TASK FAILURE PREDICTION IN CLOUD DATA CENTERS USING DEEP LEARNING

A Project Report submitted in partial fulfillment of the requirements for

the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

221910306022	**K. DIVYA TEJA VENKAT**
221910306044	**P. SURENDRA BABU**
221910306046	**T. SAI PAVAN**
221910306057	**V. TANISHKA**

Under the esteemed guidance

of

Mr. Y. PRAKASH BABU

(Assistant Professor)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GITAM

(Deemed to be University)

HYDERABAD - 502329

APRIL - 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled "**Task Failure Prediction in Data Centers Using Deep Learning**" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Registration No(s).	Name(s)	Signature(s)
221910306022	K. Divya Teja Venkat	
221910306044	P. Surendra Babu	
221910306046	T. Sai Pavan	
221910306057	V. Tanishka	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled "**Task Failure Prediction Data Centers Using Deep Learning**" is a bonafide record of work carried out by **K. Divya Teja Venkat (221910306022), P. Surendra Babu (221910306044), T. Sai Pavan (221910306046), V. Tanishka (221910306057)** students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	**Project Coordinator**	**Head of the Department**
Mr. Y. Prakash Babu	**Dr. S. Aparna**	**Dr.K.S. Sudeep**
Assistant Professor	Assistant Professor	Associate Professor and HOD
Department of CSE	Department of CSE	Department of CSE

ACKNOWLEDGEMENT

The Major Project would not have been successful without the help of several people. We would like to thank the personalities who were part of project in numerous ways, those who gave us outstanding support from the birth of the Project.

We are extremely thankful to our honorable Pro-Vice Chancellor, **Prof. D. Sambasiva Rao** for providing necessary infrastructure and resources for the accomplishment of this project.

We are very much obliged to our beloved **Dr. K.S. Sudeep**, Associate Professor, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this project and encouragement in completion of this seminar.

We hereby wish to express our deep sense of gratitude to **Dr. S. Aparna**, Project Coordinator, Department of Computer Science and Engineering, School of Technology and to our beloved guide, **Mr. Y. Prakash Babu**, Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by him for the success of the project.

We are also thankful to all the staff members of Computer Science and engineering department who have cooperated in making this project a success.

We would like to express our gratitude towards our parents and friends for their constant love and support. Without their blessings and contribution, this would not have been possible.

K. DIVYA TEJA VENKAT (221910306022)

P. SURENDRA BABU (221910306044)

T.SAI PAVAN (221910306046)

V. TANISHKA (221910306057)

TABLE OF CONTENTS

1. ABSTRACT

A large-scale cloud data center needs to provide high service reliability and availability with low failure occurrence probability. However, current large-scale cloud data centers still face high failure rates due to many reasons such as hardware and software failures, which often result in task and job failures. Such failures can severely reduce the reliability of cloud services and also occupy huge number of resources to recover the service from failures.

Therefore, it is important to predict task or job failures before occurrence with high accuracy to avoid unexpected wastage. Many machines learning and deep learning-based methods have been proposed for the task or job failure prediction by analyzing past system message logs and identifying the relationship between the data and the failures.

In order to further improve the failure prediction accuracy of the previous machine learning and deep learning-based methods, in this paper, we propose a failure prediction algorithm based on different linear models to identify task and job failures in the cloud. The goal of Non-Linear models is to predict whether the tasks and jobs are failed or completed with high accuracy.

2. INTRODUCTION

Nowadays, cloud computing service has been widely used because it provides high reliability, resource saving and also on-demand services. The cloud data centers include processors, memory units, disk drives, networking devices, and various types of sensors that support many applications (i.e., jobs) from users. The users can send requests such as store data and run applications to the cloud. Each cloud data center is composed of physical machines (PMs) and each PM can support a set of virtual machines (VMs). The tasks that are sent from users are processed in each VM. Such a large-scale cloud data centercan host hundreds of thousands of servers which often run tons of applications and receive work requestsevery second from users all over the world. A cloud data center with such heterogeneity and intensive workloads may sometimes be vulnerable to different types of failures (e.g., hardware, software, disk failures). Take software failures as an example, Yahoo Inc. and Microsoft's search engine, Bing, crashed for 20 mins in January 2015, which cost about $9000 per minute to reboot the system. Previous research found that hardware failure, especially disk failure, is a major contributing factor to the outages of cloud services. These many different types of failures will lead to the application running failures. Thus, accurate prediction for the occurrence of application failures beforehand can improve the efficiency of recovering the failure and application running.

2.1 SOFTWARE REQUIREMENTS

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

Platform – In computing, a platform describes some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer's architecture, operating system, or programming languages and their runtime libraries.

Operating system is one of the first requirements mentioned when defining system requirements (software). Software may not be compatible with different versions of the same line of operating systems, although some measure of backward compatibility is often maintained. For example, most software designed for Microsoft Windows XP does not run on Microsoft Windows 98, although the

converse is not always true. Similarly, software designed using newer features of Linux Kernel v2.6 generally does not run or compile properly (or at all) on Linux distributions using Kernel v2.2 or v2.4.

APIs and drivers – Software making extensive use of special hardware devices, like high-end display adapters, needs special API or newer device drivers. A good example is DirectX, which is a collection of APIs for handling tasks related to multimedia, especially game programming, on Microsoft platforms.

Web browser – Most web applications and software depending heavily on Internet technologies make use of the default browser installed on system. Microsoft Internet Explorer is a frequent choiceof software running on Microsoft Windows, which makes use of ActiveX controls, despite their vulnerabilities.

1. Jupyter Notebook
2. Python IDE (Python 3.7)

2.2 HARDWARE REQUIREMENTS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following subsections discuss the various aspects of hardware requirements.

Architecture – All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture. See also a list of common operating systems andtheir supporting architectures.

Processing power – The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture defines processing power as the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored. This definition of power is often erroneous, as AMD Athlon and Intel Pentium CPUs at similar clock speed often have different throughput speeds. Intel Pentium CPUs have enjoyed a considerable degree of popularity, and are often mentioned in this category.

Memory – All software, when run, resides in the random-access memory (RAM) of a computer. Memory requirements are defined after considering demands of the application, operating system, supporting software and files, and other running processes. Optimal performance of other unrelated software running on a multi-tasking computer system is also considered when defining this requirement.

Secondary storage – Hard-disk requirements vary, depending on the size of software installation, temporary files created and maintained while installing or running the software, and possible use of swap space (if RAM is insufficient).

Display adapter – Software requiring a better than average computer graphics display, like graphics editors and high-end games, often define high-end display adapters in the system requirements.

Peripherals – Some software applications need to make extensive and/or special use of some peripherals, demanding the higher performance or functionality of such peripherals. Such peripherals include CD-ROM drives, keyboards, pointing devices, network devices, etc.

1. Operating System: Windows 7 and above
2. Processor: i3 and above
3. Ram: 8GB and above Hard Disk: 1TB

3. LITERATURE SURVEY

3.1 MODELING THE IMPACT OF CORRELATED FAILURES ON WORKLOAD RELIABILITYIN DATA CENTERS

Mina (et.al,) In this paper, the authors presented a statistical reliability model and an approximation method for calculating task reliability, formulated the scheduling problem as an optimization problem, and presented a scheduling algorithm. They investigated the effectiveness of the algorithm using an analytical approach and cluster simulations. They achieved an accuracy of 91%. The paper identified related failures caused by power outages and network component failures in jobs that run multiple copies of the same job as a major issue in large data centers. Future scope could involve exploring the application of the presented algorithm to real-world scenarios and improving the model's accuracy further. Additionally, it could involve investigating the impact of other sources of error on workload reliability in data centers [1].

3.2 FAILURE PREDICTION OF DATA CENTERS USING TIME SERIES AND FAULT TREEANALYSIS

Thanyalak (et.al,) In this paper, the authors propose a framework for failure prediction in online data centers using two methods: ARMA and Fault Tree Analysis. The authors achieved an accuracy of 97% in their experiments on a simulated cluster built on the Simi's platform. The paper focuses on hardware failure and highlights the potential negative impact of poorly managed outages on system performance. The authors believe that their framework is viable and adaptable for use in future data centers. Additionally, the paper mentions the potential benefits of encrypted storage as a replication alternative, as well as the drawbacks of the reconstruction of unavailable data due to network congestion. The paper suggests using additional memory or limiting the encoding parameters as potential solutions. Future research could explore these solutions further to improve the reliability and efficiency of data center systems [2].

3.3 PARTIAL-PARALLEL-REPAIR (PPR): A DISTRIBUTED TECHNIQUE FOR REPAIRING ERASURE CODED STORAGE

Haoyu (et.al,). The paper proposes a new distributed reconstruction technique called Partial Parallel Repair (PPR). The model achieved an accuracy of 93%. The paper mentions the issue of cascading failures in cloud data centers, where multiple physical machines in one failure domain fail, leading to SLO violations. However, the Cascading Failure Resilience System (CFRS) outperformed other benchmark methods in terms of the number of domain failures, the number of PM failures, and the number of SLO violations. The future scope of the proposed techniques could involve further research and development to improve efficiency and scalability in cloud data centers [4].

