CME 323: Distributed Algorithms and Optimization, Spring 2015
`http://stanford.edu/~rezab/dao`.
Instructor: Reza Zadeh, Databricks and Stanford.

Lecture 12, 5/6/2015. Scribed by Cedric Garcia, Rohit Dilip Patki and Timothy Sweetser.

12 SPARK Streaming

We will just briefly summarize the concepts involved in **SPARK streaming** in this chapter. The details can be found in the slides available on the course website.

Streaming data analysis has practical applications in fraud detection in bank transactions, identifying anomalies in sensor data, finding cat videos in tweets etc. SPARK framework provides a way to do this in a fairly simple manner.

12.1 Steps in Big *streaming* Data

1. **Ingest**: Receive and buffer the streaming data
2. **Process**: Clean, extract and transform the data
3. **Store**: Store transformed data for consumption

12.2 Systems Involved

Kafka, Kinesis and ***Flume*** are popular data ingestion systems available. For processing data, SPARK ***streaming*** is most demanded today with other systems like ***Storm*** and ***Samza*** also available. ***HDFS, Amazon S3, HBase, MongoDB*** etc help in storing data.

12.3 Understanding the Spark Streaming model

Receivers chop up the input data streams into batches of few seconds. The spark processing engine processes each batch and pushes out the results to external data stores.

Just like **RDD**s, we have **DStreams** which represent stream of data. It is implemented as infinite sequence of **RDD**s. Transformations similar to RDDs can applied to DStreams. Refer to the slides for an example where hashtags are obtained from the incoming tweets.

Spark Streaming integrates very well with the whole Spark ecosystem. We can combine machine learning with streaming by learning models offline and applying them online to the incoming stream of data. Similarly we can interactively query the streaming data with **SQL**.

13 The Perceptron Algorithm

13.1 Introduction

Perceptron is a streaming machine learning algorithm for binary classification. It assumes that the two classes possesses linear separability. In other words, it assumes that there exists a hyperplane w^* called a separator that would perfectly separate both classes of points. In addition, we assume that there exists a value δ so that the distance of every point to the hyperplane is at least δ. This value is called the margin. Note, for a given data set there can be many different separators and margins that satisfy this condition.

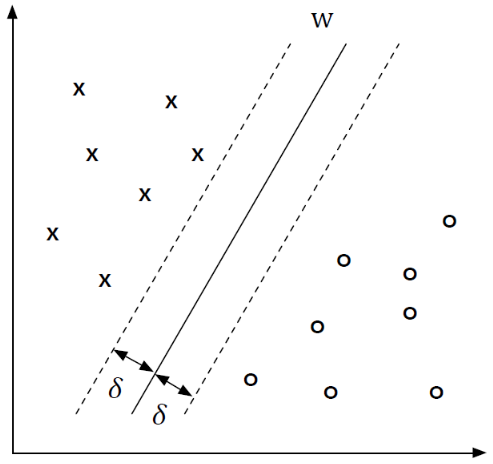

Figure 1: Linear Separability

In this situation, a support vector machine would fit the data perfectly, because it attempts to maximize the margin. Unfortunately, training an SVM requires all of the data, which are not available in the streaming scenario. The perceptron algorithm, on the other hand, is able to train adaptively in the streaming scenario.

One example setting that is similar to this situation this condition is a spam filter. The e-mails come in as a stream, and upon reception of each individual e-mail, the algorithm must classify it either as a legitimate e-mail or as spam. The user may then indicate to the algorithm the true category of the e-mail, by using the "Spam" or "Not Spam" buttons when an e-mail has been misclassified.

More generally, this approach can be summed up as.

1. A new observation X_i comes in.
2. X_i is classified by the algorithm into one of the two classes.
3. If X_i is classified into the wrong class, this is counted as a mistake.

We can prove that the number of mistakes M is bounded by $\frac{1}{\delta^2}$.

13.2 The Algorithm

We begin with an initial guess for a separator, $w_1 = 0$. For each incoming X_i, we will classify that point based on the sign of $w_i^T.X_i$. We then obtain the true class of X_i. For this algorithm we assume that for all $\|x_i\|_2 = 1$ If this is not the case, the incoming data points should be normalized.

```
for $x_i$ in stream do
    if $w^T x_i > 0$ then
        Predict that $x_i$ is in the positive class
    else
        Predict that $x_i$ is in the negative class
    end
    if $x_i$ is actually positive class, but we predicted negative class then
        $w \leftarrow w + x_i$ ;
    end
    if $x_i$ is actually negative class, but we predicted positive class then
        $w \leftarrow w - x_i$;
    end
end
```

We can note that if the point is correctly classified, the algorithm does not update w. When the algorithm misclassifies a point, adding/subtracting that point from w makes it less likely to classify other similar points the same way.
Questions for next class :

- How does the weight of w_i evolve over time?
- Given w^*, a separator that perfectly separates both classes of points, how does $w_i^T.w^*$ evolve over time?