Reservoir Sampling

Anže Pečar, Miha Zidar

***Abstract*—In this article we are going to look at some of the techniques used for learning from continuous data. We will make an overview of the algorithms used in the past and present their shortcomings as well as their advantages.**

***Index Terms*—online learning, continuous data, data mining, AUC maximization, frequent itemsets**

I. Introduction

We live in a very exciting time where processing large amounts of data is of utmost importance. Even with an almost unlimited storage capacities we are still in dire need of efficient algorithms that find only the relevant bits of information from a never ending data stream.

The subject of this paper will not only be to study different kinds of reservoir algorithms but to tackle the broader subject of learning from continuous data in general. Continuous data can be interpreted as any kind of data of which we do not know its size, which is to say that we do not know when the data stream will end. In the past applications of algorithms dealing with continuous data were for an example reading from a tape drive. Memory limitations made it difficult to copy all the data on a tape into working memory, which made it very difficult to do useful operations upon that data. Today working memory is no longer as limited as it was in the past, but that does not mean that there is no more need for such algorithms. It is in fact, quite the contrary. With the boom of the Internet there is more data out there in wild, than it has ever been before. Large websites are facing bigger and bigger problems with dealing with all of the information that they get from their visitors on a daily or even hourly basis. They have to use online machine learning techniques as processing their whole database would take a vast amount of time and resources. These algorithms will be the main subject of this article.

We shall begin with studying reservoir sampling techniques, which enable us to create a random sample of a continuous stream of data. We will take a closer look at one such algorithm named Algorithm R, as well as one of it's many improvements Algorithm Z. Later we will take a look at how Algorithm R can be used in combination with an online learning algorithm to efficiently process incoming data. As well as touching on reservoir sampling algorithms we shall touch upon online learning algorithms. We will describe the idea and implementation of the basic online learning algorithm which uses gradient descent in its implementation. We will also describe a much more complicated algorithm named Online AUC Maximization, which tries to maximize the area under the ROC curve and use this measure to predict more accurately. This very same algorithm utilizes a reservoir sampling algorithm in its implementation.

We will finish off this article with a concrete data mining problem. We will describe the issue of frequent itemsets, which is a common problem for many large supermarkets and online stores. We will describe sequential algorithms for solving this problem as well as their parallel improvements. We shall learn that the reservoir sampling algorithms can be used to make parallel algorithms more efficient. January 19, 2012

II. Algorithm R and its improvements

A. Motivation

The problem with random sampling is to select a random sample of size n from a set of size N. Many algorithms have been developed for this problem when the value of N is known beforehand. Some problems, that we encounter in the real world, do not have a specified N or N cannot be determined efficiently. Those kind of problems will be the focus of this paper. In this section we shall take a look at Algorithm R, which was developed to efficiently and accurately create a random sample from a tape in one pass. But we will not stop at the basic implementation of Algorithm R as we will try to explain the idea behind its improvement Algorithm Z.

Many reservoir algorithms make the assumption that the data does not change over time. Such is the case with Algorithm R, which samples starting data with a greater frequency than the data at the end of the tape. This is efficient, but it fails to react to emerging trends in the data. Later in the article we will learn how to use Algorithm R within an online learning algorithm, which can use the benefits of Algorithm R and still react to recent changes in trends.

B. Algorithm R

Algorithm R is a reservoir algorithm written by Alan Waterman [1]. The basic idea behind reservoir algorithms is to select a sample of size $\geq n$, from which a random sample of size n can be generated. The purpose of Algorithm R is generating a random sample small enough to fit into working memory, from a continuous data stream such as a tape. The algorithm works as follows: we fill up our reservoir of n records with the first n records of the file we are processing. The $t + 1$st record then has a $n/(t + 1)$ chance of being in our reservoir of size n. The candidate it replaces is chosen randomly from the n candidates.

Below is the implementation of Algorithm R in *pseudocode*:

```
N = [0] * n # initial reservoir
for i in range(n):
  N[i] = READ_NEXT_RECORD()
t = n
while EXISTS_NEXT_RECORD():
  t += 1
  M = randrange(0, t, 1)
  if M < n:
    N[M] = READ_NEXT_RECORD()
  else:
    READ_NEXT_RECORD()
```

Explanation: The list N is our reservoir in which we store a random sample of records. We start of with an empty list of length n in which we store the first n elements (the for loop). The function $READ\_NEXT\_RECORD()$ returns the next record in the stream. After we have inserted the first n elements into our reservoir, we enter the $while$ clause, which runs as long as there are still records in the stream. In each run of the $while$ clause we first increment the counter of records t and then calculate a random number between 0 and t. We store the random number in M. If M is lower than n we store the record at index M, otherwise we skip the record. The Algorithm R is easy to understand and we can intuitively see how the algorithm creates a random sample from our continuous data stream.

There is, however, much room for improvements. A basic improvement mentioned in [1] is Algorithm X. Algorithm X is similar to Algorithm R, but it skips a random number of records instead calculating the probability for each record. Algorithm X can further be improved by using an even more aggressive technique of skipping list items. The improvement is named Algorithm Z and will be described below.

C. Algorithm Z

It turns out we do not need to go over every single record in order to get a random set in the reservoir. We can skip a random number of records each time just as long as the probability of a record being in the reservoir does not change. This is the idea behind Algorithm Z.

Below is the first part of Algorithm Z implemented in *pseudocode*:

```
N = [0] * n # initial reservoir
for i in range(n):
  N[i] = READ_NEXT_RECORD()
t = n
thresh = T * n
num = 0
while EXISTS_NEXT_RECORD() and t <= thresh:
  t += 1
  num += 1
  V = random()
  S = 0
  quot = num/t
  while quot > V:
    S += 1
    t += 1
    num += 1
    quot = (quot * num)/t

    SKIP_RECORDS(S)

    if EXISTS_NEXT_RECORD():
      M = randrange(0, n)
      N[M] = READ_NEXT_RECORD()
```

Similarly to Algorithm R, Algorithm Z starts of by putting the first n elements into the reservoir. It then proceeds into the while loop, which runs until the variable t is below the threshold variable $thresh$. The second while loop is used to calculate the number of skipped elements S. The value of S will be larger after each iteration of the first while loop, making the algorithm skip more and more elements each time.

The above algorithm works best when its $thresh$ value T is between 10 and 40. After the t value reaches the $thres$ value the second part of the algorithm begins. The implementation of the second part of the algorithm, known as the $rejection\ technique$, is complicated, and we shall only describe the basic idea behind it. The reader can read the full algorithm in [1].

Rejection technique is a pseudo-random number sampling technique used to generate observations from a distribution. It generates sampling values from an arbitrary probability distribution function $f(x)$ by using an instrumental distribution $g(x)$, under the only restriction that

$$f(x) < Mg(X),$$

where $M > 1$ is an appropriate bound on

$$\frac{f(x)}{g(x)}.$$

The bottom line is that the rejection technique generates values S that increment faster than in the first part of $Algorithm\ Z$, which enables the algorithm to skip even more records without the risk of affecting the distribution of elements in the reservoir.

D. Conclusion

As we can now see, the main difference between $Algorithm\ R$ and $Algorithm\ Z$ is that the former is slower as it computes the probability for every single record, while the latter is faster as it skips as many records as possible. Both algorithms give us a random sample, however, none of them is ideal for detecting emerging trends in the data on its own. As we have seen in both algorithms, the values from the beginning of the data stream get inserted directly into the reservoir, while the records inserted after the reservoir is full need to pass a probability check to get inserted. In the next chapter we shall focus on algorithms that are used for detecting trends in continuous data and we will see that the reservoir algorithm can be used to improve their performance.

III. Online learning

A. Introduction

As we have mentioned in the chapters above, sometimes we do not need to process the whole data set but only the most recent and/or relevant bits. This is what online learning algorithms have been made for. A definition of online learning algorithms as it is written in [2]:

In this learning setting the learner receives a sequence of examples, making predictions after each one, and receiving feedback after each one.

Because of this, online machine learning algorithms allow us to process continuous streams of data. Today many websites use different kinds of online learning algorithms in order to learn from their users preferences. These websites use the continuous flood of data, provided by the users, in order to better present their content and improve their click through rate or other aspects of their websites.

B. A use case of an online learning algorithm

Lets suppose we run a website that provides a paying service for its users. Sometimes users choose to use the service, but some times they do not. We could use an online learning algorithm in order to fine tune our service (asking price, response time, ...), maximizing the probability of users wanting to use our service. We take a look at one example (user) at a time and try to learn from it. It is also crucial that our algorithm adopts to changing user preferences. An example of this would be changes in the economy which would make our users less likely to pay a larger price for our service. Our online learning algorithm needs to detect these kind of trends in order for us to respond to them.

C. A basic online learning algorithm

The implementation of a basic online learning algorithm would be as follows:

```
while (True):
  (x,y) = GET_USER_DATA()
  for(i = 0; i < n; i++):
    Theta[i] = Theta[i] - Alpha*(h(x) - y)*x(j)
```

We will explain exactly what Theta (Θ) is in a latter chapter. For now the important bit of the algorithm is that it is being run continuously and that it adopts to the changes being made in real time. The x variable contains the information that we receive about the visitor, while the y variable contains the information whether or not the visitor chose to use our paying service.

D. Logistic regression

Logistic regression is one of the most popular and most widely used learning algorithms today. Despite its name it is a classification algorithm. It can be used with great success in online learning algorithms so it is worth taking a closer look. Usually we want our hypothesis representation to be between

$$0 \leq h_\Theta(x) \leq 1.$$

Linear regression, as an example, does bind our hypothesis by such boundaries. To achieve this we shall use the $Logistic\ function$, which gives our algorithm its name. Our hypothesis is defined as

$$h_\Theta(x) = g(\Theta^T x)$$

Where g is Sigmoid/Logistic function

$$g(z) = \frac{1}{1+e^{-z}}$$

Our hypothesis is therefore

$$h_\Theta(x) = \frac{1}{1+e^{-\Theta^T x}}$$

Fitting parameters: We have a training set of m examples. How do we choose the parameters Θ? The basic arithmetic distance cost function defined as

$$Cost(h_\Theta(x), y) = \frac{1}{2}(h_\Theta(X) - y)^2$$

Can not be used as the function would not be convex, which would make it difficult for us to create a minimizing algorithm that would work well. For this reason we shall use the following cost function

$$Cost(h_\Theta(x), y) = \begin{cases} -\log(h_\Theta(x)) \text{ if } y = 1, \\ -\log(1 - h_\Theta(x)) \text{ if } y = 0. \end{cases}$$

We can write the logistic cost function as follows

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m} -y^{(i)} \log(h_\Theta(x^{(i)})) + (1 - y^{(i)}) - \log(h_\Theta(x^{(i)}))$$

To fit the parameters we need to minimize the cost function:

$$\min_\Theta J(\Theta)$$

And we can minimize it with the $Gradient\ Descent$ algorithm:

```
theta_old = 0
theta_new = 6 # The algorithm starts at x=6
alpha = 0.01 # step size
precision = 0.00001

while abs(theta_new - theta_old) > precision:
  theta_old = theta_new
  theta_new = theta_old - alpha * J(theta_new)
```

This is how we can use a basic online learning algorithm to predict whether a user will want to join our paying service. We can now modify some of hte parameters x and see if the outcome y will change and adjust our website accordingly.

IV. Online AUC Maximization

The problem with the basic online learning algorithm implementation is that it presumes, that the data is evenly distributed among different classes. These online algorithms are measured by their classification accuracy. But this is inappropriate if the data is unevenly distributed. One of the ways we can address this issue, is by developing an online learning algorithm for maximizing Area Under the ROC curve (AUC), which is a technique frequently used for measuring classification performance for imbalanced data sets. In the classical setup of an online algorithm overall loss is the sum of losses over individual training examples. However, in AUX Maximization, we optimize the pairwise loss between two instances from different classes.

A. Receiver Operating Characteristics (ROC)

A receiver operating characteristic (ROC) or simply a ROC curve is a graphical plot of the sensitivity. This is the true positive rate, vs. false positive rate, for a binary classifier system as its discrimination threshold is varied. The ROC can also be represented equivalently by plotting the fraction of true positives out of the positives vs. the fraction of false positives out of the negatives.

B. AUC Algorithms

Directly applying classical online learning algorithms to maximize AUC requires memorizing all the received training examples, which is of course, inconvenient. One way to overcome this problem is using a reservoir sampling technique which we described in the beginning of this article.

Online AUC Maximization algorithm focuses on learning a linear classification problem with imbalanced data distributions for the two classes. In the following example we will assume the positive class to be the rare class. $x(t)$, $y(t)$ are the training examples received at the t-th trial, where $x(t) \in \mathbb{R}^d$ and $y \in \{+1, +0\}$. $w(t)$ is the weight vector learned so far. We divide our data set D into two sets, the set of positive instances D_+ and the set of negative instances D_-.

We can use the gradient descent based online learning algorithm and update the weight vector w as follows:

$$w(t+1) = w(t) - C\Delta L_t(w)$$

where C is the step size parameter. The main problem is that in order to compute the complete gradient $\Delta L_t(w)$ we need to store all the receiving training examples. We address this problem by caching a small number of received training examples. We introduce two buffers Bp for positive instances and Bn for negative instances of sizes Np and Nn respectively.

A framework for Online AUC Maximization approach

```
w(1) = 0
Bp(1) = 0, Bn(1) = 0
Np(1) = 0, Nn(1) = 0
for t = range(1,T):
  RECEIVE_TRAINING_INSTANCE(x(t), y(t)
  if y(t) = +1
    Np(t+1) = Np(t)+1
    Nn(t+1) = Nn(t)
    Bn(t+1) = Bn(t)
    C(t) = C*max(1,Nn(t)/Nn)
    Bp(t+1) =
      UpdateBuffer(Bp(t), x(t), Np, Np(t+1)
    w(t+1)  =
      UpdateClassifier(w(t),x(t),y(t),
                        C(t),Bn(t+1))
  else
    Nn(t+1) = Nn(t)+1
    Np(t+1) = Np(t)
    Bp(t+1) = Bp(t)
    C(t) = C*max(1,Np(t)/Np)
    Bn(t+1) =
      UpdateBuffer(Bn(t), x(t), Nn, Nn(t+1)
    w(t+1)  =
      UpdateClassifier(w(t),x(t),y(t),
                        C(t),Bp(t+1))
```

UpdateBuffer: $UpdateBuffer$ function needs to keep an accurate "sketch" of history under the constraint of fixed buffer size. To achieve this we deploy our reservoir sampling algorithm Algorithm R. Specifically, given a received training example $(x(t), y(t))$, we add it to the buffer $By(t)$ if $|By(t)| < Ny$. When the buffer is full, we add new items to the buffer as we have described in our Algorithm R section. If we needed the function to be as fast as possible, we would of course implement Algorithm Z. As we have seen in the previous section, Algorithm Z differs from Algorithm R as it skips items instead of calculating the probability for each one. Even though more items get skipped in Algorithm Z we end up having a random sample of the stream in the buffer.

UpdateClassifier: This routine takes five input arguments: the current classifier $w[t]$, training example $(x[t], y[t])$, buffer B and a weight $C[t]$ which play a similar role as step size in our basic online learning algorithm implementation. $Gradient\ updating$ is one technique that could be used to implement the UpdateClassifier function. This approach is to treat $Lt(w)$ as a single loss function and apply the $gradient\ descent$ algorithm that we have also described in the previous chapters.
Gradient updating

```
def UpdateClassifier(w[t], x[t], y[t], C[t]
  for x in B:
    if y[t]*w[t]*(x[t]-x) <= 1:
      w[t+1] = w[t+1]+C[t]*y[t]*(x[t]-x)/2
  return w[t+1]
```

C. AUC Summary

Online AUC Maximization aims to online learn a model by maximizing the AUC metric. It is more challenging than conventional online learning tasks where the goal is often to minimize the mistake rate of online predictions. The key challenge, as we have seen, is that it requires to memorize all the received training instances in the online learning process. It addresses this challenge by applying the reservoir sampling technique, which is able to maintain a good representative sample of the whole dataset.

V. Frequent itemsets

In the last part of our article we will take a look at a concrete data mining problem that we can efficiently solve with the help of reservoir sampling algorithms.

Thanks to automated data collection, companies are able to collect huge amounts of data. It is impossible for them to manually analyse this vast amount of data. Therefore, automatic methods need to be developed. One such challenge is market basket analysis.

The term market basket analysis comes from the need to analyse baskets of items from supermarket customers in order to improve shelf organization and many other aspects of the supermarket. Frequent itemsets are all sets of items that occur in transactions at least $min\_support$ amount of time. The $min\_support$ is a parameter of computation. An example of a frequent itemset could be the set U in which

$$U = \{bread, milk, honey\}$$

If we say that $Supp(U) = 0.3$ it means that the itemset U occurs in 30% of transactions.

A. Sequential algorithms

There are many different algorithms for mining frequent itemsets, namely the Apriori algorithm, the Eclat algorithm, and the FP-Growth algorithm. We shall briefly describe the FPGrowth algorithm.

FPGrowth algorithm: is a depth first search algorithm which creates a frequent pattern tree that represent the whole database. This algorithm needs only two scans of the database, first to compute frequent items and second to create a frequent pattern tree. A frequent pattern tree is basically a tree with a tuple $(item, support, up\_link, link, children)$ in each node.

The $support$ field is the support of the prefix of the item field. The up_link field is the link to the node at the previous level. The $link$ field forms a tree of nodes with a particular item. We have described the basic structure of the FPGrowth algorithm but we will not go into implementation details. Readers who are interested in it should read [4].

B. Parallel algorithms

There are many parallel algorithms based on the Apriori, the FPGrowth and the Eclat algorithm, but all of the implementations have some problems. The FPGrowth algorithm, that we briefly described above, suffers from overly simplified estimate of the prefix-based classes which do not capture the real amount assigned to the processors.

C. Parallel-FIMI

Parallel-FIMI (Parallel Frequent Itemset MIning) is a new method of mining frequent itemsets proposed by R. Kessl [4]. It has three variants:

- Parallel-FIMI-SEQ
- Parallel-FIMI-PAR
- Parallel-FIMI-RESERVOIR

Parallel-FIMI method has the following advantages over other parallel algorithms that we have briefly mentioned:

1) The method is universal. We can easily parallelise any depth first search algorithm. It is even possible to parallelise breadth first search algorithms.
2) The computation is balanced statically. The static load-balancing is based on a heuristic and a sampling algorithm for estimating the size of the PBECs (prefix-based equivalence classes). The PBECs are then assigned to the processors, so that all the processors perform approximately the same amount of work.

Results are distributed among the processors. This can be an advantage if we need to query particular frequent itemsets. Each processor gets the query and finds the frequent itemsets in its own subset and sends them back to the querying processor.

The basic idea behind Parallel-FIMI is to partition all frequent itemsets into disjoint sets, using PBECs of relative sizes. Each processor then processes one partition.

Sampling based on the reservoir algorithm: We will now describe sampling based on the reservoir algorithm. The reservoir sampling algorithm is used in phase 1 of the Parallel-FIMI-RESERVOIR algorithm. The goal of phase 1 is to create an identically distributed sample of F which is essentially a stream of frequent itemsets. We do not know the size of F in advance. We take the samples F_s using the reservoir sampling algorithm that we described in the beginning of this article. This solves our problem of making a uniform sample $F_s \subseteq F$. The sampling is done using an array of frequent itemestes (a buffer or a reservoir) that holds F_s.

Instead of $READ\_NEXT\_RECORD$ and $SKIP\_RECORDS$ functions that we described in the first chapter, when we described Algorithm R, we now have functions $READ\_NEXT\_FI$ and $SKIP\_FIS$ for reading and skipping frequent itemsets. This is the only difference between the algorithm used by [4] and the algorithm that we have described above.

We shall not go into details about phases 2, 3 and 4 as there is an overwhelming amount of mathematics needed to explain them in full detail. Our goal was only to show how a reservoir sampling algorithm can be used in a parallel data mining algorithm.

VI. Conclusion

As we have seen there are many different ways of tackling continuous data. From simple online learning algorithms, that take only a few lines of code to implement, to mind bobbling algorithms that are fast and as accurate as possible. We have shown how a simple reservoir sampling algorithm can be useful for a variety of tasks. We have seen it perform well for sampling data form a continuous tape. We have also shown an example of an advanced online learning algorithm that uses it for calculating the area under the ROC curve. Lastly, we have given an concrete example of its usage in data mining task - mining frequent itemsets with a parallel algorithm.

There are countless more applications of reservoir sampling and we do not doubt that we will see it used in many more algorithms yet to be invented. There is, however, still plenty of room for improvements. There are still ways to improve the reservoir sampling algorithm itself as well as improving the performance of algorithms that use it in their implementation. The subject is as relevant as ever and we do not doubt that many more algorithms will be invented and a lot more papers shall be written on the subject.

Acknowledgement

The authors would like to thank Matjaž Kukar, PhD Assistant Professor.

References

[1] J. S. Vitter, *Random Sampling with a Reservoir*. Brown University, 1985.

[2] N. Littlestone, *Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm*. University of California, 1988

[3] P. Zhao, R. Jing, *Online AUC Maximization* School of Computer Engineering, Nanyang Technological Unoversoty & Deparment of Computer Science and Engineering, Michigan State University

[4] R. Kessl, *Parallel algorithms for mining of frequent itemsets* The Faculty of Electrical Engineering, Czech Technical University in Prague

[5] Hanley, James A. and McNeil, Barbara J. *The meaning and use of the area under of receiver operating characteristic (roc) curve.* 1982.

Anže Pečar 63060257

Miha Zidar 63060317