4. PROBLEM IDENTIFICATION AND OBJECTIVES

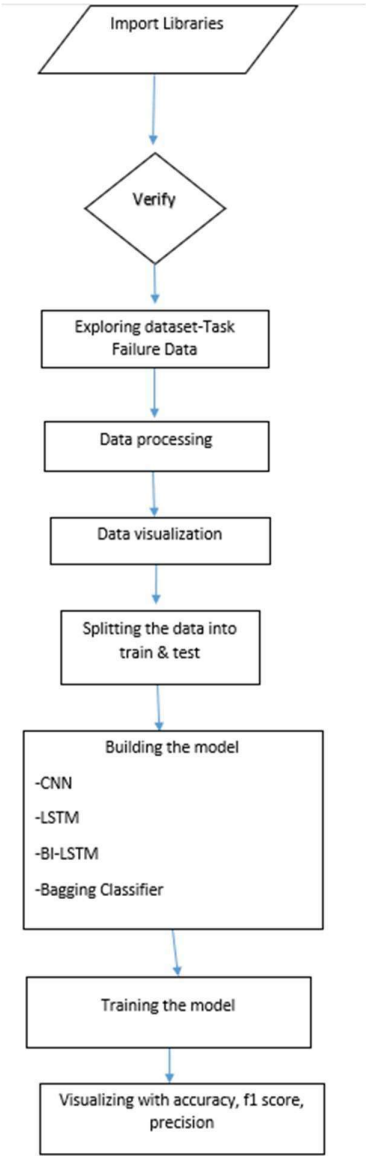
By analyzing past system message logs at cloud data centers identifying the relationship between the data and the failures. In order to further improve the failure prediction accuracy of the previous machine learning and deep learning-based methods, in this project, we propose a failure prediction model to identify task and job failures in the cloud. The goal of our project is to predict whether the tasks and jobs are failed or completed.

Failure Analysis in Cloud Data Centers. Ford etal studied the impact of correlated failures on availability of distributed storage systems for Google clusters. They found that although disk failures can result in permanent data loss, the major reason for most unavailability in the Google cloud storage system is transitory node failures.

They also developed an availability model using Markov chains to reason about past and future availability including the effects of different choices of data replication, data placement and system parameters to provide feedback and recommendations.

5. SYSTEM METHODOLOGY

5.1 SYSTEM ARCHITECTURE:



5.2 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of Object-Oriented tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

5.2.1 SEQUENCE DIAGRAM:

A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequenceof the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

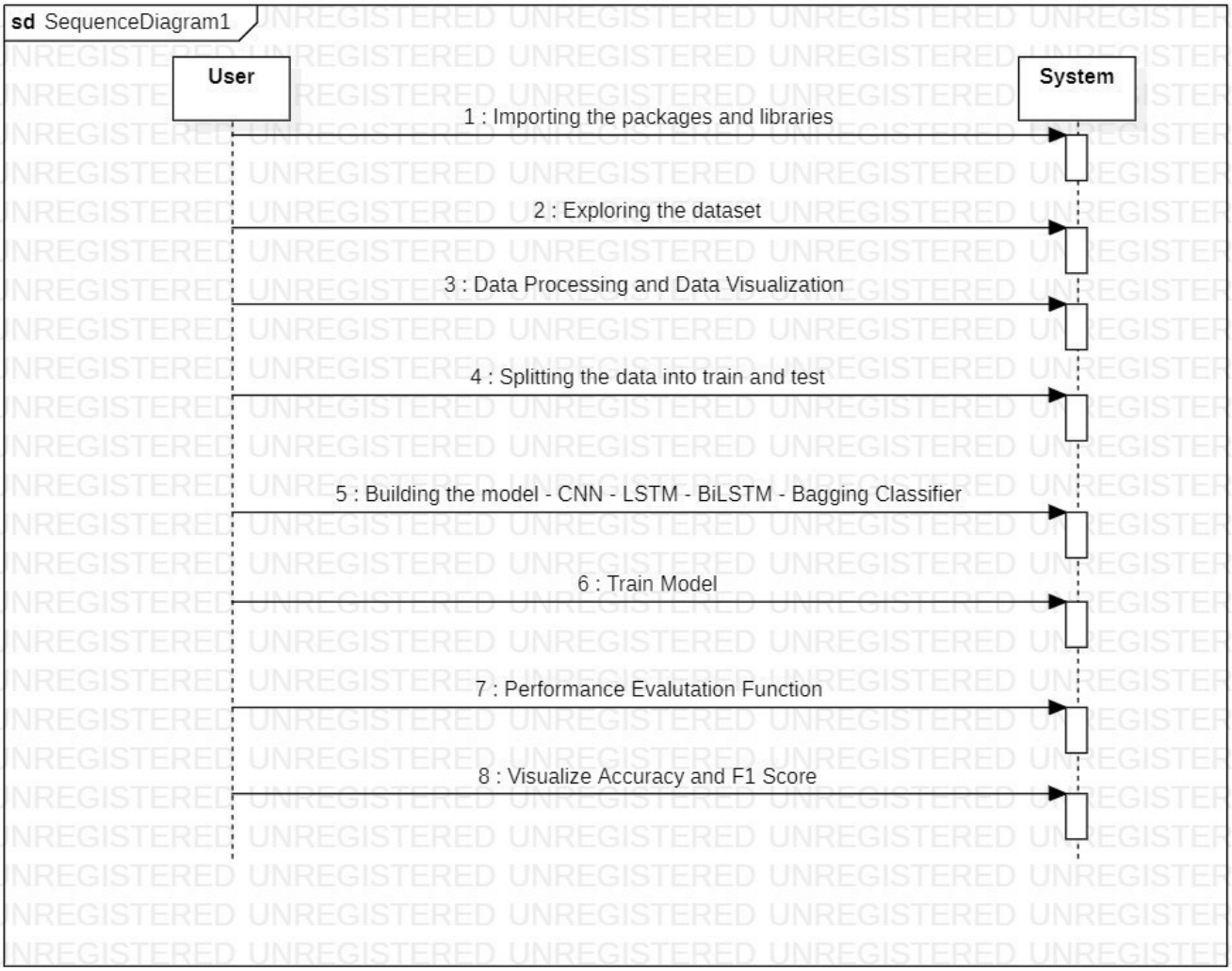


Fig.5.2.1 Sequence diagram

5.2.2 DATA FLOW DIAGRAM:

A data flow diagram shows how information flows through a process or system. This includes data input/output, data storage, and various sub processes through which data moves. DFDs are created using standardized symbols and notations to describe various entities and their relationships.

Data flow diagrams provide visual representations of systems and processes that are difficult to describe in words. You can use these diagrams to outline and improve existing systems or plan the implementation of new systems. Visualizing each element makes it easier to identify inefficiencies and create the best possible system.

