

Assignment 3 Spark and Cloud Data Platform

Yanting Chen

1 Part A

1.1 Count the odd and even numbers using the file ‘integer.txt’ and download it from Quercus. Show your code and output.

1. What is an Intrusion Detection System (IDS)?

An Intrusion Detection System (IDS) is a hardware or software application that monitors network or system activities for malicious activities or policy violations. Any detected activity or violation is typically reported to an administrator or collected centrally using a security information and event management (SIEM) system. There are three primary methods for detecting intrusions:

- **Signature-based Detection:** This method detects attacks by looking for specific patterns, such as byte sequences in network traffic or known malicious instruction sequences used by malware.
- **Anomaly-based Detection:** This method establishes a baseline of normal activities and then monitors for deviations from this baseline, which might indicate a possible intrusion.
- **Hybrid-based Detection:** This method combines signature-based and anomaly-based detection techniques to leverage the benefits of both methods and reduce the disadvantages.

2. Implementation of IDS on the KDD99 10% sub Dataset

Yes, it is possible to implement an IDS using the `kddcup.data_10_percent.gz` 10% subset. The paper describes a specific approach using machine learning techniques in a Big Data environment to improve the efficiency and accuracy of intrusion detection. The original paper used the `kdd99` dataset. This dataset is a subset of `kdd99` and has the same feature columns, except that the number is 10% of the source dataset. Its data processing can be consistent with the thesis.

3. Workflow for Implementing the Intrusion Detection System

The workflow described in the paper for implementing the Intrusion Detection System using the KDD99 dataset involves several steps:

- (a) **Data Loading:** The dataset is loaded and exported into Resilient Distributed Datasets (RDD) and DataFrame in Apache Spark. This step ensures that the data is distributed and can be processed efficiently across a cluster.
- (b) **Data Preprocessing:**
 - **Categorical to Numerical Conversion:** Categorical data in the dataset is converted into numerical data since machine learning algorithms typically require numerical input.
 - **Standardization:** The data is standardized to have a mean of zero and a variance of one. The unit-variance method uses the corrected sample standard deviation. This step ensures that all features contribute equally to the results and prevents the learning algorithm from being biased towards features with larger ranges of values.
- (c) **Feature Selection:**
 - **ChiSqSelector:** The Chi-Squared test is used to select a subset of relevant features from the dataset. This step reduces the dimensionality of the data, which helps improve the classification efficiency and reduce computation time. The `numTopFeatures` parameter is used to specify the number of top features to select.

(d) **Model Training:**

- **Support Vector Machine (SVM):** The SVM algorithm is used for data classification. Specifically, SVMWithSGD (Stochastic Gradient Descent) is used to optimize the model. The model is trained on the preprocessed and feature-selected dataset.

(e) **Model Testing and Evaluation:**

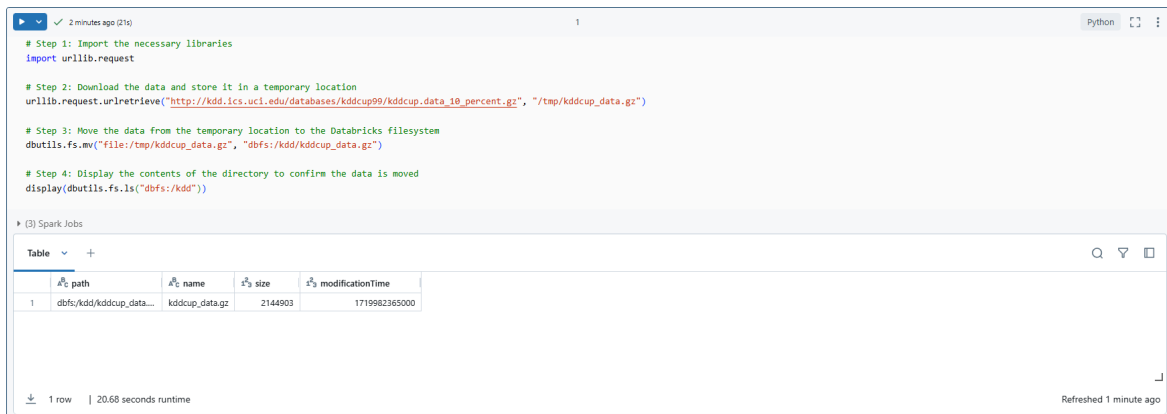
- The trained model is tested and evaluated using the KDD99 dataset. The evaluation metrics used include:
 - **Area Under Curve (AUROC):** Measures the performance of the classifier.
 - **Area Under Precision-Recall Curve (AUPR):** Shows the tradeoff between precision and recall for different thresholds.
- The results are compared with other classifiers and methods to demonstrate the performance improvements achieved by the proposed model.

1.2 Use the python urllib library to extract the KDD Cup 99 data from their web repository, store it in a temporary location and then move it to the Databricks filesystem which can enable easy access to this data for analysis.

