Educate Elevate Enlighten

JSS Mahavidyapeetha

JSS Science And Technology University

(Established Under JSS Science and Technology University Act No. 43 of 2013)

(Formerly Known as SJCE)

JSS SCIENCE AND TECHNOLOGY UNIVERSITY MYSURU

DEPARTMENT OF INFORMATION SCIENCE

SUBJECT: COMPUTER NETWORKS

CODE: IS620

SUBMITTED TO: Dr. MANJU N

PROJECT TITLE: INTERNET TRAFFIC CLASSIFICATION

SUBMITTED BY:

SL. NO.	USN	NAME
1.	01JST18IS016	HARSHIT KUMAR JAIN
2.	01JST18IS025	MOHAMMED RAYAAN HUSSAIN
3.	01JST18IS031	P. PRATHEEK

INTRODUCTION

The classification of traffic flows in today's IP networks has become an important research area with the adoption of Machine Learning (ML) techniques and Software Defined Networking (SDN) principles. Traditional methodologies including identifying traffic based on port number and payload inspection are not effective due to the dynamic and encrypted nature of current traffic. This project will attempt to utilize Supervised and Unsupervised ML algorithms to classify flows by their required bandwidth, required QoS, and their application based on various flow level details as features.

As mentioned earlier, traffic classification using Machine Learning is a growing trend in the network analytics domain. One application of these techniques is in cybersecurity. Large datasets produced from enormous Internet traffic flows are difficult to process and analyze, even for talented experts in the field using sophisticated tools. In addition, ML classification and clustering of flows can help identify network hotspots and potential bottlenecks. When using bandwidth and QoS of flows as classifiers, we can use Traffic Engineering (TE) to adjust flow paths and add virtual resources to the network infrastructure. Finally, identification of applications or web-based protocols is important for forecasting future trends and ensuring the network can meet the demand. Network classification is therefore of great interest to ISPs, governments, and enterprise alike.

LITERATURE SURVEY

Traffic Classification has become an important research area especially with the advancements of Machine Learning and Software Defined Networking. In this project, two machine learning models – Logistic Regression (supervised) and K-Means Clustering (unsupervised) were used to classify DNS, Telnet, Ping, and Voice traffic flows simulated by the Distributed Internet Traffic Generator (D-ITG) tool. Each host in the network was connected through an overlay network to an Open vSwitch (OVS). The OVS was connected to a Ryu controller which collected basic flow statistics between hosts. These statistics were then parsed by a Python traffic classification script which periodically outputted the learned traffic labels of each flow. Logistic Regression was found to work much better than K-Means Clustering. Further improvements in the project could be to add functionality to detect unique traffic flows between the same pair of source and destination.

METHODOLOGY

The datasets of each different traffic flows have been considered.

• It does not include redundant records in the train set, so the classifiers will not be biased towards more frequent records.
There are no duplicate records in the proposed test set, therefore, the performance of the learners is not biased by the methods which have better detection rates on the frequent records.

• The number of records in the train and test sets are reasonable, which makes it affordable to run the experiments on the complete set without the need to randomly select a small portion.
Consequently, evaluation results of different research works will be consistent and comparable.
The dataset is divided in the ratio 70%-30% such that the train dataset consists of 70% of data and the testing dataset consists of remaining 30% data

Then we will import the libraries we need to run the following code.

```
In [152]: import numpy as np
          import pandas as pd
          from scipy import stats
          import pickle

          # Plotting libraries
          import seaborn as sns
          import matplotlib.pyplot as plt

          # Sklearn libraries
          from sklearn.preprocessing import StandardScaler
          from sklearn.preprocessing import label_binarize
          from sklearn.model_selection import train_test_split
          from sklearn.decomposition import PCA
          from sklearn.linear_model import LogisticRegression
          from sklearn.cluster import DBSCAN, KMeans
          from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

          # Filter warnings
          warnings.filterwarnings('ignore') #filter warnings
          # Show plots inline
          %matplotlib inline
```

Loading Data

Here we load the CSV data collected from the Python script into pandas dataframe

```
In [153]: ping_df = pd.read_csv('ping_training_data.csv', delimiter='\t')
          voice_df = pd.read_csv('voice_training_data.csv', delimiter='\t')
          dns_df = pd.read_csv('dns_training_data.csv', delimiter='\t')
          telnet_df = pd.read_csv('telnet_training_data.csv', delimiter='\t')
          df = pd.concat([ping_df, voice_df, dns_df, telnet_df], ignore_index=True)
```

Cleaning Data

```
In [156]: print(df.shape)

          (5242, 13)
```

We can take a look at basic statistical information about our data now.

```
In [157]: df.describe()
```

Out[157]:

