Assignment 6: Anomaly Detection in Network Traffic Data

Arash Vahdat

Fall 2015

Readings You are highly recommended to go through the following readings while doing this assignment:

- **Clustering-based Anomaly Detection:** Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. "Intrusion detection with unlabeled data using clustering", In Proceedings of ACM CSS Workshop on Data Mining Applied to Security, 2001.
- **K-means Clustering:** Ch. 9.1, Bishop, "Pattern recognition and machine learning", Springer, 2006.
- **Spark Streaming:** Spark Documentation.
- **Streaming K-means:** Spark Documentation.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE):** `http://lvdmaaten.github.io/tsne/`

Provided Data

`/cs/vml2/avahdat/CMPT733_Data_Sets/Assignment6`

1 Introduction

Anomaly detection correspond to finding items or events that deviate from the expected normal pattern of items or events. Anomaly detection has applications in fraud detection, network intrusion, and security systems. In this assignment, we will work on intrusion detection which correspond to detecting anomalies in large networks. This problem is a practical example of a streaming application that process a large amount of streaming data. In our study scenario, a real-time process makes decision on whether a connection is an attack or not. Such system can be deployed in practical applications to alarm network experts from possible security breaches.

We will use Spark Streaming library as our processing tool. Spark Streaming is an extension of the core Spark API which enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Large datasets can be ingested in Spark Streaming library from many sources like Kafka, Flume, Twitter, ZeroMQ, Kinesis, or TCP sockets, and can be then processed using complex algorithms expressed with high-level functions like map, reduce, join and window. Finally, this processed data can be pushed out to file systems, databases, and live dashboards.

We will work with a subset of the KDD Cup 1999 dataset[1] which contains approximately 1 million samples. Each sample corresponds to a network connection and is represented using features such as "connexion type", "number of bytes sent", "login attempts", and "TCP errors" extracted from the original connection. Each sample is also labeled as "normal" vs. "attack" connection. A sample in this CSV formatted dataset looks like:

```
0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,
0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.
```

Please refer to the original dataset for detailed description of features used to represent a connection. We will use as feature descriptor for a connexion all entries except the last in each row of the CSV file. The last entry will serve as annotation. If the last entry in a row contains “normal.”, the sample will be assumed to be a normal connection. Anything other than “normal.” (including “smurf”, “neptune”, “satan”, etc.) will be assumed to be an attack. Your goal in this assignment is to train a model that is capable of detecting attacks from normal network connections data.

The data directory provided with the assignment contains a compressed file named `kddcup.data.tar.gz`. Download the file and uncompress it using:

```
tar -xvzf kddcup.data.tar.gz
```

After uncompression, you will see the following items:

- `kddcup.data_0.01.csv`: a CSV file containing about 10,000 instances (one line per sample). This file corresponds to 1% of the whole data and will be used for training K-means clusters.
- `kddcup.data_all`: a directory containing 22 CSV-formatted files. These files correspond to the whole dataset with about 1,000,000 instances and it will be used later for testing our streaming system.

2 K-means Clustering

K-means clustering algorithm is an unsupervised learning algorithm that groups training instances in clusters based on a notion of similarity. More specifically, K-means tries to form K-user-defined clusters that have minimum within cluster squared error.

Portnoy et al. in the paper “Intrusion detection with unlabeled data using clustering” proposed a simple clustering-based anomaly detection approach. Their main observation was that normal connections are frequent whereas attacks are very rare. This implies that a clustering algorithm will create large clusters for normal connections and small isolated clusters for anomalies. We will examine this observation in more details on the provided KDD Cup 1999 dataset. However, we will adopt a slightly different approach than the one proposed by Portnoy et al.’s by using K-means algorithm for clustering.

[1] The KDD Cup 1999 dataset is available at `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

Question 1 Read Data: (5 Marks) The provided dataset is organized as a CSV table. Write a function to parse each line (i.e. each sample) of the **`kddcup.data_0.01.csv`** file. Your function can return each sample using a tuple in the form of `(feature, label)`. Use the provided features to cluster the data and use the labels to evaluate the quality of your clusters. Make sure that the returned labels are 0 for "normal." connections and 1 otherwise.

As you may remember each line of our data looks like:

```
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.
```

Please note that the second, third and forth entries in each line (e.g. `tcp,http,SF`) are categorical features. For example, the second entry can be one of the following cases: "tcp", "udp", or, "icmp". You can represent these categorical features using a hot representation. In other words, the second entry can be replaced with $[1,0,0]$ for "tcp", $[0,1,0]$ for "udp", and, $[0,0,1]$ for "icmp". Find all possible cases for the second and forth entries in the dataset, and, represent each one of these entries using one hot key representation. **Please ignore the third entry.**

Question 2 Normalize Features: (2 Marks) You can now use the function implemented in Q2 to parse your data from the **`kddcup.data_0.01.csv`** file. Read the training data from this file to an RDD. You may notice that some feature dimensions have a very large range compared to others. The K-means algorithm measures distance between two instances using the Euclidean distance. This distance measure is sensitive to features' scale. As seen in Assignment 1, you can use "zero-mean unit-variance normalization" (a.k.a. standardization) normalize your features scales. Use `pyspark.mllib.feature.StandardScaler` to normalize your features.