- Code

```
1 # Step 1: Import the necessary libraries
2 import urllib.request
3
4 # Step 2: Download the data and store it in a temporary location
5 urllib.request.urlretrieve("http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz",
6                             "/tmp/kddcup_data.gz")
7
8 # Step 3: Move the data from the temporary location to the Databricks filesystem
9 dbutils.fs.mv("file:/tmp/kddcup_data.gz", "dbfs:/kdd/kddcup_data.gz")
10
11 # Step 4: Display the contents of the directory to confirm the data is moved
12 display(dbutils.fs.ls("dbfs:/kdd"))
```

- Screen shot and Output



Table

	path	name	size	modificationTime
1	dbfs:/kdd/kddcup_data...	kddcup_data.gz	2144903	1719982365000

1 row | 20.68 seconds runtime

Refreshed 1 minute ago

Figure 1: Output

1.3 After storing the data in the Databricks filesystem. Load your data from the disk into Spark's RDD. Print 10 values of your RDD and verify the type of data structure of your data (RDD)

- Code

```

1 # Step 1: Load the data from the Databricks filesystem into an RDD
2 rdd = sc.textFile("dbfs:/kdd/kddcup_data.gz")
3
4 # Step 2: Print 10 values from the RDD
5 print("First 10 values of the RDD:")
6 for line in rdd.take(10):
7     print(line)
8
9 # Step 3: Verify the type of data structure
10 print("\nType of the data structure:", type(rdd))

```

- Screen shot and Output

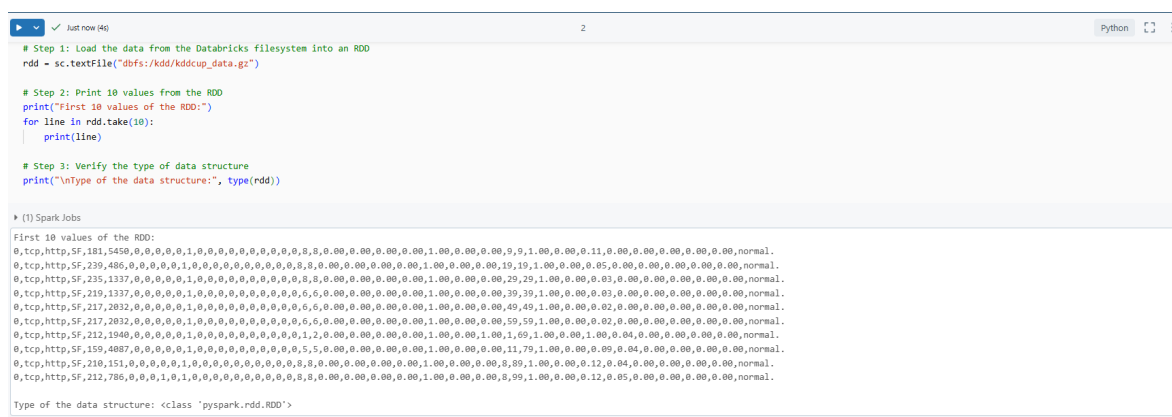


Figure 2: Output

1.4 Split the data. (Each entry in your RDD is a comma-separated line of data, which you first need to split before you can parse and build your data frame.) Show the total number of features (columns) and print results.

- Code

```

1 # Step 1: Split each line into a list of features
2 rdd_split = rdd.map(lambda line: line.split(","))
3
4 # Step 2: Show the total number of features (columns)
5 num_features = len(rdd_split.first())
6 print("Total number of features (columns):", num_features)
7
8 # Step 3: Print the first 10 rows of the split data
9 print("First 10 rows of the split data:")
10 for row in rdd_split.take(10):
11     print(row)

```

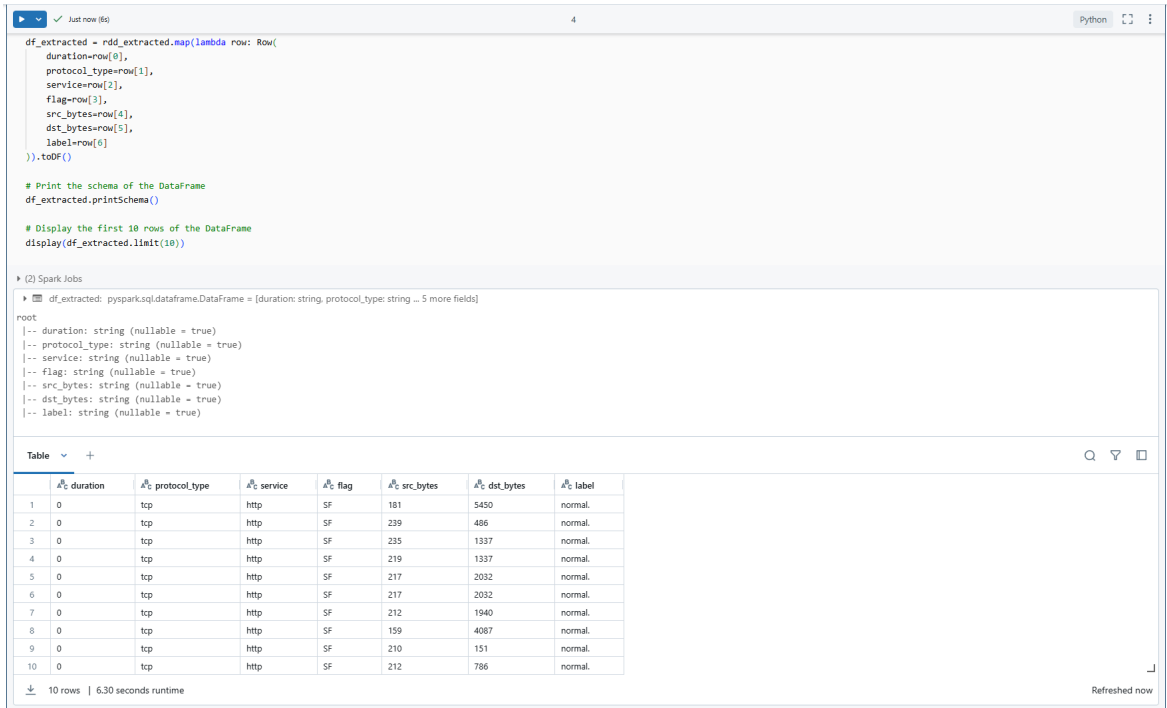
- Screen shot and Output


```

33     dst_bytes=row[5],
34     label=row[6]
35 ).toDF()
36
37 # Print the schema of the DataFrame
38 df_extracted.printSchema()
39
40 # Display the first 10 rows of the DataFrame
41 display(df_extracted.limit(10))