	Delta Forward Packets	Delta Forward Bytes	Forward Instantaneous Packets per Second	Forward Average Packets per second	Forward Instantaneous Bytes per Second	Forward Average Bytes per second	Delta Reverse Packets	Delta Reverse Bytes	DeltaReverse Instantaneous Packets per Second	Reverse Average Packets per second	Reverse Instanta Bytes p Second
count	5242.000000	5242.000000	5242.000000	5242.000000	5242.000000	5242.000000	5242.000000	5242.000000	5242.000000	5242.000000	5242.00
mean	7.814765	682.359214	7.763640	6.094360	677.842999	530.202926	17.525754	2070.781000	17.447921	15.710718	2064.79
std	27.445539	2417.020821	27.343341	10.656380	2407.997270	937.773493	31.726859	3404.747532	31.645749	18.427478	3401.35
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.000000	0.000000	0.000000	0.909309	0.000000	62.009611	1.000000	71.000000	1.000000	1.020725	71.0000
50%	1.000000	66.000000	1.000000	1.021459	66.000000	98.762821	1.000000	98.000000	1.000000	1.430508	98.0000
75%	1.000000	98.000000	1.000000	1.038462	98.000000	99.485632	37.000000	3696.000000	37.000000	34.969697	3541.00
max	211.000000	18581.000000	211.000000	76.750000	18581.000000	6711.000000	210.000000	15484.000000	210.000000	76.500000	15484.0

```
In [160]: df['Traffic Type'].cat.categories

Out[160]: Index(['dns', 'ping', 'telnet', 'voice'], dtype='object')
```

We can also get the data coded numerically using `.cat.codes`

```
In [161]: df['Traffic Type'].cat.codes.head()

Out[161]: 0    1
          1    1
          2    1
          3    1
          4    1
          dtype: int8
```

The following features will be used in the model

```
In [163]: print ('Value counts:')
          df['Traffic Type'].value_counts()

          Value counts:

Out[163]: ping      1770
          telnet    1181
          dns       1154
          voice     1137
          Name: Traffic Type, dtype: int64
```

First we will split the dataset into features and targets.

In [183]:
```
X = df.drop('Traffic Type',axis=1)
y = df['Traffic Type']
```

Create training and testing sets

We will use train_test_split with test size of 0.3 to put 70% of our data into training, and 30% into testing. The random_state is set so the results are repeatable.

In [184]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5, random_state=101)
```

Train model

Now we will create and train the model.

In [185]:
```
model = LogisticRegression()
```

In [186]:
```
model.fit(X_train,y_train)
```

Out[186]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

Make predictions

Single prediction

In [187]:
```
idx = 2590 #random number
single_x_test = [df.iloc[idx].drop('Traffic Type').tolist()]
single_y_test = df.iloc[idx]['Traffic Type']
```

In [188]:
```
single_prediction = model.predict(single_x_test)
print('For this sample, our model predicted %s and it was actually %s' % (single_prediction[0], single_y_test))
```

```
For this sample, our model predicted voice and it was actually voice
```

In [189]:
```
predictions = model.predict(X_test)
```

We can create a dataframe to see these in table form:

In [190]:
```
resultsDF = pd.DataFrame({
        'true':y_test,
        'predicted':predictions
    })
resultsDF.head()
```

Out[190]:

	true	predicted
3156	dns	dns
2530	voice	voice
3129	dns	dns
2301	voice	voice
2746	voice	voice

We see the model has a **99.68%** accuracy

In [191]:
```
print('Accuracy: %.2f%%' % (accuracy_score(predictions,y_test)*100))
```

```
Accuracy: 99.54%
```

We can save the model using the pickle library to use later in real-time

In [193]:
```
print(pickle.format_version)
```

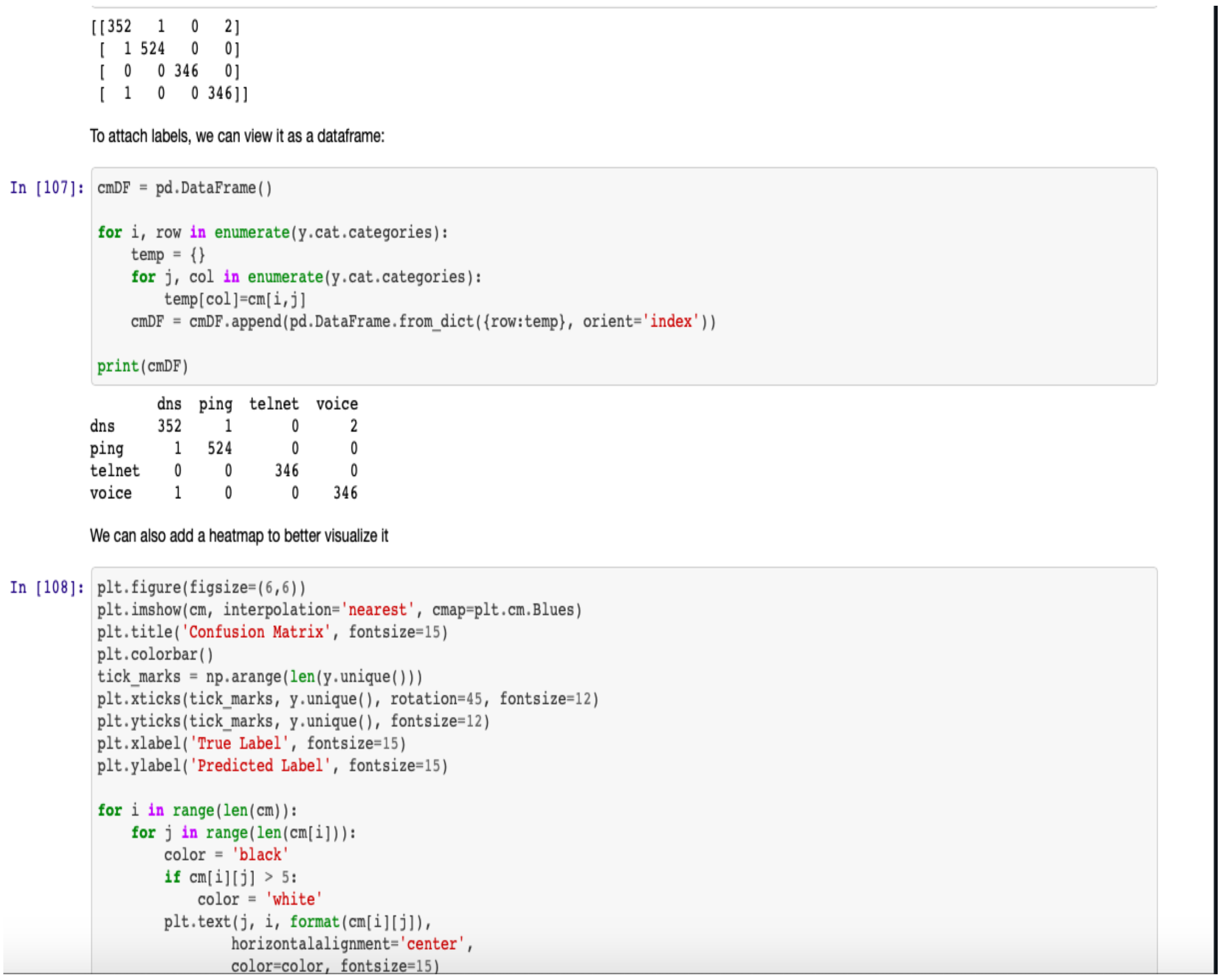
```
4.0
```

In [192]:
```
pickle.dump(model,open('LogisticRegression','wb'))
```

Confusion Matrix

The confusion matrix allows you to see the numerical breakdown of the predictions by class:

In [106]:
```
cm = confusion_matrix(predictions,y_test, labels=y.cat.categories)
```



