This notebook has solutions for both Questions 1 and 2. (Note: The solution for Q1-5 has been shared in a different file)

In [23]:
```
import pyspark
from pyspark.sql import SparkSession, SQLContext
from pyspark.ml import Pipeline,Transformer
from pyspark.ml.feature import Imputer,StandardScaler,StringIndexer,OneHotEncoder, Vecto

from pyspark.sql.functions import *
from pyspark.sql.types import *
import numpy as np

col_names = ["duration","protocol_type","service","flag","src_bytes",
"dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
"logged_in","num_compromised","root_shell","su_attempted","num_root",
"num_file_creations","num_shells","num_access_files","num_outbound_cmds",
"is_host_login","is_guest_login","count","srv_count","serror_rate",
"srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
"diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
"dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
"dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
"dst_host_rerror_rate","dst_host_srv_rerror_rate","class","difficulty"]

nominal_cols = ['protocol_type','service','flag']
binary_cols = ['land', 'logged_in', 'root_shell', 'su_attempted', 'is_host_login',
'is_guest_login']
continuous_cols = ['duration' ,'src_bytes', 'dst_bytes', 'wrong_fragment' ,'urgent', 'ho
'num_failed_logins', 'num_compromised', 'num_root' ,'num_file_creations',
'num_shells', 'num_access_files', 'num_outbound_cmds', 'count' ,'srv_count',
'serror_rate', 'srv_serror_rate' ,'rerror_rate' ,'srv_rerror_rate',
'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate' ,'dst_host_count',
'dst_host_srv_count' ,'dst_host_same_srv_rate' ,'dst_host_diff_srv_rate',
'dst_host_same_src_port_rate' ,'dst_host_srv_diff_host_rate',
'dst_host_serror_rate' ,'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
'dst_host_srv_rerror_rate']

attack_category = ['normal', 'DOS', 'R2L', 'U2R', 'probe']
normal = ['normal']
DOS = ['apache2','back','land','neptune','mailbomb','pod','processtable','smurf','teardr
R2L = ['ftp_write','guess_passwd','httptunnel','imap','multihop','named','phf','sendmail
U2R = ['buffer_overflow','loadmodule','perl','ps','rootkit','sqlattack','xterm']
probe = ['ipsweep','mscan','nmap','portsweep','saint','satan']

class OutcomeCreater(Transformer): # this defines a transformer that creates the outcome

    def __init__(self):
        super().__init__()

    def _transform(self, dataset):
        label_to_binary = udf(lambda name: 0.0 if name == 'normal' else 1.0 if name in D
        label_to_category = udf(lambda name: attack_category[0] if name == 0.0 else atta
        output_df = dataset.withColumn('outcome', label_to_binary(col('class'))).drop("c
        output_df = output_df.withColumn('outcome', col('outcome').cast(DoubleType()))
        output_df = output_df.withColumn('outcome_category',label_to_category(col('outco
        output_df = output_df.withColumn('outcome_category', col('outcome_category').cas
        output_df = output_df.drop('difficulty')
        return output_df

class FeatureTypeCaster(Transformer): # this transformer will cast the columns as approp
    def __init__(self):
        super().__init__()

    def _transform(self, dataset):
```

```
        output_df = dataset
        for col_name in binary_cols + continuous_cols:
            output_df = output_df.withColumn(col_name,col(col_name).cast(DoubleType()))

        return output_df
class ColumnDropper(Transformer): # this transformer drops unnecessary columns
    def __init__(self, columns_to_drop = None):
        super().__init__()
        self.columns_to_drop=columns_to_drop
    def _transform(self, dataset):
        output_df = dataset
        for col_name in self.columns_to_drop:
            output_df = output_df.drop(col_name)
        return output_df

def get_preprocess_pipeline():
    # Stage where columns are casted as appropriate types
    stage_typecaster = FeatureTypeCaster()

    # Stage where nominal columns are transformed to index columns using StringIndexer
    nominal_id_cols = [x+"_index" for x in nominal_cols]
    nominal_onehot_cols = [x+"_encoded" for x in nominal_cols]
    stage_nominal_indexer = StringIndexer(inputCols = nominal_cols, outputCols = nominal_

    # Stage where the index columns are further transformed using OneHotEncoder
    stage_nominal_onehot_encoder = OneHotEncoder(inputCols=nominal_id_cols, outputCols=n

    # Stage where all relevant features are assembled into a vector (and dropping a few)
    feature_cols = continuous_cols+binary_cols+nominal_onehot_cols
    corelated_cols_to_remove = ["dst_host_serror_rate","srv_serror_rate","dst_host_srv_s
                 "srv_rerror_rate","dst_host_rerror_rate","dst_host_srv_rerror_rate"
    for col_name in corelated_cols_to_remove:
        feature_cols.remove(col_name)
    stage_vector_assembler = VectorAssembler(inputCols=feature_cols, outputCol="vectoriz

    # Stage where we scale the columns
    stage_scaler = StandardScaler(inputCol= 'vectorized_features', outputCol= 'features'


    # Stage for creating the outcome column representing whether there is attack
    stage_outcome = OutcomeCreater()

    # Removing all unnecessary columbs, only keeping the 'features' and 'outcome' column
    stage_column_dropper = ColumnDropper(columns_to_drop = nominal_cols+nominal_id_cols+
        nominal_onehot_cols+ binary_cols + continuous_cols + ['vectorized_features'])
    # Connect the columns into a pipeline
    pipeline = Pipeline(stages=[stage_typecaster,stage_nominal_indexer,stage_nominal_one
        stage_vector_assembler,stage_scaler,stage_outcome,stage_column_dropper])
    return pipeline
```

In [24]:

```
import os
import sys

os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable

appName = "SystemsToolChains"
master = "local[*]"

# Create Configuration object for Spark.
conf = pyspark.SparkConf()\
    .set('spark.driver.host','127.0.0.1')\
    .set('spark.driver.memory', '15g')\
    .setAppName(appName)\
    .setMaster(master)
```

```
# Create Spark Context with the new configurations rather than relying on the default on
sc = SparkContext.getOrCreate(conf=conf)

# You need to create SQL Context to conduct some database operations like what we will s
sqlContext = SQLContext(sc)

# If you have SQL context, you create the session from the Spark Context
spark = sqlContext.sparkSession.builder.getOrCreate()

nslkdd_raw = spark.read.csv('/Users/kiranprasadjp/Downloads/NSL-KDD/KDDTrain+.txt',heade
nslkdd_test_raw = spark.read.csv('/Users/kiranprasadjp/Downloads/NSL-KDD/KDDTest+.txt',h

preprocess_pipeline = get_preprocess_pipeline()
preprocess_pipeline_model = preprocess_pipeline.fit(nslkdd_raw)

nslkdd_df = preprocess_pipeline_model.transform(nslkdd_raw)
nslkdd_df_test = preprocess_pipeline_model.transform(nslkdd_test_raw)
```

```
/Applications/ANACONDA/anaconda3/envs/aiml_env/lib/python3.10/site-packages/pyspark/sql/
context.py:112: FutureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate
() instead.
  warnings.warn(
```

Q1-1.

