UNIVERSITÀ DEGLI STUDI FIRENZE

Twitter Sentiment Analysis

Sentiment Analysis of tweets using Lambda Architecture

Claudia Raffaelli

A.A. 2019/2020

Introduction

What is Sentiment Analysis?

Sentiment analysis is the interpretation and classification of emotions within voice and text data using text analysis techniques, allowing businesses to identify customer sentiment toward products, brands or services.

Huge amounts of text data, from emails, chats, social media conversations is created every day, but it's hard to analyze, understand, and sort through, not to mention time-consuming and expensive.
Sentiment analysis models, by detecting polarity within text, can help achieve businesses a good understanding of people's thoughts upon their products.

Benefits of sentiment analysis include:

- **Fast processing of big data**, impossible goal to achieve by hand
- **Real-Time** Sentiment analysis can identify critical issues in real-time

Introduction

What is the Lambda Architecture?

Lambda Architecture is a scalable and fault-tolerant data processing architecture, developed trying to satisfy the needs of distributed data.

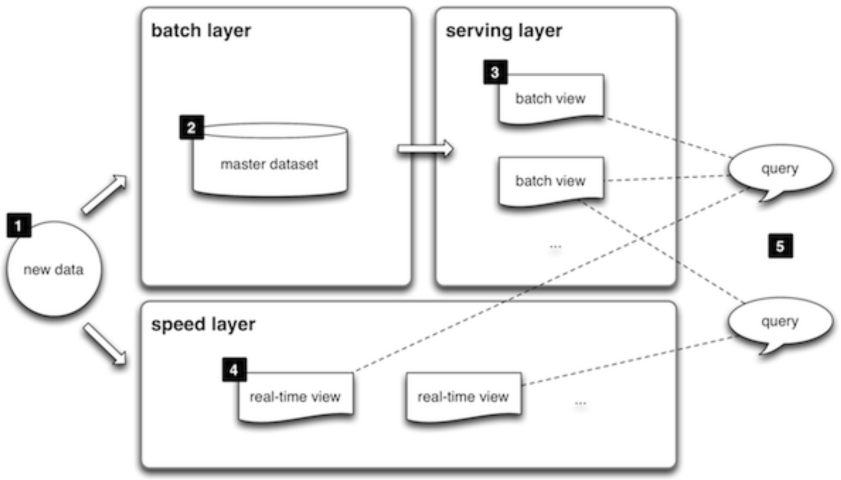
The LA aims to fulfill the basic requirements for:

- A robust system that is **fault-tolerant**, both against hardware failures and human mistakes, being able to serve a wide range of workloads and use cases,
- A system in which **low-latency** reads and updates are required,
- A linearly scalable system, that should **scale out** rather than up.

Introduction

Structure of the Lambda Architecture

Lambda Architecture is composed by three main layers:



Lambda Architecture schema

- **Batch Layer:** The batch layer pre-computes the master database (that contains all data) into batch views so that queries can be resolved with low latency. The batch views are computed each time upon the entire database.

- **Speed Layer:** Is responsible for filling the ”gap” caused by the batch layer’s lag in providing real-time views based on the most recent data. This layer’s views may not be as accurate or complete as the ones eventually produced by the batch layer, but they are available almost immediately after data is received, and can be replaced when the batch layer’s views for the same data become available.

- **Serving Layer:** The outputs from the batch layer in the form of batch views and those coming from the speed layer in the form of real-time views get forwarded to the serving. This layer is then responsible for merging both data in order to provide a complete view concerning the data.

Overview of the proposed approach

Sentiment Analysis of tweets

The realization of the Sentiment Analysis is carried out by making use of the **LingPipe** library. This library provides the main methods for the training and use of the classifier.

At the start of the computation is chosen a set of keywords upon which is done the sentiment analysis. The tweets that are continuously injected into the system are scanned in order to see if they contain one or more keywords in the white list. If this is the case, the classifier, previously trained is used to predict the sentiment.