```
[[352   1   0   2]
 [  1 524   0   0]
 [  0   0 346   0]
 [  1   0   0 346]]
```

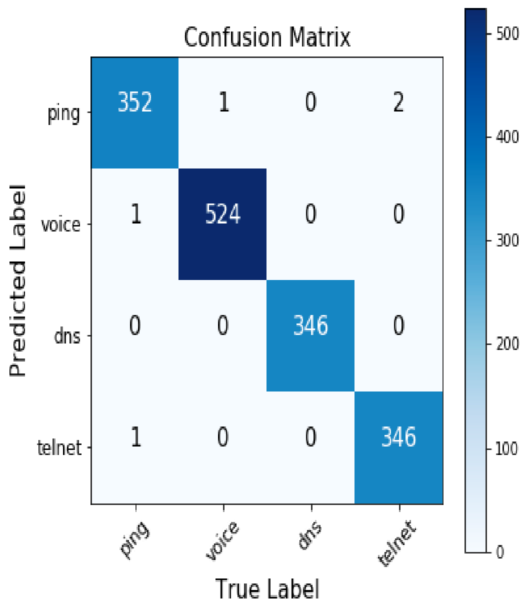
To attach labels, we can view it as a dataframe:

```
In [107]: cmDF = pd.DataFrame()

          for i, row in enumerate(y.cat.categories):
              temp = {}
              for j, col in enumerate(y.cat.categories):
                  temp[col]=cm[i,j]
              cmDF = cmDF.append(pd.DataFrame.from_dict({row:temp}, orient='index'))

          print(cmDF)
```

```
        dns  ping  telnet  voice
dns     352     1       0      2
ping      1   524       0      0
telnet    0     0     346      0
voice     1     0       0    346
```

We can also add a heatmap to better visualize it

```
In [108]: plt.figure(figsize=(6,6))
          plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
          plt.title('Confusion Matrix', fontsize=15)
          plt.colorbar()
          tick_marks = np.arange(len(y.unique()))
          plt.xticks(tick_marks, y.unique(), rotation=45, fontsize=12)
          plt.yticks(tick_marks, y.unique(), fontsize=12)
          plt.xlabel('True Label', fontsize=15)
          plt.ylabel('Predicted Label', fontsize=15)

          for i in range(len(cm)):
              for j in range(len(cm[i])):
                  color = 'black'
                  if cm[i][j] > 5:
                      color = 'white'
                  plt.text(j, i, format(cm[i][j]),
                          horizontalalignment='center',
                          color=color, fontsize=15)
```



```
In [109]: df.drop('Traffic Type',axis=1).values[0:5]

Out[109]: array([[ 0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,
            0. ],
          [ 1. , 98. ,  1. ,  1. , 98. , 98. ,  0. ,  0. ,  0. ,  0. ,  0. ,
            0. ],
          [ 1. , 98. ,  1. ,  1. , 98. , 98. ,  0. ,  0. ,  0. ,  0. ,  0. ,
            0. ],
          [ 1. , 98. ,  1. ,  1. , 98. , 98. ,  0. ,  0. ,  0. ,  0. ,  0. ,
            0. ],
          [ 1. , 98. ,  1. ,  1. , 98. , 98. ,  1. , 98. ,  1. ,  0.5, 98. ,
           49. ]])
```

Here are the means/std per feature.

```
In [110]: df.drop('Traffic Type',axis=1).values.mean(axis=0)

Out[110]: array([   7.81476536,  682.35921404,    7.76363983,    6.09435952,
           677.84299886,  530.2029264 ,   17.52575353, 2070.78099962,
            17.44792064,   15.71071753, 2064.7954979 , 1943.8291939 ])

In [111]: df.drop('Traffic Type',axis=1).values.std(axis=0)

Out[111]: array([  27.44292102, 2416.7902663 ,   27.34073283,   10.65536331,
          2407.76757632,  937.68404081,   31.72383282, 3404.4227598 ,
            31.64272994,   18.42572049, 3401.03157782, 2739.70853918])
```

Here, we fit the scaler.

