Big Data Assignment-3

Hruday Vishal Kanna Anand

1006874517

Part-a

1.

Intrusion detection system is either a hardware or software monitor that analysis the incoming data to detect any attack on the system or the network. This can be done by-

- Signature based detection- which searches the data for the signatures of known attacks.
- Anomaly based detection- which looks at user activity to figure out if there are any abnormal behaviors.
- Hybrid detection- it is a mix of the 2 or more methods.

It is possible to implement an Intrusion Detection System on this dataset as the data is the log data of connections made to a network, each log contains various network features. From which we can detect attacks or intrusions.

Workflow-

1. Load dataset and export it into Resilient Distributed Datasets (RDD) and DataFrame in Spark.
2. Data preprocessing- we have to make sure all the data is in the correct format that is required by our model-(numeric). Then we can standardize them as this will get us more reliable results.
3. Feature selection- we need to select the features that produce high accuracy and eliminates diversions. The Chi-Squared test of independence is used to decide which features to select.
4. Train Spark-Chi-SVM - split the dataset into train and test- use the train dataset to train the Spark-Chi-SVM model.
5. Test and evaluate the model- use the test dataset to get the results of the model and evaluate it using a performance metric such as Area under Curve or Area under Precision-Recall Curve.

2.

```
1  import urllib.request
2  urllib.request.urlretrieve("http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz", "/tmp/kddcup_data.gz")
3  dbutils.fs.mv("file:/tmp/kddcup_data.gz", "dbfs:/kdd/kddcup_data.gz")
4  display(dbutils.fs.ls("dbfs:/kdd"))
```

▸ (3) Spark Jobs

	path	name	size
1	dbfs:/kdd/kddcup_data.gz	kddcup_data.gz	2144903

Showing all 1 rows.

Command took 5.83 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 11:42:13 AM on firstCluster

We can see the data has been loaded into the DBFS.

3.

```
1  data = spark.sparkContext.textFile("dbfs:/kdd/kddcup_data.gz")
2  data.take(10)
```

▸ (1) Spark Jobs

```
Out[5]: ['0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,19,19,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,235,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,29,29,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,219,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,39,39,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,217,2032,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,49,49,1.00,0.00,0.02,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,217,2032,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,59,59,1.00,0.00,0.02,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,212,1940,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,2,0.00,0.00,0.00,0.00,1.00,0.00,1.00,1,69,1.00,0.00,1.00,0.04,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,159,4087,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,5,5,0.00,0.00,0.00,0.00,1.00,0.00,0.00,11,79,1.00,0.00,0.09,0.04,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,210,151,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,8,89,1.00,0.00,0.12,0.04,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,212,786,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,8,99,1.00,0.00,0.12,0.05,0.00,0.00,0.00,0.00,normal.']
```

Command took 1.78 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 11:42:17 AM on firstCluster

The data is stored in a structured manner where each row is one record, and each column is represented by the comma separated elements in each row. This dataset follows a table like structure.

4.

```
1  data2=data.map(lambda x: x.split(","))
2  print("number of features-")
3  print(len(data2.take(1)[0]))
4  data2.take(1)
```

▸ (2) Spark Jobs

```
number of features-
42
Out[6]: [['0',
  'tcp',
  'http',
  'SF',
  '181',
  '5450',
  '0',
  '0',
  '0',
  '0',
  '0',
  '1',
  '0',
  '0',
  '0',
  '0',
  '0',
  '0',
  '0',
  '0',
```

Command took 0.77 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 11:42:22 AM on firstCluster

The number of features is 41 plus 1 label leading to 42 columns.

5.

```
data3=data2.map(lambda x: [(x[0]),x[1],x[2],int(x[4]),int(x[5]),x[3],x[41]])
data3.take(10)
```

▸ (1) Spark Jobs

```
Out[29]: [[0, 'tcp', 'http', 181, 5450, 'SF', 'normal.'],
 [0, 'tcp', 'http', 239, 486, 'SF', 'normal.'],
 [0, 'tcp', 'http', 235, 1337, 'SF', 'normal.'],
 [0, 'tcp', 'http', 219, 1337, 'SF', 'normal.'],
 [0, 'tcp', 'http', 217, 2032, 'SF', 'normal.'],
 [0, 'tcp', 'http', 217, 2032, 'SF', 'normal.'],
 [0, 'tcp', 'http', 212, 1940, 'SF', 'normal.'],
 [0, 'tcp', 'http', 159, 4087, 'SF', 'normal.'],
 [0, 'tcp', 'http', 210, 151, 'SF', 'normal.'],
 [0, 'tcp', 'http', 212, 786, 'SF', 'normal.']]
```

Command took 0.41 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:06:17 PM on firstCluster

```
columns=["duration", "protocol_type", "service", "src_bytes", "dst_bytes", "flag","label"]
kdd=spark.createDataFrame(data3,columns)
kdd.printSchema()
```

▸ (1) Spark Jobs

▸ kdd: pyspark.sql.dataframe.DataFrame = [duration: long, protocol_type: string ... 5 more fields]

```
root
 |-- duration: long (nullable = true)
 |-- protocol_type: string (nullable = true)
 |-- service: string (nullable = true)
 |-- src_bytes: long (nullable = true)
 |-- dst_bytes: long (nullable = true)
 |-- flag: string (nullable = true)
 |-- label: string (nullable = true)
```

Command took 0.41 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:06:42 PM on firstCluster

```
1  display(kdd.limit(10))
```

▸ (1) Spark Jobs

	duration	protocol_type	service	src_bytes	dst_bytes	flag	label
1	0	tcp	http	181	5450	SF	normal.
2	0	tcp	http	239	486	SF	normal.
3	0	tcp	http	235	1337	SF	normal.
4	0	tcp	http	219	1337	SF	normal.
5	0	tcp	http	217	2032	SF	normal.
6	0	tcp	http	217	2032	SF	normal.
7	0	tcp	http	212	1940	SF	normal.
8	0	tcp	http	159	4087	SF	normal.
9	0	tcp	http	210	151	SF	normal.
10	0	tcp	http	212	786	SF	normal.

Showing all 10 rows.

Command took 0.56 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:06:50 PM on firstCluster

6.

Total number of connections based on Protocol type-

```
1  kdd.select("protocol_type").distinct().count()
```

▸ (3) Spark Jobs

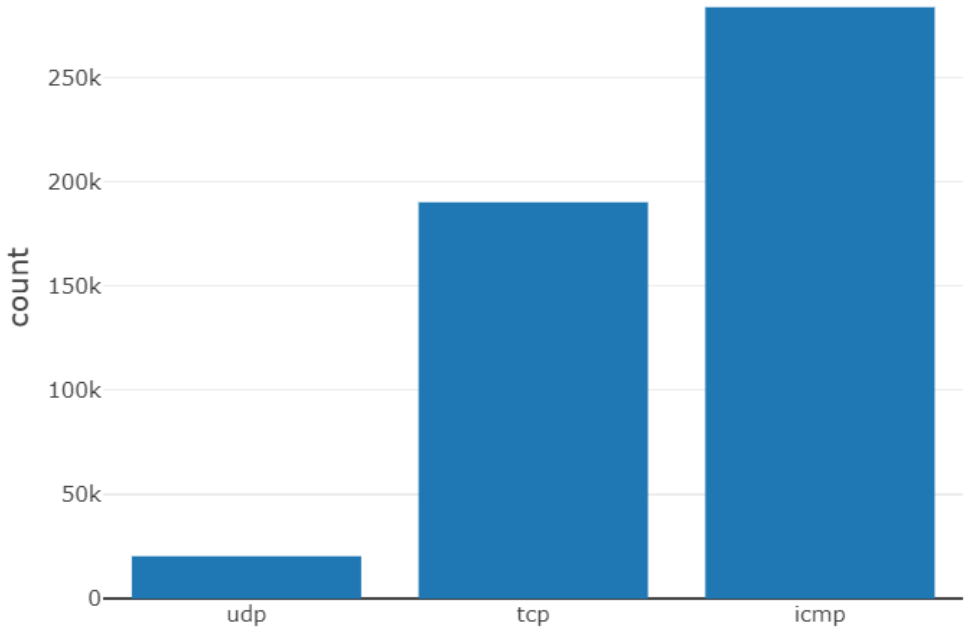
```
Out[34]: 3
```

Command took 6.01 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:08:19 PM on firstCluster

Cmd 10

```
1  display(kdd.groupBy("protocol_type").count().sort("count"))
```

▸ (2) Spark Jobs



▸ (2) Spark Jobs

	protocol_type	count
1	udp	20354
2	tcp	190065
3	icmp	283602

Showing all 3 rows.

Total number of connections based on Service-

```
1  kdd.select("service").distinct().count()
```

▸ (3) Spark Jobs

```
Out[35]: 66
```

Command took 6.53 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3,

Cmd 12

```
1  display(kdd.groupBy("service").count().sort("count"))
```

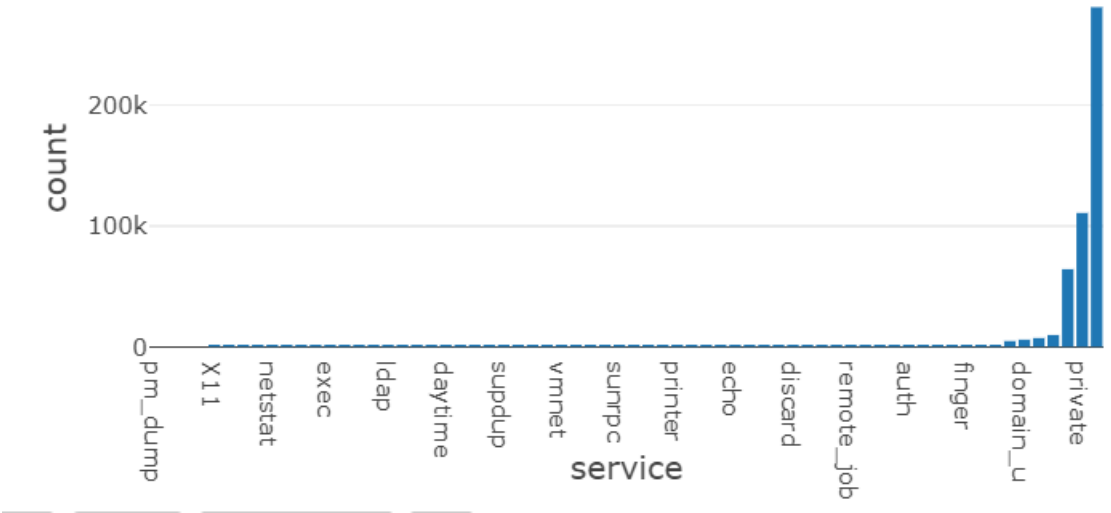
▸ (2) Spark Jobs

	service	count
1	pm_dump	1
2	red_i	1
3	tftp_u	1
4	tim_i	7
5	X11	11
6	urh_i	14
7	IRC	43
8	Z39_50	92
9	netstat	95
10	ctf	97

Showing all 66 rows.

Command took 5.78 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3,

▸ (2) Spark Jobs



7.

Label-

```
1  kdd.select("label").distinct().count()
```

▸ (3) Spark Jobs

```
Out[48]: 23
```

Command took 5.36 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:13:5

Cmd 25

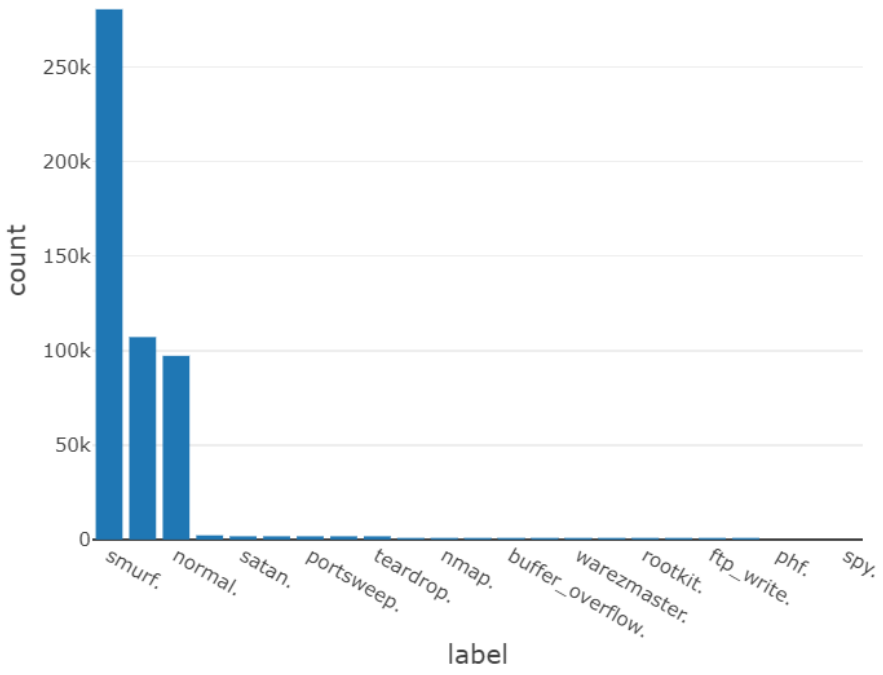
```
1  display(kdd.groupBy("label").count().sort(col("count").desc()))
```

▸ (2) Spark Jobs

	label	count
1	smurf.	280790
2	neptune.	107201
3	normal.	97278
4	back.	2203
5	satan.	1589
6	ipsweep.	1247
7	portsweep.	1040

Showing all 23 rows.

▸ (2) Spark Jobs



We can see that there in total of 23 distinct labels. The label with the most number of occurrence is “smurf” and the label with least is “spy”. The label “normal” is third highest in the bar graph.

Flag-

Cmd 22

```
1  kdd.select("flag").distinct().count()
```

▸ (3) Spark Jobs

Out[46]: 11

Command took 5.67 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:13:

Cmd 23

```
1  display(kdd.groupBy("flag").count().sort(col("count").desc()))
```

▸ (2) Spark Jobs

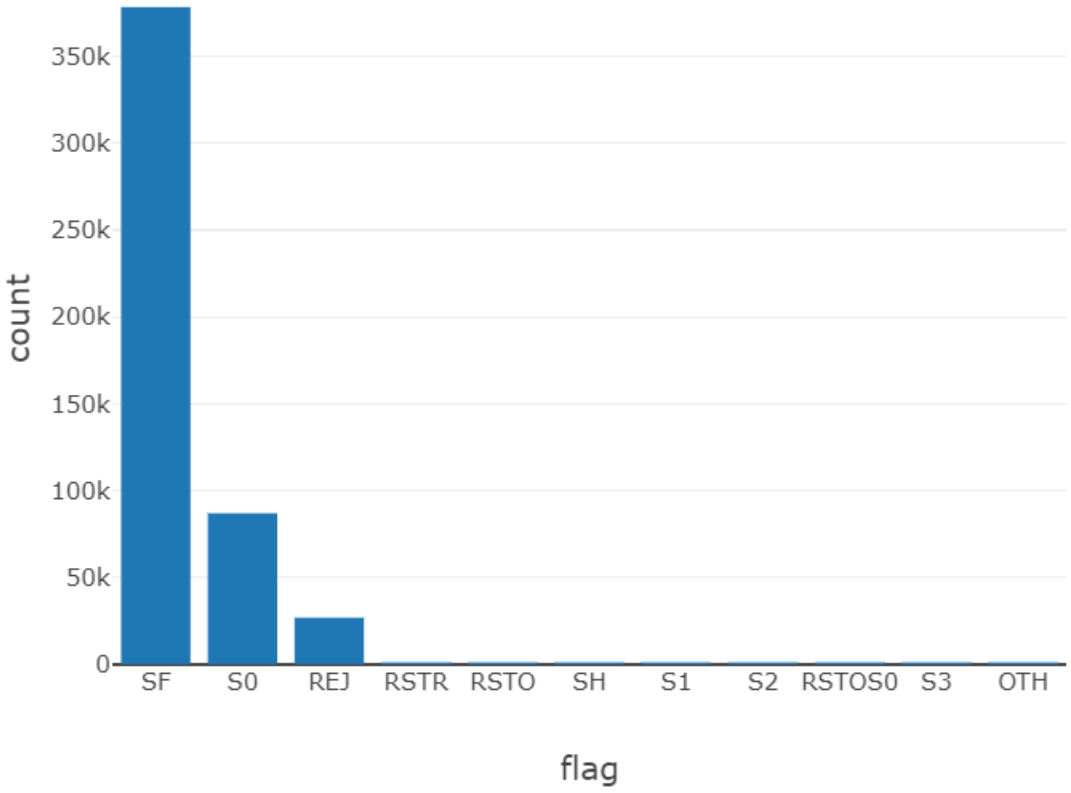
	flag	count
1	SF	378440
2	S0	87007
3	REJ	26875
4	RSTR	903
5	RSTO	579
6	SH	107
7	S1	57

Showing all 11 rows.

Command took 5.41 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:13:

Cmd 24

▸ (2) Spark Jobs



Flag has 11 distinct categories. “SF” has the highest count from the bar graph with well over 350,000 followed by “S0” and “REJ”. “OTH” has the least number of connections.

Duration-

```
1  kdd.select("duration").distinct().count()
```

▸ (3) Spark Jobs

```
Out[16]: 2495
```

```
Command took 6.18 seconds -- by vishal.kanna@mail.utoronto.ca at 7/6/2021,
```

```
Cmd 14
```

```
1  display(kdd.select("duration").agg({"duration": "avg"}))
```

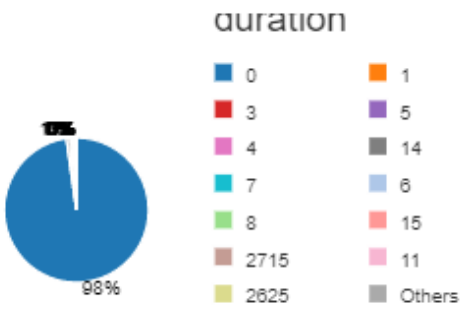
▸ (2) Spark Jobs

	avg(duration)
1	47.97930249928647

Showing all 1 rows.

```
Command took 5.72 seconds -- by vishal.kanna@mail.utoronto.ca at 7/6/2021,
```

▸ (4) Spark Jobs



Aggregated (by sum) in the backend.
Truncated results, showing first 1000 rows.

From the data above we can see that the average duration of the connections is 47.9. We can also see from the pie chart that 98% of the connections have a duration of 0 and only the remaining 2% have a duration greater than 0.

8.

Converting all the labels into either normal or attack.

```
from pyspark.sql.functions import when
kdd_final = kdd.withColumn("label", when(kdd["label"] == "normal.", "normal").otherwise("attack"))
```

kdd_final: pyspark.sql.dataframe.DataFrame = [duration: long, protocol_type: string ... 5 more fields]

Command took 0.17 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:27:52 PM on firstCluster

Cmd 27

```
display(kdd_final.groupBy("label").count().sort(col("count").desc()))
```

(2) Spark Jobs

	label	count
1	attack	396743
2	normal	97278

Showing all 2 rows.

Command took 5.17 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:28:01 PM on firstCluster

Encoding all categorical data and adding them to stages for the pipeline model. Also adding the numerical data to the vector assembler along with the encoded categorical data.

```
cols = kdd_final.columns
categoricalColumns = ["protocol_type", "service", "flag"]
stages = [] # stages in Pipeline
for categoricalCol in categoricalColumns:
    # Category Indexing with StringIndexer
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + "Index")
    # Use OneHotEncoder to convert categorical variables into binary SparseVectors
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    # Add stages.  These are not run here, but will run all at once later on.
    stages += [stringIndexer, encoder]

```

Command took 0.13 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:29:48 PM on firstCluster

Cmd 29

```
label_stringIdx = StringIndexer(inputCol="label", outputCol="labels")
stages += [label_stringIdx]
```

Command took 0.04 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:31:01 PM on firstCluster

Cmd 30

```
numericCols = ["duration", "src_bytes", "dst_bytes"]
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]

```

Command took 0.04 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:31:20 PM on firstCluster

Using pipeline to create our new dataset that will be used in our machine learning model.

```
1  from pyspark.ml.classification import LogisticRegression
2
3  partialPipeline = Pipeline().setStages(stages)
4  pipelineModel = partialPipeline.fit(kdd_final)
5  preppedDataDF = pipelineModel.transform(kdd_final)
```

▸ (8) Spark Jobs

▸ preppedDataDF: pyspark.sql.dataframe.DataFrame = [duration: long, protocol_type: string ... 13 more fields]

Command took 23.92 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:31:35 PM on firstCluster

Cmd 32

```
1  display(preppedDataDF)
```

▸ (1) Spark Jobs

	label	protocol_typeIndex	protocol_typeclassVec	serviceIndex	serviceclassVec	flagIndex	flagclassVec	labels	features
1	normal	1	[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 181, 5450]]
2	normal	1	[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 239, 486]]
3	normal	1	[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 235, 1337]]
4	normal	1	[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 219, 1337]]
5	normal	1	[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 217, 2032]]
6	normal	1	[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 217, 2032]]

Truncated results, showing first 1000 rows.

Command took 0.77 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:32:03 PM on firstCluster

Splitting our new dataset into train and test. Then training our logistics regression model with the training data. Using this model, we will make predictions using the test data.

```
1  (trainingData, testData) = dataset.randomSplit([0.8, 0.2], seed=100)
2  print(trainingData.count())
3  print(testData.count())
```

▸ (4) Spark Jobs

▸ trainingData: pyspark.sql.dataframe.DataFrame = [labels: double, features: udt ... 7 more fields]

▸ testData: pyspark.sql.dataframe.DataFrame = [labels: double, features: udt ... 7 more fields]

```
395596
98425
```

Command took 16.50 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:52:07 PM on firstCluster

Cmd 36

```
1  lr = LogisticRegression(labelCol="labels", featuresCol="features", maxIter=10)
2  lrModel = lr.fit(trainingData)
```

▸ (12) Spark Jobs

Command took 15.86 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:52:30 PM on firstCluster

Cmd 37

```
1  predictions = lrModel.transform(testData)
```

▸ predictions: pyspark.sql.dataframe.DataFrame = [labels: double, features: udt ... 10 more fields]

Command took 0.10 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 12:53:37 PM on firstCluster

Cmd 38

Evaluating these predictions with AUC (area under ROC curve) metric.

```
1  from pyspark.ml.evaluation import BinaryClassificationEvaluator
2  evaluator = BinaryClassificationEvaluator( rawPredictionCol='rawPrediction',labelCol='labels', metricName='areaUnderROC')
```

```
Command took 0.04 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 1:00:34 PM on firstCluster
```

```
Cmd 40
```

```
1  evaluator.evaluate(predictions)
```

```
▸ (3) Spark Jobs
Out[78]: 0.9962427542757365
Command took 12.82 seconds -- by vishal.kanna@mail.utoronto.ca at 7/3/2021, 1:01:36 PM on firstCluster
```

We can see that the evaluation metric used is AUC- from which we get a value of 0.996. AUC values range from 0-1 where “1” signifies 100% accuracy, hence we can confirm the model created is competent in distinguishing between a normal connection and an attack.

We use logistic regression as we need to create a binary classification (attack or normal) model with an input of 6 features. Logistic regression is a simple algorithm for that can classify classes based on the input features.

Part-b

1.

i) yes, Azure PaaS solutions continuously update and give us new features and development tools to cut coding time and make development easier.

ii) Yes, PaaS offers in built high availability to support our application so that it may be running at all times.

2. d, relational database has a defined schema, this rigid structure requires a strong consistency with the data structure.

3. c, when implementing SaaS we are only responsible for installing the software, all the other aspects are taken care by the provider.

4.

i) No, a hybrid cloud model can be achieved by migrating from a public cloud model also.

ii) Yes, a company can extend the capacity of its internal network by using the public cloud. We can connect the on premise network to the cloud storage with a secured VPN connection to increase capacity.

iii) No, different levels of access may be managed with the help of a management group to give different levels of access.

5.

a) Fault tolerance, it ensure the service is running even though faults may occur with the help of many techniques such as availability zones, availability sets, etc.

b) disaster recovery, it ensures our services can be recovered after a failure with the help of a backup and recovery plan.

c) Dynamic Scalability, ensures a service can quickly scale in or out depending on the network load or the compute required.

d) Low latency, ensures a service can be quickly accessed with the help of an internet connection so that there are minimal delays.