In [25]:
```
nslkdd_df.printSchema()
nslkdd_df.show()
```

```
root
 |-- features: vector (nullable = true)
 |-- outcome: double (nullable = true)
 |-- outcome_category: string (nullable = true)

+--------------------+-------+----------------+
|            features|outcome|outcome_category|
+--------------------+-------+----------------+
|(113,[1,13,14,17,...|    0.0|          normal|
|(113,[1,13,14,17,...|    0.0|          normal|
|(113,[13,14,15,17...|    1.0|             DOS|
|(113,[1,2,13,14,1...|    0.0|          normal|
|(113,[1,2,13,14,1...|    0.0|          normal|
|(113,[13,14,16,17...|    1.0|             DOS|
|(113,[13,14,15,17...|    1.0|             DOS|
|(113,[13,14,15,17...|    1.0|             DOS|
|(113,[13,14,15,17...|    1.0|             DOS|
|(113,[13,14,15,17...|    1.0|             DOS|
|(113,[13,14,16,17...|    1.0|             DOS|
|(113,[13,14,15,17...|    1.0|             DOS|
|(113,[1,2,13,14,1...|    0.0|          normal|
|(113,[1,13,14,17,...|    2.0|             R2L|
|(113,[13,14,15,18...|    1.0|             DOS|
|(113,[13,14,15,17...|    1.0|             DOS|
|(113,[1,2,13,14,1...|    0.0|          normal|
|(113,[1,13,14,17,...|    4.0|           probe|
|(113,[1,2,13,14,1...|    0.0|          normal|
|(113,[1,2,13,14,1...|    0.0|          normal|
+--------------------+-------+----------------+
only showing top 20 rows
```

Q1-2 Logistic Regression (Model 1)

In [26]:
```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol = 'features', labelCol = 'outcome')

lrModel = lr.fit(nslkdd_df) # fit the logistic regression model to the training dataset
```

In [27]:
```
predictions = lrModel.transform(nslkdd_df_test)
```

In [28]:
```
predictions.printSchema()
```

```
root
 |-- features: vector (nullable = true)
 |-- outcome: double (nullable = true)
 |-- outcome_category: string (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

In [29]:
```
predictions.select("rawPrediction","probability","prediction","outcome").toPandas().head
```

Out[29]:

	rawPrediction	probability	prediction	outcome
0	[-1.5050483147898488, 13.901496693741956, -11....	[2.031244249495624e-07, 0.9971021006877452, 1....	1.0	1.0
1	[-0.9299621582305706, 12.771307835461082, -10....	[1.1141079917088333e-06, 0.9938326099360288, 9...	1.0	1.0
2	[5.457682300507879, -4.91104104052788, 1.52964...	[0.9767840078973328, 3.067038370100588e-05, 0....	0.0	0.0
3	[6.778654281045073, -10.202568439867305, -4.55...	[0.025538565332638954, 1.0773211172373325e-09,...	4.0	4.0
4	[5.827755062687981, -1.5974762581092157, -5.32...	[0.9331079041425342, 0.0005561537613624257, 1....	0.0	4.0

In [30]:
```
predictions_train = lrModel.transform(nslkdd_df)# predictions using the training dataset
accuracy_train = (predictions_train.filter(predictions_train.outcome == predictions_trai
    .count() / float(predictions_train.count()))

accuracy_test = (predictions.filter(predictions.outcome == predictions.prediction)
    .count() / float(predictions.count()))
print(f"Train Accuracy : {np.round(accuracy_train*100,2)}%")
print(f"Test Accuracy : {np.round(accuracy_test*100,2)}%")
```

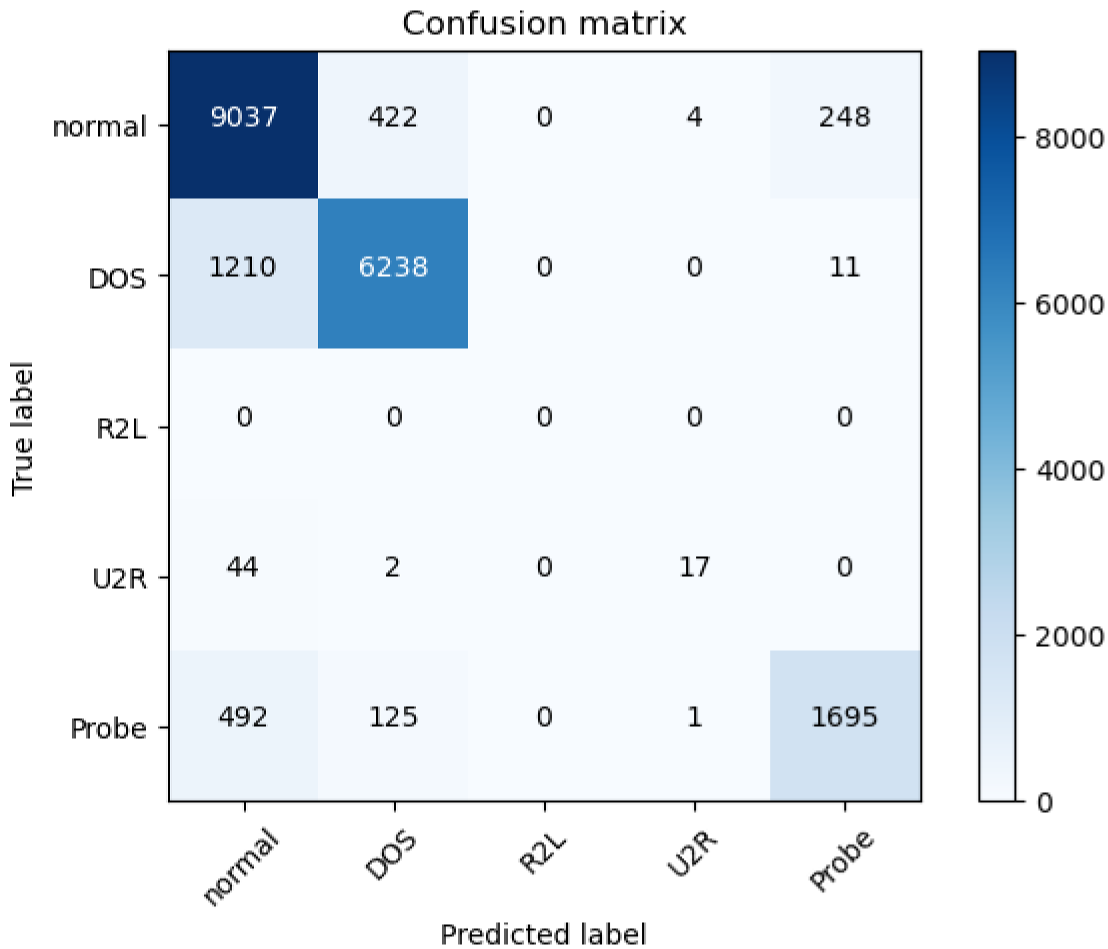
```
Train Accuracy : 98.76%
Test Accuracy : 75.92%
```

In [31]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix

import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
```

```
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [32]:
```
class_names=[0.0,1.0,2.3,3.0,4.0]
class_names_str=["normal","DOS", "R2L", "U2R", "Probe"]

outcome_true = predictions.select("outcome")
outcome_true = outcome_true.toPandas()

pred = predictions.select("prediction")
pred = pred.toPandas()

cnf_matrix = confusion_matrix(outcome_true, pred,labels=class_names)
#cnf_matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names_str,
                      title='Confusion matrix')
plt.show()
```