```
In [112]: scaler = StandardScaler()
          scaler.fit(df.drop('Traffic Type',axis=1))

Out[112]: StandardScaler(copy=True, with_mean=True, with_std=True)

In [113]: scaled_data = scaler.transform(df.drop('Traffic Type',axis=1))
          scaled_data[0:5]

Out[113]: array([[-0.28476434, -0.2823411 , -0.28395873, -0.5719523 , -0.28152344,
           -0.56543879, -0.55244754, -0.608262  , -0.55140377, -0.85265146,
           -0.60710859, -0.70950218],
          [-0.24832507, -0.24179145, -0.24738327, -0.47810285, -0.24082183,
           -0.46092597, -0.55244754, -0.608262  , -0.55140377, -0.85265146,
           -0.60710859, -0.70950218],
          [-0.24832507, -0.24179145, -0.24738327, -0.47810285, -0.24082183,
           -0.46092597, -0.55244754, -0.608262  , -0.55140377, -0.85265146,
           -0.60710859, -0.70950218],
          [-0.24832507, -0.24179145, -0.24738327, -0.47810285, -0.24082183,
           -0.46092597, -0.55244754, -0.608262  , -0.55140377, -0.85265146,
           -0.60710859, -0.70950218],
          [-0.24832507, -0.24179145, -0.24738327, -0.47810285, -0.24082183,
           -0.46092597, -0.5209255 , -0.57947592, -0.51980094, -0.82551548,
           -0.57829381, -0.69161707]])
```

Here are the new means and standard deviation per feature.

```
In [114]: scaled_data.mean(axis=0)

Out[114]: array([ 0.00000000e+00, -2.16876837e-17,  0.00000000e+00,  1.73501469e-16,
          -2.16876837e-17, -1.30126102e-16, -5.42192091e-17,  3.25315255e-17,
           3.25315255e-17, -6.50630510e-17, -9.21726555e-17, -2.16876837e-17])

In [115]: scaled_data.std(axis=0)

Out[115]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Now we will fit PCA model to the data. We will specify n_components=2, because we only want the first 2 principal components.

```
In [116]: pca = PCA(n_components=2)
          pca.fit(scaled_data)

Out[116]: PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
            svd_solver='auto', tol=0.0, whiten=False)

In [117]: scaled_data.shape

Out[117]: (5242, 12)
```

```
Out[120]: array([0.5646886, 0.2769479])
```

```
In [121]: pca.explained_variance_ratio_.sum()*100
```

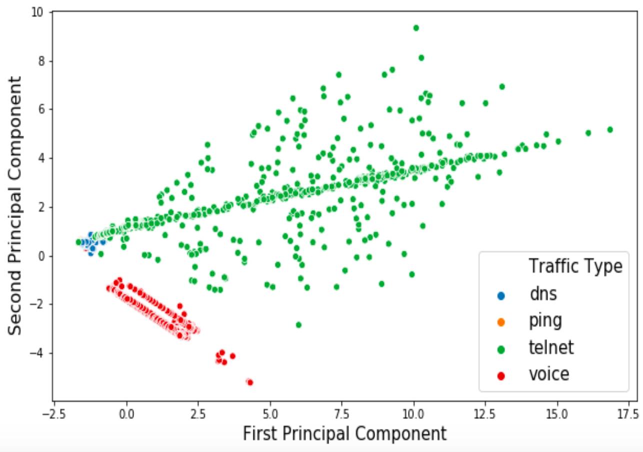
```
Out[121]: 84.16364984328655
```

From above you can see that our first 2 principal components explain 84.16% of the variance in our data. We can get higher variance explained by increasing the number of principal components to a maximum of 100% with n_components = n_features.

Plotting the principal components

```
In [122]: fig = plt.figure(figsize=(10,6))
          sns.scatterplot(x_pca[:,0], x_pca[:,1], hue=df['Traffic Type'])
          plt.xlabel('First Principal Component', fontsize=15)
          plt.ylabel('Second Principal Component', fontsize=15)
          plt.legend(fontsize=15)
```

```
Out[122]: <matplotlib.legend.Legend at 0x7ffb08ea85f8>
```



```
In [123]: X_train, X_test, y_train, y_test = train_test_split(x_pca,y.cat.codes,test_size=0.3, random_state=101)
```

```
In [124]: model = LogisticRegression()
          model.fit(X_train,y_train)
```

```
Out[124]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, max_iter=100, multi_class='warn',
                    n_jobs=None, penalty='l2', random_state=None, solver='warn',
                    tol=0.0001, verbose=0, warm_start=False)
```

As discussed above, the accuracy value is lower when we only use the first 2 principle components as opposed to the full set of features. Again, this is just for visualization purposes.

```
In [125]: predictions = model.predict(X_test)
          print('Accuracy: %.2f%%' % (accuracy_score(predictions,y_test)*100))
```

```
Accuracy: 78.89%
```

Plotting

We will first generate a grid of x[0] and x[1] values that we will use to make predictions with.

```
In [126]: x_min = x_pca[:,0].min()
          x_max = x_pca[:,0].max()
          y_min = x_pca[:,1].min()
          y_max = x_pca[:,1].max()
          spacing = 0.01
```

```
In [127]: xx, yy = np.meshgrid(np.arange(x_min, x_max, spacing), np.arange(y_min, y_max, spacing))
```