```

- Screen shot and Output



```

df_extracted = rdd_extracted.map(lambda row: Row(
    duration=row[0],
    protocol_type=row[1],
    service=row[2],
    flag=row[3],
    src_bytes=row[4],
    dst_bytes=row[5],
    label=row[6]
)).toDF()

# Print the schema of the DataFrame
df_extracted.printSchema()

# Display the first 10 rows of the DataFrame
display(df_extracted.limit(10))

```

root

```

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

```

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

10 rows | 6.30 seconds runtime

Figure 4: Output

1.6 Get the total number of connections based on the protocol_type and based on the service. Show results in an ascending order. Plot the bar graph for both.

- Code

```

1 # Step 1: Aggregate the data based on protocol_type and service
2
3 # Aggregate based on protocol_type
4 protocol_type_counts = df_extracted.groupBy("protocol_type").count().orderBy("count",
5                                     ascending=True)
6
7 # Aggregate based on service
8 service_counts = df_extracted.groupBy("service").count().orderBy("count",
9                                     ascending=True)
10
11 # Step 2: Show the results in ascending order
12
13 # Show the results for protocol_type

```

```

12 display(protocol_type_counts)
13
14 # Show the results for service
15 display(service_counts)
16
17 # Step 3: Plot the bar graphs
18
19 import matplotlib.pyplot as plt
20
21 # Convert to Pandas DataFrame for plotting
22 protocol_type_counts_pd = protocol_type_counts.toPandas()
23 service_counts_pd = service_counts.toPandas()
24
25 # Plot the bar graph for protocol_type
26 plt.figure(figsize=(10, 5))
27 plt.bar(protocol_type_counts_pd['protocol_type'], protocol_type_counts_pd['count'],
28         color='blue')
29 plt.xlabel('Protocol Type')
30 plt.ylabel('Number of Connections')
31 plt.title('Total Number of Connections Based on Protocol Type')
32 plt.xticks(rotation=45)
33 plt.show()
34
35 # Plot the bar graph for service
36 plt.figure(figsize=(15, 5))
37 plt.bar(service_counts_pd['service'], service_counts_pd['count'], color='green')
38 plt.xlabel('Service')
39 plt.ylabel('Number of Connections')
40 plt.title('Total Number of Connections Based on Service')
41 plt.xticks(rotation=90)
42 plt.show()

