Change Detection in Learning Histograms from Data Streams

Raquel Sebastião[1] and João Gama[1,2]

[1] LIAAD-INESC Porto L.A., University of Porto
[2] Faculty of Economics, University of Porto
Rua de Ceuta, 118 - 6
4050-190 Porto, Portugal
{raquel,jgama}@liacc.up.pt

Abstract. In this paper we study the problem of constructing histograms from high-speed time-changing data streams. Learning in this context requires the ability to process examples once at the rate they arrive, maintaining a histogram consistent with the most recent data, and forgetting out-date data whenever a change in the distribution is detected. To construct histogram from high-speed data streams we use the two layer structure used in the Partition Incremental Discretization (PiD) algorithm. Our contribution is a new method to detect whenever a change in the distribution generating examples occurs. The base idea consists of monitoring distributions from two different time windows: the reference time window, that reflects the distribution observed in the past; and the current time window reflecting the distribution observed in the most recent data. We compare both distributions and signal a change whenever they are greater than a threshold value, using three different methods: the Entropy Absolute Difference, the Kullback-Leibler divergence and the Cosine Distance. The experimental results suggest that Kullback-Leibler divergence exhibit high probability in change detection, faster detection rates, with few false positives alarms.

1 Introduction

Histograms are one of the most used tools for exploratory data analysis. Data analysis is complex, interactive, and exploratory over very large volumes of historic data. Traditional pattern discovery process requires online ad-hoc queries, not previously defined, that are successively refined. Due to the exploratory nature of these queries, an exact answer may not be required. A user may prefer a fast approximate answer. Histograms are one of the techniques used in data stream management systems to solve range queries and selectivity estimation (the proportion of tuples that satisfy a query), two illustrative examples where fast but approximate answers are more useful than slow and exact ones.

Another aspect is that data arrives continuously in data streams and it's necessary to evaluate it, detecting if there is a change in the distribution. Therefore it's not reasonable to allow processing algorithms enough memory capacity to store the full history of the stream. So based on a previous work [1], we perform incremental discretization and use it to detect changes. Change detection in data streams was the main motivation for this work. The paper is organized as follows. The next section presents an algorithm to continuously maintain histograms over a data stream. In Section 3 we extend the algorithm for predictive data mining. Section 4 presents preliminary evaluation of the algorithm in benchmark datasets and one real-world problem. Last section concludes the paper and presents some future research lines.

2 Histograms

Histograms are one of the most used tools in exploratory data analysis. They present a graphical representation of data, providing useful information about the distribution of a random variable. A histogram is visualized as a bar graph that shows frequency data. The basic algorithm to construct a histogram consists of sorting the values of the random variable and places them into *bins*. Next we count the number of data samples in each bin. The height of the bar drawn on the top of each bin is proportional to the number of observed values in that bin.

A histogram is defined by a set of non-overlapping intervals. Each interval is defined by its boundaries and a frequency count. In the context of open-ended data streams, we never observe all values of the random variable. For that reason, and allowing to consider extreme values and outliers, we define an histogram as a set of break points $b_1, ..., b_{k-1}$ and a set of frequency counts $f_1, ..., f_{k-1}, f_k$ that define k intervals in the range of the random variable:
$]-\infty, b_1],]b_1, b_2], ...,]b_{k-2}, b_{k-1}],]b_{k-1}, \infty[$.

The most used histograms are either *equal width*, where the range of observed values is divided into k intervals of equal length $(\forall i, j : (b_i - b_{i-1}) = (b_j - b_{j-1}))$, or *equal frequency*, where the range of observed values is divided into k bins such that the counts in all bins are equal $(\forall i, j : (f_i = f_j))$.

When all the data is available, there are exact algorithms to construct histograms [2]. All these algorithms require a user defined parameter k, the number of bins. Suppose we know the range of the random variable (domain information) and the desired number of intervals k. The algorithm to construct *equal width* histograms traverses the data once; whereas in the case of *equal frequency* histograms a sort operation is required.

One of the main problems of using histograms is the definition of the number of intervals. A rule that has been used is the Sturges' rule: $k = 1 + log_2 n$, where k is the number of intervals and n is the number of observed data points. This rule has been criticized because it is implicitly using a binomial distribution to approximate an underlying normal distribution[1]. Sturges rule has probably survived because, for moderate values of n (less than 200) produces reasonable

[1] Alternative rules for constructing histograms include Scott's (1979) rule for the class width: $k = 3.5sn^{-1/3}$ and Freedman and Diaconis's (1981) rule for the class width: $k = 2(IQ)n^{-1/3}$ where s is the sample standard deviation and IQ is the sample interquartile range.

histograms. However, it does not work for large n. In exploratory data analysis, histograms are used iteratively. The user tries several histograms using different values of k (the number of intervals), and choose the one that better fits his purposes.