Now we will make predictions on the grid that we created. The `ravel` function just makes the 2D array that we have above into a 1D array. We will reshape the predictions Z into a 2D array afterwards for plotting

```
In [128]: Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
          Z = Z.reshape(xx.shape)
```

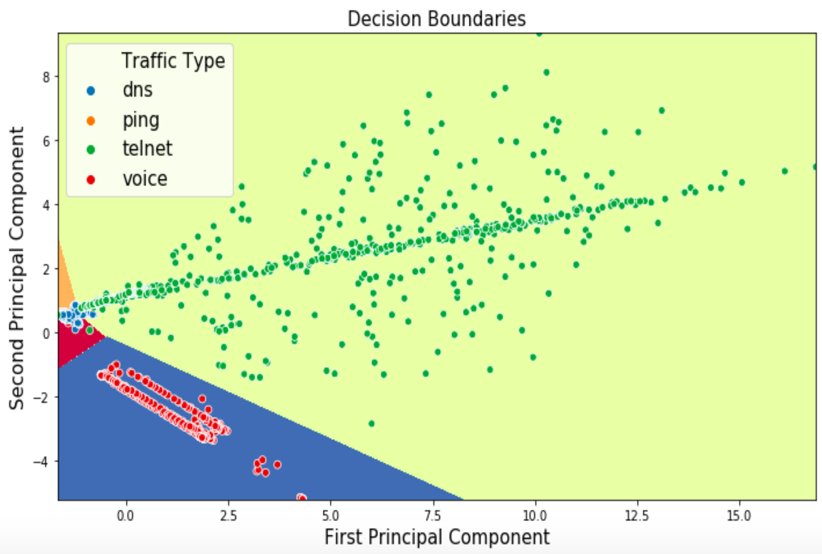
Now we will plot the data, and the decision boundaries.

```
In [129]: plt.figure(figsize=(10,6))
          plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
```

Now we will plot the data, and the decision boundaries.

```
In [129]: plt.figure(figsize=(10,6))
          plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
          sns.scatterplot(x_pca[:,0], x_pca[:,1], hue=df['Traffic Type'], cmap=plt.cm.Spectral)
          plt.title('Decision Boundaries', fontsize=15)
          plt.xlabel('First Principal Component', fontsize=15)
          plt.ylabel('Second Principal Component', fontsize=15)
          plt.tight_layout()
          plt.xlim([x_min,x_max])
          plt.ylim([y_min,y_max])
          plt.legend(fontsize=15)
```

```
Out[129]: <matplotlib.legend.Legend at 0x7ffb08e35ba8>
```



```
In [130]: X = df.drop('Traffic Type',axis=1)
          y = df['Traffic Type']
```

```
In [131]: model = KMeans(n_clusters=len(y.cat.categories))
```

Train & predict

```
In [132]: clusters = model.fit_predict(X)
```

Evaluate generated clusters

Shape of clusters

```
In [133]: model.cluster_centers_.shape
```

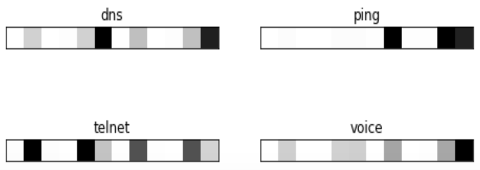
```
Out[133]: (4, 12)
```

We see that there are 4 clusters in 12 dimensions.

Visualize clusters

Let's visualize each of these cluster centers to see what they represent. The 12 size vector is reshaped into a 4x3, and visualized using matplotlib.

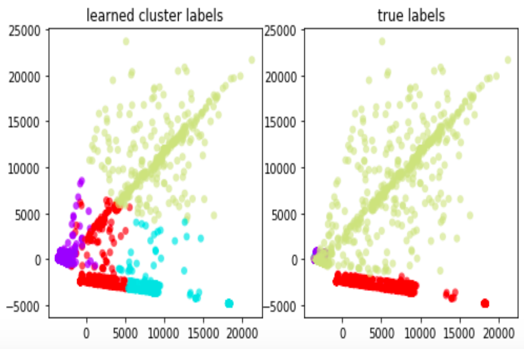
```
In [134]: fig = plt.figure(figsize=(8, 3))
          for i in range(len(y.cat.categories)):
              ax = fig.add_subplot(2, 2, 1 + i, xticks=[], yticks=[])
              ax.set_title(str(y.cat.categories[i]))
              ax.imshow(model.cluster_centers_[i].reshape((1, 12)), cmap=plt.cm.binary)
```