```
Confusion matrix, without normalization
[[9037  422    0    4  248]
 [1210 6238    0    0   11]
 [   0    0    0    0    0]
 [  44    2    0   17    0]
 [ 492  125    0    1 1695]]
```



Q1-3 Logistic Regression, hypertuning (Model 1)

In [33]:

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(predictionCol='prediction', labelCol='outc
print("Area under the curve/Accuracy is: ", evaluator.evaluate(predictions))
```

```
Area under the curve/Accuracy is:  0.7195155376924175
```

In [34]:

```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

lr = LogisticRegression(featuresCol = 'features', labelCol = 'outcome')

# Create ParamGrid for Cross Validation
lr_paramGrid = (ParamGridBuilder()
             .addGrid(lr.regParam, [0.0001, 0.001, 0.1])
             .addGrid(lr.maxIter, [10, 100, 1000])
             .build())

evaluator = MulticlassClassificationEvaluator(predictionCol='prediction', labelCol='outc

# Create a CrossValidator for multi-class classification
lr_cv = CrossValidator(estimator=lr, estimatorParamMaps=lr_paramGrid, evaluator=evaluato

# Fit the CrossValidator to your data
cv_model = lr_cv.fit(nslkdd_df)

# Make predictions on your test data
predictions = cv_model.transform(nslkdd_df_test)

# Evaluate the model's performance using the F1 score or other relevant metrics
```

```
f1_score = evaluator.evaluate(predictions)

print(predictions)
```

```
DataFrame[features: vector, outcome: double, outcome_category: string, rawPrediction: ve
ctor, probability: vector, prediction: double]
```

In [35]:
```
# Print the accuracy
print(f"Area under the curve/Accuracy is: {f1_score}")
```

```
Area under the curve/Accuracy is: 0.7158678181987708
```

Q1-2 Random Forest(Model 2)

In [36]:
```
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'outcome')
rf_model = rf.fit(nslkdd_df)
```

In [37]:
```
rf_prediction_train = rf_model.transform(nslkdd_df)
rf_prediction_test = rf_model.transform(nslkdd_df_test)

rf_accuracy_train = (rf_prediction_train.filter(rf_prediction_train.outcome == rf_predic
    .count()/ float(rf_prediction_train.count()))
rf_accuracy_test = (rf_prediction_test.filter(rf_prediction_test.outcome == rf_predictio
    .count() / float(rf_prediction_test.count()))

rf_auc = evaluator.evaluate(rf_prediction_test)

print(f"Train accuracy = {np.round(rf_accuracy_train*100,2)}%, test accuracy = {np.round
```

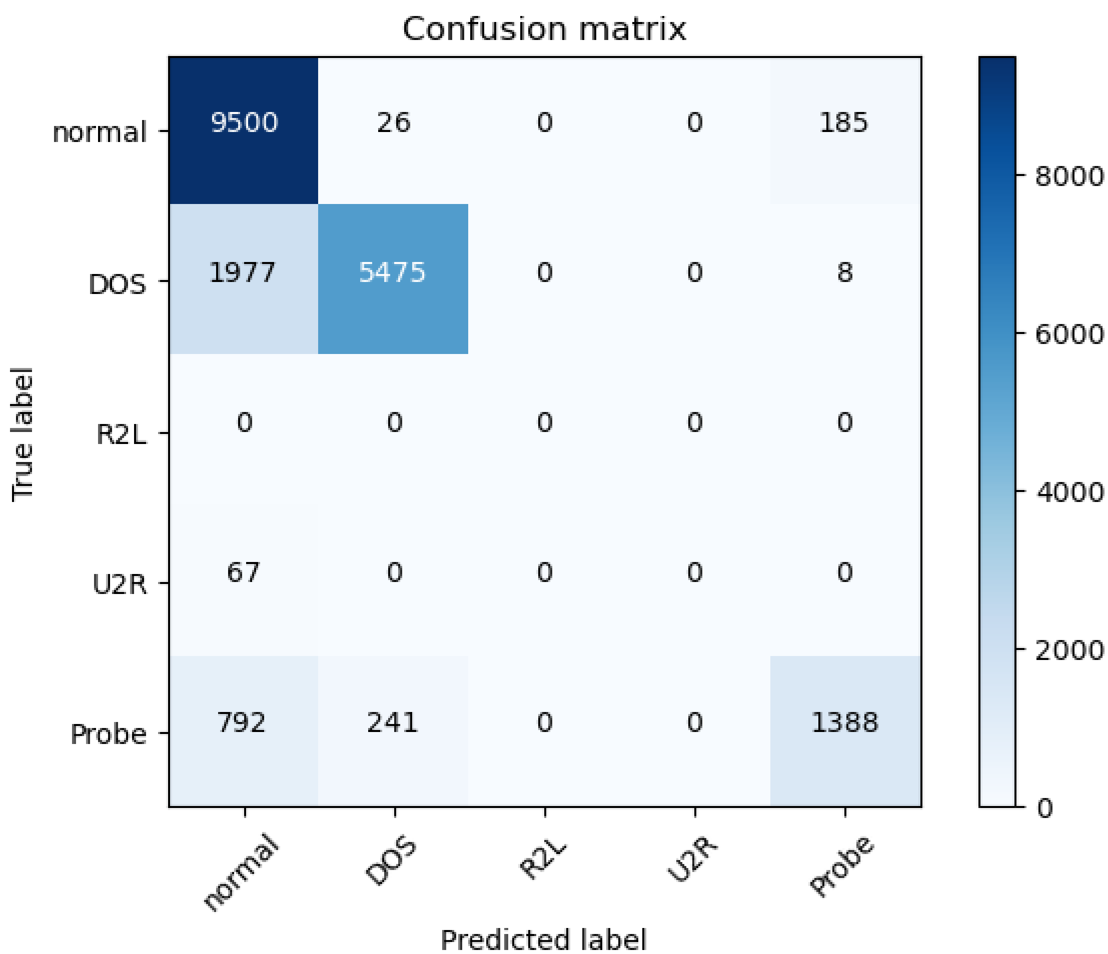
```
Train accuracy = 97.04%, test accuracy = 72.58%
```