The classifier is trained in such a way that is capable of recognizing two levels of sentiment:

- A **negative** sentiment, using the label 0
- A **positive** sentiment, with label 1

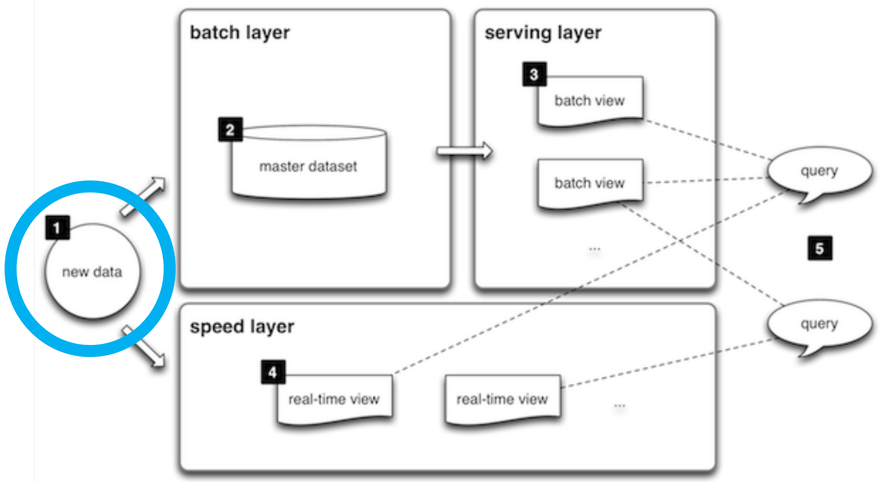
Overview of the proposed approach

Phase 1: Injecting of tweets

A CSV file is used for continuously inject new tweets into the system.
This is done in order to simulate a continuous stream of tweets, reproducing a behavior similar to the arrival of tweets directly from the Twitter platform.

The tweet input is handled using **Apache Storm**, in the form of a Tweet Spout.

As new tweets are received, are forwarded simultaneously to the Batch Layer and Speed Layer.



Lambda Architecture schema

Overview of the proposed approach

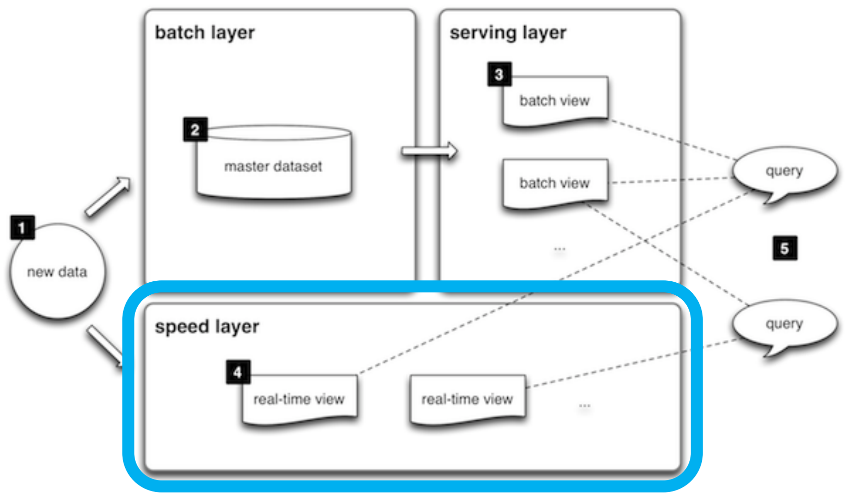
Phase 2: The Speed Layer – Storm Introduction

The **Speed Layer** is implemented using the distributed real-time computation system **Apache Storm**.

A Storm topology is mainly composed of:

- A **stream**: the core abstraction in Storm. A stream is an unbounded sequence of tuples that is processed and created in parallel in a distributed fashion.
- A **spout**: is a source of streams in a topology. Generally spouts will read tuples from an external source and emit them into the topology.
- Multiple **bolts**: a bolt handle a section of processing that is done throughout the entire topology. Bolts can do simple and complex stream transformations, as filtering, joins, functions and more.

Storm guarantees that every spout tuple will be fully processed by the topology.



