DNA CLASSIFICATION FOR DETECTING E. COLI VIRUS INFECTION

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

by

KONDA VENKATESWARA REDDY (19BCE7382)

UNDER THE GUIDANCE OF

DR. RAJAMOHAN R

SCHOOL OF COMPUTER SCIENCE ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

DECEMBER 2022

CERTIFICATE

This is to certify that the Capstone Project work titled “DNA CLASSIFICATION FOR DETECTING E. COLI VIRUS INFECTION” that is being submitted by KONDA VENKATESWARA REDDY (19BCE7382) is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. RAJAMOHAN R

Guide

THE THESIS IS SATISFACTORY / UNSATISFACTORY

Internal Examiner **External Examiner**

APPROVED BY

PROGRAM CHAIR

B. Tech. CSE

DEAN

School Of Computer Science & Engineering

ACKNOWLEDGEMENTS

I would like to express my profound gratitude to my guide, Dr. RajaMohan R, for his invaluable guidance, motivation, and support throughout the course of this project. I am immensely grateful for his effort and enthusiasm to help us in every step of the project. His encouragement, suggestions and timely feedback have been extremely helpful in completing this project.

I am also thankful to Saroj Kumar Panigrahy (PhD), Programme Chair, BTech (CSE) and Dr. Madhusudhana Rao N, Dean Academics for providing us with the necessary facilities to complete the project.

Lastly, I would like to thank my college, Vellore Institute of Technology, Amaravati for providing me this opportunity to work on this project.

CONTENTS

1. ABSTRACT

This paper presents an efficient system for detecting the presence of E. coli virus in a DNA sample. An MLP classifier model has been developed using a DNA dataset containing four types of DNA molecules (A, C, G, and T). The dataset is used to train the model, which then is used to classify the DNA sample into E. coli virus or not. The model has been tested and proven to be accurate in identifying the presence or absence of the virus.

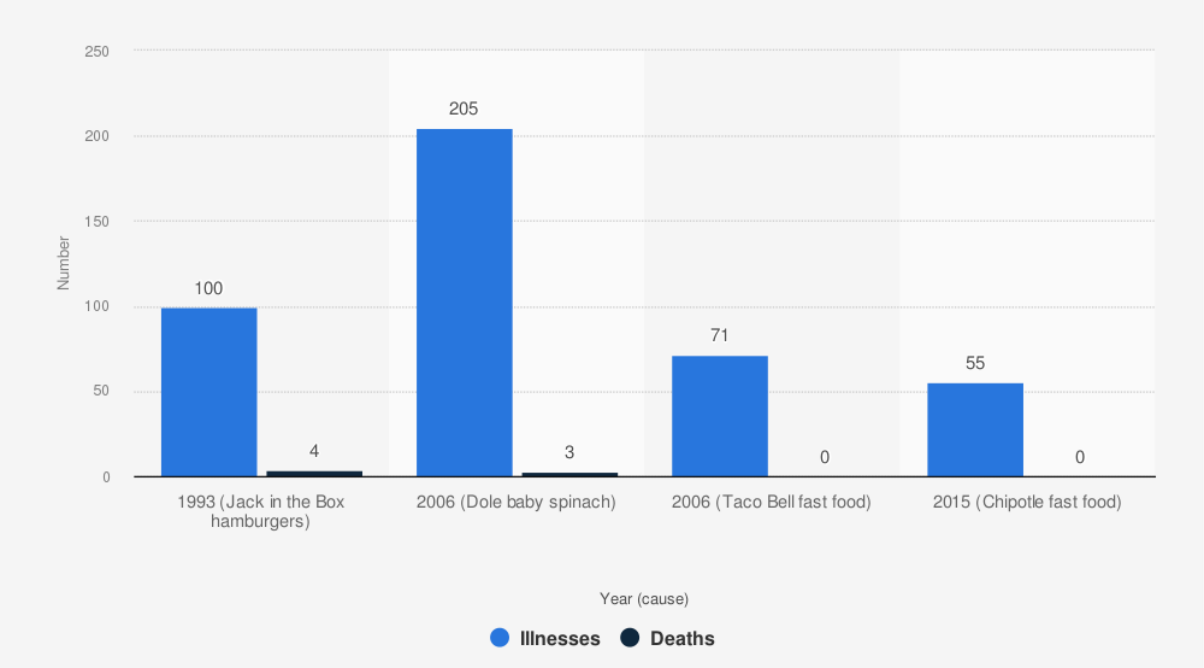
To make the system easily accessible, a website has been developed using Django Framework. The website allows users to input their own DNA samples and get the results of the classification in real-time. Furthermore, the website has been designed with user-friendly features such as a clean interface and simple workflows.

The proposed system offers an efficient and accurate way of identifying the presence of E. coli virus in a DNA sample. The results of the MLP classifier model are used to provide the classification results on the website. The system is easy to use and provides quick results, making it a useful tool for researchers and medical professionals. The system is also useful for educational purposes, allowing students to learn about the technique of virus identification in a DNA sample.

2. BACKGROUND AND OVERVIEW

2.1 - INTRODUCTION

Escherichia coli (E. coli) is a type of bacteria that is commonly found in the intestines of humans and other warm-blooded animals. It is usually harmless, but certain strains of the bacteria can cause illness, especially if it is consumed in contaminated food or water. E. coli is usually transmitted through direct contact with fecal matter, contaminated food or water, or through contact with an infected person. E. coli can cause stomach cramps, vomiting, diarrhea, fever, and dehydration. In rare cases, certain types of E. coli can cause more serious complications, including kidney failure.



Illnesses and deaths from E. coli outbreaks in U.S. from 1993-2015. (Source: Statista 2022)

2. BACKGROUND AND OVERVIEW

2.2 - PROBLEM STATEMENT

The identification and classification of E. coli is a challenging task due to the wide diversity of strains and the complexity of their genetic sequences. DNA promoter sequences are short pieces of DNA from which transcription of a genome begins, and the classification of these sequences has become complicated. Present Traditional methods of identification and classification are slow and can be prone to errors.

2.3 - PROPOSED SOLUTION

To address this problem, a machine learning model needs to be developed to quickly and accurately identify and classify E. coli strains. Such a model would be of great benefit, as it would reduce the time and effort needed to identify and classify E. coli strains. Additionally, it would ensure the accuracy and reliability of the results, which is essential for the proper management of E. coli. The development of such a model would be of great help in improving the current methods of identification and classification of E. coli, and would help to ensure the safety of our food and water supplies.