```

- Screen shot and Output

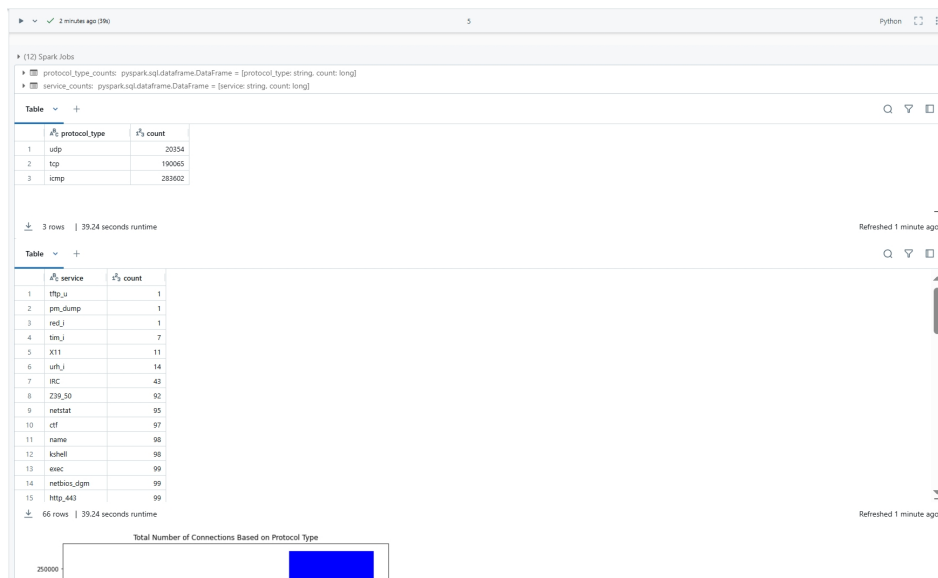


Figure 5: Output

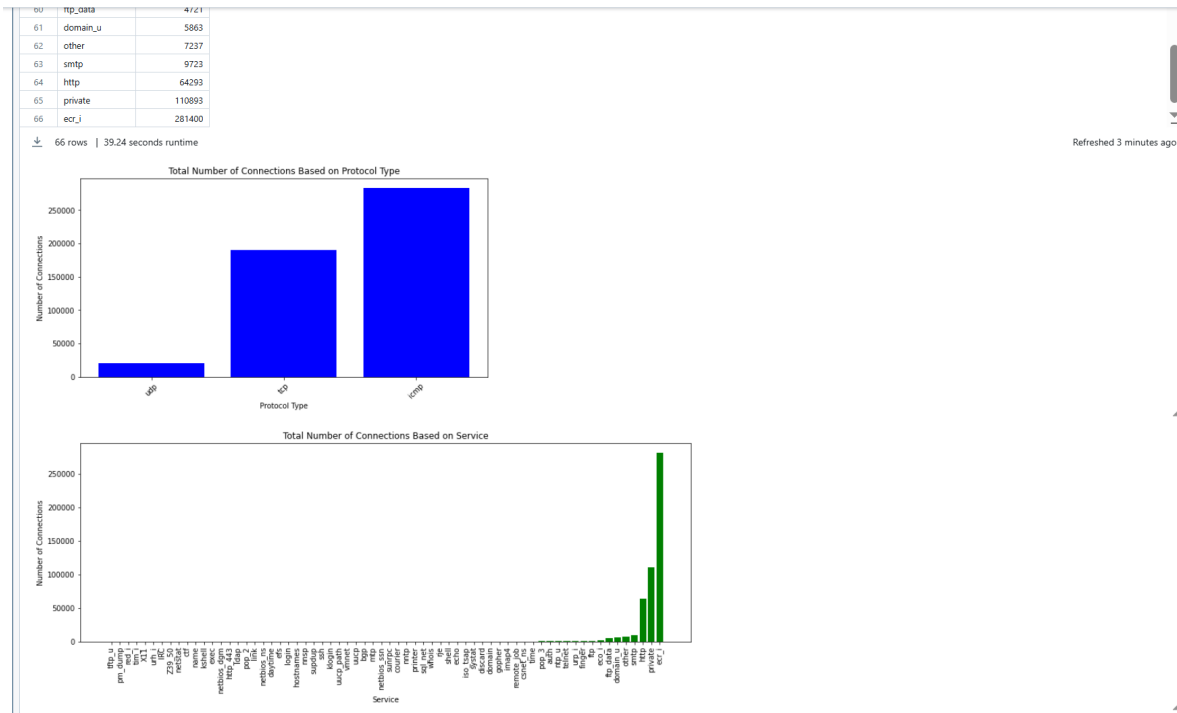


Figure 6: Output

1.7 Do a further exploratory data analysis, including other columns of this dataset and plot graphs. Plot at least 3 different charts and explain them.

To perform further exploratory data analysis (EDA) on the KDD Cup 99 dataset, I will include additional columns and plot at least three different charts. Here, I will explore the distribution of the label column, the relationship between src_bytes and dst_bytes, and the count of connections based on the flag column.

• Code

```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3
4 # Set up the plotting environment
5 sns.set(style="whitegrid")
6
7 # 1. Distribution of Attack Types (label column)
8 label_counts = df_extracted.groupby("label").count().orderBy("count", ascending=False)
9 label_counts_pd = label_counts.toPandas()
10
11 plt.figure(figsize=(10, 5))
12 sns.barplot(x='label', y='count', data=label_counts_pd, palette='viridis')
13 plt.xlabel('Label')
14 plt.ylabel('Number of Connections')
15 plt.title('Distribution of Attack Types')
16 plt.xticks(rotation=45)
17 plt.show()
18
19 # 2. Relationship between Source Bytes (src_bytes) and Destination Bytes (dst_bytes)
20 # with log scale
21 # and distinct color for 'normal' points
```

```

22 # Convert to Pandas DataFrame for plotting
23 data_pd = df_extracted.toPandas()
24
25 # Dynamically create a color palette
26 unique_labels = data_pd['label'].unique()
27 palette = {label: 'red' if label == 'normal' else sns.color_palette('viridis',
28         len(unique_labels))[i] for i, label in enumerate(unique_labels)}
29
30 plt.figure(figsize=(10, 5))
31 sns.scatterplot(x='src_bytes', y='dst_bytes', data=data_pd, hue='label',
32         palette=palette)
33 plt.xscale('log')
34 plt.yscale('log')
35 plt.xlabel('Source Bytes (log scale)')
36 plt.ylabel('Destination Bytes (log scale)')
37 plt.title('Relationship between Source Bytes and Destination Bytes')
38 plt.legend(loc='upper right')
39 plt.show()
40
41 # 3. Count of Connections based on Flag
42 flag_counts = df_extracted.groupBy("flag").count().orderBy("count", ascending=False)
43 flag_counts_pd = flag_counts.toPandas()
44
45 plt.figure(figsize=(10, 5))
46 sns.barplot(x='flag', y='count', data=flag_counts_pd, palette='viridis')
47 plt.xlabel('Flag')
48 plt.ylabel('Number of Connections')
49 plt.title('Count of Connections based on Flag')
50 plt.xticks(rotation=45)
51 plt.show()