Question 3 Train K-means: (2 Marks) Use Spark's MLlib library to train a K-means model. Set the number of clusters K to the following values: $\{5,8,10,12,20\}$. Use maxIterations=40, runs=5, initializationMode="random". Fix the seed argument so that your K-means uses a similar initial point in different experiments. See the next question for setting K and seed.

Question 4 Evaluate Clusters: (8 Marks) As you may remember, the intuition of Portnoy et al. was that clusters with a small number of instances will correspond to attacks or anomalies. We use this intuition to generate a confidence score from the clustering model's output. The confidence score reflects how much the clustering model believes an instance is an attack or not. Let's assume x_i is a data point describing a network connection, and, it is assigned to the $c(x_i)$ cluster by our K-means model. We can use:

$$f(x_i) = \frac{N_{max} - N_{c(x_i)}}{N_{max} - N_{min}} \tag{1}$$

to score x_i as being an anomaly. Note that in this equation, N_{max} and N_{min} reflect the size of the largest and smallest clusters, respectively. $N_{c(x_i)}$ represents the size of the cluster assigned to x_i. If you check the equation carefully, you will notice that $f(x_i) = 1$ when x_i is assigned to the smallest cluster and $f(x_i) = 0$ when x_i is assigned to a large cluster.

Use the function in Eq. 1 to score every instance in **`kddcup.data_0.01.csv`**. Evaluate the quality of your clusters using these confidence scores and the ground truth labels:

- Plot a ROC curve to visualize the behaviour of your model when using different thresholds. Use semilogx as axis to your plot.
- Compute the area under ROC curve as a clustering measure. Try different values as K and seed, and, choose the setting that gives the highest area under ROC curve you achieved.
- As discussed during lectures, ROC curves can be used to select a threshold on the confidence score while considering both True Positive Rate (TPR) and False Positive Rate (FPR). Here, we are looking for a threshold that results in a high TPR ($\sim 90\%$) and low FPR ($\sim 0.01\%$). Such threshold may correspond to predicting data points in certain clusters with the anomaly label. Which clusters are those? Explain why assigning data points in those clusters will result in a high TPR and low FPR.

You can use `pyspark.mllib.evaluation.BinaryClassificationMetrics` to compute the area under ROC curve and `sklearn.metrics.roc_curve` to draw the ROC curve.

3 Cluster Visualization Based on the t-SNE Method

The previous section described a simple technique to quantitatively measure the quality of a clustering model. Despite its simplicity, this technique requires ground truth annotations on the test set which limits its applicability to real-life scenarios. An alternative is to qualitatively evaluate the K-means clusters obtained using some visualization techniques. The below describes how such techniques can be performed and their advantages.

Training samples can be visualized in their feature space. In this case, different colors can be used to visualize samples assigned to different clusters. This type of visualization not only helps a data scientist get an overall idea of the data distribution in the feature space, but also, illustrates how clusters created in the feature space match the data distribution.

However, when the feature space is very high dimensional, visualization becomes infeasible. One solution is to reduce the dimensionality of the feature space using some dimensionality reduction methods. These methods map data points from a high dimensional feature space to a low dimensional space. In the series of dimensionality reduction techniques, t-Distributed Stochastic Neighbor Embedding (t-SNE) is a prize-winning method proposed by Maaten and Hinton[2] to visualize high-dimensional datasets. In this section, we will use this popular technique to visualize our samples and their clusters.

Question 5: (10 Marks) The t-SNE method is available in `sklearn.manifold`. After training K-means clusters in the previous section, use this method to map your training samples to a two-dimensional space, and, visualize them in this space. Use colors and markers to visualize cluster indices and ground truth labels respectively. For example, if you obtained four clusters, use four distinct colors. Use two markers to represent the ground truth labels "normal" and "attack". Fig. 1 illustrates a depiction of clusters and ground truth labels

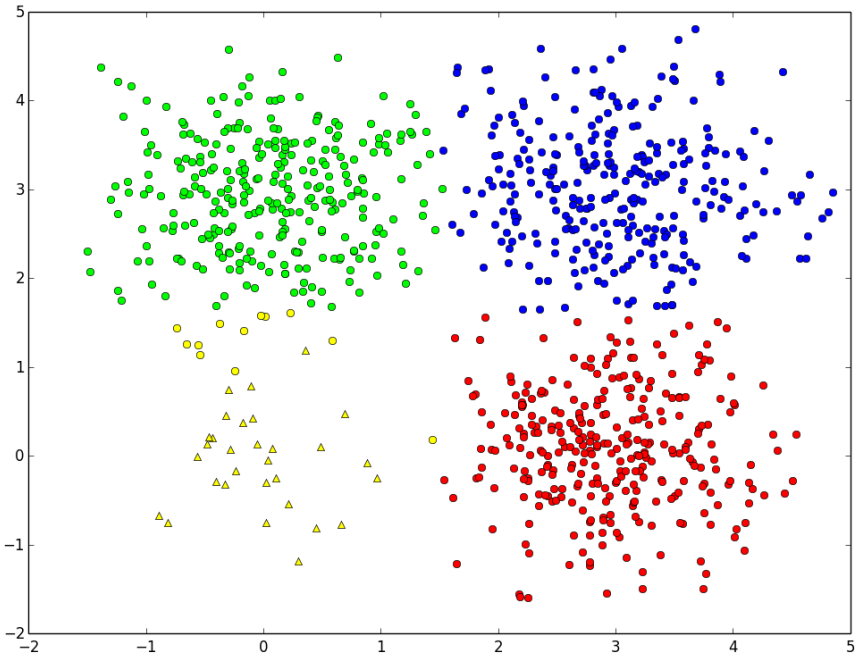

Figure 1: Visualization of clustering and ground truth labels in a toy example. Note that four clusters are visualized with four different colors. Ground truth labels are visualized with markers, i.e., circles for "normal" and triangles for "attack" samples.

in a toy example.

Report your t-SNE visualization for this question using the K-means models that resulted in the highest and lowest area under ROC in your experiments. Compare both models' figures and provide and explanation of why some settings resulted in a low or hight area under ROC. Note that running t-SNE can be very slow, try sampling your training data with relatively small rates to speed-up the visualization.

4 Spark Streaming

In previous questions, you have trained and evaluated (quantitatively and qualitatively) a K-means model for anomaly detection. In this section, you will deploy the trained model using Spark Streaming Library in a real-time system that processes network data on-the-fly on our system.

Question 6 Hello World! (3 Marks) Before diving into the Spark Streaming library, let's try a simple "Hello World!" example available in Spark documentation:

[2]Interested readers refer to `http://lvdmaaten.github.io/tsne/`

https://spark.apache.org/docs/latest/streaming-programming-guide.html#a-quick-example

This is a basic introduction example to streaming applications. The example reads data from a socket and counts the number of words in it. Follow this example and recreate it. Modify it such that it processes the data in batch intervals of 5 seconds. Instead of writing your result to the standard output, create text files.

Question 7 Streaming K-means (10 Marks) In the previous question, you learned how the `nc` command can be used to write data to a port on a host. Here, you can use the same technique to write the network data to a port:

```
$ cd /put/path/to/kddcup.data_all/here
$ ls part-000* | xargs -I % sh -c '{ cat  %; sleep 5;}' | nc -lk 9999
```

The above command will change the current directory to the data directory (the second item in the list of data items provided with this assignment). Then, it will start writing data to a port on your machine, file by file with a 5s pause in between iterations.

Write a Spark Streaming script that reads instances from a socket and decides whether they are "normal" or "attack" connections. The following steps can guide your implementation:

1. Parse socket data into **normalized** feature vectors using the function created in Q1.
2. Create a streaming K-means model as described in Sec. 2. For doing so, you need to first create a `StreamingKMeans` object, then use the `setInitialCenters` method to set cluster centers to those trained in Sec. 2.
3. Use the streaming K-means model to predict labels on the DStreams obtained when parsing the socket data.
4. Store your ground truth and predicted labels in a text file.
5. Write a python script to read the created text file and compute the False Positive Rate (FPR) and True Positive Rate (TPR) using the following equations:

$$TPR = \frac{TP}{P}, \quad FPR = \frac{FP}{N} \tag{2}$$

   where TP, P, FP, and N indicates the number of true positives, positives, false positives and negatives, respectively. Note that in our case, the positive class is "attack" and the negative class is "normal".
6. Visualize the TPR and FPR over time. What are your average TPR and FPR?

Note that to answer this question you may need to store some data computed in the first section of this assignment including your K-means model, word-to-index look-up tables for categorical features, and the standard scaler model. All these models can be easily stored in a file system. However, storing a standard scaler model isn't trivial. For doing so, you can get the numpy vector representation of the mean and standard deviation using:

```
mn = scaler.call('mean').toArray()
st = scaler.call('std').toArray()
```

where scaler is a StandardScaler object.

5 Submission

Your assignment should be submitted online at the following URL: `https://courses.cs.sfu.ca/`. Your submission should include:

1. **Report:** A PDF report including your answers to all questions.
2. **Code:** An archive file containing all scripts you have designed for this assignment. You should create your scripts such that your results can be easily understood and reproduced. Please include a small readme.txt file briefly explaining which script was used for each question.