2. BACKGROUND AND OVERVIEW

2.4 - PROJECT OBJECTIVE

The purpose of this project is to design a machine learning model that can accurately identify and classify Escherichia coli (E. coli) using DNA promoter sequences as input. To do this, a dataset of DNA sequences will be collected from the UCI Machine Learning Repository website and pre-processed for use in machine learning. The data will then be classified using a deep learning model, such as a Multi-Layer Perceptron (MLP) classifier, which can learn from the data and make accurate classifications. This model will be tested and evaluated to ensure that it is able to accurately identify and classify E. coli and can be used to further research in the field of microbiology.

2.5 - PROJECT JUSTIFICATION

The development of a Deep learning model to identify and classify E. coli is essential for improving public health and reducing the spread of this virus. A successful model would be able to quickly and accurately diagnose individuals who may be at risk of infection. In addition, this project could have a wide range of applications, including the development of a new therapeutic treatment for E. coli-related infections. With accurate detection, individuals can receive timely treatment and reduce the spread of this virus. Furthermore, the development of a machine learning model to detect and classify E. coli could potentially save numerous lives and improve public health outcomes. With this in mind, this project could be an important step in the fight against E. coli and other infectious diseases.

3. METHODOLOGY AND FRAMEWORK

In order to create a successful deep learning model, it is necessary to go through the following steps.

3.1 - COLLECT DATA

Data gathering is the first step of any Machine Learning or Deep Learning project. Without proper data, no model can give accurate results. Data can be gathered from various sources like websites, databases, surveys, etc. The data should be properly cleaned and pre-processed before training the model. It is important to select the right features for the model which will help in better predictions.

3.2 - EXPLORE AND PRE-PROCESS DATA

Once the data is collected, it is important to explore it and understand its characteristics. This helps in identifying the irrelevant data or any outliers which can affect the accuracy of the model. The data should be pre-processed accordingly such as normalizing, scaling, removing the outliers, etc. After pre-processing the data, the features should be selected and the data should be split into train and test sets.

3.3 - CHOOSE A MODEL ARCHITECTURE

The model architecture is an important part of any Deep Learning project. It should be chosen carefully based on the type and size of the data. The model should be capable of handling the data and giving accurate results. There are a variety of architectures available. The choice of architecture will depend on the problem that needs to be solved and the data that is available.

3. METHODOLOGY AND FRAMEWORK

3.4 - TRAIN AND EVALUATE THE MODEL

After the model architecture is chosen, the model should be trained by using various techniques like batch training, cross-validation, etc. The model should be evaluated by using different metrics such as confusion matrix, accuracy, precision, recall, etc. This will help in understanding the performance of the model and can be used to further improve the model.

3.5 – DEPLOY MODEL

Finally, the model can be deployed in production. This involves setting up the infrastructure for the model, such as a web server, and making sure that the model is able to handle the traffic that comes through it. Once this is done, the model can be used to make predictions or generate insights from the data.

CONCLUSION

Creating a successful deep learning model involves several steps, starting with data gathering and ending with deployment. Each step is essential for creating an accurate, efficient, and effective model. By following this process, organizations can create deep learning models that are tailored to their specific needs and applications.

4. DATASET AND FEATURES

Title of Database: E. coli promoter gene sequences (DNA)

Dataset URL: promoters.data

Sources:

(a) Creators:

- promoter instances: Harley(CHARLEY@McMaster.CA) and R. Reynolds
- non-promoter instances: M. Noordewier (non-promoters derived from work of lab of Prof. Tom Record, University of Wisconsin Biochemistry Department)

(b) Donor: M. Noordewier and J. Shavlik, {noordewi, shavlik} @cs.wisc.edu

(c) Date received: 6/30/90

Past Usage:

(a) Biological:

Harley, C. and Reynolds, R. 1987.
"Analysis of E. Coli Promoter Sequences."
Nucleic Acids Research, 15:2343-2361.

(b) Machine Learning:

Towell, G., Shavlik, J. and Noordewier, M. 1990.
"Refinement of Approximate Domain Theories by Knowledge-Based Artificial Neural Networks." In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90).

4. DATASET AND FEATURES

Number of Instances: 106

Number of Attributes: 59

- class (positive or negative)
- instance name
- 57 sequential nucleotide ("base-pair") positions

Missing Attribute Values: none

Class Distribution: 50% (53 positive instances, 53 negative instances)

Sample Dataset Overview:

```
[ ]  data.head(10)
```

	Class	id	Sequence
0	+	S10	\t\ttactagcaatacgcttgcgttcggtggttaagtatgtataat...
1	+	AMPC	\t\ttgctatcctgacagttgtcacgctgattggtgtcgttacaat...
2	+	AROH	\t\tgtactagagaactagtgcattagcttatttttttgttatcat...
3	+	DEOP2	\taattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaa...
4	+	LEU1_TRNA	\ttcgataattaactattgacgaaaagctgaaaaccactagaatgc...
5	+	MALEFG	\taggggcaaggaggatggaaagaggttgccgtataaagaaactag...
6	+	MALK	\t\tcaggggtggaggatttaagccatctcctgatgacgcatagt...
7	+	RECA	\t\ttttctacaaaacacttgatactgtatgagcatacagtataat...
8	+	RPOB	\t\tcgacttaatatactgcgacaggacgtccgttctgtgtaaatc...
9	+	RRNAB_P1	\tttttaaatttcctcttgtcaggccggaataactccctataatgc...

5. DATA PREPROCESSING

5.1 - LIBRARIES USED

Before embarking on a deep learning project, it is essential to import the necessary libraries. The libraries will provide the necessary tools for data preprocessing, feature engineering, model building, and model evaluation.

The libraries used in this project are as follows:

1. NumPy: NumPy is a library for scientific computing with tools for creating and manipulating arrays, and performing math operations.

2. Matplotlib: Matplotlib is a library which offers functions for plotting data in 2D and 3D graphs, as well as histograms, bar charts, and scatter plots.

3. Pandas: Pandas is a library for data manipulation and analysis, providing data structures and tools for transforming data.

4. TensorFlow: TensorFlow is an open-source library for machine learning that allows users to create, train, and deploy powerful models.

5. Seaborn: Seaborn is a library for data visualization. It provides functions for creating attractive statistical graphics.