```

1.7.1 Distribution of Attack Types (label column)

– Screen shot and Output

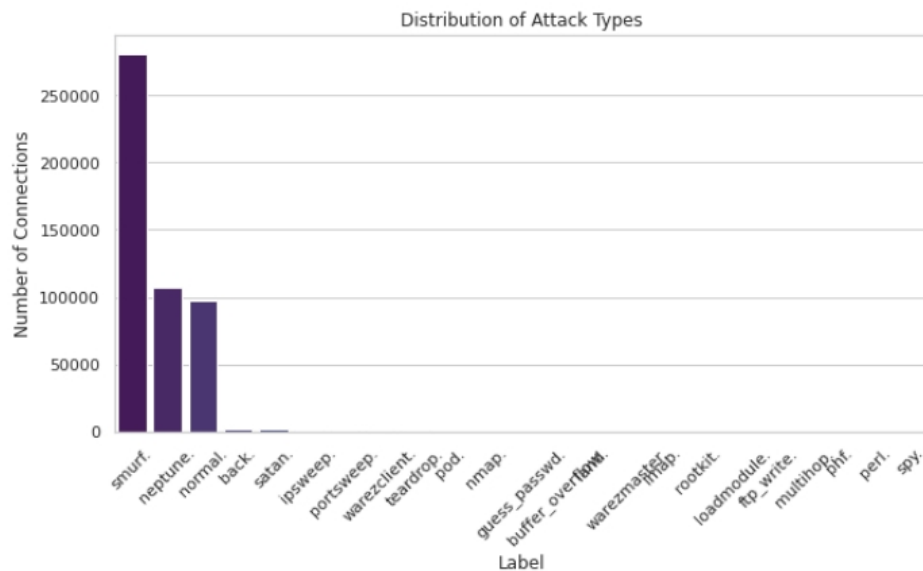


Figure 7: Output

- Explanation

This chart shows the distribution of different attack types and normal connections in the dataset. It helps in understanding the frequency of various attack types compared to normal connections. The bars represent the count of connections for each label (attack type or normal).

1.7.2 Relationship between Source Bytes (src_bytes) and Destination Bytes (dst_bytes) with Log Scale

- Screen shot and Output

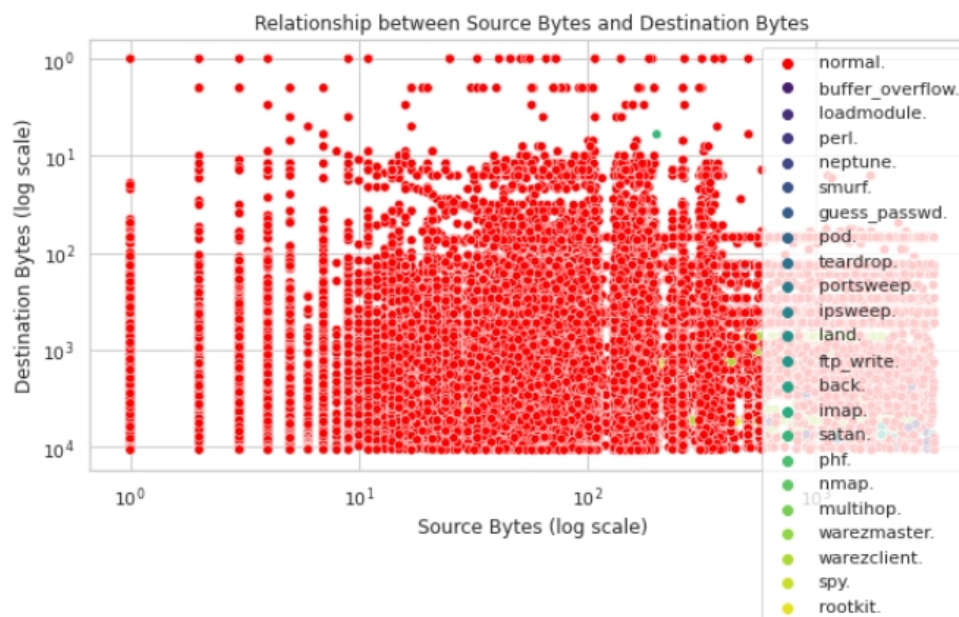


Figure 8: Output

- Explanation

This scatter plot shows the relationship between src_bytes and dst_bytes for each connection using log scales for both axes. Different colors represent different attack types or normal connections. This plot helps in identifying patterns or anomalies in data transfer sizes that might indicate specific types of attacks.

Using log scales for both axes makes the plot more readable by spreading out the data points, especially when there is a large range of values. The normal points are highlighted in red for better distinction.

1.7.3 Distribution of Attack Types (label column)

- Screen shot and Output
See figure 9.
- Explanation

This bar chart shows the count of connections for each type of flag. The flag column represents the status of the connection. This plot helps in understanding the distribution of connection statuses in the dataset.