In [38]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix

import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
```

```
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [39]:
```
class_names=[0.0,1.0,2.3,3.0,4.0]
class_names_str=["normal","DOS", "R2L", "U2R", "Probe"]

outcome_true = rf_prediction_test.select("outcome")
outcome_true = outcome_true.toPandas()

pred = rf_prediction_test.select("prediction")
pred = pred.toPandas()

cnf_matrix = confusion_matrix(outcome_true, pred,labels=class_names)
#cnf_matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names_str,
                      title='Confusion matrix')
plt.show()
```

```
Confusion matrix, without normalization
[[9500   26    0    0  185]
 [1977 5475    0    0    8]
 [   0    0    0    0    0]
 [  67    0    0    0    0]
 [ 792  241    0    0 1388]]
```



In [40]:
```
rf_paramGrid = (ParamGridBuilder()
               .addGrid(rf.maxDepth, [5, 10, 15])# maximum depth for each tree
               .addGrid(rf.numTrees,[10, 20, 40])# number of trues
               .build())
```

```python
rf_cv = CrossValidator(estimator=rf, estimatorParamMaps=rf_paramGrid,
                       evaluator=evaluator, numFolds=5)

rf_cv_model = rf_cv.fit(nslkdd_df)

rf_cv_prediction_test = rf_cv_model.transform(nslkdd_df_test)
rf_cv_auc = evaluator.evaluate(rf_cv_prediction_test)
```

```
23/10/26 23:37:31 WARN DAGScheduler: Broadcasting large task binary with size 1023.1 KiB
23/10/26 23:37:33 WARN DAGScheduler: Broadcasting large task binary with size 1103.0 KiB
23/10/26 23:37:33 WARN DAGScheduler: Broadcasting large task binary with size 1469.3 KiB
23/10/26 23:37:34 WARN DAGScheduler: Broadcasting large task binary with size 1874.9 KiB
23/10/26 23:37:34 WARN DAGScheduler: Broadcasting large task binary with size 1257.5 KiB
23/10/26 23:37:35 WARN DAGScheduler: Broadcasting large task binary with size 1064.4 KiB
23/10/26 23:37:37 WARN DAGScheduler: Broadcasting large task binary with size 1023.1 KiB
23/10/26 23:37:37 WARN DAGScheduler: Broadcasting large task binary with size 1247.6 KiB
23/10/26 23:37:37 WARN DAGScheduler: Broadcasting large task binary with size 1472.7 KiB
23/10/26 23:37:37 WARN DAGScheduler: Broadcasting large task binary with size 1673.4 KiB
23/10/26 23:37:37 WARN DAGScheduler: Broadcasting large task binary with size 1855.4 KiB
23/10/26 23:37:38 WARN DAGScheduler: Broadcasting large task binary with size 2009.6 KiB
23/10/26 23:37:38 WARN DAGScheduler: Broadcasting large task binary with size 1221.6 KiB
23/10/26 23:37:40 WARN DAGScheduler: Broadcasting large task binary with size 1103.0 KiB
23/10/26 23:37:40 WARN DAGScheduler: Broadcasting large task binary with size 1469.3 KiB
23/10/26 23:37:40 WARN DAGScheduler: Broadcasting large task binary with size 1874.9 KiB
23/10/26 23:37:41 WARN DAGScheduler: Broadcasting large task binary with size 2.2 MiB
23/10/26 23:37:41 WARN DAGScheduler: Broadcasting large task binary with size 2.7 MiB
23/10/26 23:37:41 WARN DAGScheduler: Broadcasting large task binary with size 3.0 MiB
23/10/26 23:37:42 WARN DAGScheduler: Broadcasting large task binary with size 3.4 MiB
23/10/26 23:37:42 WARN DAGScheduler: Broadcasting large task binary with size 3.7 MiB
23/10/26 23:37:43 WARN DAGScheduler: Broadcasting large task binary with size 2.1 MiB
23/10/26 23:37:48 WARN DAGScheduler: Broadcasting large task binary with size 1003.3 KiB
23/10/26 23:37:50 WARN DAGScheduler: Broadcasting large task binary with size 1084.2 KiB
23/10/26 23:37:50 WARN DAGScheduler: Broadcasting large task binary with size 1440.1 KiB
23/10/26 23:37:51 WARN DAGScheduler: Broadcasting large task binary with size 1838.7 KiB
23/10/26 23:37:51 WARN DAGScheduler: Broadcasting large task binary with size 1254.6 KiB
23/10/26 23:37:52 WARN DAGScheduler: Broadcasting large task binary with size 1052.4 KiB
23/10/26 23:37:54 WARN DAGScheduler: Broadcasting large task binary with size 1003.3 KiB
23/10/26 23:37:54 WARN DAGScheduler: Broadcasting large task binary with size 1210.7 KiB
23/10/26 23:37:54 WARN DAGScheduler: Broadcasting large task binary with size 1420.2 KiB
23/10/26 23:37:54 WARN DAGScheduler: Broadcasting large task binary with size 1617.9 KiB
23/10/26 23:37:55 WARN DAGScheduler: Broadcasting large task binary with size 1796.7 KiB
23/10/26 23:37:55 WARN DAGScheduler: Broadcasting large task binary with size 1954.3 KiB
23/10/26 23:37:55 WARN DAGScheduler: Broadcasting large task binary with size 1188.9 KiB
23/10/26 23:37:57 WARN DAGScheduler: Broadcasting large task binary with size 1084.2 KiB
23/10/26 23:37:57 WARN DAGScheduler: Broadcasting large task binary with size 1440.1 KiB
23/10/26 23:37:57 WARN DAGScheduler: Broadcasting large task binary with size 1838.7 KiB
23/10/26 23:37:58 WARN DAGScheduler: Broadcasting large task binary with size 2.2 MiB
23/10/26 23:37:58 WARN DAGScheduler: Broadcasting large task binary with size 2.6 MiB
23/10/26 23:37:58 WARN DAGScheduler: Broadcasting large task binary with size 3.0 MiB
23/10/26 23:37:59 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
23/10/26 23:37:59 WARN DAGScheduler: Broadcasting large task binary with size 3.6 MiB
23/10/26 23:38:00 WARN DAGScheduler: Broadcasting large task binary with size 2.1 MiB
23/10/26 23:38:05 WARN DAGScheduler: Broadcasting large task binary with size 1071.8 KiB
23/10/26 23:38:07 WARN DAGScheduler: Broadcasting large task binary with size 1105.2 KiB
23/10/26 23:38:07 WARN DAGScheduler: Broadcasting large task binary with size 1470.3 KiB
23/10/26 23:38:08 WARN DAGScheduler: Broadcasting large task binary with size 1874.4 KiB
23/10/26 23:38:08 WARN DAGScheduler: Broadcasting large task binary with size 1274.8 KiB
23/10/26 23:38:10 WARN DAGScheduler: Broadcasting large task binary with size 1064.4 KiB
23/10/26 23:38:11 WARN DAGScheduler: Broadcasting large task binary with size 1071.8 KiB