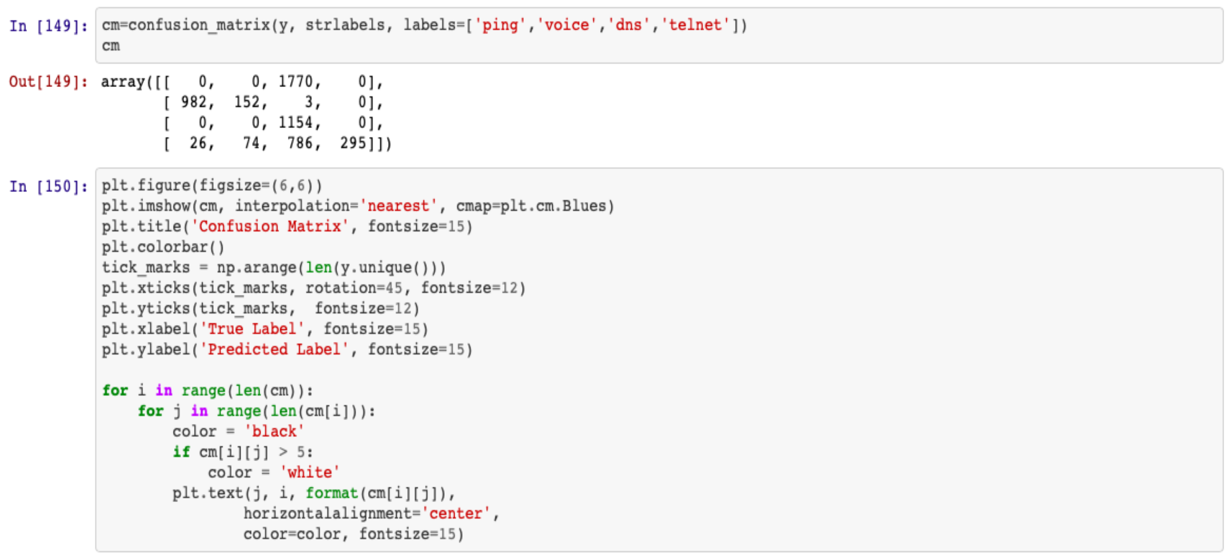


In [137]:
```
df['Traffic Type'].value_counts()
```
Out[137]:
```
ping      1770
telnet    1181
dns       1154
voice     1137
Name: Traffic Type, dtype: int64
```
In [138]:
```
y_codes.tolist().count(0) #DNS
```
Out[138]:
```
1154
```
In [139]:
```
y_codes.tolist().count(1) #Ping
```
Out[139]:
```
1770
```
In [140]:
```
y_codes.tolist().count(2) #Telnet
```
Out[140]:
```
1181
```
In [142]:
```
y_codes.tolist().count(3) #Voice
```
Out[142]:
```
1137
```
In [143]:
```
strlabels = ['']*len(y)
for i in range(len(clusters)):
    if clusters[i]==0: strlabels[i] = 'dns'
    elif clusters[i]==1: strlabels[i] = 'ping'
    elif clusters[i]==2: strlabels[i] = 'telnet'
    elif clusters[i]==3: strlabels[i] = 'voice'
```
Accuracy:

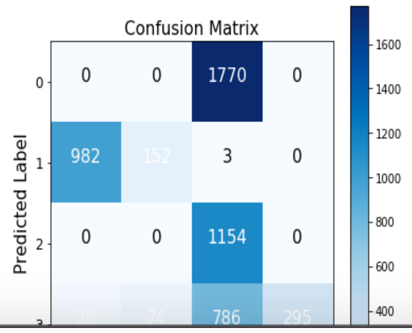
In [144]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y, strlabels)*100.0
```
Out[144]:
```
30.541777947348344
```
The accuracy is very poor at just **%30.54**. With only 4 possible cluster labels, the model is pretty much guessing the right label. We will see why using PCA analysis

In [145]:
```
from sklearn.decomposition import PCA
X = PCA(2).fit_transform(X)
```
Plot clusters from PCA and true labels:

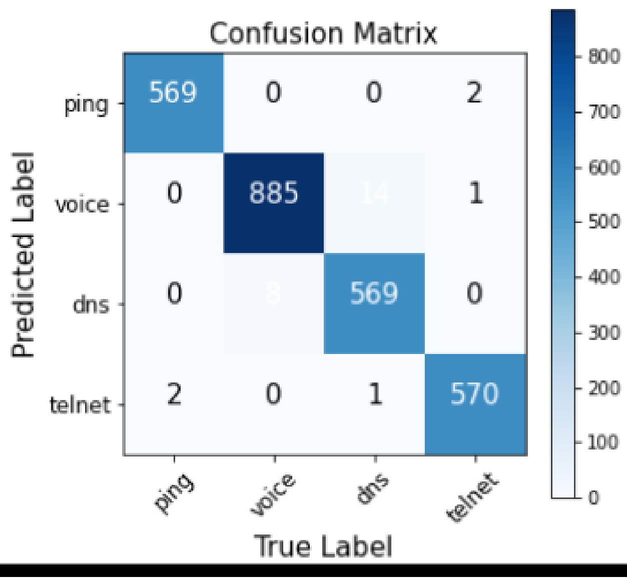
In [146]:
```
y_codes =  np.asarray(y.cat.codes)
```
In [147]:
```
labels = np.zeros_like(clusters) # Create an array of 0s with equal length ot the number of clusters