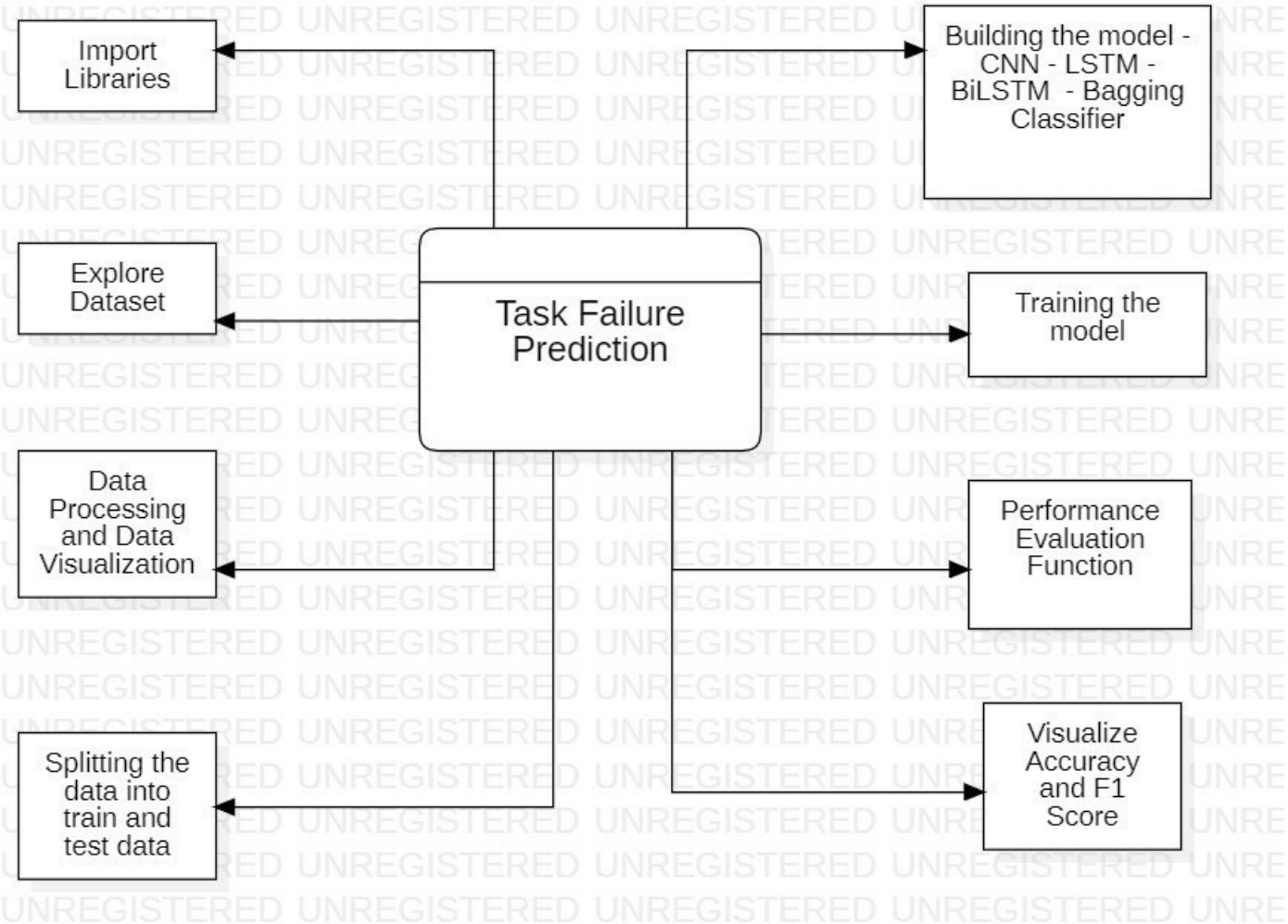


Fig.5.2.2 DFD diagram

6. OVERVIEW OF TECHNOLOGIES

TECHNOLOGIES USED -

- Star UML - Used for creating UML diagrams (DFD and Sequence Diagrams)
- Jupyter Notebook - Implementing code
- Libraries and Packages Used –
  - NumPy
  - Pandas
  - Seaborn
  - Matplotlib
  - Plotly
  - Sklearn
  - Tensorflow
  - Keras

7. IMPLEMENTATION

MODULES:

1. Importing the packages: using this module we will import all packages.
2. Exploring the dataset – Task failure Data: Using this module we will upload the dataset.
3. Data Processing: Using this module we will read data for processing.
4. Visualization using seaborn & matplotlib: Using this module will get graphical representation of information and data.
5. Splitting the data to train and test: Using this module will divide dataset into train & test for processing.
6. Applying SMOTE (Synthetic Minority Oversampling Technique) to the train dataset for removing the bias.
7. Building the model: Using this module we will build all algorithms.
   - Convolutional Neural Network (CNN)
   - Long Short-Term Memory (LSTM)
   - Bi – Long Short-Term Memory (Bi-LSTM)
   - Bagging Classifier
8. Training the model: Using this module algorithms trained for processing & prediction.
9. Building the model with Bagging Classifier since it gives better accuracy comparing with other models.

ALGORITHMS:

CONVOLUTIONAL NEURAL NETWORKS:

Convolutional Neural Networks (CNNs) have been successfully applied to various computer vision and natural language processing tasks. However, they have also shown great promise in predicting failures in cloud data centers. CNNs can automatically learn features from the raw data, such as logs and metrics, which are often used for failure prediction.

In the context of failure prediction, a CNN can take in a sequence of input data points, such as CPU

usage, memory usage, disk usage, and network traffic, and learn to identify patterns that indicate an impending failure. The output of the CNN is a probability score that represents the likelihood of a failure occurring in the near future.

Deep learning techniques, including CNNs, have been shown to outperform traditional machine learning algorithms in many failure prediction tasks. They can handle large and complex datasets, learnfrom unlabeled data, and generalize well to new and unseen data.

Overall, CNNs are a promising approach to predicting failures in cloud data centers, and they have the potential to help operators prevent or mitigate service disruptions, improve availability, and reduce operational costs.

LONG SHORT-TERM MEMORY (LSTM):

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) that has been successfully applied to various sequential data analysis tasks. LSTM is a powerful tool for modeling time-series data, making it a suitable choice for predicting failures in cloud data centers.

In the context of failure prediction, LSTM can be trained on time-series data, such as log files, metrics, and system events, to identify patterns that indicate an impending failure. LSTM is particularly useful for modeling complex and nonlinear relationships between variables, such as the interdependencies between different system components in a cloud data center.

One of the main advantages of LSTM is its ability to remember long-term dependencies and avoid the vanishing gradient problem that often occurs in standard RNNs. This property allows LSTM to capture long-term dependencies between input variables and make accurate predictions over extended periods of time.

LSTM models can be trained on a large dataset of historical failure events to learn patterns and predict future failures. The output of an LSTM model is a probability score that represents the likelihood of a failure occurring in the near future. This information can be used to alert operators to potential issues before they occur, allowing them to take proactive measures to prevent or mitigate service disruptions.

In summary, LSTM is a powerful tool for failure prediction in cloud data centers. Its ability to model

complex relationships between variables and remember long-term dependencies makes it an excellent choice for this task. LSTM can help operators improve the availability and reliability of their services while reducing operational costs.

Bi LONG SHORT-TERM MEMORY (Bi-LSTM):

Bidirectional Long Short-Term Memory (BI-LSTM) is a type of neural network that is particularly well-suited for modeling sequences of data. It is an extension of the standard LSTM that uses two LSTM layers, one that processes the input sequence from beginning to end and another that processes the input sequence from end to beginning. By combining these two layers, BI-LSTM can capture both forward and backward dependencies in the input sequence, making it an excellent choice for predicting failures in cloud data centers.

In the context of failure prediction, BI-LSTM can be trained on time-series data, such as log files, metrics, and system events, to identify patterns that indicate an impending failure. BI-LSTM is particularly useful for modeling complex and nonlinear relationships between variables, such as the interdependencies between different system components in a cloud data center.

One of the main advantages of BI-LSTM is its ability to capture dependencies over longer time periods than standard LSTM. By processing the input sequence in both directions, BI-LSTM can take into account not only past events but also future events that may have an impact on the likelihood of a failure occurring.

BI-LSTM models can be trained on a large dataset of historical failure events to learn patterns and predict future failures. The output of a BI-LSTM model is a probability score that represents the likelihood of a failure occurring in the near future. This information can be used to alert operators to potential issues before they occur, allowing them to take proactive measures to prevent or mitigate service disruptions.

In summary, BI-LSTM is a powerful tool for failure prediction in cloud data centers. Its ability to capture dependencies over longer time periods and model complex relationships between variables makes it an excellent choice for this task. BI-LSTM can help operators improve the availability and reliability of their services while reducing operational costs.

BAGGING CLASSIFIER:

Bagging Classifier is an ensemble learning method that uses multiple base classifiers to improve the accuracy and stability of predictions. In the context of failure prediction in cloud data centers, Bagging Classifier can be used to combine the predictions of multiple deep learning models, such as LSTM or CNN, to make more accurate and robust predictions.

The basic idea behind Bagging Classifier is to create multiple subsets of the training data by randomly sampling with replacement. Each subset is used to train a base classifier, such as an LSTM or CNN, on a different subset of the data. Once all the base classifiers are trained, their predictions are combined using an aggregation method, such as voting or averaging, to produce the final prediction.

One of the main advantages of Bagging Classifier is its ability to reduce overfitting and improve the stability of predictions. By training multiple base classifiers on different subsets of the data, Bagging Classifier can capture different aspects of the data and reduce the impact of outliers or noisy data points.

Bagging Classifier can be particularly effective in the context of failure prediction in cloud data centers, where the data can be complex and noisy, and there may be a high degree of variability in the data due to factors such as changes in workload or hardware failures.

Bagging Classifier has been shown to be effective in improving the accuracy and stability of predictions in many deep learning applications, including failure prediction in cloud data centers. By combining the predictions of multiple deep learning models, Bagging Classifier can help operators improve the reliability and availability of their services while reducing the risk of service disruptions.

In summary, Bagging Classifier is a powerful tool for improving the accuracy and stability of predictions in failure prediction tasks in cloud data centers. By combining the predictions of multiple deep learning models, Bagging Classifier can help operators make more accurate and robust predictions, reducing the risk of service disruptions and improving the reliability and availability of their services.