23/10/26 23:38:11 WARN DAGScheduler: Broadcasting large task binary with size 1297.1 KiB
23/10/26 23:38:11 WARN DAGScheduler: Broadcasting large task binary with size 1516.9 KiB
23/10/26 23:38:11 WARN DAGScheduler: Broadcasting large task binary with size 1717.9 KiB
23/10/26 23:38:12 WARN DAGScheduler: Broadcasting large task binary with size 1895.8 KiB
23/10/26 23:38:12 WARN DAGScheduler: Broadcasting large task binary with size 2.0 MiB
```

```
23/10/26 23:38:12 WARN DAGScheduler: Broadcasting large task binary with size 1237.8 KiB
23/10/26 23:38:14 WARN DAGScheduler: Broadcasting large task binary with size 1105.2 KiB
23/10/26 23:38:14 WARN DAGScheduler: Broadcasting large task binary with size 1470.3 KiB
23/10/26 23:38:14 WARN DAGScheduler: Broadcasting large task binary with size 1874.4 KiB
23/10/26 23:38:15 WARN DAGScheduler: Broadcasting large task binary with size 2.2 MiB
23/10/26 23:38:15 WARN DAGScheduler: Broadcasting large task binary with size 2.6 MiB
23/10/26 23:38:16 WARN DAGScheduler: Broadcasting large task binary with size 3.0 MiB
23/10/26 23:38:16 WARN DAGScheduler: Broadcasting large task binary with size 3.4 MiB
23/10/26 23:38:16 WARN DAGScheduler: Broadcasting large task binary with size 3.6 MiB
23/10/26 23:38:17 WARN DAGScheduler: Broadcasting large task binary with size 2.1 MiB
23/10/26 23:38:23 WARN DAGScheduler: Broadcasting large task binary with size 1025.8 KiB
23/10/26 23:38:25 WARN DAGScheduler: Broadcasting large task binary with size 1081.7 KiB
23/10/26 23:38:25 WARN DAGScheduler: Broadcasting large task binary with size 1440.2 KiB
23/10/26 23:38:25 WARN DAGScheduler: Broadcasting large task binary with size 1845.3 KiB
23/10/26 23:38:26 WARN DAGScheduler: Broadcasting large task binary with size 1246.5 KiB
23/10/26 23:38:27 WARN DAGScheduler: Broadcasting large task binary with size 1000.9 KiB
23/10/26 23:38:27 WARN DAGScheduler: Broadcasting large task binary with size 1071.7 KiB
23/10/26 23:38:28 WARN DAGScheduler: Broadcasting large task binary with size 1025.8 KiB
23/10/26 23:38:29 WARN DAGScheduler: Broadcasting large task binary with size 1236.9 KiB
23/10/26 23:38:29 WARN DAGScheduler: Broadcasting large task binary with size 1442.9 KiB
23/10/26 23:38:29 WARN DAGScheduler: Broadcasting large task binary with size 1629.7 KiB
23/10/26 23:38:29 WARN DAGScheduler: Broadcasting large task binary with size 1798.1 KiB
23/10/26 23:38:29 WARN DAGScheduler: Broadcasting large task binary with size 1939.6 KiB
23/10/26 23:38:30 WARN DAGScheduler: Broadcasting large task binary with size 1178.9 KiB
23/10/26 23:38:31 WARN DAGScheduler: Broadcasting large task binary with size 1081.7 KiB
23/10/26 23:38:32 WARN DAGScheduler: Broadcasting large task binary with size 1440.2 KiB
23/10/26 23:38:32 WARN DAGScheduler: Broadcasting large task binary with size 1845.3 KiB
23/10/26 23:38:32 WARN DAGScheduler: Broadcasting large task binary with size 2.2 MiB
23/10/26 23:38:33 WARN DAGScheduler: Broadcasting large task binary with size 2.6 MiB
23/10/26 23:38:33 WARN DAGScheduler: Broadcasting large task binary with size 3.0 MiB
23/10/26 23:38:33 WARN DAGScheduler: Broadcasting large task binary with size 3.3 MiB
23/10/26 23:38:34 WARN DAGScheduler: Broadcasting large task binary with size 3.6 MiB
23/10/26 23:38:34 WARN DAGScheduler: Broadcasting large task binary with size 2.1 MiB
23/10/26 23:38:39 WARN DAGScheduler: Broadcasting large task binary with size 1080.7 KiB
23/10/26 23:38:41 WARN DAGScheduler: Broadcasting large task binary with size 1098.4 KiB
23/10/26 23:38:42 WARN DAGScheduler: Broadcasting large task binary with size 1484.0 KiB
23/10/26 23:38:42 WARN DAGScheduler: Broadcasting large task binary with size 1915.4 KiB
23/10/26 23:38:42 WARN DAGScheduler: Broadcasting large task binary with size 1304.7 KiB
23/10/26 23:38:44 WARN DAGScheduler: Broadcasting large task binary with size 1009.6 KiB
23/10/26 23:38:45 WARN DAGScheduler: Broadcasting large task binary with size 1080.7 KiB
23/10/26 23:38:45 WARN DAGScheduler: Broadcasting large task binary with size 1316.5 KiB
23/10/26 23:38:45 WARN DAGScheduler: Broadcasting large task binary with size 1552.6 KiB
23/10/26 23:38:46 WARN DAGScheduler: Broadcasting large task binary with size 1764.7 KiB
23/10/26 23:38:46 WARN DAGScheduler: Broadcasting large task binary with size 1938.6 KiB
23/10/26 23:38:46 WARN DAGScheduler: Broadcasting large task binary with size 2.0 MiB
23/10/26 23:38:46 WARN DAGScheduler: Broadcasting large task binary with size 1253.1 KiB
23/10/26 23:38:48 WARN DAGScheduler: Broadcasting large task binary with size 1098.4 KiB
23/10/26 23:38:48 WARN DAGScheduler: Broadcasting large task binary with size 1484.0 KiB
23/10/26 23:38:48 WARN DAGScheduler: Broadcasting large task binary with size 1915.4 KiB
23/10/26 23:38:49 WARN DAGScheduler: Broadcasting large task binary with size 2.3 MiB
23/10/26 23:38:49 WARN DAGScheduler: Broadcasting large task binary with size 2.7 MiB
23/10/26 23:38:49 WARN DAGScheduler: Broadcasting large task binary with size 3.1 MiB
23/10/26 23:38:50 WARN DAGScheduler: Broadcasting large task binary with size 3.5 MiB
23/10/26 23:38:50 WARN DAGScheduler: Broadcasting large task binary with size 3.8 MiB
23/10/26 23:38:51 WARN DAGScheduler: Broadcasting large task binary with size 2.2 MiB
23/10/26 23:38:53 WARN DAGScheduler: Broadcasting large task binary with size 1022.1 KiB
23/10/26 23:38:53 WARN DAGScheduler: Broadcasting large task binary with size 1245.6 KiB
23/10/26 23:38:53 WARN DAGScheduler: Broadcasting large task binary with size 1461.6 KiB
23/10/26 23:38:54 WARN DAGScheduler: Broadcasting large task binary with size 1678.1 KiB
23/10/26 23:38:54 WARN DAGScheduler: Broadcasting large task binary with size 1882.8 KiB
23/10/26 23:38:54 WARN DAGScheduler: Broadcasting large task binary with size 2.0 MiB
23/10/26 23:38:55 WARN DAGScheduler: Broadcasting large task binary with size 1227.5 KiB
```