Lambda Architecture schema

Overview of the proposed approach

Phase 2: The Speed Layer

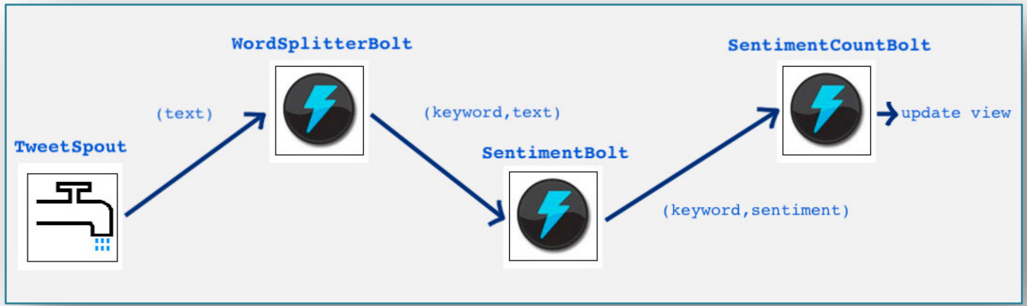
The main stages of the Speed Layer are:

- **TweetSpout**: the tweets are continuously injected into the system, reading them from a CSV dataset. This spout is responsible for simulating a continuous stream of tweets, reproducing a behavior similar to the Twitter platform. From the spout is emitted a stream of tweet texts.

- **WordSplitterBolt**: has the task of splitting the tweet text, dividing each word from the others. It also checks if the word under exam belongs to a list of keywords. If this is the case it outputs the pair *(keyword, text)* that will be received from the next bolt. If more than one keyword is contained in that text, more tuple will be emitted.

- **SentimentBolt**: is in charge of taking the text of the tweet and use the classifier in order to compute the sentiment.

- **SentimentCountBolt:** receives the pair *(keyword, sentiment)* and simply updates a database table, henceforth called *Realtime View*, incrementing the count of positive or negative sentiments for that particular keyword.



Overview of the proposed approach

Phase 3: The Batch Layer

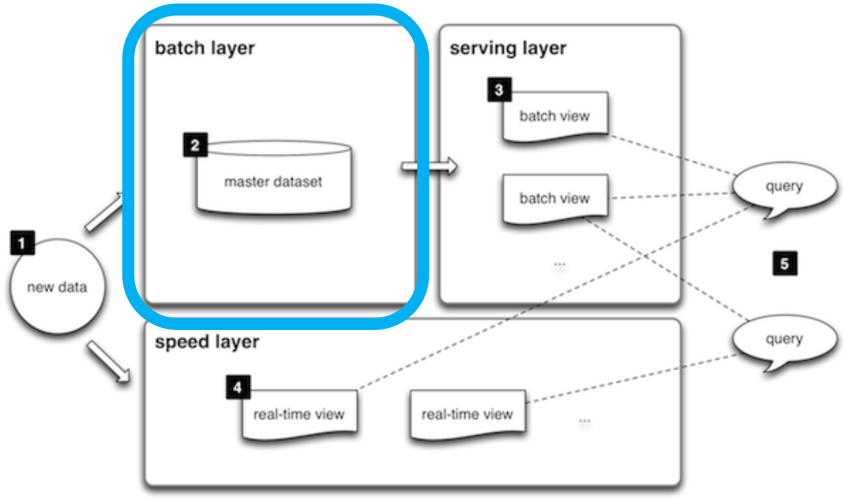
The Batch Layer is responsible for receiving tweets from the TweetSpout. When received, new tweets are stored at once in the **master database**, realized with Apache HBase.

To maintain the rawness of data, new tweets are stored with:

- A **key** assigned incrementally as new tweets come,
- The original **date** of the tweet,
- The tweet **text**,
- A **timestamp**

The other task of the Batch Layer is to compute **batch views**, that will be stored in a batch view table. In order to achieve this task is used **Apache Hadoop**, performing a MapReduce job that is the analogous of a Storm topology.

The main difference resides in the fact that a MapReduce job eventually finishes, whereas a topology runs (theoretically) forever.