7.1 CODING

In [2]:
```
#Data pre-processing
import numpy as np
import pandas as pd

#Data visualzation
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px# functions to visualize a variety of types of data.

#Prediction scores
from sklearn.metrics import accuracy_score, f1_score, precision_score, classification_report,confusion_matrix #used to define the performan
from sklearn.preprocessing import StandardScaler

#machine learning algorithms
from sklearn.model_selection import KFold,StratifiedKFold #Provides train/test indices to split data in train/test sets.
from sklearn.model_selection import train_test_split #Split arrays or matrices into random train and test subsets.
from keras.utils.np_utils import to_categorical #Converts a class vector (integers) to binary class matrix.
from sklearn.metrics import log_loss #loss functions are a measurement of how good your model is in terms of predicting the expected outcom
from sklearn import linear_model #contain different functions for performing machine learning with linear models

from sklearn.ensemble import BaggingClassifier
from sklearn.neural_network import MLPClassifier
from imblearn.over_sampling import SMOTE
import tensorflow as tf
from tensorflow import keras
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.utils import class_weight
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [3]:
```
#importing the dataset
train_df=pd.read_csv('train_data.csv')
```

In [4]:
```
#Datasets columns
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   job_id               20000 non-null  object
 1   memory_GB            20000 non-null  float64
 2   network_log10_MBps   20000 non-null  float64
 3   local_IO_log10_MBps  20000 non-null  float64
 4   NFS_IO_log10_MBps    20000 non-null  float64
 5   failed               20000 non-null  int64
dtypes: float64(4), int64(1), object(1)
memory usage: 937.6+ KB
```

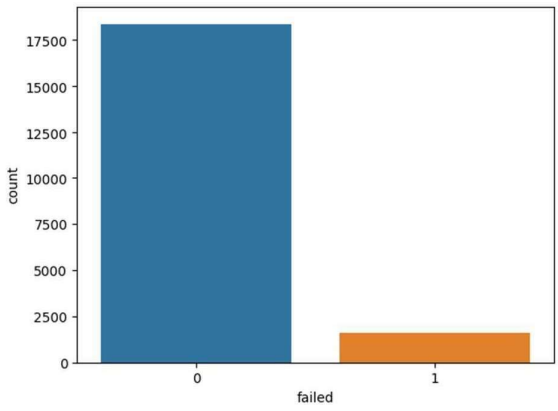
In [5]:
```
#Dataset Size
print(train_df.shape)
```

```
(20000, 6)
```

Before SMOTE:

In [6]:
```
#Dataset visualization
import seaborn as sns
sns.countplot(x="failed", data = train_df)#it shows the counts of observations in each categorical bin using bars.
```

Out[6]: <Axes: xlabel='failed', ylabel='count'>



In [7]:
```
#iloc-helps us select a specific row or column from the data set.
X = train_df.drop(['job_id','failed'],axis=1) #Independent columns
y = train_df['failed']#dependent columns
```

In [8]:
```
#Dataset preprocessing
#Dataset divided into train dataset ,test dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
print(X_train)
print(X_test)
print(y_train)
print(y_test)
```

```
       memory_GB  network_log10_MBps  local_IO_log10_MBps  NFS_IO_log10_MBps
5514      9.9907              3.0017              -1.0306            -3.0000
1266     68.3678              3.6583               0.3965            -2.9789
5864     30.4616              3.1174               0.0286             1.6032
15865    64.8335              2.7876               0.1579            -3.0000
12892    29.0354             -0.8687               0.4797            -3.0000
...          ...                 ...                  ...               ...
11284    22.8092              3.6074              -0.6133            -3.0000
11964    15.4002              3.1133              -0.3565            -3.0000
5390     31.3535              2.7512              -0.2257             1.5171
860      22.4325              3.6977              -1.3013             0.8938
15795     7.4687              0.9667               0.6496            -3.0000

[15000 rows x 4 columns]
       memory_GB  network_log10_MBps  local_IO_log10_MBps  NFS_IO_log10_MBps
10650    31.8525             -1.9383              -0.8963            -3.0000
2041      7.3693              2.9138              -0.9154            -3.0000
8668     10.2089             -2.3445              -0.7959            -3.0000
1114     10.2756              2.9883              -0.9035            -3.0000
13902     9.8465              1.5807              -0.4440             1.5580
...          ...                 ...                  ...               ...
3761     19.8527              3.1227              -1.4124            -2.1073
5478      9.9907              0.6636               0.1532            -0.3913
5805     74.1367              0.8782               0.5388             1.2031
10084   100.2638              2.3703               0.0959            -3.0000
[5000 rows x 4 columns]
5514     0
1266     0
5864     0
15865    0
12892    0
        ..
11284    0
11964    0
5390     0
860      0
15795    0
Name: failed, Length: 15000, dtype: int64
10650    0
2041     0
8668     0
1114     0
13902    0
        ..
3761     0
5478     0
5805     0
10084    0
13494    0
Name: failed, Length: 5000, dtype: int64
```

In [9]:
```
# now we reshape train,test dataset .
X_train = X_train.values
X_test = X_test.values
print(X_train)
print(X_test)
print(X_train.shape[1])
print(X_test.shape[1])

X_train = X_train.reshape(-1, X_train.shape[1],1)
X_test = X_test.reshape(-1, X_test.shape[1],1)
print(X_train)
print(X_test)
```

```
[[ 9.99070e+00  3.00170e+00 -1.03060e+00 -3.00000e+00]
 [ 6.83678e+01  3.65830e+00  3.96500e-01 -2.97890e+00]
 [ 3.04616e+01  3.11740e+00  2.86000e-02  1.60320e+00]
 ...
 [ 3.13535e+01  2.75120e+00 -2.25700e-01  1.51710e+00]
 [ 2.24325e+01  3.69770e+00 -1.30130e+00  8.93800e-01]
 [ 7.46870e+00  9.66700e-01  6.49600e-01 -3.00000e+00]]
[[ 3.185250e+01 -1.938300e+00 -8.963000e-01 -3.000000e+00]
 [ 7.369300e+00  2.913800e+00 -9.154000e-01 -3.000000e+00]
 [ 1.020890e+01 -2.344500e+00 -7.959000e-01 -3.000000e+00]
 ...
 [ 7.413670e+01  8.782000e-01  5.388000e-01  1.203100e+00]
 [ 1.002638e+02  2.370300e+00  9.500000e-02 -3.000000e+00]
 [ 1.942320e+01  3.520000e-02 -9.370000e-02  1.160000e-02]]
4
4
[[[ 9.99070e+00]
  [ 3.00170e+00]
  [-1.03060e+00]
  [-3.00000e+00]]

 [[ 6.83678e+01]
  [ 3.65830e+00]
  [ 3.96500e-01]
  [-2.97890e+00]]
```

In [10]:
```
#return size,shape,dimensions of the dataframe
print(X_train.shape)
print(X_test.shape)
print(X_train.shape[2])
```

```
(15000, 4, 1)
(5000, 4, 1)
1
```

Convert the Target label to categorical

In [11]:
```
target_train = y_train
target_test = y_test
Y_train=to_categorical(target_train)
Y_test=to_categorical(target_test)
```

In [12]:
```
print(Y_train)
print(Y_train.shape)
print(Y_test.shape)
```

```
[[1. 0.]
 [1. 0.]
 [1. 0.]
 ...
 [1. 0.]
 [1. 0.]
 [1. 0.]]
(15000, 2)
(5000, 2)
```

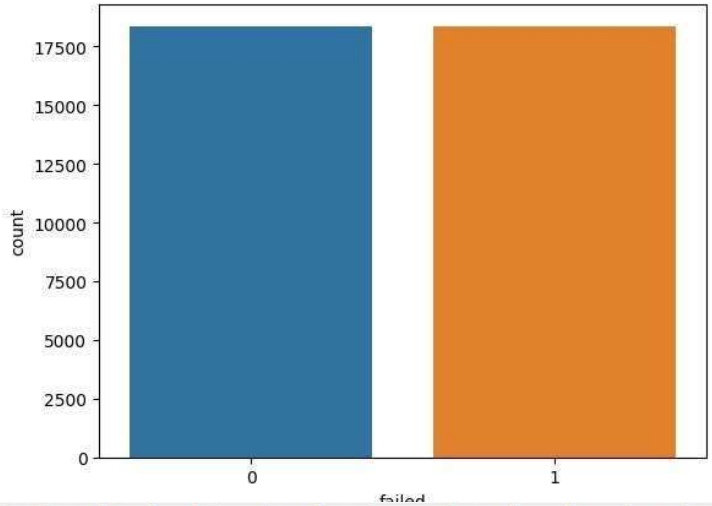
Performance Evaluation Function

In [13]:
```
#To find the accuracy,precision,f1score for each algorithms
def showResults(test, pred):
    #For Example target_names = ['positive', 'negative']
    # print(classification_report(test, pred, target_names=target_names))
    accuracy = accuracy_score(test, pred)
    precision=precision_score(test, pred, average='weighted')
    f1Score=f1_score(test, pred, average='weighted')
    loss=log_loss(test,pred)
    print("Accuracy  : {}".format(accuracy))
    print("Precision : {}".format(precision))
    print("f1Score : {}".format(f1Score))
    print("Loss : {}".format(loss))
    cm=confusion_matrix(test, pred)
    print(cm)
```

After SMOTE:

In [50]:
```
import seaborn as sns
X = train_df.drop(['job_id','failed'], axis=1)
y = train_df['failed']
smote = SMOTE(sampling_strategy='minority')