In [41]:
```
print(f"Before cross-validation and parameter tuning, AUC/accuracy={np.round(rf_auc,2)}"
print(f"After cross-validation and parameter tuning, AUC/accuracy={np.round(rf_cv_auc,2)
```

```
Before cross-validation and parameter tuning, AUC/accuracy=0.68
After cross-validation and parameter tuning, AUC/accuracy=0.71
```

Q1-4

Machine Learning Models:

1. Logistic Regression:

- Logistic Regression is a classic and interpretable model, often used as a baseline in classification tasks.
- Logistic Regression is well-suited for cases where model interpretability and feature importance are crucial.

1. Random Forest:

- Random Forest was selected for its versatility and ability to handle complex, non-linear relationships in data.
- it combines multiple decision trees to provide robust predictions.

Hyperparameters to Tune:

1. Logistic Regression:

- Regularization Parameter (regParam): The choice to tune the regularization parameter is driven by the need to control overfitting. By tuning regParam, we can adjust the balance between model complexity and generalization. For the regParam hyperparameter, a grid was constructed with values [[0.0001, 0.001, 0.1] to span a range from light to strong regularization.
- Maximum Iterations (maxIter): The maxIter parameter was selected to define the maximum number of iterations for model convergence during training. The maxIter parameter grid included values [10, 100, 1000], representing different maximum iteration settings. This allows for the exploration of the trade-off between training time and model convergence.

1. Random Forest:

- Number of Trees (numTrees): This hyperparameter was selected for tuning as it determines the size of the ensemble. The goal is to find the optimal number of trees to balance model complexity and predictive power. The numTrees hyperparameter grid was defined with values [5, 10, 15], covering different ensemble sizes.
- Maximum Depth (maxDepth): Tuning the maximum depth of individual decision trees was chosen to investigate different levels of tree complexity. The maxDepth parameter grid contained values [10, 20, 40] to analyze various levels of tree complexity.

- Hyperparameter tuning is crucial for fine-tuning both models to find the best configurations, as it significantly impacts model performance.
- Logistic Regression serves as a good starting point and may suffice for simpler tasks, while Random Forest is a versatile and powerful choice for more intricate classification problems where accuracy and capturing non-linear patterns

Q2-1 GBT

Gradient Boosted Trees is an ensemble learning method that combines multiple decision trees to create a powerful predictive model. It works by sequentially training decision trees in such a way that each new tree corrects the errors made by the previous ones. This is achieved by assigning more weight to misclassified instances and less weight to correctly classified instances. The trees are added iteratively, and the final prediction is made by combining the predictions of all trees. This process continues until a predefined number of trees are built or until a specified stopping condition is met. GBTs can be used for both regression and classification tasks and are known for their strong predictive performance.

Pros:

- Excellent Predictive Accuracy: GBTs are known for their high predictive accuracy and are among the top-performing algorithms in various machine learning competitions.
- Handles Non-linearity: GBTs can capture complex, non-linear relationships in the data and are robust to outliers.
- Feature Importance: They provide information about feature importance, allowing you to understand which features are most relevant in making predictions.
- Automatic Feature Selection: GBTs can perform automatic feature selection by giving more importance to informative features.

Cons:

- Complexity: GBTs can be more complex than some other models, which can lead to longer training times and increased memory usage.
- Overfitting: GBTs are susceptible to overfitting, especially if the model is allowed to become too complex. Careful tuning of hyperparameters is necessary to mitigate this.
- Less Interpretable: While GBTs provide feature importance scores, the model itself is less interpretable compared to simpler models like Logistic Regression.

hyper-parameters and their meanings:

1. gbt.maxDepth, [2, 4, 6]): This part of the code defines a grid for the maxDepth hyperparameter of the GBT model. The maxDepth controls the maximum depth or levels of the individual decision trees within the GBT ensemble. The grid includes three values: 2, 4, and 6. These values represent different levels of tree complexity that the hyperparameter tuning process will explore.

2. .addGrid(gbt.maxIter, [10, 20, 30]): Here, a grid is created for the maxIter hyperparameter. maxIter specifies the maximum number of iterations or the maximum number of trees to be added to the ensemble. The grid contains three values: 10, 20, and 30. These values represent different settings for the maximum number of iterations.

3. .addGrid(gbt.stepSize,[0.1, 0.01]): This part of the code defines a grid for the stepSize hyperparameter, which is equivalent to the learning rate in Gradient Boosting. The learning rate controls the step size at which the model updates during training. The grid includes two values: 0.1 and 0.01. These values represent different learning rates that will be explored during hyperparameter tuning.

