Technische Universität Berlin

Report

Advanced Information Management III: Scalable Data Analytics and Data Mining

A Comparison of Online learning Naïve Bayes Classifier on RSS Feeds using SPARK

Authors:

Ahmet Anil Pala

Franziska Adler

February 17, 2015

Table of Contents

1 Theory

1.1 Motivation

In Text Classification Naïve Bayes Classifiers are popular due to their simplicity and find application in the field of spam email detection and discovery of specified web content (**?**, p. 225). Classifier aim the prediction of a category like spam or ham on the basis of previous examples. This can be easily realised with a probabilistic classifier like Naïve Bayes assuming that the data distribution is static over time, all data is always accessible and query time does not play a major role. For adapting real world problems and their dynamics this might not be sufficient. The development of the internet and the need of processing huge amounts of available data and answering queries in time-critical situations needs to be taken in account for handling model construction and model updating. The reason why model updates are necessary is the prevalence of shifts in the statistical distribution of the target classes in online-learning scenario. This is called *concept drift*. Also aspects of scalability might be considered since Concept Drift occurs mainly in situations of large data quantities arriving via stream (**?**, p. 4)
In this project we use RSS feeds from BCC for classification on a distributed system. Focus is the evaluation of several Naïve Bayes classifiers which implement the online learning paradigm by using different model update techniques. The evaluation concentrates on the handling of Concept Drift. First part of this report explores the theoretical foundations of the relevant concepts as well as the explanations of the challenges we face and the methodology we follow for implementation and the evaluation. The second part digs into more details of our implementation such as how the different classifier approaches are realised. Furthermore the evaluation of their performances are analysed and presented.

1.2 Naïve Bayes classifier

As mentioned Naïve Bayes classifiers are probabilistic classifiers which are simple but effective in text classification. They operate on the basis of the Bayes Theorem and assume independence of features. Here the feature values are normalised words frequencies occurring in a document. The probability of a word w belonging to class $y \in Y$ is given by $P(y|w_i)$ and can be reformulated with Bayes Theorem to its conditional probability and a priori probability $P(y)$ of class y. Both can be derived from frequencies of documents and words.

$$P(y|w_i) \propto P(y)P(w_i|y)$$

The computation of the joint probability over a documents features will give the probability for a document d belonging to a certain class y. Instead of multiplication the logarithm is used to avoid underflow by multiplying with zero:

$$P(y|d) \propto P(y) \prod_i P(w_i|y), \text{ bzw.: } P(y|d) \propto logP(y)log \sum_i P(w_i|y),$$

Applying for all classes will give the decision rule which categorise a new text document according to its most likely class which is the one with the highest joint probability value.

$$\arg\max_y \{logP(y)log \sum_i P(w_i|y)\}$$

1.3 Challanges/ Concepts

texttexttext

1.3.1 Concept Drift

Most online learning scenarios including ours, news feed mining, is susceptible to the phenomenon called *concept drift*. Concept drift occurs when the

statistical properties of the target variable to be predicted change over time. Concept drift can manifest itself in different ways as explained in (**?**, p. 5) . Firstly, shifts in the prior class probabilities or class-conditional probabilities could be the indicator of a concept drift. Likewise, if the relevance between a certain set of attribute values, clues, and certain class predictions change, this could signal a concept drift. Moreover, abrupt increase in the model complexity can be a sign of the concept drift as well for the applicable predictive models. Finally and most importantly, prediction accuracy serves as the most widely-used criterion for detecting the presence of a concept drift. The sources of the concept drifts are diverse depending on the application. For example, in spam filtering applications, user changing his/her mind about what is spam and what is not is an example of a concept drift. In our classification scenario, news feed items categorisation, concept drifts can directly stem from the shift in the media attention which is naturally drawn to the emerging real-life events and its implications on the set of vocabulary news writers use to explain emerging phenomena. For example, a new music album titled "Anarchy in the UK" is introduced and sold millions of copies in short time hence became the topic of many news articles. Since the terms in the title suggests that the chances are that this item could be linked to the category 'politics', most probably the naive bayesian classifier will capture the statistical correspondence of these words to its corresponding what-it-would-be-categorized-as-in-past category and lead to false predictions.
Different ways of handling concept drifts are proposed in literature (**?**, p. 5). These different approaches can be classified according to 4 different criteria. First criterium is whether instance-based or batch-based processing of the stream items are employed. Since, instance-based processing is mostly too costly in terms of CPU load, batch-based processing where the continuous stream is discretised as batches are mostly used. Secondly, concept drift handling differs by the mechanism of the drift detection. There are two common kinds of concept drifts detection mechanism namely explicit detection