# fit and transform the data
X_resampled, y_resampled = smote.fit_resample(X, y)
X_train, X_val, y_train, y_val = train_test_split(X_resampled, y_resampled, test_size=0.2)
# Load the resampled data
X_resampled_df = pd.DataFrame(X_resampled, columns=X.columns)
y_resampled_df = pd.DataFrame(y_resampled, columns=['failed'])
df_resampled = pd.concat([X_resampled_df, y_resampled_df], axis=1)

# Visualize class distribution
sns.countplot(x="failed", data=df_resampled)
```

Out[50]: <Axes: xlabel='failed', ylabel='count'>

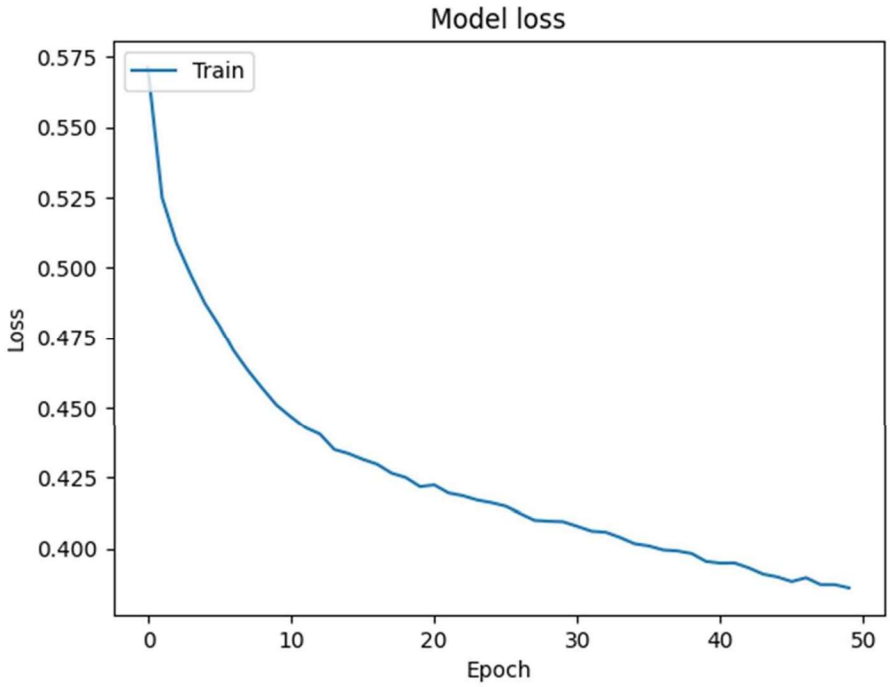


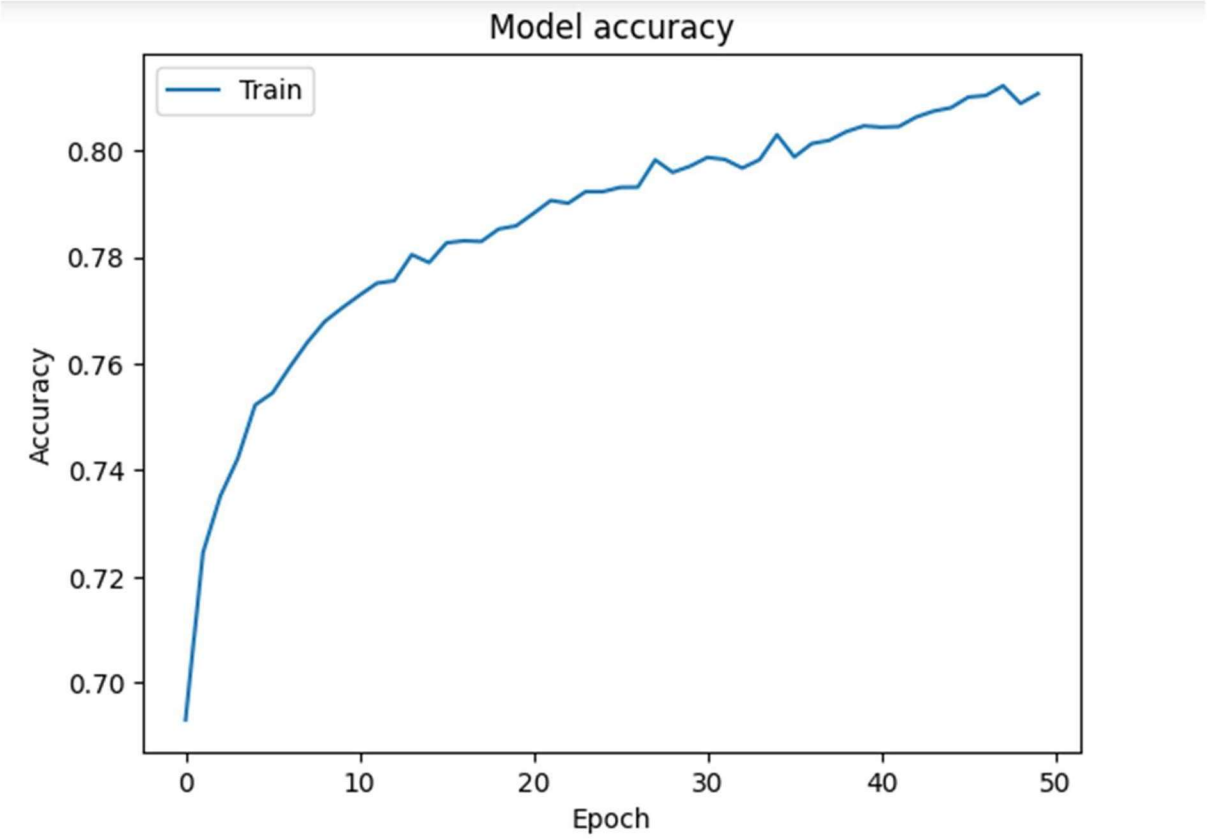
After Smoting the dataset performing various deep learning algorithms

CNN

In [27]:
```
# Load the data
df = pd.read_csv('train_data.csv')

# Separate the features and target variables
X = df.drop(['job_id', 'failed'], axis=1)
y = df['failed']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform SMOTE on the dataset to address class imbalance
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Define the model architecture
def getModel():
    cnnmodel = keras.Sequential([
        layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
        layers.Dense(32, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])

    # Compile the model
    cnnmodel.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return cnnmodel
cnnmodel=getModel()
    # Train the model
history = cnnmodel.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

In [28]:
```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig('loss.png', format='png', dpi=1200)
plt.show()

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig('accuracy.png', format='png', dpi=1200)
plt.show()
```





In [29]:
```
cnnpredictions=cnnmodel.predict(X_test,verbose=1)
```

```
230/230 [==============================] - 1s 4ms/step
```

In [30]:
```
# Evaluate the model on the testing set
cnn_predict=([1 if y >= 0.5 else 0 for y in cnnpredictions])
cnn_actual_value=y_test
showResults(cnn_actual_value,cnn_predict)
```

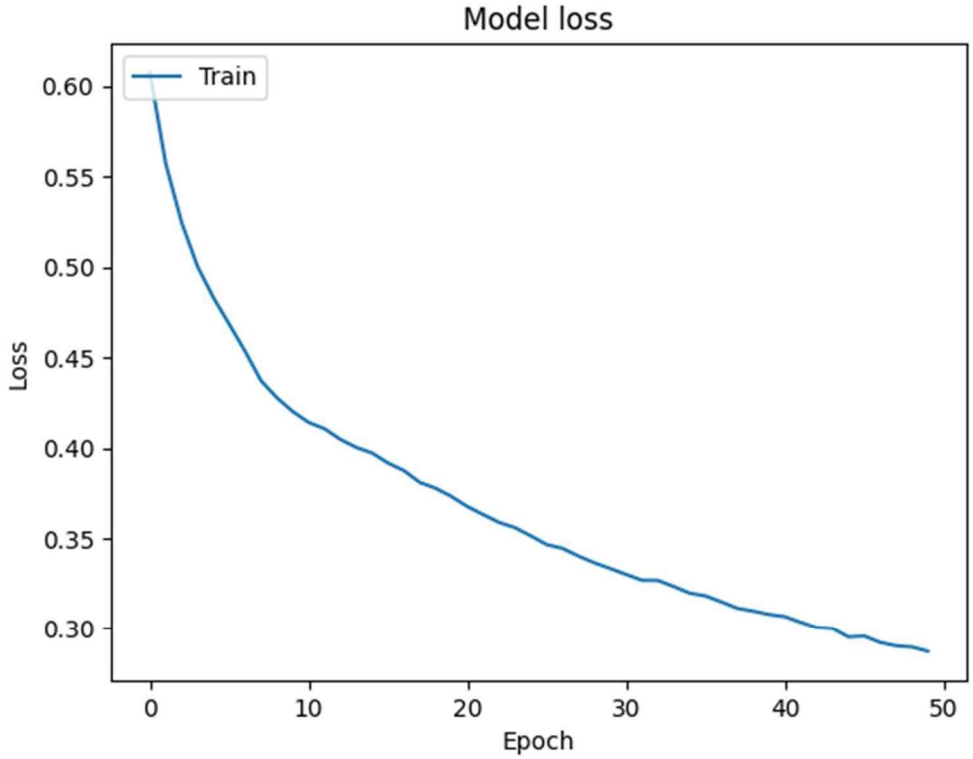
```
Accuracy  : 0.7915646258503402
Precision : 0.7949046061951345
f1Score : 0.7910331860256046
Loss : 7.51277237988129
[[3087  573]
 [ 959 2731]]
```

