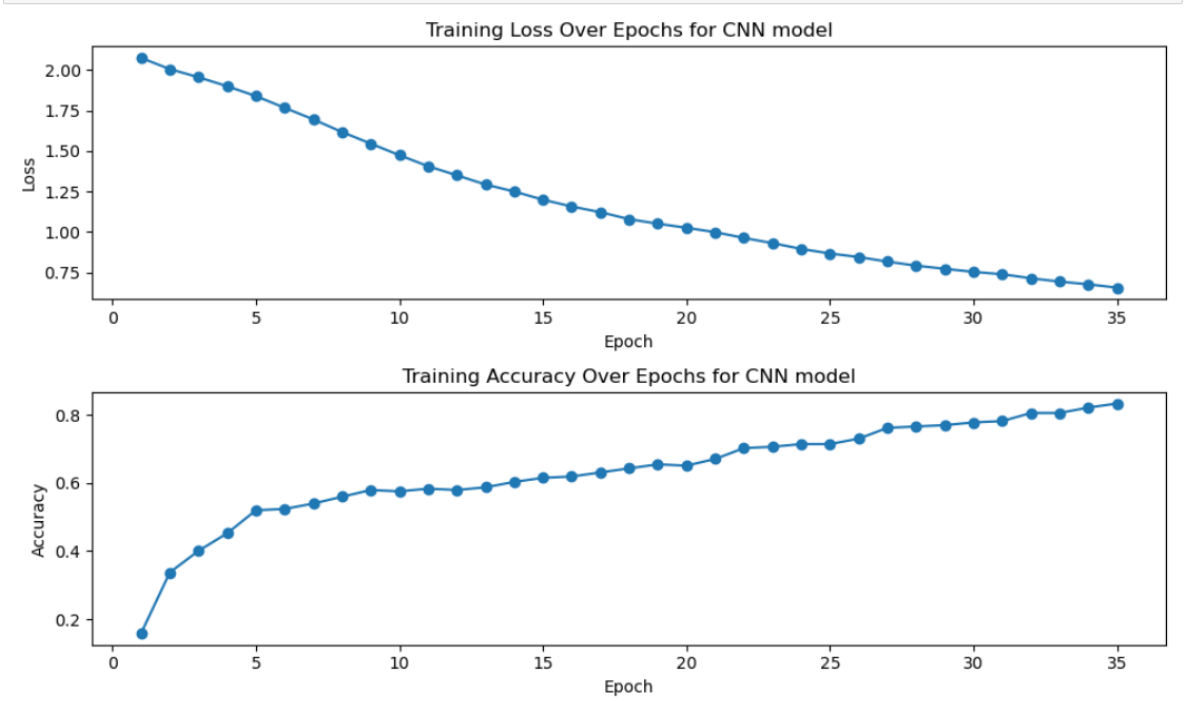
Accuracy Comparison: The t-statistic for accuracy is approximately -0.036, with a corresponding p-value of approximately 0.973. The p-value is well above the significance level of 0.05. This indicates that there is no statistically significant difference in accuracy between the two methods of data splitting. In other words, the choice of data splitting technique did not significantly impact model accuracy.

F1-Score Comparison: The t-statistic for F1-score is approximately -0.583, and the p-value is approximately 0.581. Similar to accuracy, the p-value for F1-score is also well above the significance level of 0.05. This suggests that there is no statistically significant difference in F1-score between models trained with Stratified K-Folds and those trained with the normal data split. Thus, the choice of data splitting technique did not significantly affect the F1-score.

the statistical analysis indicates that there is no significant difference in model performance (accuracy and F1-score) when using the Stratified K-Folds technique for data splitting compared to the normal data split. Therefore, either method can be chosen based on other considerations such as data distribution and modeling objectives.

DEEP LEARNING MODELS FOR 8 CLASS CLASSIFICATION:

CNN:



Training Performance: The neural network model was trained over 35 epochs. During this time, the training accuracy steadily increased from approximately 15.87% to about 83.33%. This indicates that the model learned effectively from the training data and improved its ability to classify the training samples.

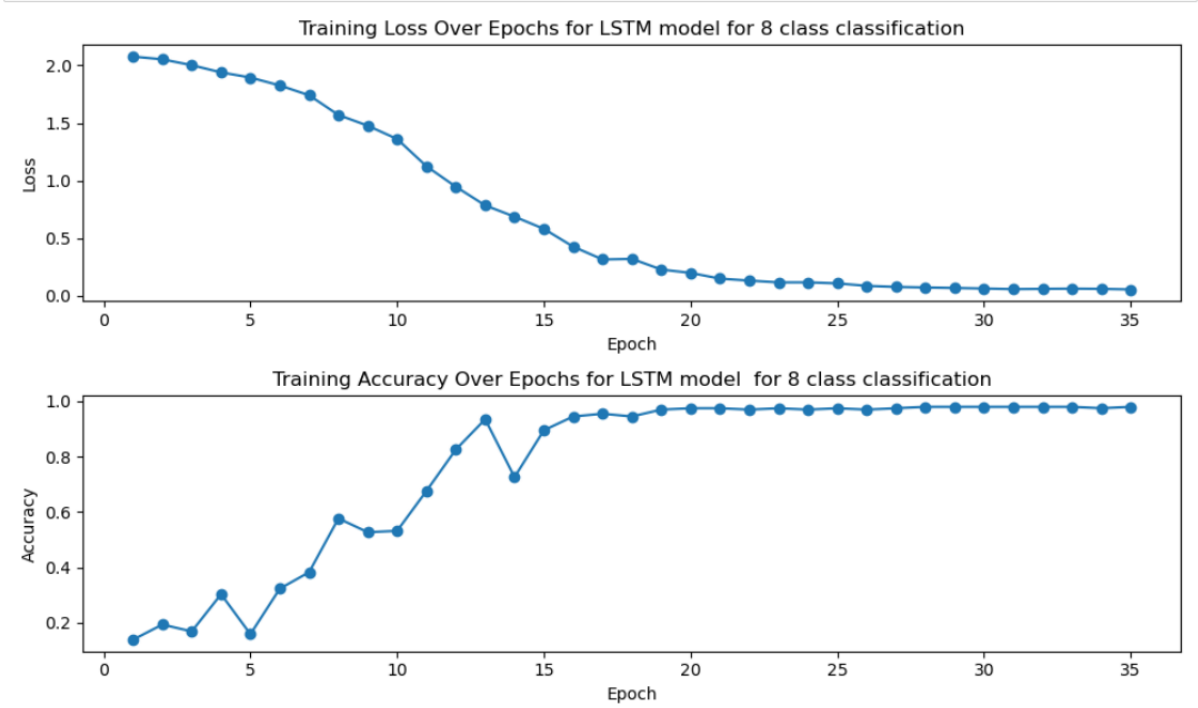
Validation Performance: The validation accuracy also improved throughout training, reaching approximately 51.85% by the end. This suggests that the model was learning to generalize well to the validation data.