6. Scikit-learn: Scikit-learn is a library for Machine Learning. It provides functions for training, evaluating and pre-processing.

7. Pickle: Pickle is a library for serializing and de-serializing Python objects. It allows objects to be stored and retrieved from a file.

5. DATA PREPROCESSING

5.2 - REFINING AND STRUCTURING DATA

The dataset contains three columns: class (positive or negative), instance name (id) and 57 sequential nucleotide ("base-pair") positions. In order to build our custom dataset, the following preprocessing steps were taken.

1. Dropping the unnecessary columns from the dataset
2. Generating a list of DNA sequences
3. Removing the tab from each sequence
4. Appending the class assignment to each sequence
5. Converting the class and sequence objects into a pandas data frame.

5.2.1 - DROPPING THE UNNECESSARY COLUMNS FROM THE DATASET

The project requires only working with nucleotide sequences and class labels, so the instance name column (id) can be removed. This was done by using the pandas library and the '.drop()' function. The function was used to remove the unnecessary columns from the dataset.

5.2.2 - GENERATING A LIST OF DNA SEQUENCES

The DNA sequences were initially presented in an object format, which is not suitable for further operations. To make the data more suitable, a list of DNA sequences was generated from the dataset using the pandas library and the 'loc()' function. The list data type is more efficient for further operations, as it allows for quick access of individual elements, iteration over elements, and manipulation of the list. Converting the DNA sequences from an object to a list also allows for more efficient data processing and manipulation.

5. DATA PREPROCESSING

5.2.3 - REMOVING THE TAB FROM EACH SEQUENCE

The tab character is a non-printing character used to separate fields in a text file. When importing data into a program such as Python, the tab character can lead to incorrect parsing. To avoid this, it is necessary to remove the tab characters from the dataset. This was done by creating a loop which iterated through each sequence and removed the tab characters.

5.2.4 - APPENDING THE CLASS ASSIGNMENT TO EACH SEQUENCE

After removing the tab character from each sequence, the class assignment was appended to each sequence. This was done by creating a loop which iterated through each sequence and appended the class assignment to the end of the sequence.

5.2.5 - CONVERTING CLASS AND SEQUENCE OBJECTS INTO A DATA FRAME.

Converting the class and sequence objects into a dataframe can be useful for data analysis, visualizations, and machine learning tasks. Dataframes are an efficient way to store data as they have a tabular format that can be easily manipulated. This was done by using the pandas library and the '.DataFrame()' function. The function was used to convert the dictionary object into a dataframe. The resulting dataframe contained 16 nucleotide sequences and a class label associated with each sequence.

After structuring the data, we have to encode this categorical data into numerical values. It can be done using One-Hot Encoding.

5.3 - ONE HOT ENCODING

Categorical variables typically contain string values, which most machine learning algorithms cannot process. To use these variables in the algorithm, the strings must be replaced with numerical values through a process known as categorical variable encoding. This allows the algorithms to interpret the values and use them in the model.

One-Hot encoding is a popular method of categorical variable encoding. It is simple to implement and does not introduce additional bias into the model. This process involves creating a new dummy binary variable for each of the categories of the original categorical variable. The dummy variables are then assigned a value of 0 or 1 depending on whether they are present or absent in the data.

For instance, if the DNA sequence is A, C, G, and T, then the one hot encoding of the sequence would be [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], and [0, 0, 0, 1], respectively.

DNA
A
C
G
T

One-Hot Encoding →

A	C	G	T
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

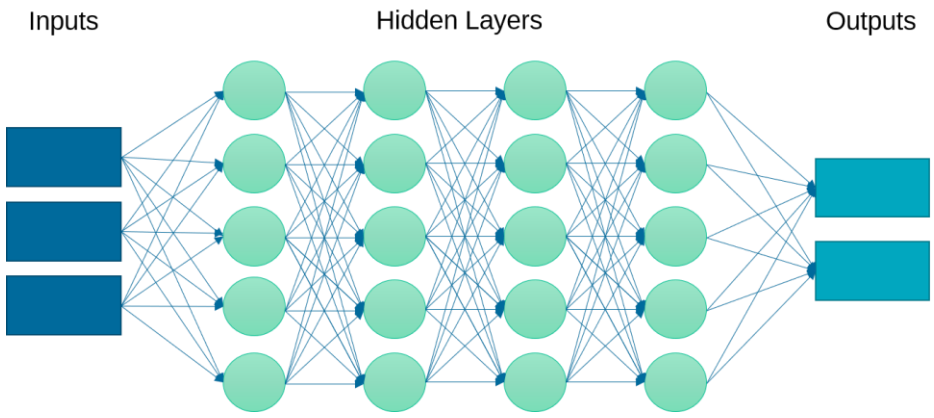
This process is useful because it reduces the dimensionality of the data while still retaining the information present in the original variable.

6. MODEL ARCHITECTURE

In deep learning, an architecture is a combination of layers and parameters that are connected to create a neural network. The architecture specifies how data is passed through the network and how it is processed and modified as it passes through each layer. Different architectures can be used to solve different tasks, and in this project, an MLP Classifier architecture is being used for classification purposes.

6.1 - MLP CLASSIFIER

MLP stands for Multilayer Perceptron, which is a type of artificial neural network. It consists of an input layer, one or more hidden layers, and an output layer. Each layer is made up of interconnected 'nodes', which contain an activation function that determines the output of the node. The connections between nodes are represented by weights, which can be adjusted based on the output of the network. The architecture of an MLP is typically feed-forward, meaning that the information flows in one direction from the input layer to the output layer, with no cycles or loops.



6. MODEL ARCHITECTURE

6.2 - TRAIN AND TEST SETS

when working on deep learning projects, it is common to separate the original dataset into two distinct groups: the training dataset and the test dataset. The training dataset is used to build the model, while the test dataset is used to evaluate the model's performance and the ability to generalize to new or unseen data. This process is essential for correctly assessing the model's accuracy and effectiveness.

For splitting the dataset, we can use the train_test_split function of scikit-learn. The bellow line of code can be used to split dataset:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=1)
```

In the first line of the above code, we have imported the train_test_split function from the sklearn library.

In the second line, we have used four variables, which are

- x_train: It is used to represent features for the training data
- x_test: It is used to represent features for testing data
- y_train: It is used to represent dependent variables for training data
- y_test: It is used to represent independent variable for testing data

In the train_test_split() function, we have passed four parameters. Which first two are for arrays of data, and test_size is for specifying the size of the test set. The test_size may be .5, .3, or .2, which tells the dividing ratio of training and testing sets. The last parameter, random_state, is used to set a seed for a random generator so that you always get the same result.

6. MODEL ARCHITECTURE

6.3 - BUILDING THE MLP CLASSIFIER

Finally, we will build the Multi-layer Perceptron classifier.