Q2-2 GBT

In [42]:

```python
import pyspark
from pyspark.sql import SparkSession, SQLContext
from pyspark.ml import Pipeline,Transformer
from pyspark.ml.feature import Imputer,StandardScaler,StringIndexer,OneHotEncoder, Vecto

from pyspark.sql.functions import *
from pyspark.sql.types import *
import numpy as np

col_names = ["duration","protocol_type","service","flag","src_bytes",
"dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
"logged_in","num_compromised","root_shell","su_attempted","num_root",
"num_file_creations","num_shells","num_access_files","num_outbound_cmds",
"is_host_login","is_guest_login","count","srv_count","serror_rate",
"srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
"diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
"dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
"dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
"dst_host_rerror_rate","dst_host_srv_rerror_rate","class","difficulty"]

nominal_cols = ['protocol_type','service','flag']
binary_cols = ['land', 'logged_in', 'root_shell', 'su_attempted', 'is_host_login',
'is_guest_login']
continuous_cols = ['duration' ,'src_bytes', 'dst_bytes', 'wrong_fragment' ,'urgent', 'ho
'num_failed_logins', 'num_compromised', 'num_root' ,'num_file_creations',
'num_shells', 'num_access_files', 'num_outbound_cmds', 'count' ,'srv_count',
'serror_rate', 'srv_serror_rate' ,'rerror_rate' ,'srv_rerror_rate',
'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate' ,'dst_host_count',
'dst_host_srv_count' ,'dst_host_same_srv_rate' ,'dst_host_diff_srv_rate',
'dst_host_same_src_port_rate' ,'dst_host_srv_diff_host_rate',
'dst_host_serror_rate' ,'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
'dst_host_srv_rerror_rate']

class OutcomeCreater(Transformer): # this defines a transformer that creates the outcome

    def __init__(self):
        super().__init__()

    def _transform(self, dataset):
        label_to_binary = udf(lambda name: 0.0 if name == 'normal' else 1.0)
        output_df = dataset.withColumn('outcome', label_to_binary(col('class'))).drop("c
        output_df = output_df.withColumn('outcome', col('outcome').cast(DoubleType()))
        output_df = output_df.drop('difficulty')
        return output_df

class FeatureTypeCaster(Transformer): # this transformer will cast the columns as approp
    def __init__(self):
        super().__init__()

    def _transform(self, dataset):
        output_df = dataset
        for col_name in binary_cols + continuous_cols:
            output_df = output_df.withColumn(col_name,col(col_name).cast(DoubleType()))

        return output_df
class ColumnDropper(Transformer): # this transformer drops unnecessary columns
    def __init__(self, columns_to_drop = None):
        super().__init__()
        self.columns_to_drop=columns_to_drop
    def _transform(self, dataset):
        output_df = dataset
        for col_name in self.columns_to_drop:
            output_df = output_df.drop(col_name)
        return output_df

def get_preprocess_pipeline():
```

```
    # Stage where columns are casted as appropriate types
    stage_typecaster = FeatureTypeCaster()

    # Stage where nominal columns are transformed to index columns using StringIndexer
    nominal_id_cols = [x+"_index" for x in nominal_cols]
    nominal_onehot_cols = [x+"_encoded" for x in nominal_cols]
    stage_nominal_indexer = StringIndexer(inputCols = nominal_cols, outputCols = nominal

    # Stage where the index columns are further transformed using OneHotEncoder
    stage_nominal_onehot_encoder = OneHotEncoder(inputCols=nominal_id_cols, outputCols=n

    # Stage where all relevant features are assembled into a vector (and dropping a few)
    feature_cols = continuous_cols+binary_cols+nominal_onehot_cols
    corelated_cols_to_remove = ["dst_host_serror_rate","srv_serror_rate","dst_host_srv_s
                    "srv_rerror_rate","dst_host_rerror_rate","dst_host_srv_rerror_rate"
    for col_name in corelated_cols_to_remove:
        feature_cols.remove(col_name)
    stage_vector_assembler = VectorAssembler(inputCols=feature_cols, outputCol="vectoriz

    # Stage where we scale the columns
    stage_scaler = StandardScaler(inputCol= 'vectorized_features', outputCol= 'features'


    # Stage for creating the outcome column representing whether there is attack
    stage_outcome = OutcomeCreater()

    # Removing all unnecessary columbs, only keeping the 'features' and 'outcome' column
    stage_column_dropper = ColumnDropper(columns_to_drop = nominal_cols+nominal_id_cols+
        nominal_onehot_cols+ binary_cols + continuous_cols + ['vectorized_features'])
    # Connect the columns into a pipeline
    pipeline = Pipeline(stages=[stage_typecaster,stage_nominal_indexer,stage_nominal_one
        stage_vector_assembler,stage_scaler,stage_outcome,stage_column_dropper])
    return pipeline
```

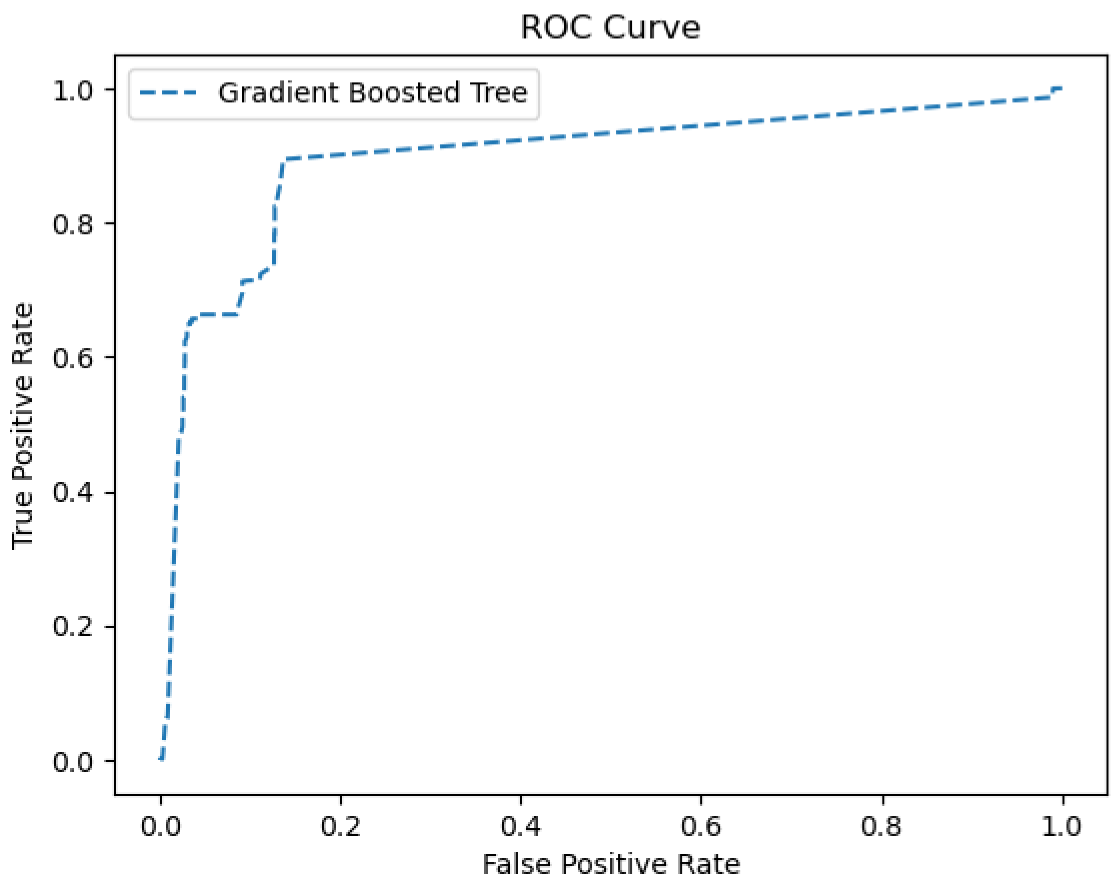
In [43]:

```
import os
import sys

os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable

spark = SparkSession.builder \
    .master("local[*]") \
    .appName("SystemsToolChains") \
    .getOrCreate()

nslkdd_raw = spark.read.csv('/Users/kiranprasadjp/Downloads/NSL-KDD/KDDTrain+.txt',heade
nslkdd_test_raw = spark.read.csv('/Users/kiranprasadjp/Downloads/NSL-KDD/KDDTest+.txt',h

preprocess_pipeline = get_preprocess_pipeline()
preprocess_pipeline_model = preprocess_pipeline.fit(nslkdd_raw)

nslkdd_df = preprocess_pipeline_model.transform(nslkdd_raw)
nslkdd_df_test = preprocess_pipeline_model.transform(nslkdd_test_raw)
```