# Set labels based on the modes of the target
for i in range(len(y.cat.categories)):
    mask = (clusters == i)
    labels[mask] = stats.mode(y_codes[mask])[0]
print (labels)
```
```
[1 1 1 ... 1 1 1]
```
In [148]:
```
kwargs = dict(cmap = plt.cm.get_cmap('rainbow', 4),
              edgecolor='none', alpha=0.6)
fig, ax = plt.subplots(1, 2, figsize=(8, 4))
ax[0].scatter(X[:, 0], X[:, 1], c=clusters, **kwargs)
ax[0].set_title('learned cluster labels')

ax[1].scatter(X[:, 0], X[:, 1], c=y_codes, **kwargs)
ax[1].set_title('true labels');
```


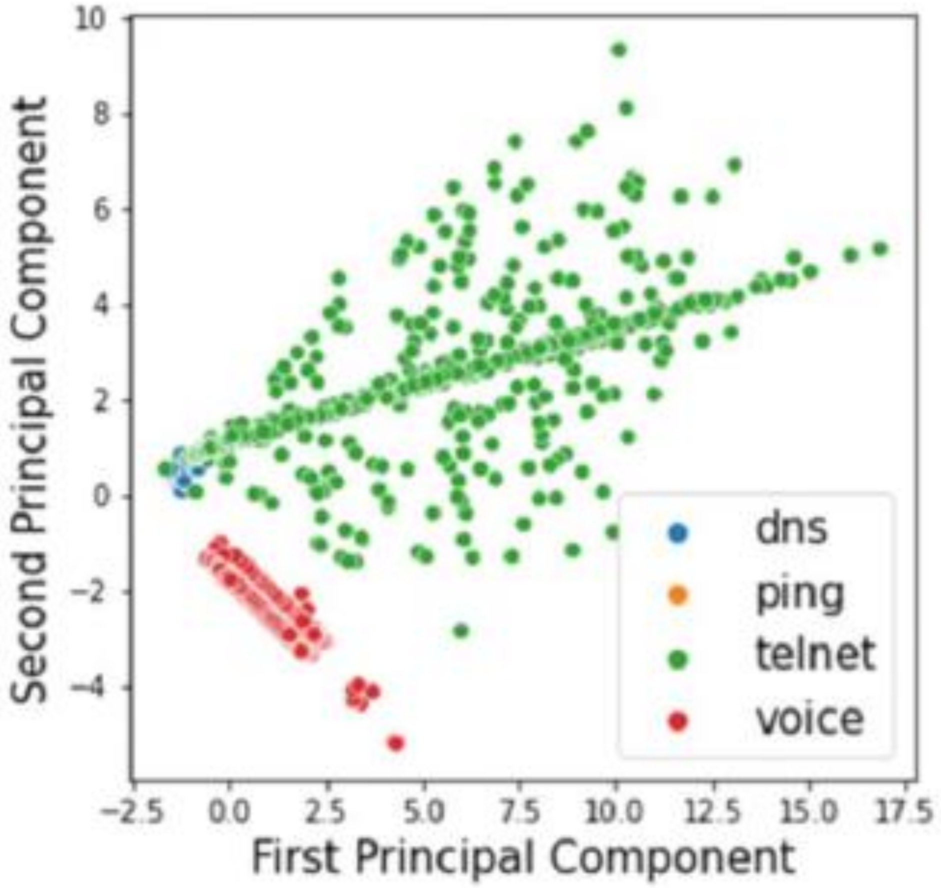


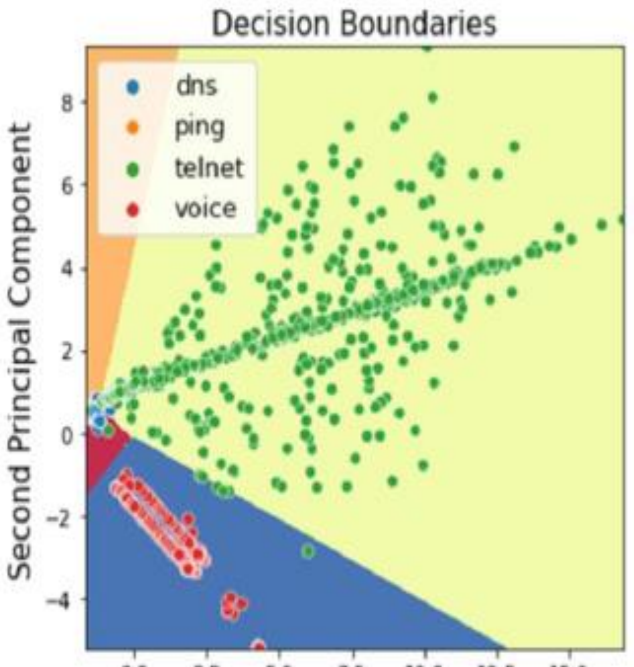
```
In [149]: cm=confusion_matrix(y, strlabels, labels=['ping','voice','dns','telnet'])
          cm

Out[149]: array([[   0,    0, 1770,    0],
                 [ 982,  152,    3,    0],
                 [   0,    0, 1154,    0],
                 [  26,   74,  786,  295]])

In [150]: plt.figure(figsize=(6,6))
          plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
          plt.title('Confusion Matrix', fontsize=15)
          plt.colorbar()
          tick_marks = np.arange(len(y.unique()))
          plt.xticks(tick_marks, rotation=45, fontsize=12)
          plt.yticks(tick_marks,  fontsize=12)
          plt.xlabel('True Label', fontsize=15)
          plt.ylabel('Predicted Label', fontsize=15)