```
#Importing MLPClassifier
from sklearn.neural_network import MLPClassifier
```

```
#Initializing the MLPClassifier
model=MLPClassifier(hidden_layer_sizes=(150,100,50),max_iter=300,
activation='relu', solver='adam', random_state=1)
```

- hidden_layer_sizes : This parameter allows us to set the number of layers and the number of nodes we wish to have in the Neural Network Classifier. Each element in the tuple represents the number of nodes at the ith position where i is the index of the tuple. Thus the length of tuple denotes the total number of hidden layers in the network.
- max_iter: It denotes the number of epochs.
- activation: The activation function for the hidden layers.
- solver: This parameter specifies the algorithm for weight optimization across the nodes.
- random_state: The parameter allows to set a seed for reproducing the same results

After initializing we can now give the data to train the Neural Network.

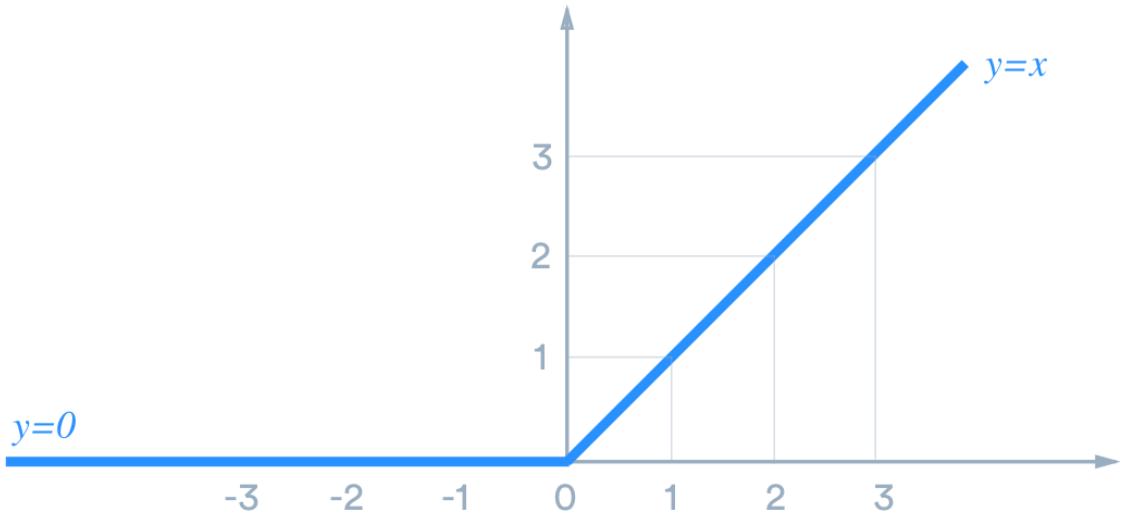
```
#Fitting the training data to the network
model.fit(X_train, Y_train)
```

6. MODEL ARCHITECTURE

6.3.1 - ACTIVATION FUNCTION: ReLU

An activation function is a mathematical equation that determines the output of a neural network. It maps the input signal to an output signal and is used to determine whether a neuron should be activated or not. It is a nonlinear transformation that is applied to the weighted sum of all the inputs from the previous layer to the current layer.

ReLU (Rectified Linear Unit) is a type of activation function used in neural networks. It is a non-linear function that is used to transform a linear input signal into a non-linear output signal. It is the most commonly used activation function and is defined as: f(x) = max(0, x). ReLU is simple to compute, as it requires only one computation step. It is also computationally efficient and has been found to improve the performance of deep learning networks.



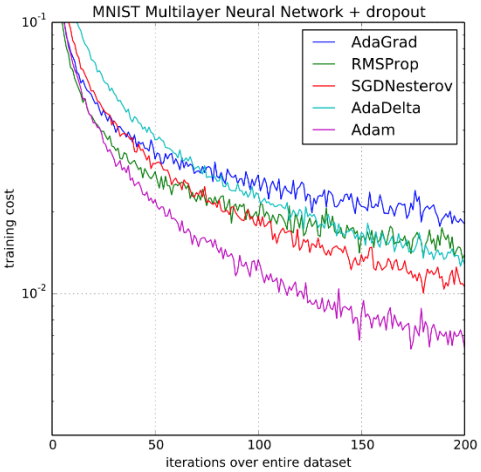
Graphical Representation of ReLU function : f(x) = max(0, x)

6. MODEL ARCHITECTURE

6.3.2 - SOLVER: ADAM OPTIMIZER

A solver is an algorithm used to optimize the parameters of a machine learning model. It is used to adjust the weights of a model in order to minimize the loss function and maximize the accuracy. The type of solver used depends on the type of model being used, as well as the size and complexity of the data set.

Adam (Adaptive Moment Estimation) is a popular optimization algorithm used in deep learning. It is a variant of SGD (Stochastic Gradient Descent) and combines the benefits of both Momentum and RMSprop. Adam uses adaptive learning rates and works well with large datasets and non-convex optimization problems. It is a popular choice for many deep learning tasks and is well-suited for problems with large data sets and multiple local minima.



comparison of Adam to other optimization algorithms training a MLP.

Taken from Adam: A Method for Stochastic Optimization, 2015.

The results of the Adam optimizer are generally better than every other optimization algorithms, have faster computation time, and require fewer parameters for tuning. Because of all that, Adam is recommended as the default optimizer for most of the applications

7. MODEL EVALUATION

In this project, I have used various metrics and techniques to evaluate the performance of the model on the given dataset. These include the accuracy score, Classification Report, confusion matrix, Jaccard Index, Mathews correlation coefficient, dice coefficient, and the loss curve.

7.1 - ACCURACY SCORE

Accuracy score is a metric used to evaluate the performance of a classification model. It is the percentage of correct predictions made by the model. It is calculated by dividing the total number of correct predictions by the total number of predictions made.

Code:

```
from sklearn.metrics import accuracy_score
y_pred = model.predict(x_test)
accuracy_score(y_test, y_pred)
```

7.2 - CLASSIFICATION REPORT

Classification report is a detailed report of the performance of a classification model on a given dataset. It is used to evaluate the precision, recall, f1-score and support of the model. It is calculated by comparing the predicted values with the actual values in the test dataset

Code:

```
from sklearn.metrics import classification_report
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred)
```

7. MODEL EVALUATION

7.3 - CONFUSION MATRIX

Confusion matrix is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with the counts of the correct and incorrect predictions made by the model.

Code:

```
from sklearn.metrics import confusion_matrix
y_pred = model.predict(x_test)
confusion_matrix(y_test, y_pred)
```

7.4 - JACCARD INDEX

The Jaccard index is a statistic used for gauging the similarity and diversity of sample sets. It is measured by the size of the intersection divided by the size of the union of the sample sets. This is a measure of the similarity between two samples and can range from 0 (no similarity) to 1 (complete similarity).

Code:

```
# import the necessary libraries
from sklearn.metrics import jaccard_score

# calculate the Jaccard Index
jaccard_index = jaccard_score(y_test, y_pred)

# print the result
print("The Jaccard Index is:", jaccard_index)
```

7. MODEL EVALUATION

7.5 - MATHEWS CORRELATION COEFFICIENT

The Matthews Correlation Coefficient (MCC) is a measure of the quality of binary classification problems. It takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes.

The MCC is in essence a correlation coefficient value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 an average random prediction and -1 an inverse prediction.

Code:

```
# import the necessary libraries
from sklearn.metrics import matthews_corrcoef

# calculate the mcc
mcc = matthews_corrcoef(y_test, y_pred)

# print the result
print("The matthews correlation coefficient is:", mcc)
```

7.6 - DICE COEFFICIENT

Dice coefficient is a measure of the similarity between two samples. It is calculated by dividing the size of the intersection of two samples by the size of their union. A Dice coefficient of 1 represents a perfect overlap, whereas a coefficient of 0 indicates no overlap.

7. MODEL EVALUATION

Code:

```
import tensorflow as tf
def dice_coef(y_test, y_pred):
    y_test_f = tf.reshape(tf.dtypes.cast(y_test, tf.float32), [-1])
    y_pred_f = tf.reshape(tf.dtypes.cast(y_pred, tf.float32), [-1])
    intersection = tf.reduce_sum(y_test_f * y_pred_f)
    return (2. * intersection + 1.) / (tf.reduce_sum(y_test_f) + tf.reduce_sum(y_pred_f) + 1)

# calculate and print the result
dice = dice_coef(y_test, y_pred)
print("Dice Coefficient: {:.3f}".format(dice))
```

7.7 - LOSS CURVE

A Loss Curve is a chart that shows the decrease in loss over time as a deep learning model is trained. It is used to measure the performance of the model and diagnose problems with the model. This is typically done by plotting the model's training and validation loss values over time.

Code:

```
import matplotlib.pyplot as plt
plt.plot(model.loss_curve_)