In [44]:

```
from pyspark.ml.classification import GBTClassifier
from sklearn.metrics import roc_curve
import pyspark.sql.functions as F
import pyspark.sql.types as T
import numpy
from matplotlib import pyplot as plt

gbt = GBTClassifier(featuresCol = 'features', labelCol = 'outcome')

gbt_model = gbt.fit(nslkdd_df) # fit the logistic regression model to the training datas
```

```
gbt_predictions = gbt_model.transform(nslkdd_df_test) # make predictions
outcome_true = gbt_predictions.select('outcome').toPandas() # the true outcome label as


# make the ROC curve
gbt_pred_prob = gbt_predictions.select("probability")
to_array = F.udf(lambda v: v.toArray().tolist(), T.ArrayType(T.FloatType()))
gbt_pred_prob = gbt_pred_prob.withColumn('probability', to_array('probability'))
gbt_pred_prob = gbt_pred_prob.toPandas()
gbt_pred_prob_nparray = np.array(gbt_pred_prob['probability'].values.tolist())

gbt_fpr, gbt_tpr, gbt_thresholds = roc_curve(outcome_true, gbt_pred_prob_nparray[:,1])

plt.plot(gbt_fpr, gbt_tpr, linestyle='--', label='Gradient Boosted Tree')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()

# calculate the area under curve
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(rawPredictionCol='rawPrediction',
    labelCol='outcome', metricName='areaUnderROC')

print("Area under the curve is: ", evaluator.evaluate(gbt_predictions))
```




```
Area under the curve is:  0.8955875210476721
```

In [45]:
```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator

gbt = GBTClassifier(featuresCol = 'features', labelCol = 'outcome')

# Create ParamGrid for Cross Validation
gbt_paramGrid = (ParamGridBuilder()
```

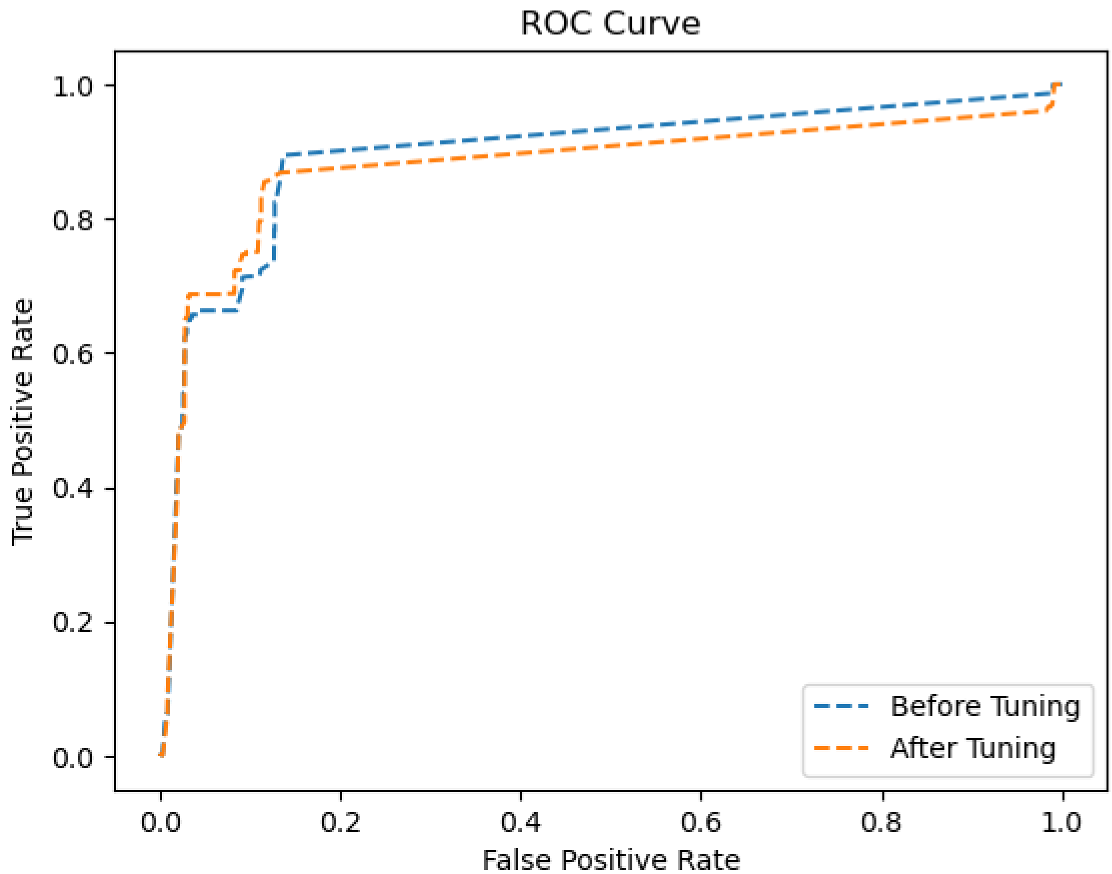
```
            .addGrid(gbt.maxDepth, [2, 4, 6])
            .addGrid(gbt.maxIter, [10, 20, 30])
            .addGrid(gbt.stepSize,[0.1, 0.01])
            .build())

evaluator = BinaryClassificationEvaluator(rawPredictionCol='rawPrediction',
    labelCol='outcome', metricName='areaUnderROC')

gbt_cv = CrossValidator(estimator=gbt, estimatorParamMaps=gbt_paramGrid,
                    evaluator=evaluator, numFolds=5)
```

In [46]:
```
gbt_cv_model = gbt_cv.fit(nslkdd_df)
```

In [47]:
```
gbt_cv_prediction_test = gbt_cv_model.transform(nslkdd_df_test)
print('Test Area Under ROC (AUC) after Cross-Validation:', evaluator.evaluate(gbt_cv_pre
print('Test Area Under ROC (AUC) before Cross-Validation:', evaluator.evaluate(gbt_predi
```

```
Test Area Under ROC (AUC) after Cross-Validation: 0.8782859791751592
Test Area Under ROC (AUC) before Cross-Validation: 0.8955875210476721
```

In [49]:
```
gbt_cv_pred_prob = (gbt_cv_prediction_test.select("probability").
        withColumn('probability', to_array('probability')).toPandas())
gbt_cv_pred_prob = np.array(gbt_cv_pred_prob['probability'].values.tolist())

gbt_cv_fpr, gbt_cv_tpr, gbt_cv_thresholds = roc_curve(outcome_true, gbt_cv_pred_prob[:,1
# plot the roc curve for the model
plt.plot(gbt_fpr, gbt_tpr, linestyle='--', label='Before Tuning')
plt.plot(gbt_cv_fpr, gbt_cv_tpr, linestyle='--', label='After Tuning')

# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
# show the legend
plt.legend()
# show the plot
plt.show()
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Before Tuning
After Tuning

In [ ]: