Assignment 3 Report

Aditya Jain

1 Part A

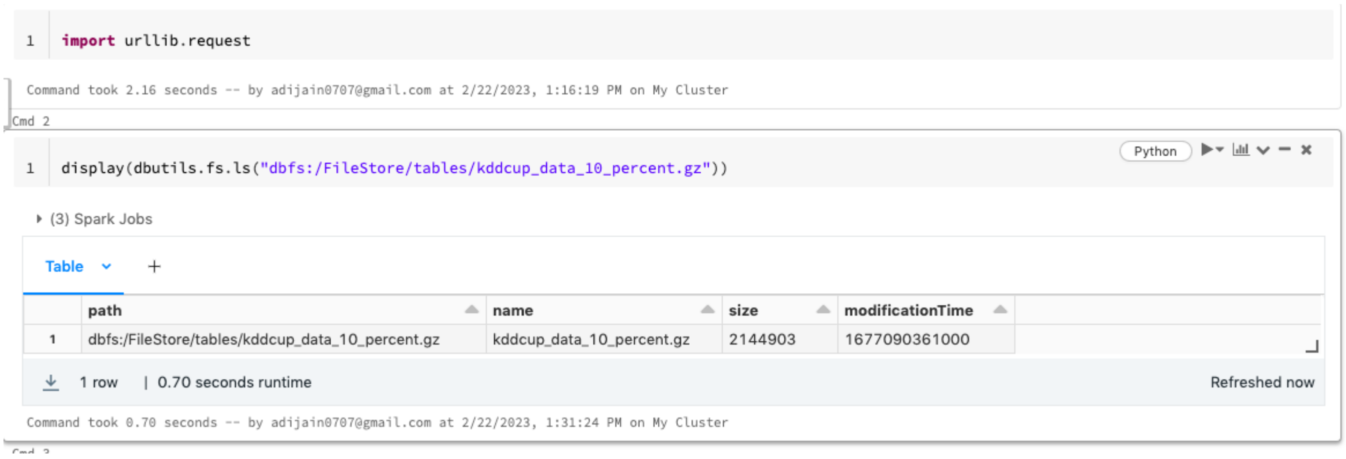
1.1 Part A.1

An Intrusion Detection System (IDS) checks, monitors, and analyzes unauthorized and malicious attempts to gain access to a system or a network. Yes, we can implement IDS on the KDD Cup 99 dataset. The workflow for implementing an IDS is as follows:

1. Data processing: The first step is to process and standardize the input data. The categorical data is converted to numerical entries, followed by scaling to zero mean and unit variance.
2. Feature selection: Chi-Squared test of independence is performed to remove redundant and irrelevant features from the dataset. This step gives us important features that should be used for model training.
3. Model training: Support vector machine (SVM) is used for classification, which classifies the data into different classes by an N-dimensional hyperplane.
4. Model evaluation: Finally, the model's accuracy is evaluated on the test dataset.

1.2 Part A.2

Since retrieving from *urllib* was throwing an error, I directly uploaded the data to Databricks filesystem (as explained in the lecture). Figure 1 shows the display result.



```
import urllib.request
```

```
Command took 2.16 seconds -- by adijain0707@gmail.com at 2/22/2023, 1:16:19 PM on My Cluster
```

Cmd 2

```
display(dbutils.fs.ls("dbfs:/FileStore/tables/kddcup_data_10_percent.gz"))
```

	path	name	size	modificationTime
1	dbfs:/FileStore/tables/kddcup_data_10_percent.gz	kddcup_data_10_percent.gz	2144903	1677090361000

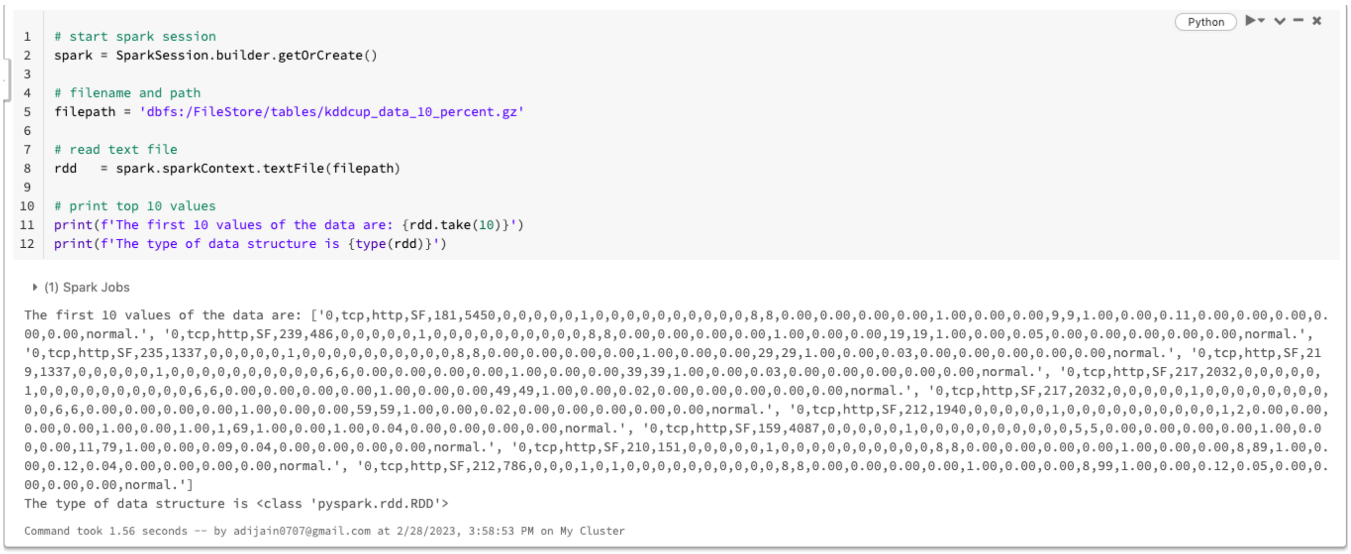
1 row | 0.70 seconds runtime Refreshed now

```
Command took 0.70 seconds -- by adijain0707@gmail.com at 2/22/2023, 1:31:24 PM on My Cluster
```

Figure 1: Result of Part A.2

1.3 Part A.3

Figure 2 shows the 10 values of the data and its type.



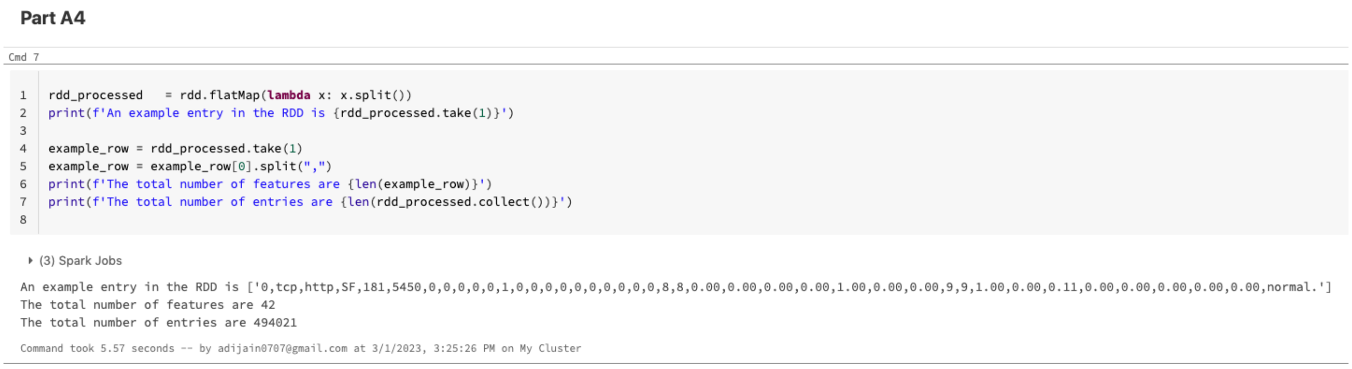
```python
# start spark session
spark = SparkSession.builder.getOrCreate()

# filename and path
filepath = 'dbfs:/FileStore/tables/kddcup_data_10_percent.gz'

# read text file
rdd   = spark.sparkContext.textFile(filepath)

# print top 10 values
print(f'The first 10 values of the data are: {rdd.take(10)}')
print(f'The type of data structure is {type(rdd)}')
```

```
The first 10 values of the data are: ['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,21
9,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,39,39,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,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,159,4087,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,5,5,0.00,0.00,0.00,0.00,1.00,0.0
0,0.00,11,79,1.00,0.00,0.09,0.04,0.00,0.00,0.00,0.00,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,212,786,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,8,99,1.00,0.00,0.12,0.05,0.00,0.
00,0.00,0.00,normal.']
The type of data structure is <class 'pyspark.rdd.RDD'>
Command took 1.56 seconds -- by adijain0707@gmail.com at 2/28/2023, 3:58:53 PM on My Cluster
```

Figure 2: Result of Part A.3

1.4 Part A.4

Figure 3 shows an example entry of the data. There are a total of 41 features, with an additional column determining the label of connection type (normal or attack). There are a total of 494,021 row entries in the data.



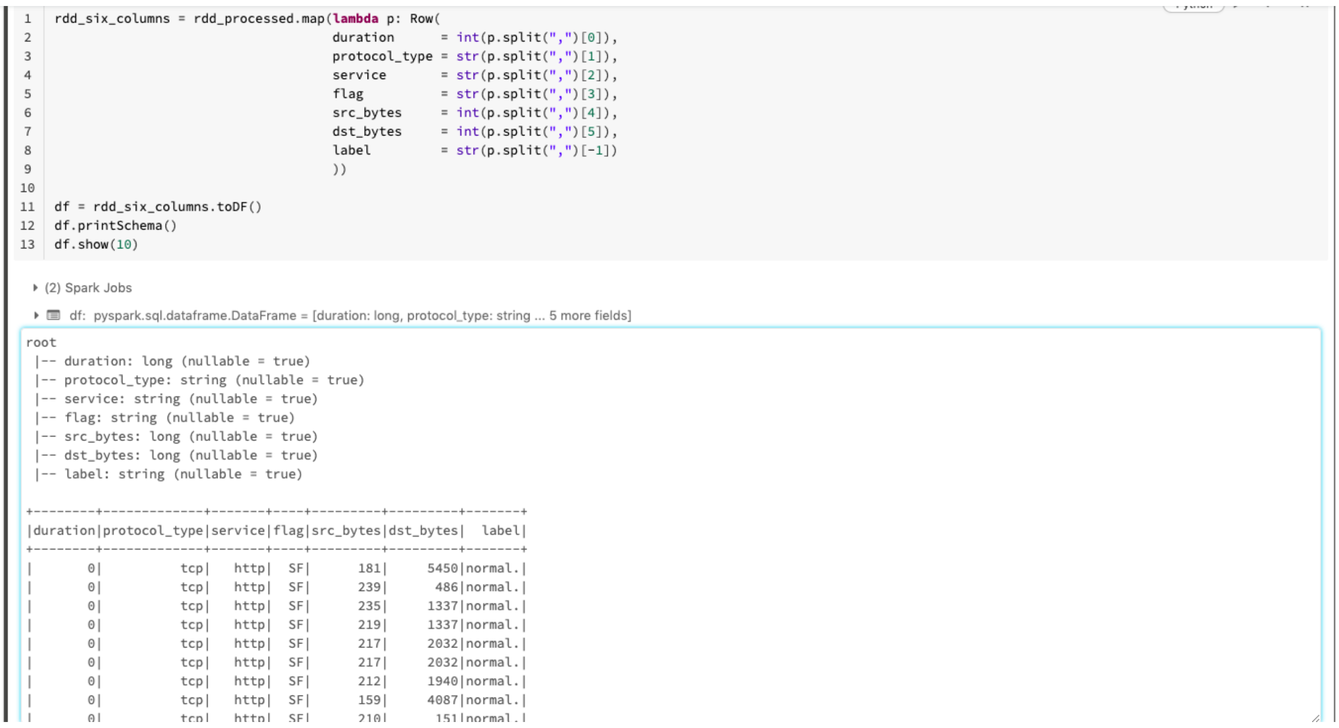
```python
rdd_processed   = rdd.flatMap(lambda x: x.split())
print(f'An example entry in the RDD is {rdd_processed.take(1)}')

example_row = rdd_processed.take(1)
example_row = example_row[0].split(",")
print(f'The total number of features are {len(example_row)}')
print(f'The total number of entries are {len(rdd_processed.collect())}')

```

```
An example entry in the RDD is ['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.']
The total number of features are 42
The total number of entries are 494021
Command took 5.57 seconds -- by adijain0707@gmail.com at 3/1/2023, 3:25:26 PM on My Cluster
```

Figure 3: Result of Part A.4

1.5 Part A.5

A new RDD and dataframe is built using the given six columns and label. Figure 4 shows the dataframe schema and ten values.



```python
rdd_six_columns = rdd_processed.map(lambda p: Row(
                                    duration      = int(p.split(",")[0]),
                                    protocol_type = str(p.split(",")[1]),
                                    service       = str(p.split(",")[2]),
                                    flag          = str(p.split(",")[3]),
                                    src_bytes     = int(p.split(",")[4]),
                                    dst_bytes     = int(p.split(",")[5]),
                                    label         = str(p.split(",")[-1])
                                    ))

df = rdd_six_columns.toDF()
df.printSchema()
df.show(10)
```

```
(2) Spark Jobs
df: 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)
 |-- flag: string (nullable = true)
 |-- src_bytes: long (nullable = true)
 |-- dst_bytes: long (nullable = true)
 |-- label: string (nullable = true)

+--------+-------------+-------+----+---------+---------+-------+
|duration|protocol_type|service|flag|src_bytes|dst_bytes|  label|
+--------+-------------+-------+----+---------+---------+-------+
|       0|          tcp|   http|  SF|      181|     5450|normal.|
|       0|          tcp|   http|  SF|      239|      486|normal.|
|       0|          tcp|   http|  SF|      235|     1337|normal.|
|       0|          tcp|   http|  SF|      219|     1337|normal.|
|       0|          tcp|   http|  SF|      217|     2032|normal.|
|       0|          tcp|   http|  SF|      217|     2032|normal.|
|       0|          tcp|   http|  SF|      212|     1940|normal.|
|       0|          tcp|   http|  SF|      159|     4087|normal.|
|       0|          tcp|   http|  SF|      210|      151|normal.|
```

Figure 4: Result of Part A.5

1.6 Part A.6

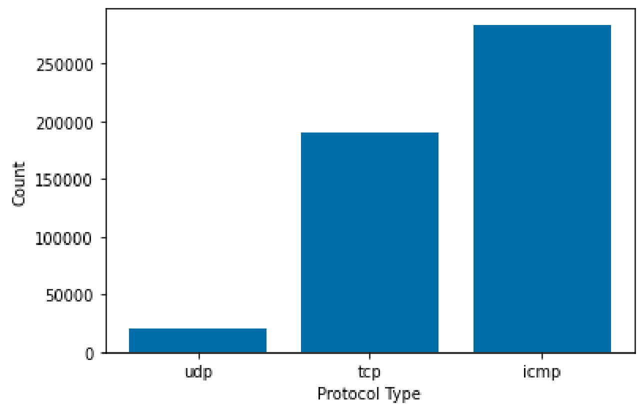
Figure 5 shows the total number of connections based on *protocol_type*.

Figure 6 shows the total number of connections based on *service* and figure 7 is the corresponding bar graph.

```python
# analysis based on protocol type
data_protocol_type = df.rdd.map(lambda x: x.protocol_type).collect()

dictionary = {}
for item in data_protocol_type:
    if item not in dictionary.keys():
        dictionary[item] = 1
    else:
        dictionary[item] += 1

dictionary = dict(sorted(dictionary.items(), key=lambda item: item[1]))
print(f'The total number of connections based on protocol_type are {dictionary}')

# plot
names  = list(dictionary.keys())
values = list(dictionary.values())

plt.bar(range(len(dictionary)), values, tick_label=names)
plt.xlabel('Protocol Type')
plt.ylabel('Count')
plt.show()
```

▸ (1) Spark Jobs

```
The total number of connections based on protocol_type are {'udp': 20354, 'tcp': 190065, 'icmp': 283602}
```



```
Command took 14.12 seconds -- by adijain0707@gmail.com at 3/2/2023, 10:17:02 AM on My Cluster
```

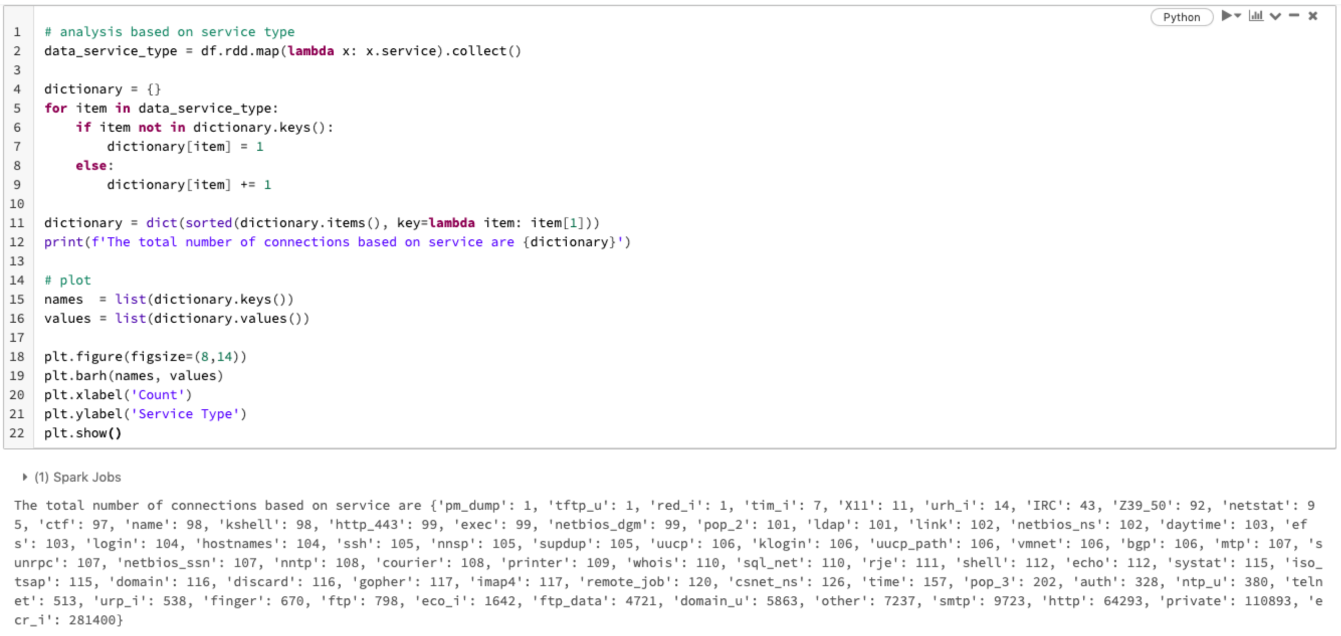
Figure 5: Result of Part A.6 *protocol_type*

1.7 Part A.7

Figure 8 shows the number of connections based on the type of connection. The 'normal' connections are only a small fraction and the others represent the attack type.

Figure 9 shows a bar plot for the counts of various error status of the connection.

Figure 10 is a comparison between average *src_bytes* and average *dst_bytes* transferred. It is evident that a lot more data is transferred from source to destination than other way round.



```python
# analysis based on service type
data_service_type = df.rdd.map(lambda x: x.service).collect()

dictionary = {}
for item in data_service_type:
    if item not in dictionary.keys():
        dictionary[item] = 1
    else:
        dictionary[item] += 1

dictionary = dict(sorted(dictionary.items(), key=lambda item: item[1]))
print(f'The total number of connections based on service are {dictionary}')

# plot
names  = list(dictionary.keys())
values = list(dictionary.values())

plt.figure(figsize=(8,14))
plt.barh(names, values)
plt.xlabel('Count')
plt.ylabel('Service Type')
plt.show()
```

▸ (1) Spark Jobs

```
The total number of connections based on service are {'pm_dump': 1, 'tftp_u': 1, 'red_i': 1, 'tim_i': 7, 'X11': 11, 'urh_i': 14, 'IRC': 43, 'Z39_50': 92, 'netstat': 9
5, 'ctf': 97, 'name': 98, 'kshell': 98, 'http_443': 99, 'exec': 99, 'netbios_dgm': 99, 'pop_2': 101, 'ldap': 101, 'link': 102, 'netbios_ns': 102, 'daytime': 103, 'ef
s': 103, 'login': 104, 'hostnames': 104, 'ssh': 105, 'nnsp': 105, 'supdup': 105, 'uucp': 106, 'klogin': 106, 'uucp_path': 106, 'vmnet': 106, 'bgp': 106, 'mtp': 107, 's
unrpc': 107, 'netbios_ssn': 107, 'nntp': 108, 'courier': 108, 'printer': 109, 'whois': 110, 'sql_net': 110, 'rje': 111, 'shell': 112, 'echo': 112, 'systat': 115, 'iso_
tsap': 115, 'domain': 116, 'discard': 116, 'gopher': 117, 'imap4': 117, 'remote_job': 120, 'csnet_ns': 126, 'time': 157, 'pop_3': 202, 'auth': 328, 'ntp_u': 380, 'teln
et': 513, 'urp_i': 538, 'finger': 670, 'ftp': 798, 'eco_i': 1642, 'ftp_data': 4721, 'domain_u': 5863, 'other': 7237, 'smtp': 9723, 'http': 64293, 'private': 110893, 'e
cr_i': 281400}
```

Figure 6: Result of Part A.6 *service*

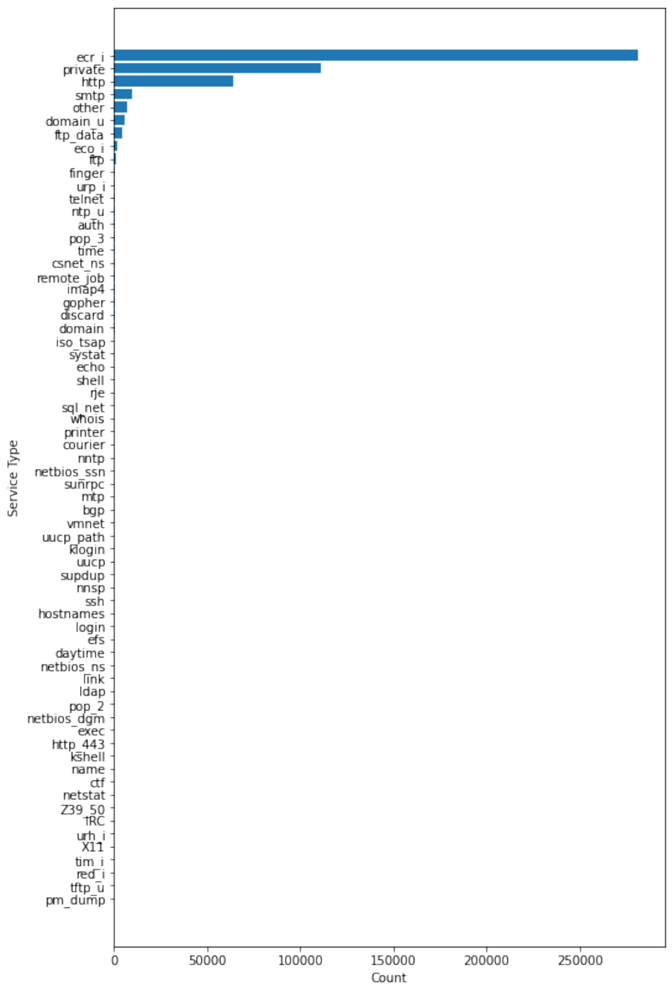


Figure 7: Result of Part A.6 *service* bar graph

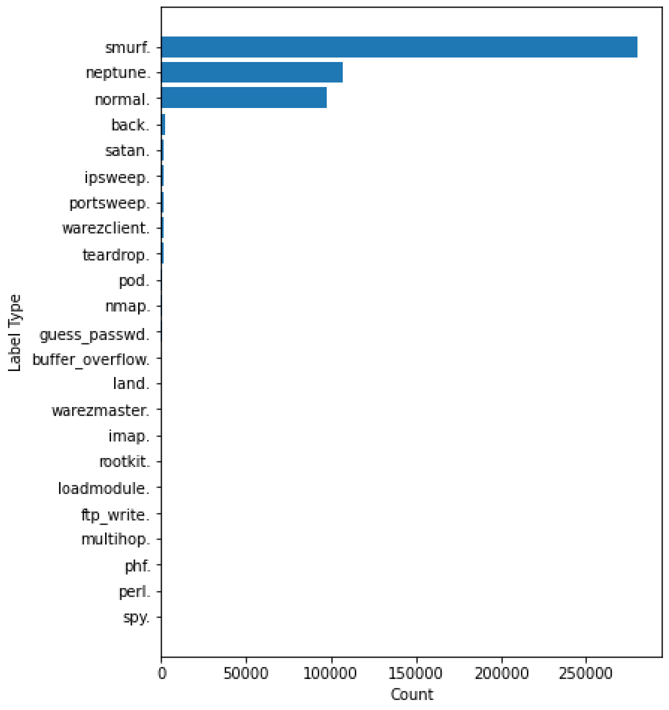


Figure 8: Counts of type of connection

1.8 Part A.8 and A.9

The six attributes from part 1.5 are used for training models. Figure 11 shows the data transformation that is applied before training. The categorical data is converted to numerical data and features are concatenated into a single vector. The data is split into training (80%) and test (20%) sets.

The two models trained are: **linear SVM classifier** (LinearSVC) and **multilayer perceptron classifier** (MLP). LinearSVC is a classical ML algorithm for classification, which constructs a hyperplane(s) in high-dimensional space between the datapoints of different classes. MLP is a deep-learning based algorithm that uses a neural network architecture for learning patterns in the data. The test classification accuracy for LinearSVC is 93.85% (figure 12) and 99% (figure 13) for MLP. The near perfect accuracy for MLP is attributed to its architecture: the structure of hidden layers make it theoretically an universal approximator. The other contributing factors are simplicity of the data and large number of epochs used for training.

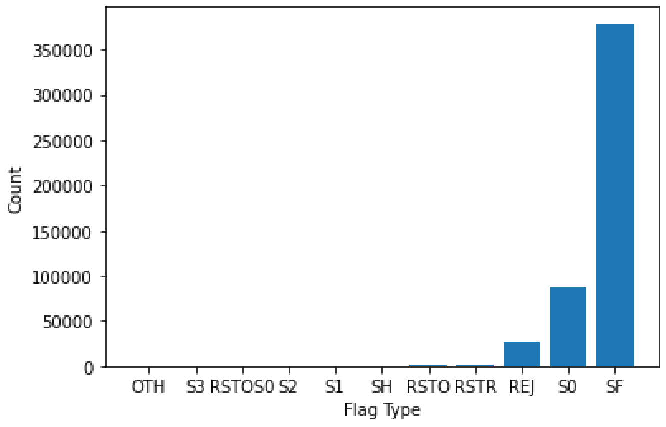


Figure 9: Comparison based on flag type

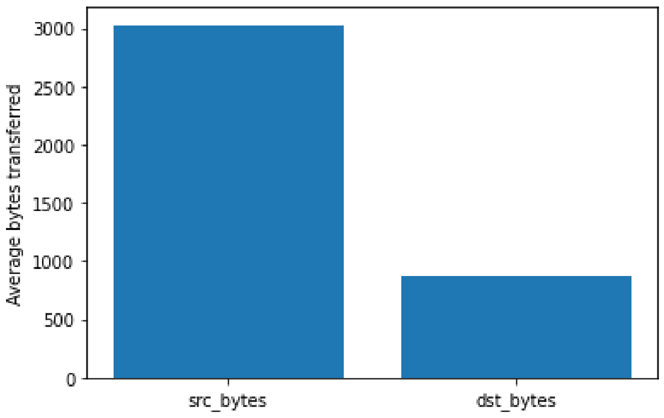


Figure 10: Data transfer

2 Part B

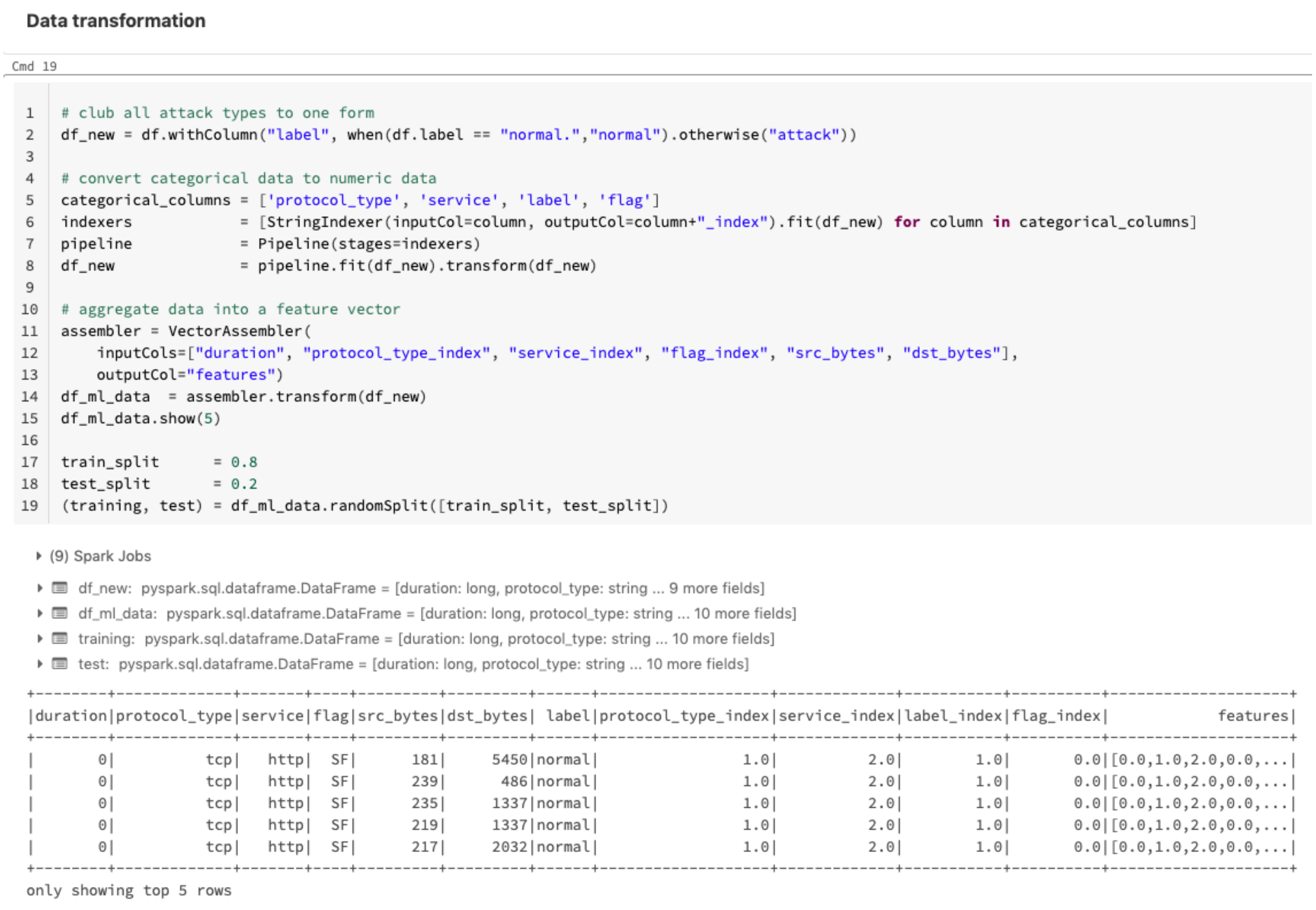
2.1 Part B.1

1. A platform as a service (PaaS) solution that hosts web apps in Azure provide professional development services to continuously add features to custom applications.
Yes - PaaS provides a framework where developers can build or customize cloud-based applications. Being able to add custome features to applications is one of the core features of PaaS.

2. A platform as a service (PaaS) database offering in Azure provides built-in high availability.
Yes - Just like other cloud computing services (SaaS, IaaS), high availability is one of the default components of PaaS.

2.2 Part B.2

A relational database must be used when: **Data will be stored as key/value pairs**.
A relational database is used when we have structured data and adheres to a strict schema. If the data needs to be stored in a structured format like a dictionary (key/value pairs), then relational database must be used.



Data transformation

Cmd 19

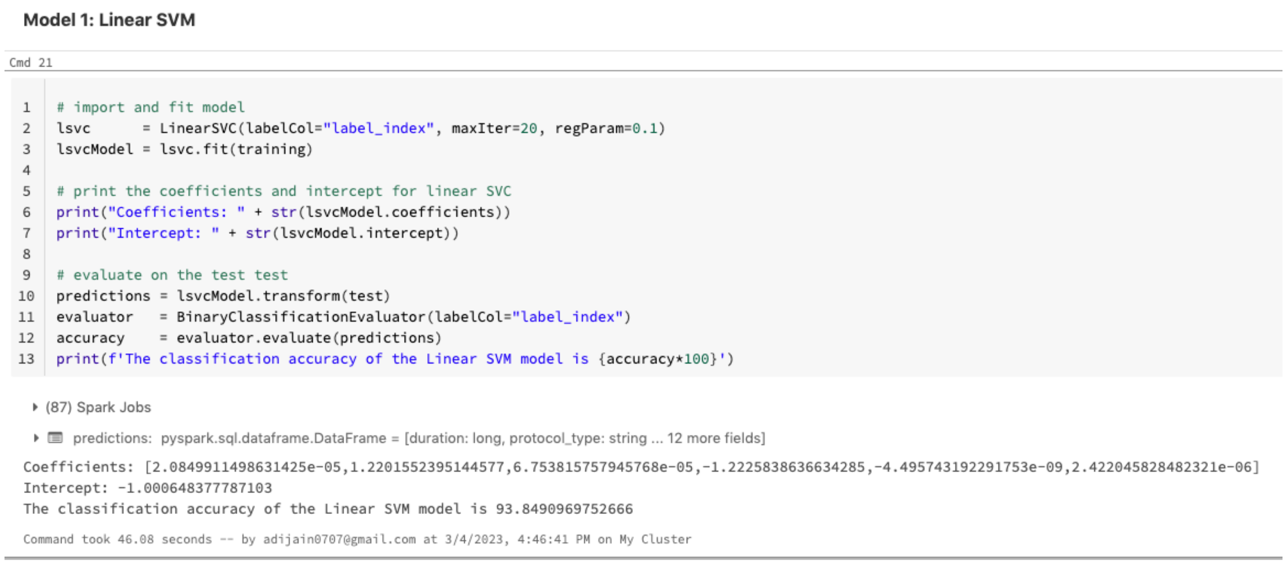
```
# club all attack types to one form
df_new = df.withColumn("label", when(df.label == "normal.","normal").otherwise("attack"))

# convert categorical data to numeric data
categorical_columns = ['protocol_type', 'service', 'label', 'flag']
indexers            = [StringIndexer(inputCol=column, outputCol=column+"_index").fit(df_new) for column in categorical_columns]
pipeline            = Pipeline(stages=indexers)
df_new              = pipeline.fit(df_new).transform(df_new)

# aggregate data into a feature vector
assembler = VectorAssembler(
    inputCols=["duration", "protocol_type_index", "service_index", "flag_index", "src_bytes", "dst_bytes"],
    outputCol="features")
df_ml_data  = assembler.transform(df_new)
df_ml_data.show(5)

train_split      = 0.8
test_split       = 0.2
(training, test) = df_ml_data.randomSplit([train_split, test_split])
```

(9) Spark Jobs

df_new: pyspark.sql.dataframe.DataFrame = [duration: long, protocol_type: string ... 9 more fields]
df_ml_data: pyspark.sql.dataframe.DataFrame = [duration: long, protocol_type: string ... 10 more fields]
training: pyspark.sql.dataframe.DataFrame = [duration: long, protocol_type: string ... 10 more fields]
test: pyspark.sql.dataframe.DataFrame = [duration: long, protocol_type: string ... 10 more fields]

```
+--------+-------------+-------+----+---------+---------+------+-------------------+-------------+-----------+----------+--------------------+
|duration|protocol_type|service|flag|src_bytes|dst_bytes| label|protocol_type_index|service_index|label_index|flag_index|            features|
+--------+-------------+-------+----+---------+---------+------+-------------------+-------------+-----------+----------+--------------------+
|       0|          tcp|   http|  SF|      181|     5450|normal|                1.0|          2.0|        1.0|       0.0|[0.0,1.0,2.0,0.0,...|
|       0|          tcp|   http|  SF|      239|      486|normal|                1.0|          2.0|        1.0|       0.0|[0.0,1.0,2.0,0.0,...|
|       0|          tcp|   http|  SF|      235|     1337|normal|                1.0|          2.0|        1.0|       0.0|[0.0,1.0,2.0,0.0,...|
|       0|          tcp|   http|  SF|      219|     1337|normal|                1.0|          2.0|        1.0|       0.0|[0.0,1.0,2.0,0.0,...|
|       0|          tcp|   http|  SF|      217|     2032|normal|                1.0|          2.0|        1.0|       0.0|[0.0,1.0,2.0,0.0,...|
+--------+-------------+-------+----+---------+---------+------+-------------------+-------------+-----------+----------+--------------------+
only showing top 5 rows
```

Figure 11: Data transformation



Model 1: Linear SVM

Cmd 21

```
# import and fit model
lsvc      = LinearSVC(labelCol="label_index", maxIter=20, regParam=0.1)
lsvcModel = lsvc.fit(training)

# print the coefficients and intercept for linear SVC
print("Coefficients: " + str(lsvcModel.coefficients))
print("Intercept: " + str(lsvcModel.intercept))

# evaluate on the test test
predictions = lsvcModel.transform(test)
evaluator   = BinaryClassificationEvaluator(labelCol="label_index")
accuracy    = evaluator.evaluate(predictions)
print(f'The classification accuracy of the Linear SVM model is {accuracy*100}')
```

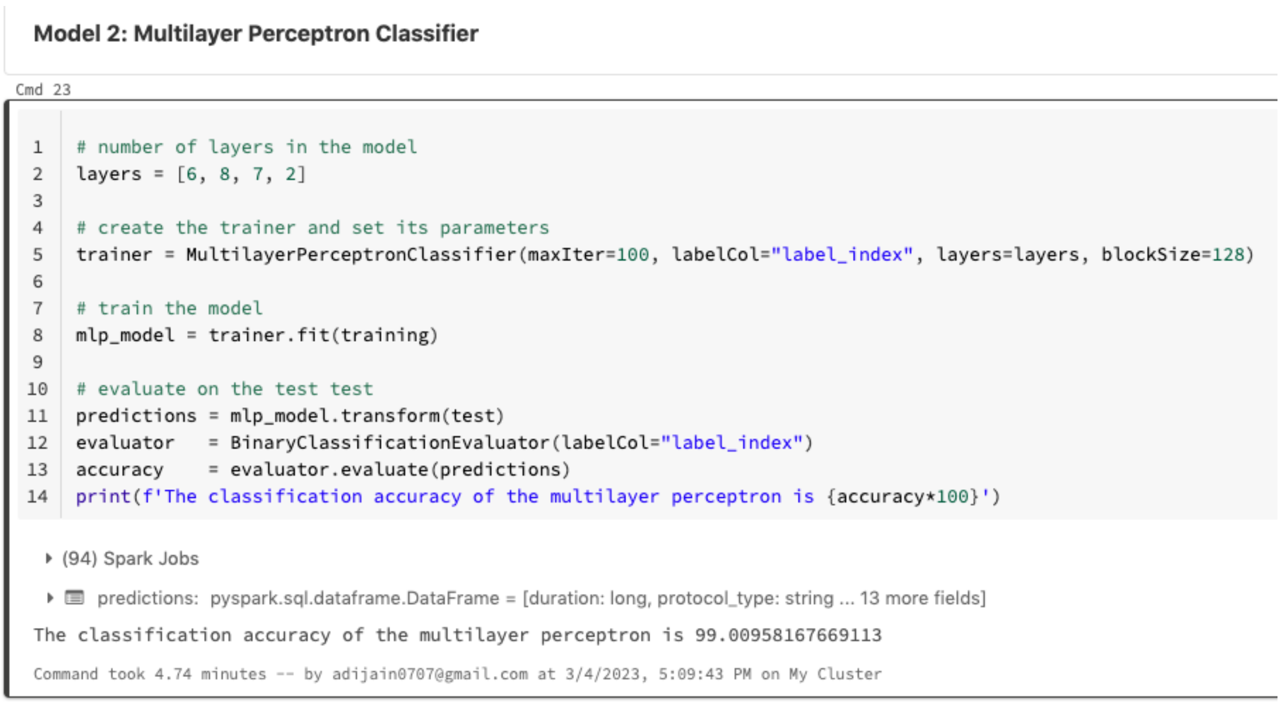
(87) Spark Jobs

predictions: pyspark.sql.dataframe.DataFrame = [duration: long, protocol_type: string ... 12 more fields]

```
Coefficients: [2.0849911498631425e-05,1.2201552395144577,6.753815757945768e-05,-1.2225838636634285,-4.495743192291753e-09,2.422045828482321e-06]
Intercept: -1.000648377787103
The classification accuracy of the Linear SVM model is 93.8490969752666
```

Command took 46.08 seconds -- by adijain0707@gmail.com at 3/4/2023, 4:46:41 PM on My Cluster

Figure 12: Linear SVM


Model 2: Multilayer Perceptron Classifier

Cmd 23

```
# number of layers in the model
layers = [6, 8, 7, 2]

# create the trainer and set its parameters
trainer = MultilayerPerceptronClassifier(maxIter=100, labelCol="label_index", layers=layers, blockSize=128)

# train the model
mlp_model = trainer.fit(training)

# evaluate on the test test
predictions = mlp_model.transform(test)
evaluator   = BinaryClassificationEvaluator(labelCol="label_index")
accuracy    = evaluator.evaluate(predictions)
print(f'The classification accuracy of the multilayer perceptron is {accuracy*100}')
```

```
(94) Spark Jobs
predictions: pyspark.sql.dataframe.DataFrame = [duration: long, protocol_type: string ... 13 more fields]
The classification accuracy of the multilayer perceptron is 99.00958167669113
Command took 4.74 minutes -- by adijain0707@gmail.com at 3/4/2023, 5:09:43 PM on My Cluster
```

Figure 13: Multilayer Perceptron Classifier

2.3 Part B.3

When you are implementing a Software as a Service solution, you are responsible for: **Defining scalability rules**.
In SaaS framework, the installation, configuration, and deployment of the software is taken by the service provider. The user needs to just define the scalability rules depending upon requirement.

2.4 Part B.4

1. To achieve a hybrid cloud model, a company must always migrate from a private cloud model.
No - A hybrid cloud model is a mix of public cloud and on-premise deployment. Hence, to achieve a hybrid cloud model, a company needs to migrate only *some* of its infrastructure to a cloud service provider.

2. A company can extend the capacity of its internal network by using a public cloud.
Yes - Through the hybrid cloud model, a company extend its on-premise capacity by using a public cloud.

3. In a public cloud model, only guest users at your company can access the resources in the cloud.
No - Specific access privileges can be given to specific users of the company in the cloud.

2.5 Part B.5

A. A cloud service that remains available after a failure occurs - **Fault Tolerance**
Fault tolerance is the property of a cloud service in which it remains accessible after a failure.

B. A cloud service that can be recovered after a failure occurs - **Disaster Recovery**
Disaster recovery is the property of a cloud service in which data can be recovered after a failure.

C. A cloud service that performs quickly when demand increases - **Dynamic Scalability**
A cloud service has dynamic scalability if its infrastucture scales according to the demand.

D. A cloud service that can be accessed quickly from the internet - **Low Latency**
A cloud service has low latency if it can be accessed from the internet in a small amount of time.