# Add labels and title
plt.title("Loss Curve"); plt.xlabel("Epoch"); plt.ylabel("Loss")

# Show the plot
plt.show()
```

8. MODEL DEPLOYMENT

we are now moving to the deployment phase of our Deep Learning Model, and using the Django Framework to create a website for it.

Django is an open-source web framework written in Python. It is used to build complex web applications quickly. It is also suitable for deploying deep learning models as it provides support for custom database models and a wide range of libraries for building interactive web applications.

Steps to Deploy a Deep Learning Model Using Django Version 3.2.0:

1. INSTALL THE DJANGO FRAMEWORK:

First, we need to install the Django framework to deploy a Deep Learning Model. We can install Django using pip by running the following command:
pip install django

2. CREATE A DJANGO PROJECT:

Once the Django framework is installed, we can create a Django project by running the following command: *django-admin startproject myproject*

3. INSTALL TENSORFLOW:

Next, we need to install the Tensorflow library so that we can use it to develop and deploy our Deep Learning model. We can install Tensorflow using pip by running the following command: *pip install tensorflow*

4. CREATE A DEEP LEARNING MODEL:

Once Tensorflow is installed, we can create our Deep Learning model using the Tensorflow library.

8. MODEL DEPLOYMENT

5. TEST THE MODEL:

Once the model is created, we can test it to make sure it is working correctly.

6. CREATE A DJANGO APP:

Once the model is tested, we can create a Django app to deploy the model. We can create a Django app by running the following command:
python manage.py startapp myapp

7. CONNECT THE MODEL TO THE APP:

Next, we need to connect the Deep Learning model to the Django app. We can do this by writing a Python script that connects the model to the app.

8. DEPLOY THE MODEL:

Finally, we can deploy the Deep Learning model using Django. We can do this by running the following command: *python manage.py runserver*

9. RESULTS AND DISCUSSION

The model was evaluated using several evaluation metrics. The results of the evaluation of the model are as follows:

Accuracy: The model achieved an accuracy of 0.9259 on the test set, indicating that it was able to correctly classify 92.59% of the instances.

Classification Report: The precision for class 0 was 1.00, indicating all predictions for class 0 were correct, while the precision for class 1 was 0.83, indicating 83% of the predictions were correct. The recall for class 0 was 0.88, indicating 88% of the instances of class 0 were correctly classified, and the recall for class 1 was 1.00, with all instances of class 1 correctly classified. The F1-score for class 0 was 0.94 and for class 1 was 0.91.

Confusion Matrix: The confusion matrix shows that 15 of the 17 instances of class 0 were correctly classified and all 10 instances of class 1 were correctly classified.

Jaccard Index: The Jaccard Index was 0.83, indicating that 83% of the predictions were correct.

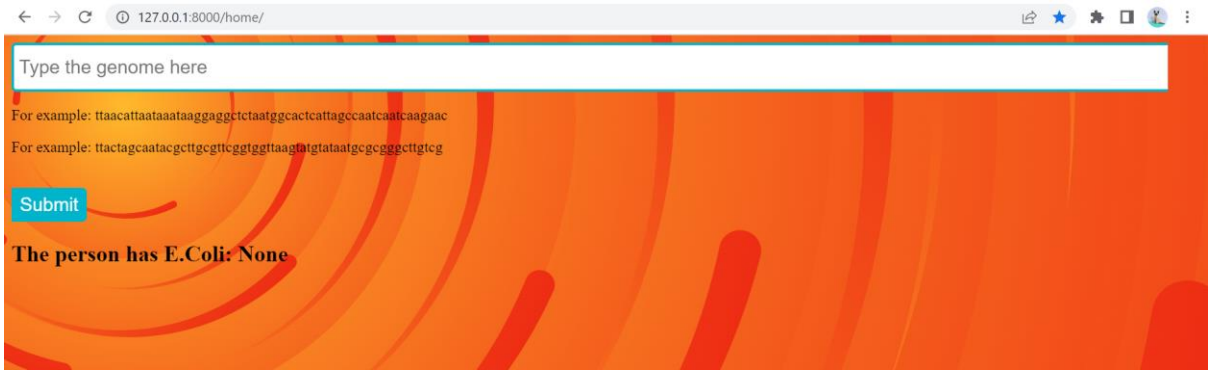
Mathews Correlation Coefficient: The MCC score was 0.86, indicating that the model was able to correctly classify 86% of the instances.

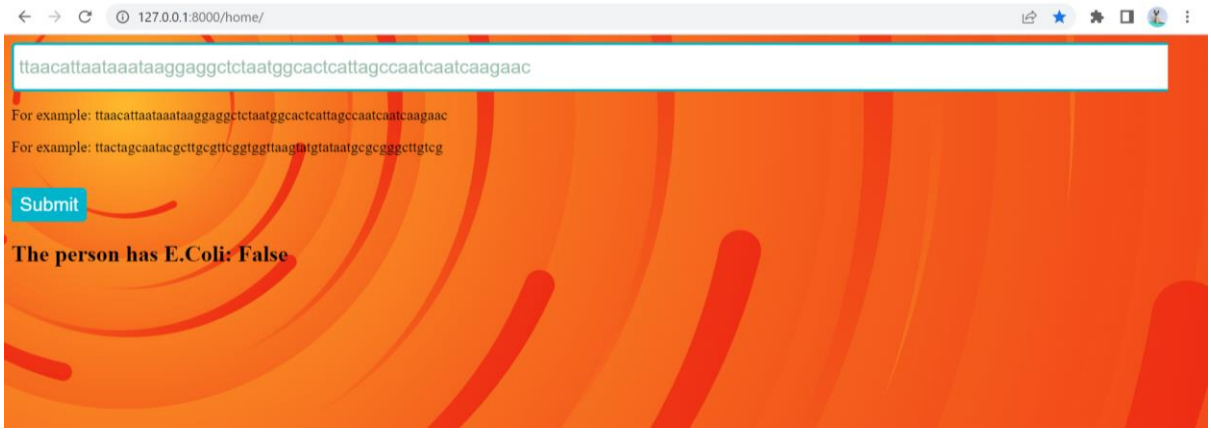
Dice Coefficient: The Dice Coefficient was 0.91, indicating that 91% of the predictions were correct.

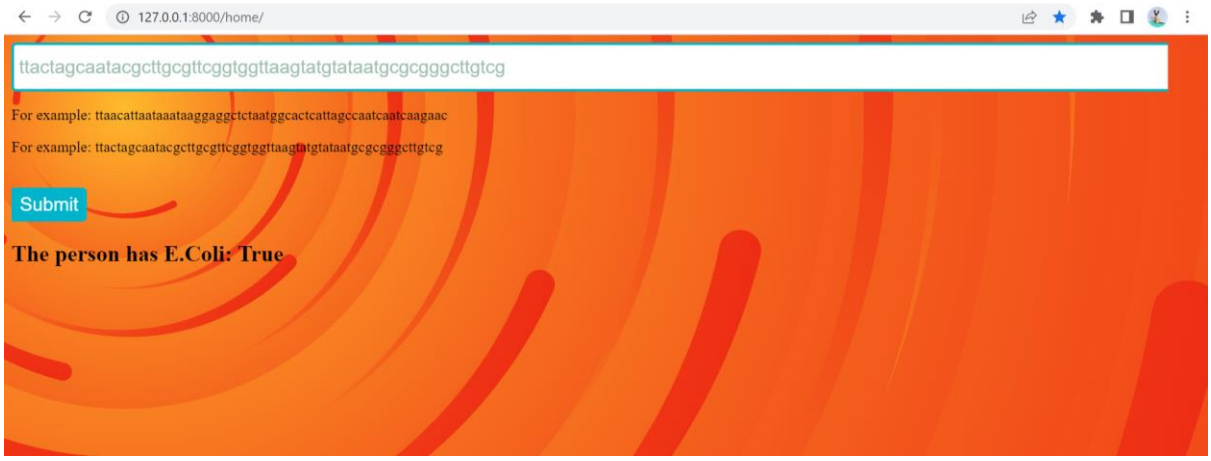
Overall, the model achieved good results and was able to correctly classify a majority of the instances.

9. RESULTS AND DISCUSSION

OUTPUT SCREENSHOTS AFTER DEPLOYING THE MODEL:







10. CONCLUSION AND FUTURE SCOPE

In conclusion, the proposed system offers an efficient and accurate way of identifying the presence of E. coli virus in a DNA sample. The MLP classifier model that has been developed is accurate in classifying the DNA sample into E. coli virus or not. Furthermore, the website developed with user-friendly features allows users to input their own DNA samples and get the results of the classification in real-time. This system is useful for medical purposes, as it can provide quick and accurate results. In addition, it is also useful for educational purposes, allowing students to learn about the technique of virus identification in a DNA sample.

The future of this project could involve expanding its capabilities to identify not only E. coli virus, but other related viruses and bacteria as well. This would allow the system to be more widely used by medical professionals and researchers for the identification and classification of various microorganisms. Furthermore, the system could be further improved by exploring different machine learning and deep learning models to ensure better accuracy and faster classification results. Finally, the system could be adapted to be used in different fields, such as the detection of genetic diseases and the identification of food-borne bacteria.

11. REFERENCES

1. Hensel, M., J. E. Shea, C. Gleeson, M. D. Jones, E. Galton, and D. W. Holden. 1995. Simultaneous identification of bacterial virulence genes by negative selection. Science 269:400-403.
2. Handfield, M., L. J. Brady, A. Progulske-Fox, and J. D. Hillman. 2000. IVIAT: a novel method to identify microbial genes expressed specifically during human infection. Trends Microbiol. 8:336-339.
3. Berg, R., & Doolittle, R. F. (1982). A hierarchical classification of bacterial viruses. Journal of Molecular Evolution, 18(3), 224-229.
4. Chaudhuri, S., & Chaudhuri, S. (2015). Machine learning for microbial genomics. Trends in Genetics, 31(9), 481-490.
5. Fiehn, O. (2002). Metabolomics—the link between genotypes and phenotypes. Plant Molecular Biology, 48(1-2), 155-171.
6. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
7. UCI Machine Learning Repository (2020). E. Coli promoter data set. Retrieved from https://archive.ics.uci.edu/ml/machine-learning-databases/molecular-biology/promoter-gene-sequences/
8. Wirawan, D. (2018). Machine learning for medical diagnosis and treatment. International Journal of Computer Science & Applications, 15(3), 4-1
9. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC128117/
10. https://www.sciencedirect.com/science/article/abs/pii/S0168160500002063?via%3Dihub

12. APPENDIX

```
> from google.colab import drive
> drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

DATA PREPROCESSING

Importing the necessary libraries

```
> import numpy as np
> import matplotlib.pyplot as plt
> import pandas as pd
> import seaborn as sns
> from sklearn.preprocessing import OneHotEncoder
> import pickle
> from sklearn.neural_network import MLPClassifier
> from sklearn.metrics import classification_report, accuracy_score
> from sklearn.model_selection import train_test_split
> from sklearn.metrics import confusion_matrix
```

Loading the Dataset

```
> url = 'https://archive.ics.uci.edu/ml/machine-learning-
databases/molecular-biology/promoter-gene-sequences/promoters.data'
> names = ['Class', 'id', 'Sequence']
> data = pd.read_csv(url, names = names)
```

```
> data.head(4)
```

	Class	id	Sequence
0	+	S10	\t\ttactagcaatacgcttgcgttcggtggttaagtatgtataat...
1	+	AMPC	\t\ttgctatcctgacagttgtcacgctgattggtgtcgttacaat...
2	+	AROH	\t\tgtactagagaactagtgcattagcttatttttttgttatcat...
3	+	DEOP2	\taattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaa...

12. APPENDIX

```
> data.shape
```

```
(106, 3)
```

```
> data.dtypes
```

```
Class       object
id          object
Sequence    object
dtype: object
```

REFINING AND STRUCTURING THE DATA

```
# Dropping Unnecessary Columns
```

```
> classes = data.loc[:,'Class']
> print(classes.value_counts())
```

```
+    53
-    53
Name: Class, dtype: int64
```

```
# generate list of DNA sequence
```

```
> sequence = list(data.loc[:, 'Sequence'])
> sequence[-1]
```

```
'\t\ttaacattaataaataaggaggctctaatggcactcattagccaatcaatcaagaact'
```

```
#Remove tab from each sequence
```

```
> dic = {}
> for i, seq in enumerate(sequence):
>     nucleotides = list(seq)
>     nucleotides = [char for char in nucleotides if char != '\t']
```

12. APPENDIX

```
>    #append class assignment
>    nucleotides.append(classes[i])
>    dic[i] = nucleotides
```

Convert Dict object into dataframe

```
> df = pd.DataFrame(dic)
> df.head()
```

0	1	2	3	4	5	6	7	8	9	...	96	97
	98	99	100	101	102	103	104	105				
0	t	t	g	a	t	a	c	t	c	t	...	c
	c	t	a	g	c	g	c	c	t			
1	a	g	t	a	c	g	a	t	g	t	...	c
	g	a	g	a	c	t	g	t	a			
2	c	c	a	t	g	g	g	t	a	t	...	g
	c	t	a	g	t	a	c	c	a			
3	t	t	c	t	a	g	g	c	c	t	...	a
	t	g	g	a	c	t	g	g	c			
4	a	a	t	g	t	g	g	t	t	a	...	g
	a	a	g	g	a	t	a	t	a			

5 rows × 106 columns

transpose dataframe into correct format

```
> df = df.transpose()
> df.head()
```

Rename the 57th column as it is our classes

```
> df.rename(columns = {57:'Class'}, inplace = True)
```

12. APPENDIX

```
> temp = df.copy(deep=True)
> temp = temp.drop(['Class'], axis = 1)
> temp.head()
```

0	1	2	3	4	5	6	7	8	9	...	47	48
	49	50	51	52	53	54	55	56				
0	t	a	c	t	a	g	c	a	a	t	...	g
	g	c	t	t	g	t	c	g	t			
1	t	g	c	t	a	t	c	c	t	g	...	g
	c	a	t	c	g	c	c	a	a			
2	g	t	a	c	t	a	g	a	g	a	...	c
	c	a	c	c	c	g	g	c	g			
3	a	a	t	t	g	t	g	a	t	g	...	t
	a	a	c	a	a	a	c	t	c			
4	t	c	g	a	t	a	a	t	t	a	...	t
	c	c	g	t	g	g	t	a	g			

5 rows × 57 columns

ONE-HOT ENCODING

Encoding using one-hot encoder:

```
> enc = OneHotEncoder(handle_unknown='ignore')
> enc.fit(temp)
> print(enc.categories_)
> df1 = enc.transform(temp).toarray()
> del temp
```

[array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g',

```
't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object), array(['a', 'c', 'g', 't'], dtype=object)]
```

Saving the one-hot encoder

```
> with open("drive/MyDrive/Dataset/Models/EColi-encoder.pickle", "wb") as
  f: pickle.dump(enc, f)

> df_new = pd.DataFrame(df1)
> df_new.head()
```

	0	1	2	3	4	5	6	7	8	9	...	218	219	220	221	222	223	224	225	226	227
0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	...	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

12. APPENDIX

Fixing the classes column:

```
> df["Class"] = df["Class"].replace(to_replace =["+"], value =1)
> df["Class"] = df["Class"].replace(to_replace =["-"], value =0)
> df_new["Classes"] = df['Class']
> df_new.head()
```

	0	1	2	3	4	5	6	7	8	9	...	219	220
	221	222	223	224	225	226	227	Classes					
0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.0	
	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1				
1	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.0	
	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1				
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	
	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1				
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	
	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1				
4	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	
	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1				

5 rows × 229 columns

MODEL ARCHITECTURE

Train and Test sets

```
y = df_new['Classes'].values
x = df_new.drop(['Classes'], axis = 1).values

#define a seed for reproducibility
seed = 1

# Splitting data into training and testing data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = seed)
```

12. APPENDIX

BUILDING THE MLP CLASSIFIER

```
> model = MLPClassifier(hidden_layer_sizes=(150,100,50),
   max_iter=300,activation = 'relu',solver='adam',random_state=1)
> model.fit(x_train, y_train)
> print(model.score(x_train, y_train))
```

1.0

MODEL EVALUATION

accuracy score

```
> y_pred = model.predict(x_test)
> accuracy_score(y_test, y_pred)
```

0.9259259259259259

classification report

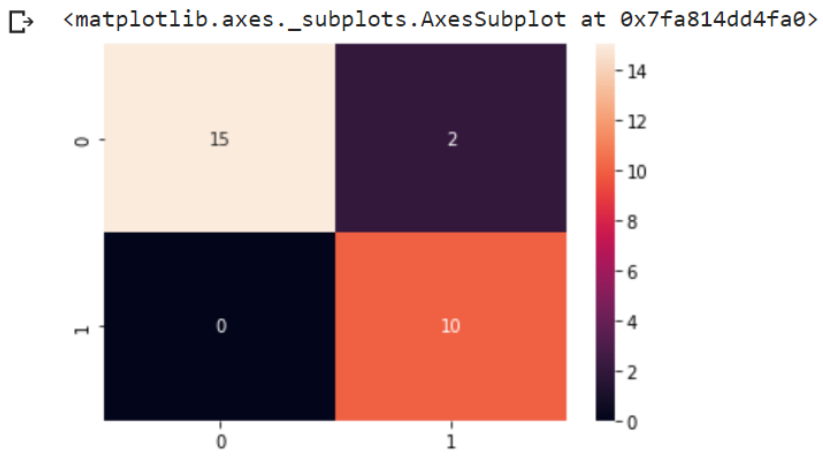
```
> y_pred = model.predict(x_test)
> print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.88	0.94	17
1	0.83	1.00	0.91	10
accuracy			0.93	27
macro avg	0.92	0.94	0.92	27
weighted avg	0.94	0.93	0.93	27

12. APPENDIX

confusion matrix

> cm = confusion_matrix(y_test, y_pred)
> sns.heatmap(cm, annot = True)



Jaccard Index

> from sklearn.metrics import jaccard_score
> jaccard_index = jaccard_score(y_test, y_pred)
> print("The Jaccard Index is:", jaccard_index)

The Jaccard Index is: 0.8333333333333334

Mathews Correlation Coefficient

> from sklearn.metrics import matthews_corrcoef
> mcc = matthews_corrcoef(y_test, y_pred)
> print("The matthews correlation coefficient is:", mcc)

The matthews correlation coefficient is: 0.8574929257125441

12. APPENDIX

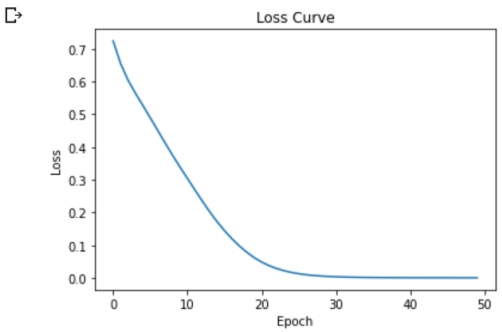
Dice Coefficient

```
> import tensorflow as tf

> def dice_coef(y_test, y_pred):
    y_test_f = tf.reshape(tf.dtypes.cast(y_test, tf.float32), [-1])
    y_pred_f = tf.reshape(tf.dtypes.cast(y_pred, tf.float32), [-1])
    intersection = tf.reduce_sum(y_test_f * y_pred_f)
    return (2. * intersection + 1.) / (tf.reduce_sum(y_test_f) +
    tf.reduce_sum(y_pred_f) + 1)

> dice = dice_coef(y_test, y_pred)
> print("Dice Coefficient: {:.3f}".format(dice))
```

```
Dice Coefficient: 0.913
```

Loss Curve

```
plt.plot(model.loss_curve_)
plt.title("Loss Curve"); plt.xlabel("Epoch"); plt.ylabel("Loss")
plt.show()
```




```
# save the model to disk
filename = 'drive/MyDrive/Dataset/Models/E-Coli_model.pickle'
pickle.dump(model, open(filename, 'wb'))
```

12. APPENDIX

RESULTS

```
# Checking For E. coli(True Case)
> genome = "ttactagcaatacgcttgcgttcggtggttaagtatgtataatgcgcgggcttgtcg"
> genome_list = list(genome)
> print(genome_list)
> df_test = pd.DataFrame(genome_list)
> df_test = df_test.transpose()
```

['t', 't', 'a', 'c', 't', 'a', 'g', 'c', 'a', 'a', 't', 'a', 'c', 'g', 'c', 't', 't', 'g', 'c', 'g', 't', 't', 'c', 'g', 'g', 't', 'g', 'g', 't', 't', 'a', 'a', 'g', 't', 'a', 't', 'g', 't', 'a', 't', 'a', 'a', 't', 'g', 'c', 'g', 'c', 'g', 'g', 'g', 'c', 't', 't', 'g', 't', 'c', 'g']

```
> encoder = pickle.load(open("drive/MyDrive/Dataset/Models/EColi-
  encoder.pickle", 'rb'))
> data_test = encoder.transform(df_test).toarray()
> print(model.predict(data_test))
```

[1]

```
# Checking For E. coli(FALSE Case)
> genome = "ttaacattaataaataaggaggctctaatggcactcattagccaatcaatcaagaac"
> genome_list = list(genome)
> print(genome_list)
> df_test = pd.DataFrame(genome_list)
> df_test = df_test.transpose()
```

['t', 't', 'a', 'a', 'c', 'a', 't', 't', 'a', 'a', 't', 'a', 'a', 'a', 't', 'a', 'a', 'g', 'g', 'a', 'g', 'g', 'c', 't', 'c', 't', 'a', 'a', 't', 'g', 'g', 'c', 'a', 'c', 't', 'c', 'a', 't', 't', 'a', 'g', 'c', 'c', 'a', 'a', 't', 'c', 'a', 'a', 't', 'c', 'a', 'a', 'g', 'a', 'a', 'c']

```
> encoder = pickle.load(open("drive/MyDrive/Dataset/Models/EColi-
  encoder.pickle", 'rb'))
> data_test = encoder.transform(df_test).toarray()
> print(model.predict(data_test))
```

[0]