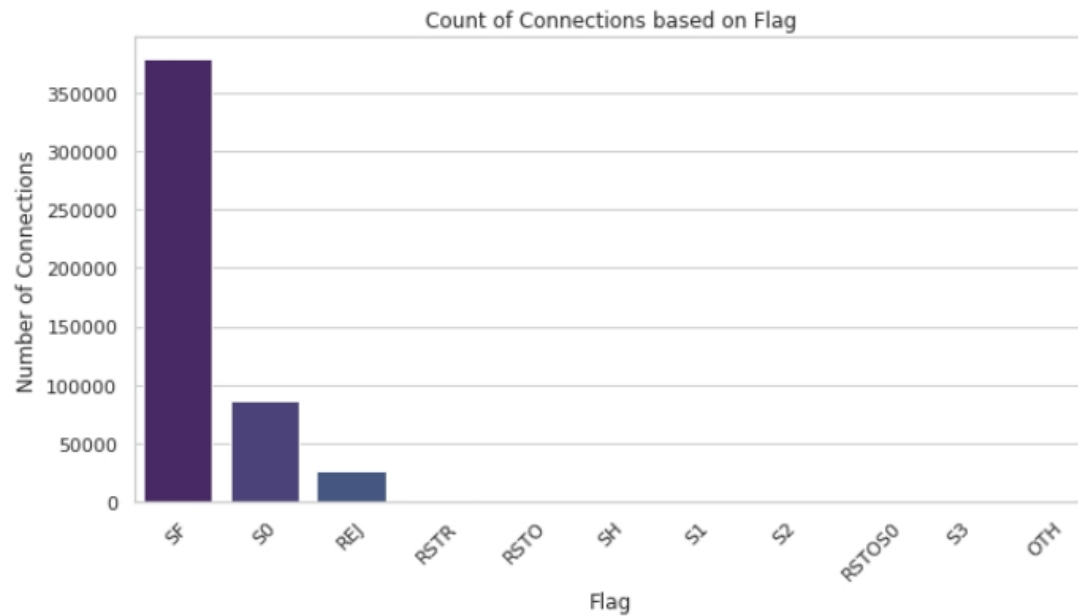


Figure 9: Output

1.8

1.8.1 Create the New Label Column:

- Code

```
1 from pyspark.sql.functions import when
2
3 # Create a new label column where 'normal' is 'normal' and everything else is 'attack'
4 df_extracted = df_extracted.withColumn("new_label", when(df_extracted["label"] ==
5     "normal.", "normal").otherwise("attack"))
6
7 # Show the updated DataFrame
8 display(df_extracted.limit(10))
```

- Screen shot and Output

```
from pyspark.sql.functions import when

# Create a new label column where 'normal' is 'normal' and everything else is 'attack'
df_extracted = df_extracted.withColumn("new_label", when(df_extracted["label"] == "normal.", "normal").otherwise("attack"))

# Show the updated DataFrame
display(df_extracted.limit(10))
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	label	new_label
1	0	tcp	http	SF	101	5420	normal	normal
2	0	tcp	http	SF	239	486	normal	normal
3	0	tcp	http	SF	235	1337	normal	normal
4	0	tcp	http	SF	219	1337	normal	normal
5	0	tcp	http	SF	217	2032	normal	normal
6	0	tcp	http	SF	217	2032	normal	normal
7	0	tcp	http	SF	212	1940	normal	normal
8	0	tcp	http	SF	159	4087	normal	normal
9	0	tcp	http	SF	210	151	normal	normal
10	0	tcp	http	SF	212	786	normal	normal

10 rows | 4.55 seconds runtime

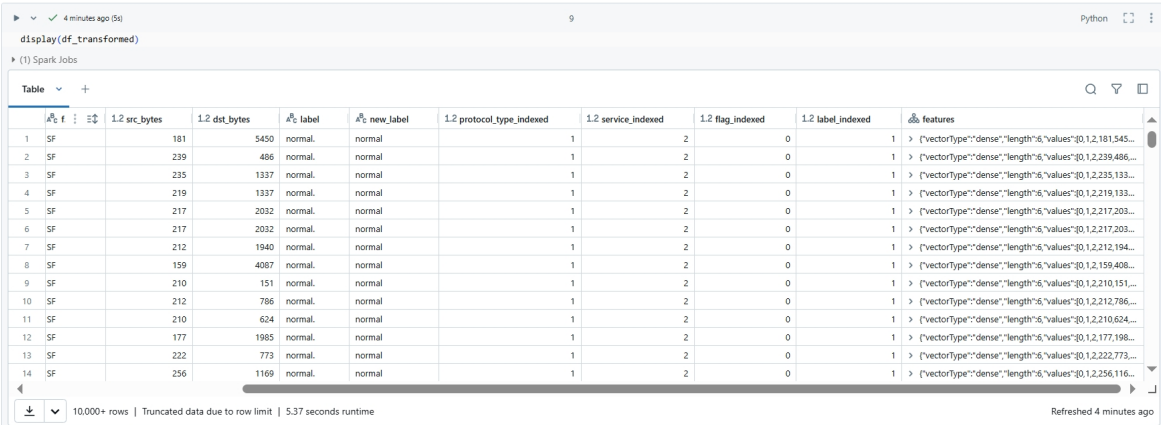
Figure 10: Output

1.8.2 Convert Data Types and Index Categorical Columns:

- Code