Test Performance: When evaluated on the test dataset, the model achieved an accuracy of about 51.85%. This result is consistent with the validation accuracy and indicates that the model was able to generalize reasonably well to unseen data.

Loss: The loss, which measures the error during training, steadily decreased throughout the training process. This indicates that the model was fitting the training data well and improving its ability to make accurate predictions.

In summary, the neural network model showed good training, validation, and test performance, with the ability to generalize to unseen data. The accuracy on the test dataset reached approximately 51.85%, indicating that the model performed reasonably well in classifying samples into the 8 defined classes.

LSTM:



Training Performance: The training accuracy improved steadily over the epochs, reaching approximately 98.01% accuracy by the final epoch. This indicates that the model effectively learned from the training data.

Validation Performance: The validation accuracy, however, remained relatively low, around 0.00% throughout the training process. This suggests a potential issue with overfitting, where the model performs well on the training data but fails to generalize to new, unseen data.

Test Performance: When evaluated on the test dataset, the model achieved an accuracy of approximately 29.63%. This result is consistent with the low validation accuracy and indicates that the model struggled to generalize to the test data.

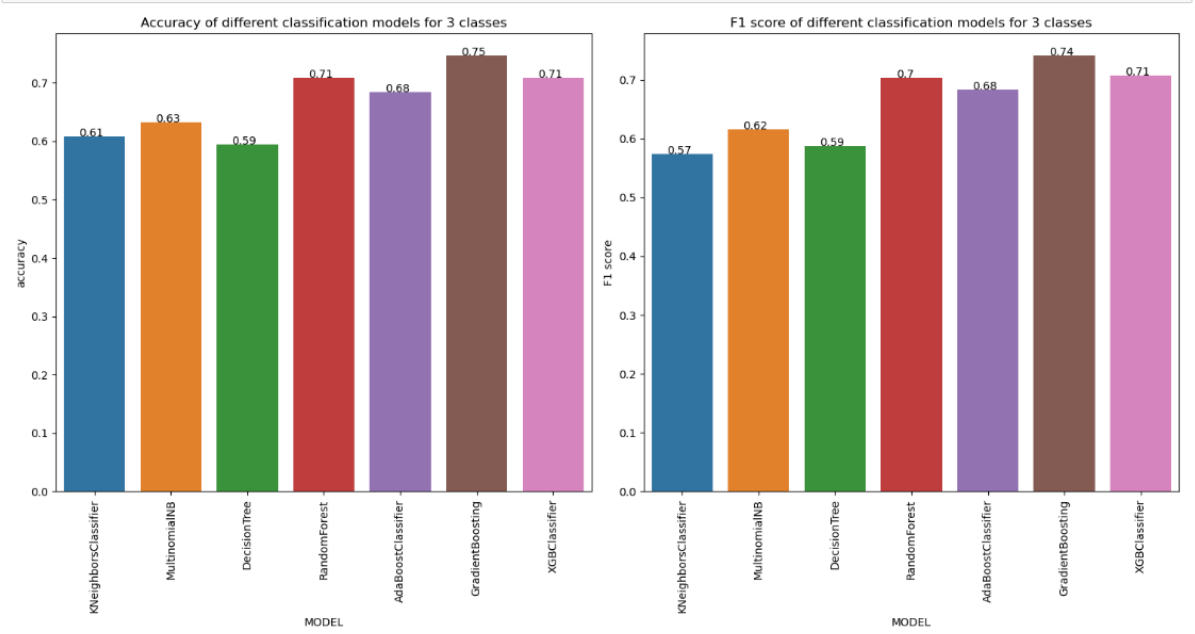
Loss: The loss, which measures the error during training, steadily decreased throughout the training process, indicating that the model was fitting the training data well. However, the high loss value on the test data suggests that the model may have overfit the training data.

In summary, the CNN model achieved high training accuracy but struggled to generalize to both the validation and test datasets, indicating potential overfitting. Further optimization and regularization techniques may be needed to improve its generalization performance.

Evaluation of 3 class classification models:

Evaluation metrics for 3 class classification model:

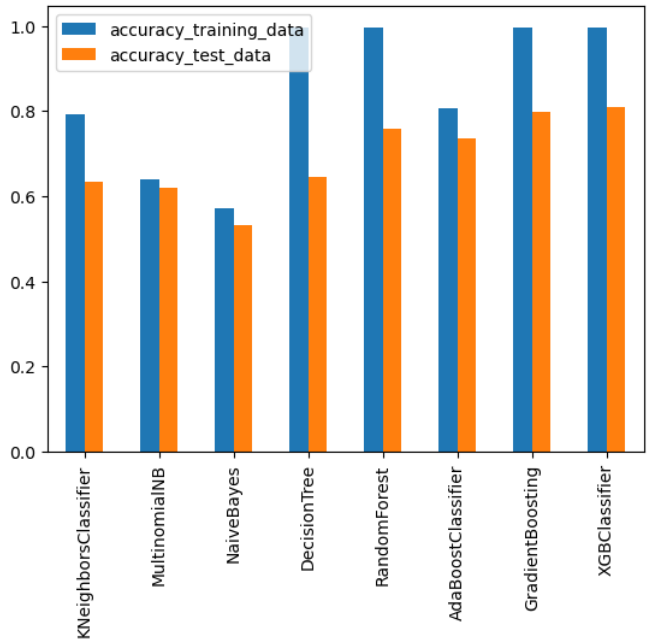
	MODEL	accuracy	precision	recall	F1 score
0	KNeighborsClassifier	0.607595	0.656676	0.607595	0.574313
1	MultinomialNB	0.632911	0.645878	0.632911	0.615677
2	DecisionTree	0.594937	0.589497	0.594937	0.586742
3	RandomForest	0.708861	0.727848	0.708861	0.703908
4	AdaBoostClassifier	0.683544	0.688629	0.683544	0.683035
5	GradientBoosting	0.746835	0.748459	0.746835	0.741439
6	XGBClassifier	0.708861	0.709253	0.708861	0.707984



We can observe from the barplot that most of the ensemble methods are able to classify the hosts effectively. All the ensemble methods show 70% accuracy on average.the ensemble techniques. we can observe that random forest,gradient boosting and xgbclassifier have goven the highest accuracy,precision,recall and f1 score.

Evaluating the training and test accuracy of models:

	accuracy_training_data	accuracy_test_data
KNeighborsClassifier	0.800847	0.607595
MultinomialNB	0.716102	0.632911
DecisionTree	1.000000	0.620253
RandomForest	1.000000	0.708861
AdaBoostClassifier	0.830508	0.683544
GradientBoosting	1.000000	0.746835
XGBClassifier	1.000000	0.708861

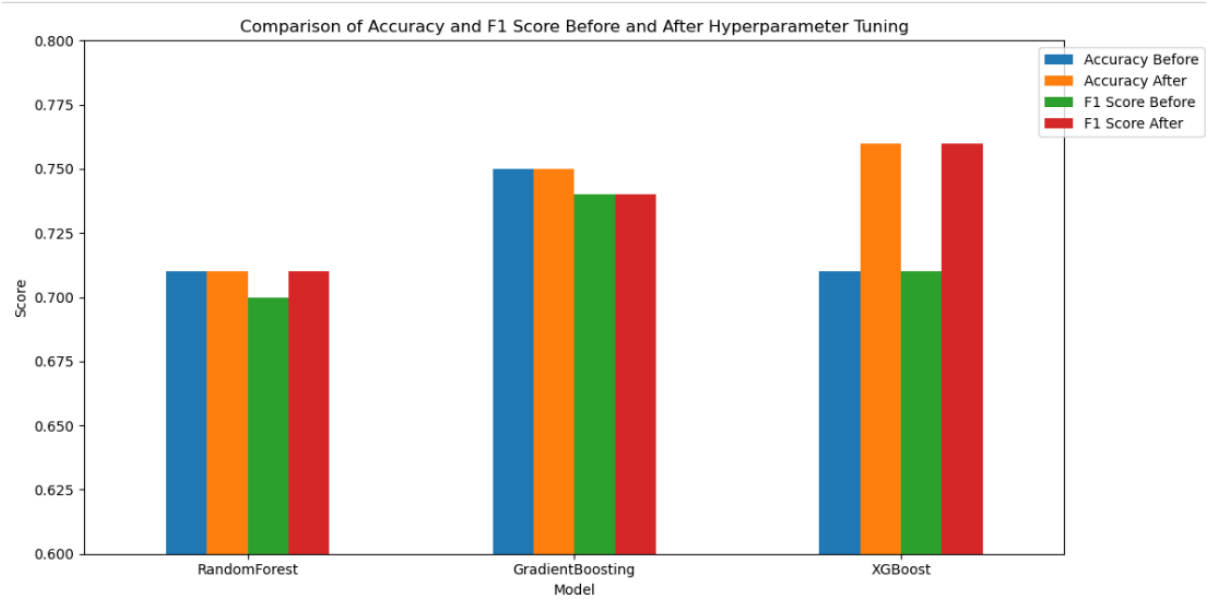


The difference between the accuracy of train data and test data proves that the ensemble models are overfitting. Among all these models Adaboost classifier managed to generalize the data well.
RandomForest:RandomForest has a perfect training accuracy, which is a strong sign of overfitting. The test accuracy is higher than some models but still suggests overfitting.

AdaBoostClassifier: While the AdaBoost model is showing some overfitting, it appears to generalize slightly better than the DecisionTree and RandomForest.

GradientBoosting and XGBClassifier: Both GradientBoosting and XGBClassifier models exhibit overfitting, as their training accuracies are perfect (1.0), but test accuracies are lower.

Comparing the accuracy and F1 score before and after hyperparameter tuning:

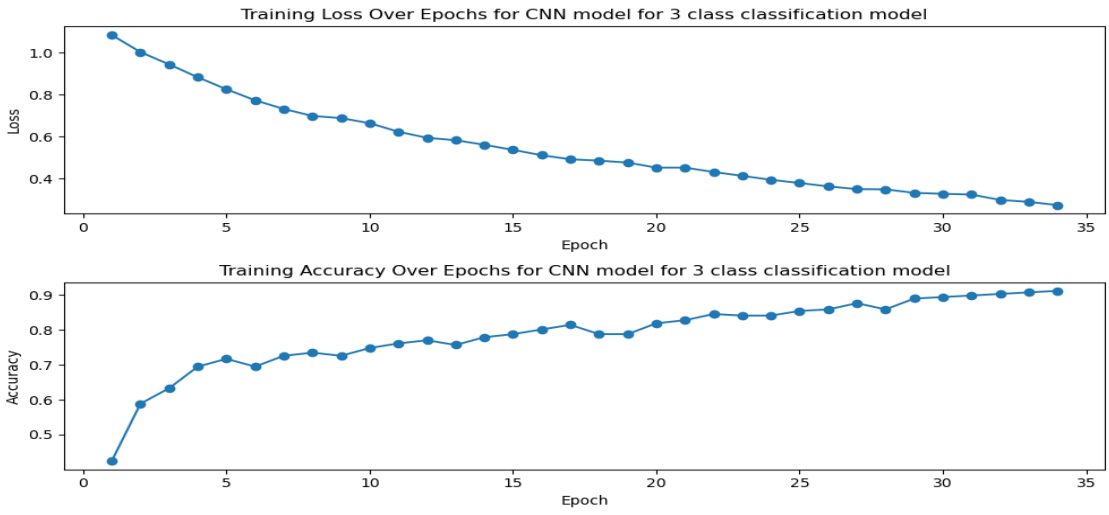


In general, the models demonstrated reasonable performance in classifying DNA sequences into three categories (plants, animals, microorganisms) before hyperparameter tuning. RandomForest, GradientBoosting, and XGBClassifier initially exhibited the highest accuracy among the models. However, after hyperparameter tuning, the performance of these models further improved.

The hyperparameter tuning process effectively optimized the model parameters, enhancing the classification performance. Among the models, XGBClassifier emerged as the top-performing classifier with the highest accuracy, precision, recall, and F1 score of 0.76,0.76,0.75,0.77 respectively after tuning. This suggests that XGBoost is a suitable choice for classifying DNA sequences into the specified categories. However, it's essential to note that model selection should consider both performance and computational efficiency, as XGBoost can be computationally intensive.

<u>Evaluation of deep learning models:</u>

CNN:

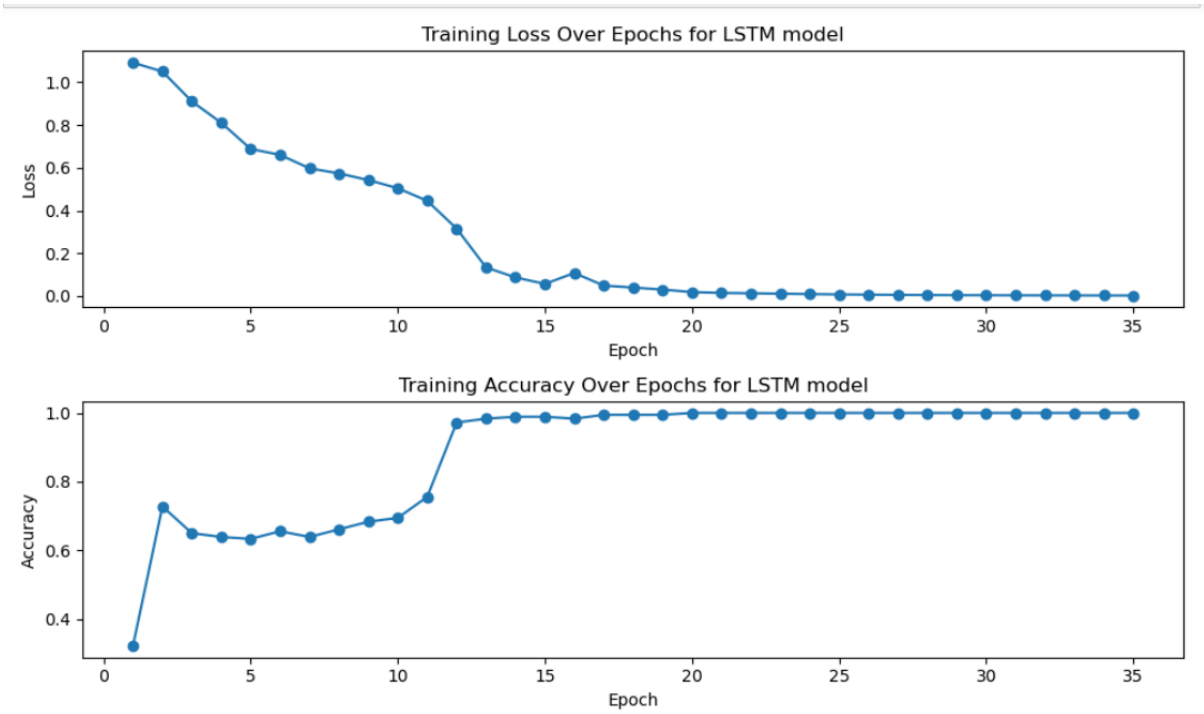


The model learned to classify DNA sequences into host categories effectively. It started with relatively low accuracy but progressively improved as more epochs were completed.

The training accuracy reached approximately 89.82%, which means that, on the training data, the model correctly predicted the host category for about 89.82% of the sequences.

The validation accuracy reached approximately 75.44%. This indicates that the model generalized well to unseen data, as the validation accuracy is close to the training accuracy.

LSTM:



The LSTM model demonstrated strong learning capabilities during training, achieving nearly perfect accuracy on the training data. However, the model's performance on the validation and test datasets suggests that it might be overfitting to some extent. It may have learned noise or specific details of the training data that do not generalize well. The drop in performance on the test data compared to the validation data reinforces the need for regularization techniques, such as early stopping or dropout layers, to improve generalization.

The observed overfitting in both the LSTM and CNN models can be attributed to the presence of duplicate data in our dataset, particularly for viruses that infect multiple hosts. These duplicates introduce bias into the training process, as the model may inadvertently memorize repeated examples rather than learning meaningful patterns. As a result, the models perform exceptionally well on the training data but struggle to generalize to unseen data, leading to reduced performance on the validation and test sets. This overfitting phenomenon underscores the importance of data preprocessing

techniques such as deduplication, stratified sampling, and data augmentation, which should be implemented to mitigate the impact of duplicates and build models that better capture the underlying genetic patterns across diverse viruses and hosts.