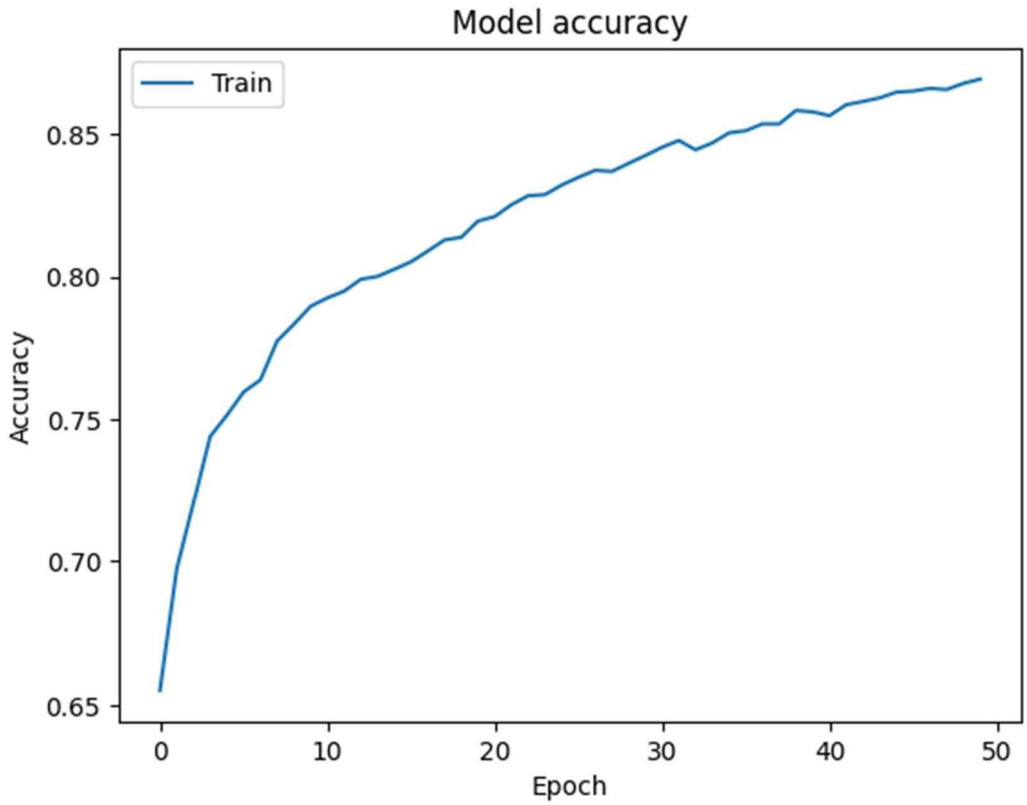
In [31]:
```
cnn=accuracy_score(cnn_actual_value,cnn_predict)
f1cnn=f1_score(cnn_actual_value,cnn_predict,average='weighted')
```

LSTM

In [32]:
```
# Load the data
df = pd.read_csv('train_data.csv')

# Separate the features and target variables
X = df.drop(['job_id', 'failed'], axis=1)
y = df['failed']
S
# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform SMOTE on the dataset to address class imbalance
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Reshape the data for LSTM input
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1],1) #->(1,2,3)1) # 1)batch_size
#2)a way of examining and analyzing your data through specified time intervals
#3)it defines the features per  time step.
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Define the model architecture
def lstmmodel():
    model = keras.Sequential([
        layers.LSTM(64, input_shape=(X_train.shape[1], 1)),
        layers.Dense(32, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
lstm=lstmmodel()
# Train the model
history = lstm.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

In [33]:
```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig('loss.png', format='png', dpi=1200)
plt.show()

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig('accuracy.png', format='png', dpi=1200)
plt.show()
```


Model loss
Train
0.60
0.55
0.50
0.45
0.40
0.35
0.30
Loss
0
10
20
30
40
50
Epoch


Model accuracy
Train
0.85
0.80
0.75
0.70
0.65
Accuracy
0
10
20
30
40
50
Epoch

In [34]:
```
# Evaluate the model on the testing set
predictions=lstm.predict(X_test,verbose=1)
predictcv =[1 if y >= 0.5 else 0 for y in predictions]
actual_valuecv=y_test
showResults(actual_valuecv,predictcv)
```

```
230/230 [==============================] - 2s 4ms/step
Accuracy  : 0.8574149659863946
Precision : 0.8587365428882745
f1Score : 0.8572620774782884
Loss : 5.139285544461874
[[3023  637]
 [ 411 3279]]
```

In [35]:
```
lstm = accuracy_score(actual_valuecv, predictcv)
f1lstm=f1_score(actual_valuecv, predictcv, average='weighted')
```

Bi-LSTM

In [36]:
```
# Load the data
df = pd.read_csv('train_data.csv')

# Separate the features and target variables
X = df.drop(['job_id', 'failed'], axis=1)
y = df['failed']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform SMOTE on the dataset to address class imbalance
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Reshape the data for LSTM input
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Define the model architecture
def bilstm():
    model = keras.Sequential([
        layers.Bidirectional(layers.LSTM(64, input_shape=(X_train.shape[1], 1))),
        layers.Dense(32, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])

    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
# Train the model
bilstm=bilstm()
history = bilstm.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

In [38]:
```
# Evaluate the model on the testing set
predictions=bilstm.predict(X_test,verbose=1)
predictcv =[1 if y >= 0.5 else 0 for y in predictions]
actual_valuecv=y_test
showResults(actual_valuecv,predictcv)
```

```
230/230 [==============================] - 2s 6ms/step
Accuracy  : 0.8653061224489796
Precision : 0.8653103232763311
f1Score : 0.8653066211111107
Loss : 4.854859436085167
[[3170  490]
 [ 500 3190]]
```

In [39]:
```
bilstm = accuracy_score(actual_valuecv, predictcv)
f1bilstm=f1_score(actual_valuecv, predictcv, average='weighted')
```

7.2 TESTING

7.2.1 BLACK BOX TESTING

Bagging Classifier

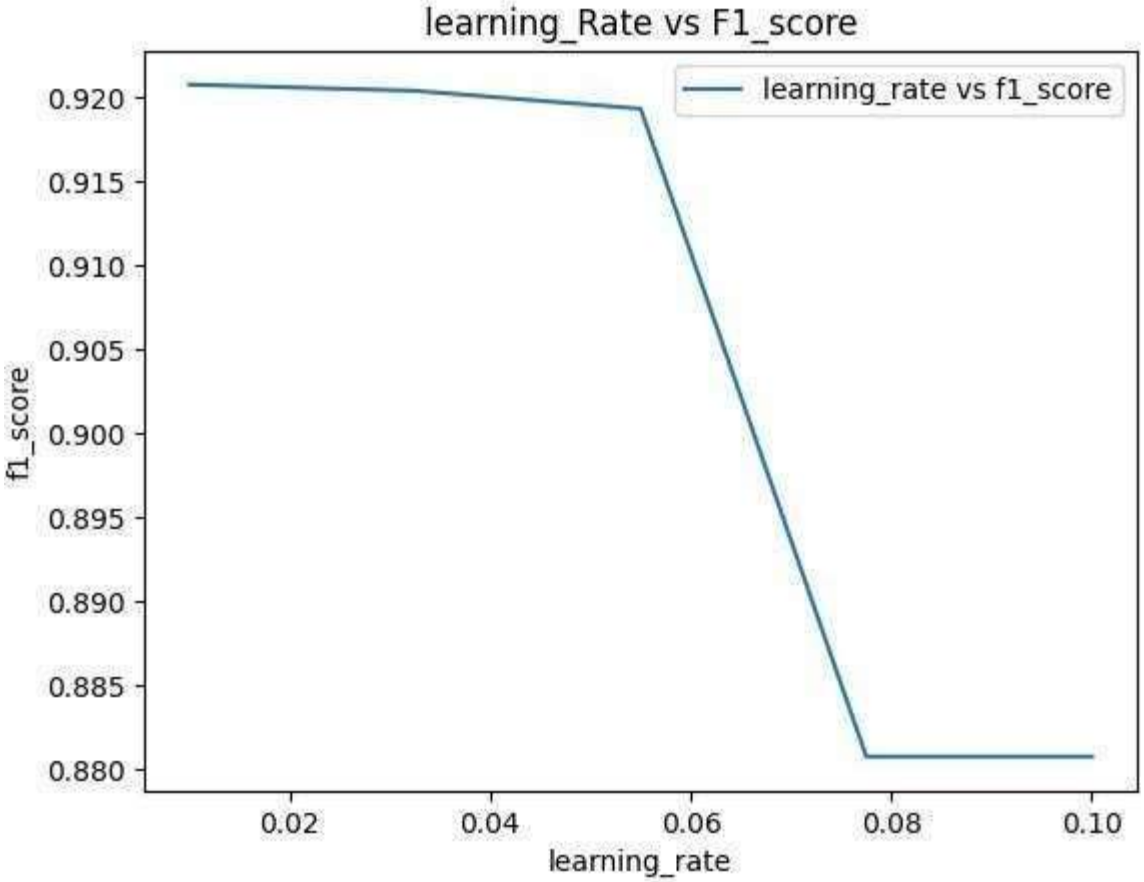
In [25]:
```
X = train_df.drop(['job_id','failed'], axis=1)
y = train_df['failed']
smote = SMOTE(sampling_strategy='minority')

# fit and transform the data
X_resampled, y_resampled = smote.fit_resample(X, y)
X_train, X_val, y_train, y_val = train_test_split(X_resampled, y_resampled, test_size=0.2)

# train a Random Forest classifier on the training data
clfbagging = BaggingClassifier(n_estimators=100)
clfbagging.fit(X_train, y_train)

# make predictions on the validation set
y_pred = clfbagging.predict(X_val)

# evaluate the performance of the classifier
showResults(y_val,y_pred)
```

```
Accuracy  : 0.9239455782312925
Precision : 0.9246527050249083
f1Score : 0.9239292272285324
Loss : 2.741279216941019
[[3356  353]
 [ 206 3435]]
```

In [26]:
```
val2=accuracy_score(y_val,y_pred)*100
f1bagging=f1_score(y_val,y_pred,average='weighted')
```

7.2.2 WHITE BOX TESTING

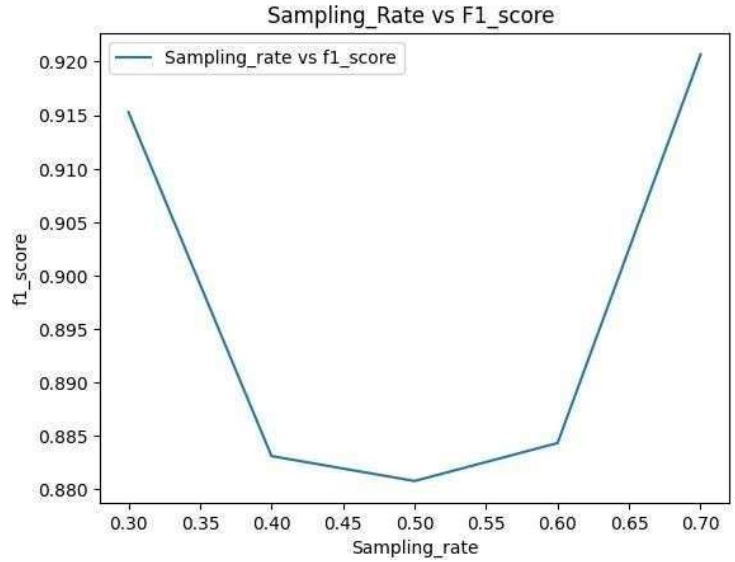
In [103]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, f1_score
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

s = np.linspace(0.01, 0.1, 5)
learning_rate = []
f1 = []

for i in s:
    clf1 = MLPClassifier(hidden_layer_sizes=(6,5),
                         learning_rate_init=i)
    learning_rate.append(i)

    bagging_clf = BaggingClassifier(base_estimator=clf1, n_estimators=10, random_state=42)
    bagging_clf.fit(X_train, y_train)
    y_pred = bagging_clf.predict(X_test)

    f1.append(f1_score(y_test, y_pred, average='weighted'))

plt.plot(learning_rate, f1, label='learning_rate vs f1_score')
plt.xlabel("learning_rate")
plt.ylabel("f1_score")
plt.title('learning_Rate vs F1_score')
plt.legend()
plt.show()
```


learning_Rate vs F1_score
learning_rate vs f1_score
0.920
0.915
0.910
0.905
0.900
0.895
0.890
0.885
0.880
f1_score
0.02
0.04
0.06
0.08
0.10
learning_rate

In [104]:

```
from sklearn.ensemble import BaggingClassifier

a=[]
p=[]
f1=[]
Test_rate=[]
def different_rates():
    s=np.linspace(0.01,0.1,5)# tool in Python for creating numeric sequences
    l=[]

    t=[0.3,0.4,0.5,0.6,0.7] #test sizes
    for i in s:
        l.append(float('{:.2f}'.format(i)))

    clf1 = MLPClassifier(hidden_layer_sizes=(6,5),
                         learning_rate_init=0.08)

    for i in t:
        X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=i,random_state=42)
        Test_rate.append(i)

        clf_bagging = BaggingClassifier(base_estimator=clf1, n_estimators=10)
        clf_bagging.fit(X_train, y_train)
        ypred = clf_bagging.predict(X_test)
        a.append(accuracy_score(y_test,ypred))
        p.append(precision_score(y_test,ypred, average='weighted'))
        f1.append(f1_score(y_test,ypred, average='weighted'))

different_rates()

plt.plot(Test_rate,f1,label='Sampling_rate vs f1_score')
plt.xlabel("Sampling_rate")
plt.ylabel("f1_score")
print(plt)
plt.title('Sampling_Rate vs F1_score ')
plt.legend()
plt.show()
```