Lambda Architecture schema

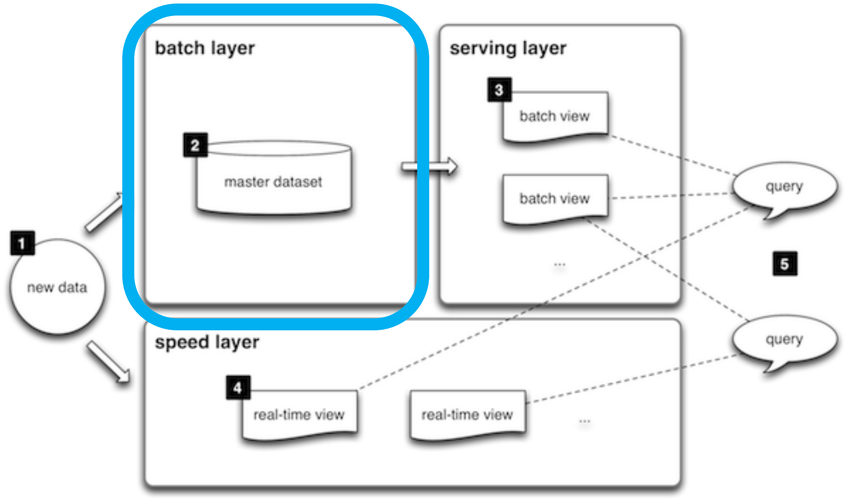
Overview of the proposed approach

Phase 3: The Batch Layer - MapReduce

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data in parallel on large clusters in a reliable, fault-tolerant manner.
A MapReduce *job* splits the input data-set into independent chunks which are processed by the *map tasks* in a parallel manner. The framework sorts the outputs of the maps, which are then input to the *reduce tasks*.
Data is shuffled so that data based on the same key is located on the same worker node, making performances better.

In our case:

- The **Mapping** job receives as input the records from the master database, calculates the sentiment upon the tweet texts that contains a keyword. Then it emits the keyword and sentiment computed.
- The **Reducer** does the sum part, taking as key, the keyword, and updating the batch view table after computing the number of positive and negative sentiment for that keyword.



Lambda Architecture schema

Overview of the proposed approach

Phase 4: The Serving Layer

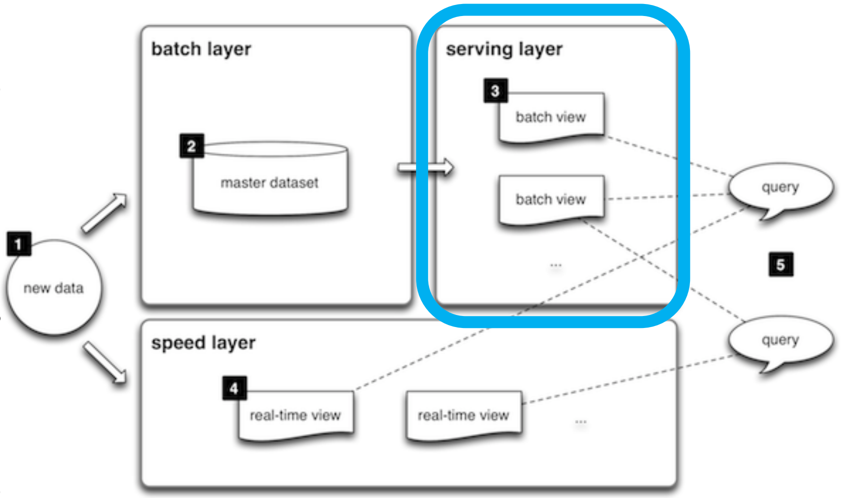
The **Serving Layer** plays the role of a supervisor of both Batch and Speed layers. It is responsible for querying the HBase database to retrieve their instances.

Then the Serving Layer is in charge of merging the two views, obtaining a full view of the sentiment of each keyword for the whole database.
The crucial aspect of the serving, is how the **synchronization** between the two layers is performed. The sum of the two tables needs to yield to a correct representation of the master database.

To do so we always have at hand two batch view tables:

- *BatchViewInComputation* upon which is performed the MapReduce job,
- *BatchViewComplete* which is the consistent and queriable view

At the start of every new computation the realtime view is initialized, so that only the new tweets that will arrive from that moment forward will be counted in the speed layer.
Also the new MapReduce computation is performed in *BatchViewInComputation*, and after it has finished the results are safely copied back to *BatchViewComplete*.



Lambda Architecture schema

Conclusion

In conclusion, at any given time the two tables Realtime view and BatchViewComplete, can be **queried** by the serving layer in order to get a complete view of the system.

The resulting view is an **actual representation** of the master database. Some small errors of synchronization with this method can occur (that is 1-2 tweets missing) due to the, even small but present, time gap between the initialization of the speed layer and subsequent computation of the batch view.

However because of the nature of the batch view, the next computation will re-compute the whole database, adjusting any possible error.