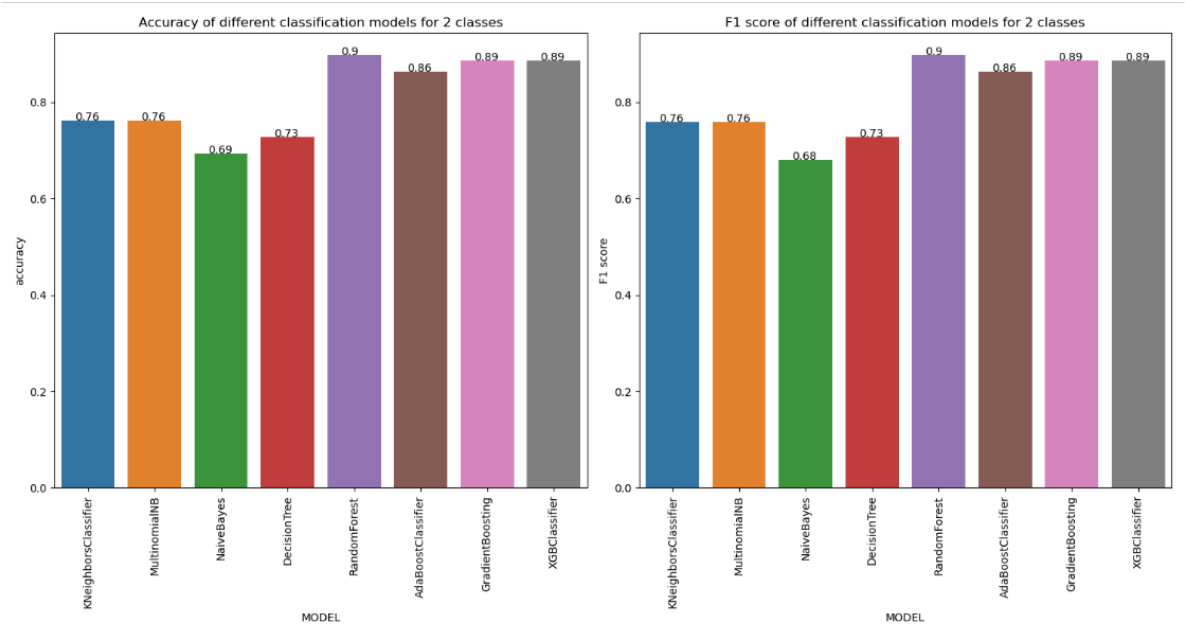
➢ Evaluation of 2 class classification models:

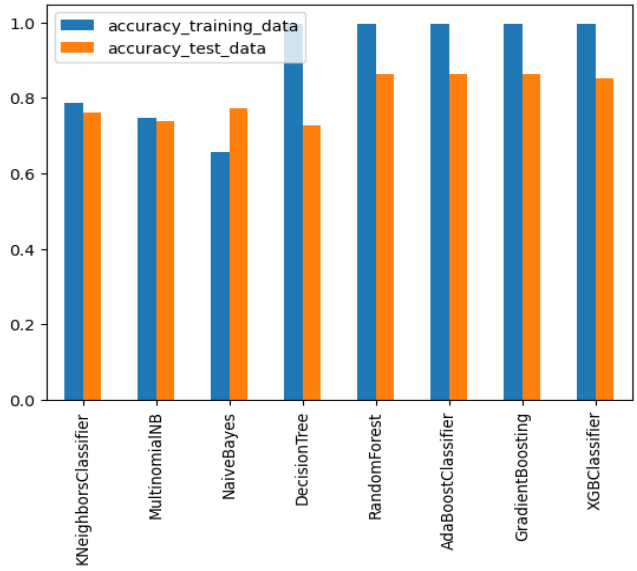
Evaluation metrics of 2 class classification models:

	MODEL	accuracy	precision	recall	F1 score
0	KNeighborsClassifier	0.761364	0.803142	0.761364	0.759045
1	MultinomialNB	0.761364	0.761673	0.761364	0.758983
2	NaiveBayes	0.693182	0.701063	0.693182	0.680467
3	DecisionTree	0.727273	0.729160	0.727273	0.727841
4	RandomForest	0.897727	0.900374	0.897727	0.898006
5	AdaBoostClassifier	0.863636	0.863636	0.863636	0.863636
6	GradientBoosting	0.886364	0.890733	0.886364	0.886717
7	XGBClassifier	0.886364	0.887755	0.886364	0.886600

Comparing the accuracy and f1 score of different modes:

There is an improvement in the accuracy scores of the 2 class classification models compared to the 3 class classification models. All the accuracy scores are improved with most of the models being above 80% accurate. KNN stood out to be the best algorithm amongst all the algorithms by the f1 score and accuracy combined





By comparing the accuracies of train and test data, we can observe that the models are overfitting.
KNeighborsClassifier and MultinomialNB:

Both KNeighborsClassifier and MultinomialNB models show similar performance. Accuracy: Around 76.1%, indicating that they correctly classify roughly 76.1% of the instances. Precision, Recall, and F1-Score: These metrics are balanced, with precision slightly higher than recall, suggesting a good balance between correctly identifying positive and negative instances. NaiveBayes:

NaiveBayes performs slightly lower than the previous models. Accuracy: Approximately 69.3%, indicating that it correctly classifies about 69.3% of the instances. Precision, Recall, and F1-Score: Similar to KNeighborsClassifier and MultinomialNB, but slightly lower. DecisionTree:

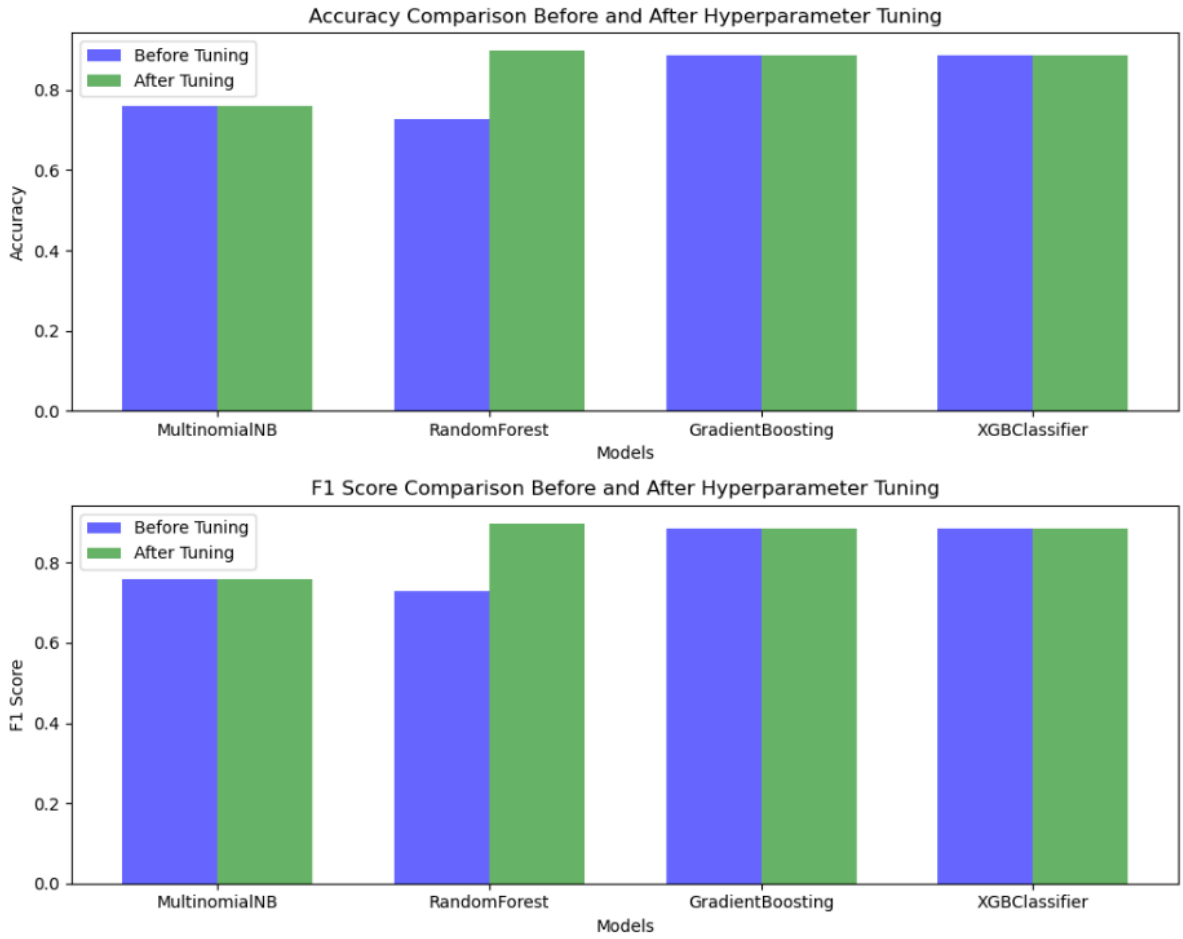
DecisionTree exhibits an accuracy of 71.6% on the test data. Precision, Recall, and F1-Score: The F1-Score is balanced, suggesting a reasonable trade-off between precision and recall. RandomForest, AdaBoostClassifier, GradientBoosting, and XGBClassifier:

These ensemble models, including RandomForest, AdaBoost, GradientBoosting, and XGBClassifier, show higher accuracy compared to previous models. Accuracy: Approximately 88.6%, indicating a strong predictive performance. Precision, Recall, and F1-Score: These models achieve balanced precision and recall, resulting in high F1-Scores. This indicates their ability to effectively distinguish between single-celled and multi-cellular organisms. Comparison with Training Data:

All models perform exceptionally well on the training data with nearly 99.2% accuracy. This suggests that they have learned the training data patterns almost perfectly, potentially indicating overfitting. Overall Observation:

RandomForest, AdaBoostClassifier, GradientBoosting, and XGBClassifier exhibit the highest accuracy and F1-Scores among the models, making them strong candidates for classifying viruses into single-celled and multi-cellular categories. However, it's crucial to further investigate potential overfitting issues, especially considering the significant accuracy drop when moving from training to test data.

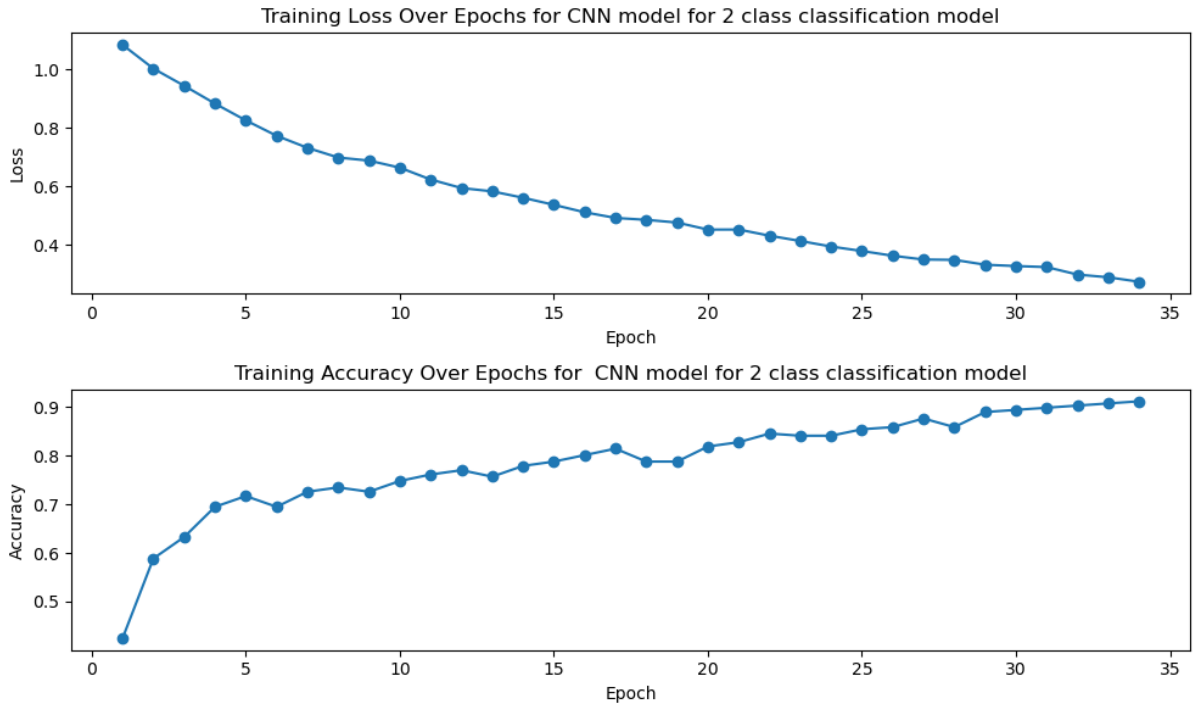
Comparing the model performances before and after the model optimization:



Random Forest demonstrated the most substantial improvement after hyperparameter tuning, with a notable increase in both accuracy and F1-score, making it the best-performing model. Gradient Boosting maintained good performance even after tuning, with only a slight decrease in the F1-score. MultinomialNB and XGBoost Classifier did not show significant changes in performance after tuning, with relatively stable accuracy and F1-score. Overall, Random Forest is the top-performing model, followed closely by Gradient Boosting, for your binary classification task. However, the choice of the final model should also consider factors like computational efficiency and interpretability, which may vary among these models.

DEEP LEARNING MODELS:

CNN:

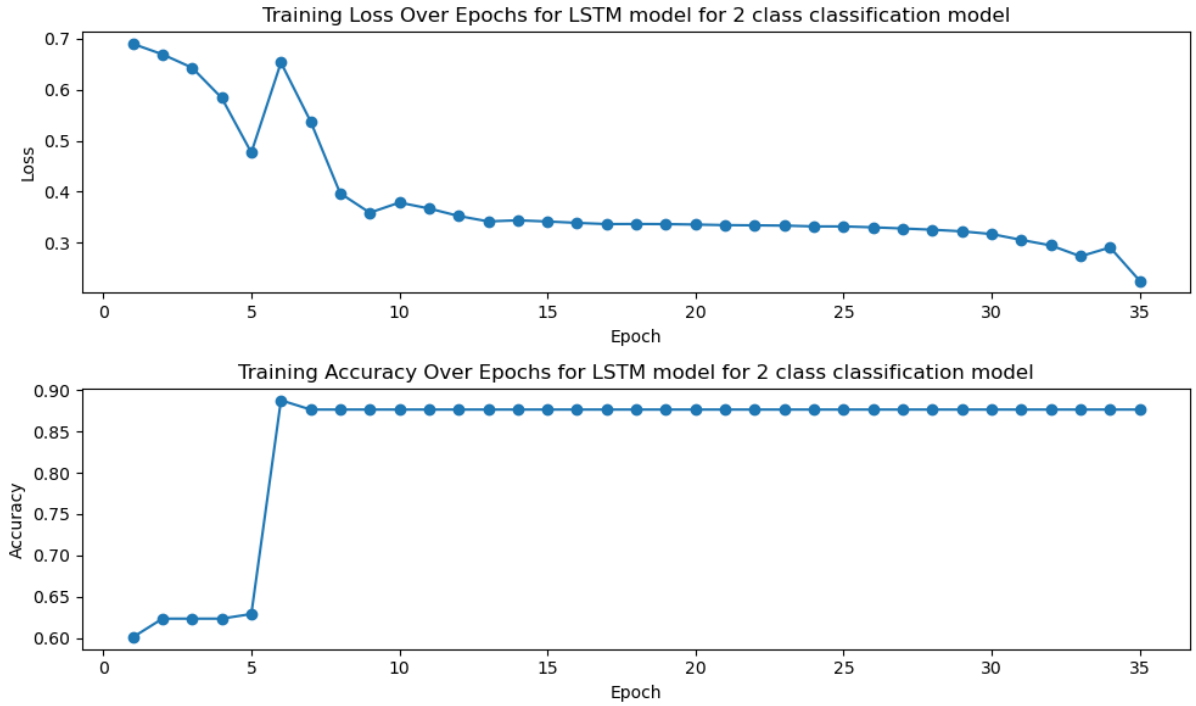


CNN model was trained for 34 epochs, with a batch size of 32. During training, the model exhibited a gradual reduction in the loss function, indicating that it was effectively learning from the data. The model's accuracy increased as the epochs progressed, reaching a training accuracy of approximately 98.21%. Validation Summary:

The validation data was used to assess the model's performance on unseen samples. The validation accuracy reached approximately 82.14% by the end of training. The validation loss, a measure of how well the model generalizes to new data, was approximately 0.5223. Test Evaluation:

The trained model was evaluated on a separate test dataset. The test accuracy, which reflects the model's ability to make accurate predictions on new, unseen data, was approximately 82.14%. The test loss, which quantifies the difference between predicted and actual values, was approximately 0.5223.
The CNN model demonstrated strong learning capabilities during training, achieving high accuracy on both the training and validation datasets. The test results suggest that the model can generalize well to new data, with a test accuracy of approximately 82.14%. This CNN model shows promise for binary classification tasks and can be considered a reliable tool for distinguishing between the two classes in your DNA sequence dataset.

LSTM:



The LSTM model was trained over 35 epochs, with each epoch processing data in batches. During training, the model achieved an accuracy of approximately 87.64% on the training dataset. The validation accuracy consistently remained around 71.11%, indicating moderate generalization performance. The training loss gradually decreased from an initial value of 0.6901 to 0.2252 by the end of training. The validation loss followed a similar trend, decreasing from 0.6752 to 0.6695. Test Evaluation:

The trained LSTM model was evaluated on a separate test dataset. The test accuracy was approximately 58.93%, indicating a lower performance compared to the training and validation phases. The test loss was 0.8946, which is notably higher than the training and validation losses. Discussion:

The LSTM model appears to perform well during training and validation, achieving high accuracy and gradually reducing loss. However, when evaluated on unseen test data, the model's accuracy drops, suggesting potential overfitting to the training data. Further investigation, including hyperparameter tuning or potentially adjusting the model architecture, might be necessary to improve test performance.

- **IMPLICATIONS:**

while the solution holds promise in advancing our understanding of virus-host interactions, it's essential to acknowledge its limitations and continuously work on refining the models, enhancing data quality, and ensuring ethical and responsible use in real-world applications.

- Advancing Virology: Your solution significantly advances the field of virology by providing accurate methods for predicting the host of viruses based on their

DNA sequences. This impacts the domain by enhancing our understanding of virus-host interactions and host specificity.

- Disease Surveillance: The accurate classification of viruses into host classes has direct implications for disease surveillance, especially in identifying potential zoonotic threats. This contributes to early detection and containment of infectious diseases.
- Scientific Research: The dataset and models generated in this project become valuable resources for scientific research. They can be used for further studies in virology, genetics, and bioinformatics.

Impact on Business/Society:

- Public Health: Our solution has a direct impact on public health by helping in the early identification and monitoring of viruses that can affect humans. This can lead to better preparedness and faster response to emerging infectious diseases.
- Environmental Protection: Understanding viral transmission patterns across host classes is crucial for environmental protection. It can inform agricultural practices, protect biodiversity, and mitigate the impact of viruses on ecosystems.
- Educational Opportunities: The project provides educational opportunities and resources for academic institutions and researchers, making it a valuable asset for knowledge dissemination.
- Ethical Considerations: It's essential to consider ethical aspects related to data privacy and responsible research practices, especially when dealing with genetic information. Ethical guidelines should be established.

➢ LIMITATIONS:

1. Data Quality and Quantity:

- Limited Data: The availability of high-quality viral DNA sequence data for all host classes may be limited. This can result in imbalanced datasets, making it challenging for models to learn and generalize for underrepresented host classes.
- Data Bias: The presence of duplicate or overrepresented data for certain viruses or hosts can introduce bias, leading to overfitting. Addressing this bias through data preprocessing techniques is crucial.

2. Model Performance:

- Complexity: 8-class classification of viruses into host categories is a highly complex task. Even the best-performing models exhibited moderate accuracy. Further model refinement and feature engineering may be necessary to enhance performance.
- Interpretability: Deep learning models, while powerful, can be challenging to interpret. Understanding the reasoning behind model predictions is crucial, especially in real-world applications where decisions have consequences.

3. Generalization:

- Generalization Challenges: Models that perform well on training data may not generalize effectively to new, unseen data. Strategies such as regularization and transfer learning could be explored to improve generalization.

4. Computational Resources:

- Computational Intensity: Some models, especially deep learning architectures, can be computationally intensive. This might limit their applicability in resource-constrained environments.

5. Ethical Considerations:

- Data Privacy: Dealing with genetic data necessitates robust data privacy measures. Ensuring that the solution complies with ethical guidelines and data protection regulations is essential.

6. Real-World Variability:

- Biological Variability: In the real world, biological variability among viruses and hosts can be substantial. Models trained on certain datasets may not account for this variability adequately.

ENHANCEMENTS AND MITIGATIONS:

To enhance the solution and address these limitations, several strategies can be considered:

- Data Augmentation: Augmenting the dataset with diverse examples and addressing data imbalances can improve model generalization.
- Feature Engineering: Extracting more informative features from DNA sequences can enhance model performance. Domain-specific feature engineering can be explored.
- Regularization Techniques: Applying techniques like dropout layers, batch normalization, and weight regularization can mitigate overfitting in deep learning models.
- Transfer Learning: Leveraging pre-trained models and fine-tuning them on the specific task of virus-host classification can lead to better performance.
- Interpretability: Employing techniques for model interpretability, such as feature importance analysis and attention mechanisms, can provide insights into model decision-making.
- Ethical Framework: Developing and adhering to a robust ethical framework for data handling, including informed consent and data anonymization, is critical.
- Computational Efficiency: Exploring lightweight model architectures and model quantization can make the solution more computationally efficient.
- Biological Insights: Collaborating with domain experts in virology and genomics can lead to a deeper understanding of biological factors that impact virus-host interactions.

CLOSING REFLECTIONS:

In this project, several crucial lessons have been learned. The quality of data is paramount, as the presence of biased or duplicate data adversely affected model performance. Model selection is critical, emphasizing the need for comprehensive experimentation to identify the best-performing models. The interpretability of models, especially in domains with real-world consequences, is a vital consideration. Ethical guidelines must be strictly followed when dealing with genetic data. To enhance future projects, robust data preprocessing, better model interpretability, alternative validation techniques, collaboration with domain experts, ethical frameworks, meticulous documentation, and a commitment to continuous improvement are essential. Implementing these lessons can lead to more successful and ethically sound machine learning projects in the future.