```
<module 'matplotlib.pyplot' from 'C:\\Users\\divya\\anaconda3\\envs\\new-env\\lib\\site-packages\\matplotlib\\pyplot.py
'>
```



8. RESULTS AND DISCUSSIONS

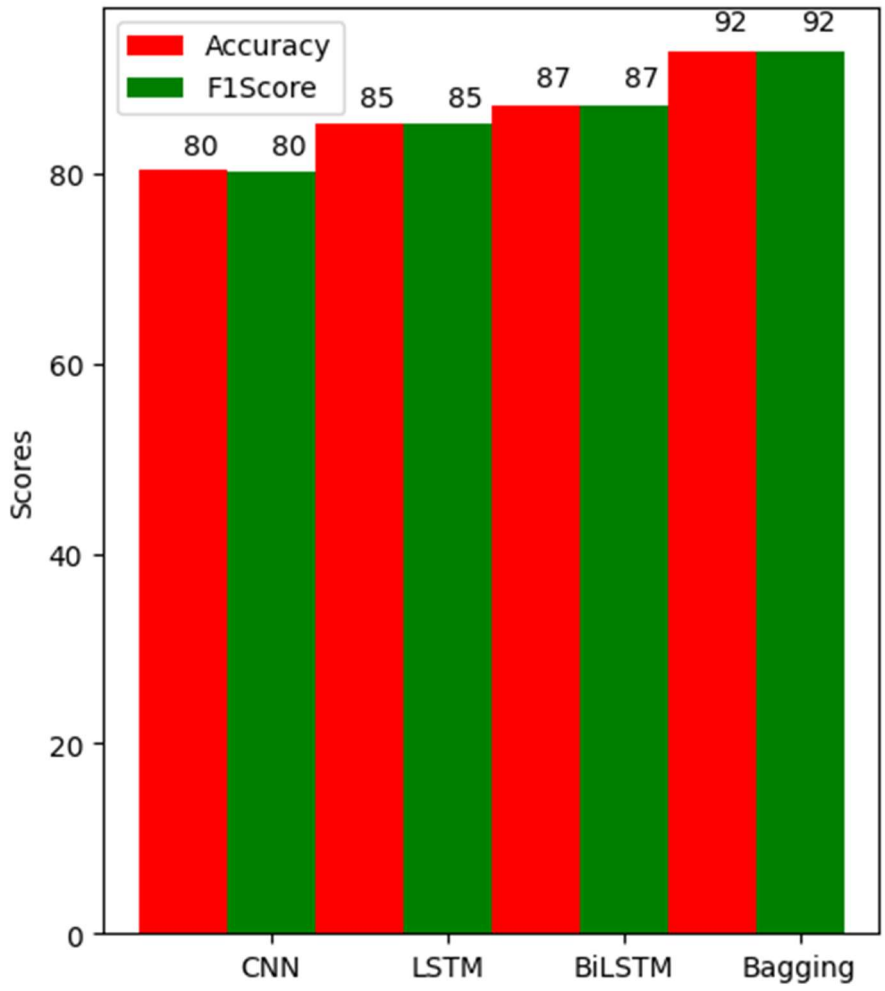
RESULTS

S. No	Model	Accuracy	F1 Score	Precision
1	Convolutional Neural Network (CNN)	79.15%	79.49%	79.10%
2	Long-Short term Memory (LSTM)	85.74%	85.87%	85.72%
3	Bi-directional Long-Short term Memory (Bi-LSTM)	86.53%	86.53%	86.53%
4	Bagging Classifier	92.39%	92.46%	92.39%

Comparison

In [96]:

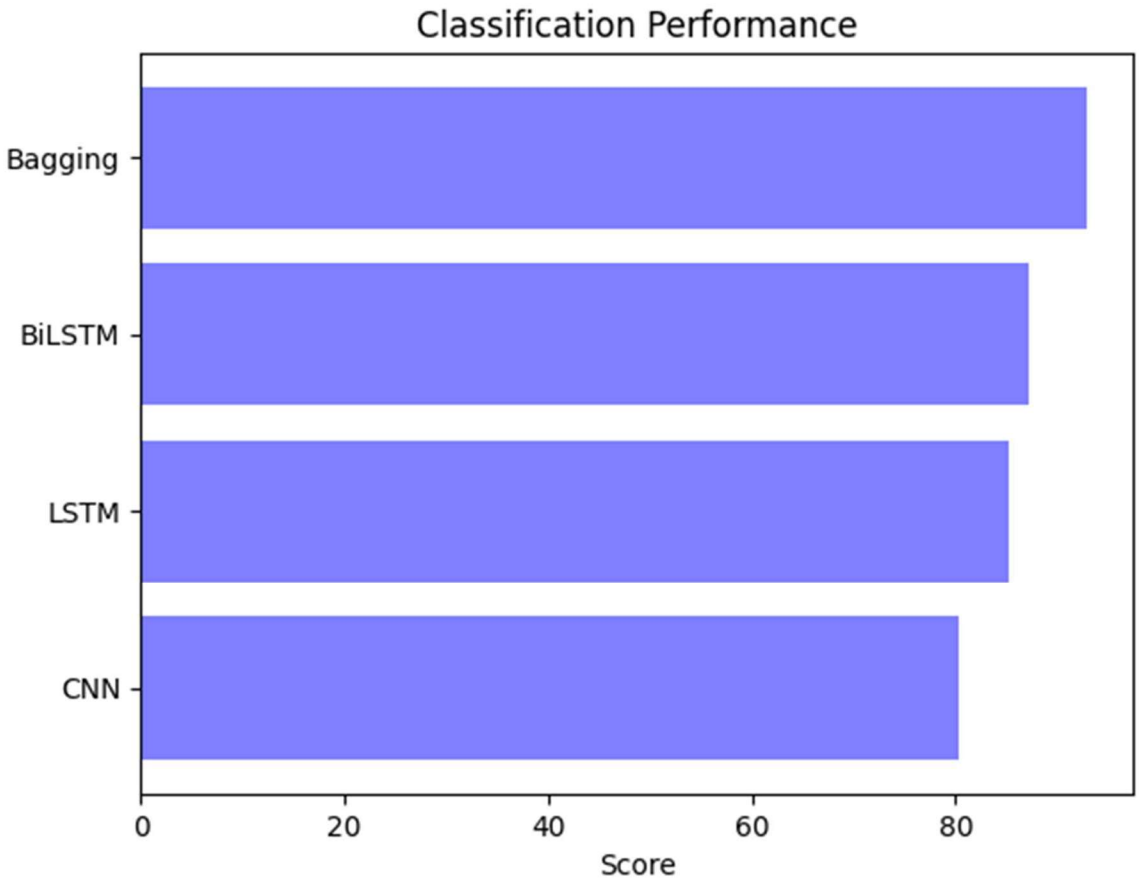
```
import numpy as np
import matplotlib.pyplot as plt

N = 4
ind = np.arange(N)  # the x locations for the groups
width = 0.5      # the width of the bars

fig = plt.figure(figsize=(5,6))
ax = fig.add_subplot(111)

yvals = [cnn*100,lstm*100,bilstm*100,val2]

rects1 = ax.bar(ind, yvals, width, color='r')
zvals = [f1cnn*100,f1lstm*100,f1bilstm*100,f1bagging*100]
rects2 = ax.bar(ind+width, zvals, width, color='g')


ax.set_ylabel('Scores')
ax.set_xticks(ind+width)
ax.set_xticklabels( ('CNN','LSTM','BiLSTM','Bagging') )
ax.legend( (rects1[0], rects2[0]), ('Accuracy', 'F1Score') )

def autolabel(rects):
    for rect in rects:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='left', va='top')

autolabel(rects1)
autolabel(rects2)

plt.show()
```



In [68]:
```
score = [cnn*100,lstm*100,bilstm*100,val2]
classifier=('CNN','LSTM','BiLSTM','Bagging')
y_pos=np.arange(len(classifier))
print(y_pos)
print(score)
```

```
[0 1 2 3]
[80.39455782312925, 85.26530612244898, 87.17006802721089, 92.81632653061224]
```

In [70]:
```
import matplotlib.pyplot as plt2
plt2.barh(y_pos, score, alpha=0.5,color='blue')
plt2.yticks(y_pos, classifier)
plt2.xlabel('Score')
plt2.title('Classification Performance')
plt2.show()
```



DISCUSSIONS

- Firstly, we implemented CNN algorithm and achieved an accuracy of 79.15%.
- Later we realized our dataset was biased towards one value so we used SMOTE.
- SMOTE stands for Synthetic Minority Oversampling Technique which is an improved method of dealing with imbalanced data in classification problems.
- After implementing SMOTE on our dataset, we later implemented various other algorithms such as LSTM, BiLSTM and Bagging Classifier.
- Out of all the algorithms we achieved a maximum accuracy of 92.39% from bagging classifier.
- Hence, we completed our project implementation using bagging classifier.

9.CONCLUSIONS AND FUTURE SCOPE

CONCLUSION

In cloud data centers, high service reliability and availability are crucial to application QoS. In our method, we first input the data into a trained model. We then find that the further input features are essential to achieve high prediction accuracy. We found the various linear models by using statistical, machine learning and deep learning-based methods and evaluated the performance with three metrics: accuracy and F1 score, precision score.

We finally implemented Bagging Classifier from which we achieved maximum accuracy of 92.39% by performing both white box and black box testing and proved that this is our best fit linear model for our major project.

FUTURE SCOPE

Try to implement our methods in real data center to examine the real-world performance and then work on how to use the failure prediction results to build a failure recovery system in data center which can further improve the fault tolerance performance.

10. REFERENCES

[1] Mina Sedaghat, **Eddie Wadbro** (2016). DieHard: Reliable Scheduling to Survive Correlated Failures in Cloud Data Centers, *IEEE Publishers*, ISBN:978-1-5090-2453-7)

[2] Thanyalak Chalermarrewong, **Tiranee Achalakul** (2012), Failure Prediction of Data Centers Using Time Series and Fault Tree Analysis*, IEEE Publishers*, ISBN:978-1-4673-4565-1

[3] Rajesh Patna, **Subrata Mitra (**2016), Partial-parallel-repair (ppr): a distributed technique for repairing erasure coded storage*, Published by Association for Computing Machinery*, ISBN:9781450342407

[4] Haoyu Wang, **Haiying Shen** (2018), Approaches for Resilience against Cascading Failures in Cloud Datacenters, *IEEE Publishers*, ISBN:978-1-5386-6871-9

[5] H. Wang and H. Shen, "Proactive incast congestion control in a datacenter serving web applications," in *Proc. of INFOCOM*, 2018.

[6] R. Baldoni, L. Montanari, and M. Rizzuto, "On-line failure prediction in safety-critical systems," *Future Generation Computer Systems*, 2015.

[7] Y. Zhao, X. Liu, S. Gan, and W. Zheng, "Predicting disk failures with hmm-and hsmm-based approaches," in *Industrial Conference on Data Mining*, 2010.

[8] J. Murray, G. Hughes, and K. Kreutz-Delgado, "Machine learning methods for predicting failures in hard drives: A multiple-instance application," *Journal of Machine Learning Research*, 2005.

[9] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure prediction based on log files using random indexing and support vector machines," *Journal of Systems and Software*, 2013.