and implicit-detection. Explicit-detection mechanism takes a detection action upon the detection such as retraining the model based on the recent items. On the other hand, implicit-detection only continuously adjusts the weights of different classification parameters as a function of concept drift indicators such as error rate, accuracy, etc. These parameters could be the weights of the members of the ensemble classifiers letting the models trained with the obsolete data fade. Moreover, another criterium for the above-mentioned classification is whether classifier-specific or classifier-free action mechanism is employed. In the former, the detection and action mechanism depends on the nature of the classifier and cannot be applied to the all kinds of classifiers. In the latter, the detection is bound to the accuracy and the action is updating the training data so it works with any classifier. Finally, concept drifts can be handled by a single classifier or an ensemble of them. When an ensemble is employed, the prediction decision is jointly taken by a dynamic combiner logic that forgets some of the classifiers which performs bad. Furthermore, although the drift detection action is usually implicit in the ensembles, every now and then ensemble can drop a member by running 'replace the loser' policy.

For this project, we use three different stream learning approaches. They all do batch-based processing using a single classifier for explicit concept drift detection. In terms of classifier-dependency of the concept-drift handling methods, two of them implement classifier-free methods for detection and taking action for the drifting concepts and one of them employs classifier-specific approach. Details regarding these different stream learning approaches we use are discussed in the Methods subsection.

1.3.2 Streaming

In order to deal with the continuous nature of the data, traditional programming primitives are not of much help and mostly distracting. In order to make programmerss job easier, libraries providing higher level abstractions

for stream data are introduced. These libraries usually discretise the continuous input stream into batches and give the programmer a local view of the stream allowing him/her to grab only the data from one batch at a time. Hence, programmer writes code to handle one batch of the discretised stream and the streaming library run this piece of code on all the batches sequentially as they arrive. For instance, programmer wants to use a flatmap function and he writes only one line of code where the flatmap function is called and the streaming library executes this line many times for each arriving batch as if in a loop. This abstraction takes care of all the chore of updating the data structures storing batch elements and bookkeeping loop variables to implement repetitive execution logic, letting the programmer concentrate only on the streaming logic of the application.

1.3.3 Distributed Systems

With the Internet data availability seems not to constitute a problem anymore. Aspects of storing, processing and mining those data amounts were reconsidered in the past years and led to the raise of new technologies and the mostly unloved buzzword Big Data. A single CPU can not accomplish the processing of those data quantities, especially if the algorithms are computationally intensive like most of the prediction algorithms from the field of Machine Learning which are widely used in data mining. To handle that the parallel execution of calculations is spread over a cluster of machines with a underlying system responsible for scheduling, load balance and fault tolerance (**?**, p. 10) . The forerunner of this cluster model was Hadoop, now several frameworks which extend this work are developed and allow wider functionality and significant performance improvements. Even though those frameworks are centered around data processing and ease of use software developers need to be aware of the distributed nature of such systems and parallel computation execution. This might be clear to those coming from a Distributed Systems background but challenging for data engineers or devel-

opers related to data mining who are more used to sequential arrangement of data processing.

1.4 Methods

1.4.1 Learning Methods

texttexttext

1.4.1.1 Offline Learning

1.4.1.2 Online Learning

1.5 Evaluation Measurse

texttexttext

2 Empirics

2.1 Data

Data we are using is basically RSS(Really simple syndication) items streamed from BBC. An RSS Item is combination of some tags describing the content of the feed item and some meta-data about it such as publication date, url, etc. in XML structure. An example RSS item is given below.

Listing 1: An Example RSS Item

```
<item>
    <title>VIDEO: Jaguar announces UK jobs boost</title>
    <description>Jaguar Land Rover says it will create 1,300
        new jobs to build Jaguar's first sports utility
        vehicle (SUV) at its Solihull plant in the West
        Midlands.</description>
    <link>http://www.bbc.co.uk/news/business-30787327#sa-
        ns_mchannel=rss&amp;ns_source=PublicRSS20-sa</link>
```

```
    <guid isPermaLink="false">http://www.bbc.co.uk/news/
        business-30787327</guid>
    <pubDate>Mon, 12 Jan 2015 21:02:57 GMT</pubDate>
</item>
```

.

Here, we are only interested in what is in the publication date, description and title tags. Publication date is crucial for computing interval-based prediction metrics and in-batch ordering. Description and title constitute the document we want to categorise in our case as we want to mine RSS feed items as opposed to mining news articles. Therefore, there is no need for fetching the actual article's text.

One difficulty of working with data streams is the volatility of the data hence it inherently requires real-time data processing. In other words, data should be handled as soon as it arrives. Since it is only possible to process data after implementing the text classifier and get it running, there was no straightforward way of utilising the data streamed during the development of the project. As Data scarcity evidently would pose a serious problem for a text classifier especially considering that we aim to detect concept drifts, we were compelled to find a way to facilitate the data being streamed while developing the application. To this end, we decided to first archive the stream data, then broadcast/stream this data through one of the local ports on the development environment and finally let the streaming application consume this archived data broadcasted data once the development is complete. This way, we could take the time needed for the development and meanwhile did not lose any data since the very first day we conceived the idea of mining news feeds items for this project.

2.2 Software

For realising the stated approaches we are using Apache Spark and MLlib. Apache Spark is a open source processing engine for parallel processing of

large scale data. Spark works on top of a distributed storage and uses a cluster manager cluster manager like Yarn or Mesos. For the purpose of this project the locale storage was used as simulated distributed storage which is integrated in spark for developing and testing reasons.
While the Spark core handles scheduling and load balancing the on top working modules provide additional functionality for streaming, Machine Learning algorithms and graph computation. The main programming abstraction in Spark is called RDD (resilient distributed dataset), a collection of objects partitioned across different machines for parallel programming. Beside map and reduce parallel operations on RDDs like filter, collect and foreeach etc. are provided. For shared variables broadcast variables and accumulators can be used.
TODO: Something about Streaming
TODO: MLlib if used

2.3 Data

texttexttext

2.4 Implementation

For streaming: Since our feed classifier is essentially a streaming application, we decided to use a streaming library. In particular, we employed Apache Spark Streaming assuming that it is nicely integrable into Apache Spark Core which provides distributed systems architecture and facilities and Apache Spark providing the machine learning libraries. U However, it introduces some difficulties as well such as lack of state variables that can be easily updated although there are ways to accomplish it such as state update methods provided in the streaming API, However, this particular method

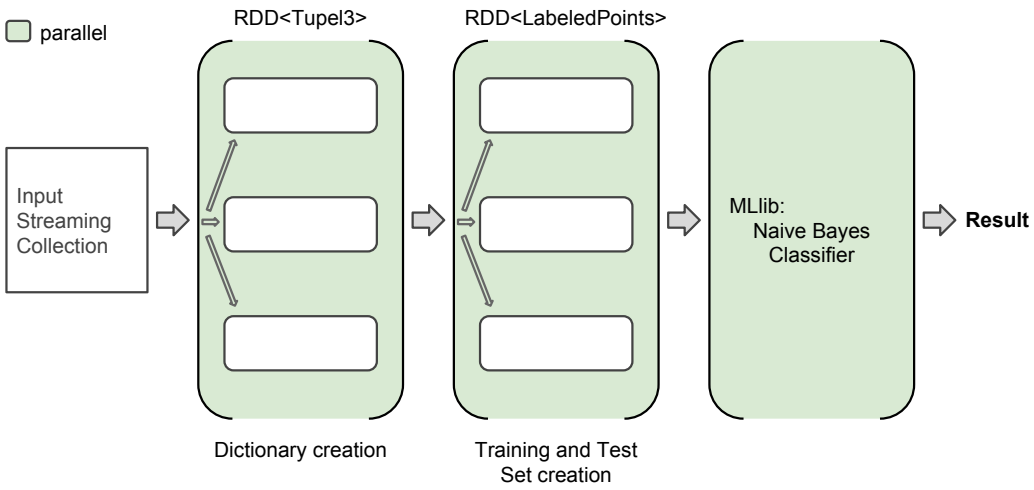

Figure 1: Offline Workflow

2.5 Evaluation

texttexttext

2.6 Results

texttexttext

2.7 Summary

texttexttext