```
1 from pyspark.sql.functions import col
2 from pyspark.ml.feature import StringIndexer, VectorAssembler
3 from pyspark.ml import Pipeline
4
5 # Convert columns to appropriate types
6 df_extracted = df_extracted.withColumn("duration", col("duration").cast("double"))
7 df_extracted = df_extracted.withColumn("src_bytes", col("src_bytes").cast("double"))
8 df_extracted = df_extracted.withColumn("dst_bytes", col("dst_bytes").cast("double"))
9
10 # Index the categorical columns
11 protocol_indexer = StringIndexer(inputCol="protocol_type",
12     outputCol="protocol_type_indexed")
13 service_indexer = StringIndexer(inputCol="service", outputCol="service_indexed")
14 flag_indexer = StringIndexer(inputCol="flag", outputCol="flag_indexed")
15 label_indexer = StringIndexer(inputCol="new_label", outputCol="label_indexed")
16
17 # Assemble the feature columns
18 feature_assembler = VectorAssembler(inputCols=[
19     "duration",
20     "protocol_type_indexed",
21     "service_indexed",
22     "src_bytes",
23     "dst_bytes",
24     "flag_indexed"
25 ], outputCol="features")
26
27 # Create a pipeline
28 pipeline = Pipeline(stages=[protocol_indexer, service_indexer, flag_indexer,
29     label_indexer, feature_assembler])
30 pipeline_model = pipeline.fit(df_extracted)
31
32 # Transform the data
33 df_transformed = pipeline_model.transform(df_extracted)
```

- Screen shot and Output



display(df_transformed)

(1) Spark Jobs

	id	src_bytes	dst_bytes	label	new_label	protocol_type_indexed	service_indexed	flag_indexed	label_indexed	features
1	SF	181	5450	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,181,545...
2	SF	239	486	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,239,486...
3	SF	235	1337	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,235,133...
4	SF	219	1337	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,219,133...
5	SF	217	2032	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,217,203...
6	SF	217	2032	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,217,203...
7	SF	212	1940	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,212,194...
8	SF	159	4087	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,159,408...
9	SF	210	151	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,210,151...
10	SF	212	786	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,212,786...
11	SF	210	624	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,210,624...
12	SF	177	1985	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,177,198...
13	SF	222	773	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,222,773...
14	SF	256	1169	normal	normal	1	2	0	1	> (vectorType<"dense";"length"<6;"values"<[0.1,2,256,116...

10,000+ rows | Truncated data due to row limit | 5.37 seconds runtime

Refreshed 4 minutes ago

Figure 11: Output

1.8.3 Split the Data into Training and Test Sets:

- Code

```
1 # Split the data into training and test sets
2 train_data, test_data = df_transformed.randomSplit([0.8, 0.2], seed=42)
```

1.8.4 Build a Simple Machine Learning Model (SVM):

- Code

```
1 from pyspark.ml.classification import LinearSVC
2
3 # Create a Linear SVM model
4 svm = LinearSVC(labelCol="label_indexed", featuresCol="features", maxIter=10)
5
6 # Train the model
7 svm_model = svm.fit(train_data)
8
9 # Make predictions
10 predictions = svm_model.transform(test_data)
11
12 # Show some predictions
13 display(predictions.select("new_label", "prediction", "rawPrediction").limit(10))
```

- Screen shot and Output

```
from pyspark.ml.classification import LinearSVC

# Create a Linear SVM model
svm = LinearSVC(labelCol="label_indexed", featuresCol="features", maxIter=10)

# Train the model
svm_model = svm.fit(train_data)

# Make predictions
predictions = svm_model.transform(test_data)

# Show some predictions
display(predictions.select("new_label", "prediction", "rawPrediction").limit(10))
```

100 Spark Jobs

predictions: pyspark.sql.dataframe.DataFrame

	new_label	1,2 prediction	rawPrediction
1	attack	0	[{"vectorType":"dense","length":2,"values":{"1.0355017417600882,-1.0355017417600...
2	attack	0	[{"vectorType":"dense","length":2,"values":{"1.0355017417600882,-1.0355017417600...
3	attack	0	[{"vectorType":"dense","length":2,"values":{"1.0355017417600882,-1.0355017417600...
4	attack	0	[{"vectorType":"dense","length":2,"values":{"1.0355017417600882,-1.0355017417600...
5	attack	0	[{"vectorType":"dense","length":2,"values":{"1.0355017417600882,-1.0355017417600...
6	attack	0	[{"vectorType":"dense","length":2,"values":{"1.0355017417600882,-1.0355017417600...
7	attack	0	[{"vectorType":"dense","length":2,"values":{"1.0355017417600882,-1.0355017417600...
8	attack	0	[{"vectorType":"dense","length":2,"values":{"1.0355017417600882,-1.0355017417600...
9	attack	0	[{"vectorType":"dense","length":2,"values":{"1.0355017417600882,-1.0355017417600...
10	attack	0	[{"vectorType":"dense","length":2,"values":{"1.0355017417600882,-1.0355017417600...

10 rows | 43.24 seconds runtime

Figure 12: Output

1.8.5 Evaluate the Model:

- Code

```
1 from pyspark.ml.evaluation import MulticlassClassificationEvaluator
2
3 # Evaluate the model
4 evaluator = MulticlassClassificationEvaluator(labelCol="label_indexed",
5       predictionCol="prediction", metricName="accuracy")
6 accuracy = evaluator.evaluate(predictions)
```

```

6 print(f"Accuracy: {accuracy}")
7
8 # Additional metrics
9 evaluator_f1 = MulticlassClassificationEvaluator(labelCol="label_indexed",
    predictionCol="prediction", metricName="f1")
10 f1_score = evaluator_f1.evaluate(predictions)
11 print(f"F1 Score: {f1_score}")

```

- Screen shot and Output

```

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Evaluate the model
evaluator = MulticlassClassificationEvaluator(labelCol="label_indexed", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print(f"Accuracy: {accuracy}")