          for i in range(len(cm)):
              for j in range(len(cm[i])):
                  color = 'black'
                  if cm[i][j] > 5:
                      color = 'white'
                  plt.text(j, i, format(cm[i][j]),
                          horizontalalignment='center',
                          color=color, fontsize=15)
```



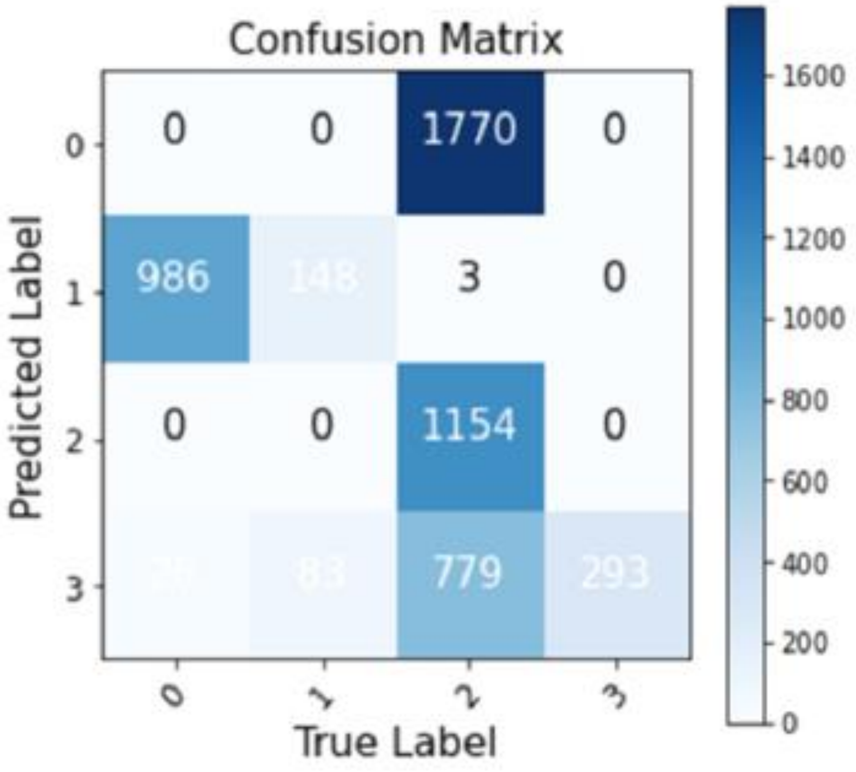
RESULTS

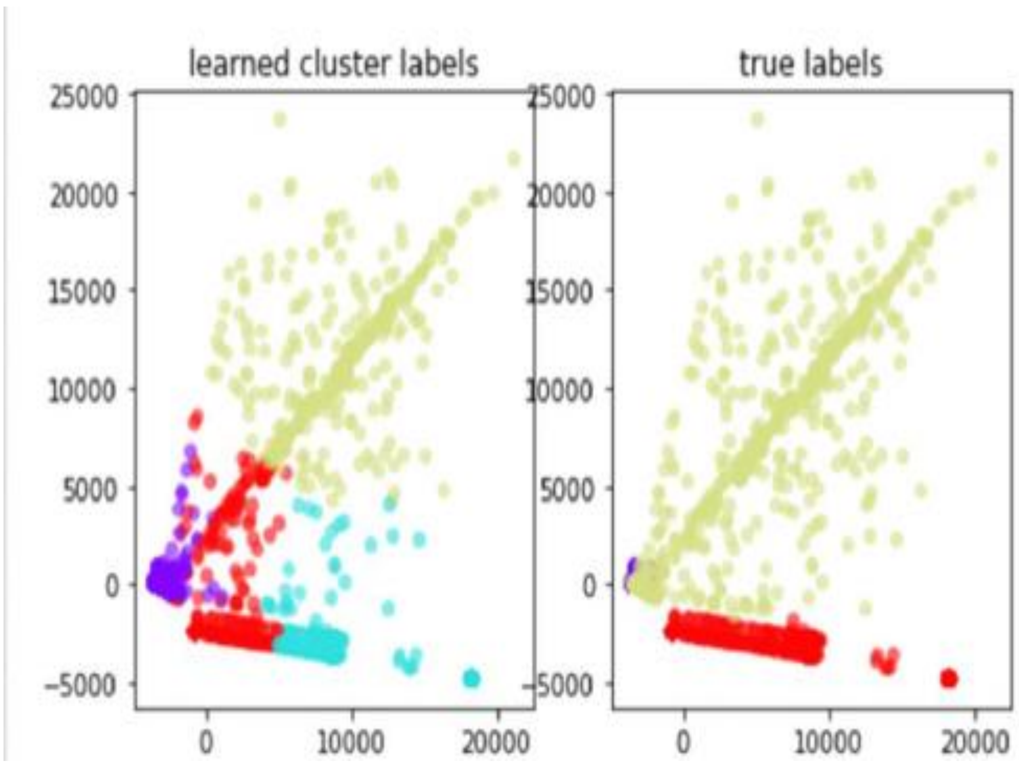



Second Principal Component
First Principal Component
dns
ping
telnet
voice


Decision Boundaries
dns
ping
telnet
voice
Second Principal Component
First Principal Component

K-Means Clustering:





From confusion matrix and the above graphs, that Logistic Regression works much better in terms of classifying the traffic flows (telnet, DNS, ping, voice) in SDN based networks environment into corresponding class labels compared to K-Means Clustering.

CONCLUSION:

After analysing network data collected by SDN controller and classifying the traffic of the considered SDN based networks and integrate the model in software-defined networking platform we arrive at the conclusion that Logistic Regression works much better in terms of classifying the traffic flows (telnet, DNS, ping, voice) of SDN based networks into corresponding class labels compared to K-Means Clustering.
Thus, for network traffic classification. It can be inferred that traffic classification using machine learning algorithms provides good results within SDN environment because of the ability of collecting information in this type of architecture.
Therefore, Logistic Regression can be used for traffic classification in SDN environment.