# Additional metrics
evaluator_f1 = MulticlassClassificationEvaluator(labelCol="label_indexed", predictionCol="prediction", metricName="f1")
f1_score = evaluator_f1.evaluate(predictions)
print(f"F1 Score: {f1_score}")

```

(2) Spark Jobs
 Accuracy: 0.9768257433347483
 F1 Score: 0.97678739851267

Figure 13: Output

1.8.6 Explanation of the Selected Algorithm: Linear Support Vector Machine (LinearSVM)

The Support Vector Machine (SVM) is a supervised machine learning algorithm used primarily for classification tasks, but also for regression. LinearSVM is a variant that uses a linear kernel, meaning it attempts to separate the data with a hyperplane that maximizes the margin between the two classes. By choosing LinearSVM, we capitalize on its strengths in binary classification tasks, especially its effectiveness in high-dimensional spaces and robustness to overfitting. The resulting model can efficiently and accurately distinguish between normal and attack connections in the KDD Cup 99 dataset.

2 Part B

2.1 Read the below statements, choose the correct answer, and provide explanations.

2.1.1 A platform as a service (PaaS) solution that hosts web apps in Azure provide professional development services to continuously add features to custom applications.

- Answer

Yes

- Explanation

Azure PaaS solutions, such as Azure App Service, enable developers to build, deploy, and scale web apps easily. These services provide the necessary infrastructure and tools, including CI/CD pipelines, to continuously integrate and deploy new features to custom applications.

2.1.2 A platform as a service (PaaS) database offering in Azure provides built-in high availability

- Answer

Yes

- Explanation

Azure's PaaS database offerings, such as Azure SQL Database, Azure Database for MySQL, and Azure Database for PostgreSQL, include built-in high availability features. These services are designed to ensure that databases are resilient to failures and can maintain continuous operation with minimal downtime.

2.2 A relational database must be used when

- Answer

d. Strong consistency guarantees are required

- Explanation

Relational databases (RDBMS) like SQL Server, MySQL, and PostgreSQL are designed to store data in tables with predefined schemas. They are optimized for transactions that require ACID (Atomicity, Consistency, Isolation, Durability) properties, which ensure strong consistency guarantees. This means that any transaction will bring the database from one valid state to another, ensuring data integrity.

2.3 When you are implementing a Software as a Service solution, you are responsible for:

- Answer

d. Configuring the SaaS solution

- Explanation

Software as a Service (SaaS) is a model where the provider delivers software applications over the internet. Users can access the software through a web browser, and the provider manages the infrastructure, middleware, application software, and security. The installation, maintenance, and updates of the SaaS solution are handled by the provider.

2.4 Read the below statements, choose the correct answer, and provide explanations.

2.4.1 To achieve a hybrid cloud model, a company must always migrate from a private cloud model

- Answer

No

- Explanation

A company does not need to migrate from a private cloud model to achieve a hybrid cloud model. It is possible to integrate traditional on-premises IT infrastructure directly with public cloud services to create a hybrid cloud environment. Tools and services from cloud providers (like Azure Arc, AWS Outposts, or Google Anthos) facilitate the integration of on-premises systems with public clouds, enabling a hybrid model. Therefore, the correct answer is No.

2.4.2 A company can extend the capacity of its internal network by using a public cloud

- Answer

Yes

- Explanation

Public clouds offer virtually unlimited scalability. Companies can easily extend their internal network capacity by provisioning additional resources (e.g., compute, storage, networking) from the public cloud as needed. Companies can establish secure connections between their internal

networks and public cloud environments using VPNs or dedicated connections (e.g., Azure ExpressRoute, AWS Direct Connect). This ensures secure and reliable extension of their internal network to the public cloud.

2.4.3 In a public cloud model, only guest users at your company can access the resources in the cloud

- Answer

No

- Explanation

Public cloud providers like AWS, Azure, and Google Cloud Platform implement Role-Based Access Control (RBAC), allowing organizations to assign different levels of access to various users, including employees, administrators, and service accounts, based on their roles.

2.5 Read the below statements, choose the correct answer, and provide explanations.

2.5.1 A cloud service that remains available after a failure occurs.

- Answer

Fault Tolerance

- Explanation

Fault tolerance is the ability of a system to continue operating without interruption when one or more of its components fail. In the context of cloud services, fault tolerance ensures that the service remains available even when there are hardware, software, or network failures.

2.5.2 A cloud service that can be recovered after a failure occurs.

- Answer

Disaster Recovery

- Explanation

Disaster recovery refers to the set of policies, tools, and procedures that enable the recovery or continuation of vital technology infrastructure and systems following a natural or human-induced disaster. In the context of cloud services, disaster recovery ensures that services can be restored to a functional state after a significant failure or disruption.

2.5.3 A cloud service that performs quickly when demand increases.

- Answer

Dynamic Scalability

- Explanation

Dynamic scalability, also known as elastic scalability, is the ability of a cloud service to automatically adjust its resources (such as computing power, storage, and network capacity) in response to changes in demand. This ensures that the service can handle varying workloads efficiently without manual intervention.

2.5.4 A cloud service that can be accessed quickly from the internet.

- Answer

Low Latency

- Explanation

Low latency refers to the quick response time or minimal delay experienced when accessing a service over the internet. It is a critical performance metric for cloud services, especially those that require real-time or near-real-time interactions.