2.1 The Partition Incremental Discretization

The *Partition Incremental Discretization* algorithm (*PiD* for short) is composed by two layers. The first layer simplifies and summarizes the data; the second layer constructs the final histogram.

The first layer is initialized without seeing any data. As described in [1], the input for the initialization phase is the number of intervals (that should be much larger than the desired final number of intervals) and the range of the variable.

Instead of initialize the algorithm with the number of bins, we manage a way such that the numbers of bins depends on the upper bound on relative error and on the desirable confidence level:

$$nBins1 = 50INT(ln(\tfrac{1}{\delta}) * ln(\tfrac{1}{\epsilon^2})).$$

Figure 1 shows that the number of bins increase when the relative error (ϵ) decreases and the confidence ($1-\delta$) increases.

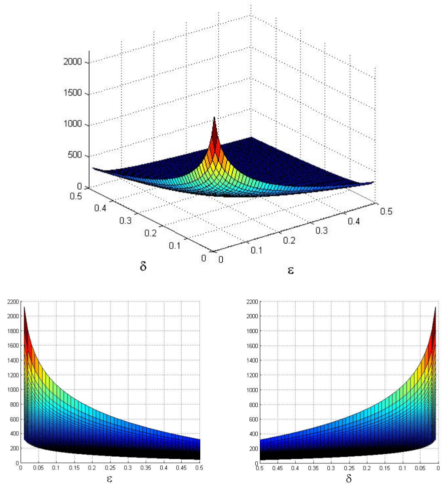

Fig. 1. Representation of the number of bins of $layer_1$. The top figure shows the dependency from ϵ and δ and bottom figures shows it according to only one variable.

Figure 1 (top) represents the number of bins of $layer_1$ in function of ϵ and δ. The bottom figures give a projection of the number of bins according with the variables ϵ and δ (respectively).

So, the input for the initialization phase is a pair of parameters (that will be used to express accuracy guarantees) and the range of the variable:

- The upper bound on relative error ϵ.
- The desirable confidence level δ.
- The range of the variable.

The range of the variable is only indicative. It is used to initialize the set of breaks using an equal-width strategy. Each time we observe a value of the random variable, we update $layer_1$. The update process determines the interval corresponding to the observed value, and increments the counter of this interval. Whenever the counter of an interval is above a user defined threshold (a percentage of the total number of points seen so far), a split operator triggers. The split operator generates new intervals in $layer_1$.

If the interval that triggers the split operator is the first or the last, a new interval with the same step is inserted. In all the other cases, the interval is split into two, generating a new interval. Figure 2 describes the $layer_1$ update process. The process of updating $layer_1$ works online, performing a single scan over the data stream. It can process infinite sequences of data, processing each example in constant time and space.

The second layer merges the set of intervals defined by the first layer. It triggers whenever it is necessary (e.g. by user action). The input for the second

```
Update-Layer1(x, breaks, counts, NrB, alfa, Nr)
x - Observed value of the random variable
breaks - Vector of actual set of break points
counts - Vector of actual set of frequency counts
NrB    - Actual number of breaks
alfa   - Threshold for Split an interval
Nr     - Number of observed values

 If (x < breaks[1]) k = 1; Min.x = x
 Else If (x > breaks[NrB] k = NrB; Max.x = x
 Else k = 2 + integer((x - breaks[1]) / step)

 while(x < breaks[k-1]) k <- k - 1
 while(x > breaks[k]) k <- k + 1

 counts[k] = 1 + counts[k]
 Nr = 1 + Nr
 If ((1+counts[k])/(Nr+2) > alfa)  {
      val = counts[k] / 2
      counts[k] = val
      if (k == 1) {
        breaks = append(breaks[1]-step, breaks)
        counts <- append(val,counts)
      }
      else {
        if(k == NrB) {
          breaks <- append(breaks, breaks[NrB]+step)
          counts <- append(counts,val)
        }
        else {
          breaks <- Insert((breaks[k]+ breaks[k+1])/2, breaks, k)
          counts <- Insert(val, counts, k)
        }
      }
   NrB = NrB + 1
 }
```

Fig. 2. The PiD algorithm for updating $layer_1$

layer is the breaks and counters of $layer_1$, the type of histogram (equal-width or equal-frequency) and the number of intervals. The algorithm for the $layer_2$ is very simple. For equal-width histograms, it first computes the breaks of the final histogram, from the actual range of the variable (estimated in $layer_1$). The algorithm traverses the vector of breaks once, adding the counters corresponding to two consecutive breaks. For equal-frequency histograms, we first compute the exact number F of points that should be in each final interval (from the total number of points and the number of desired intervals). The algorithm traverses the vector of counters of $layer_1$ adding the counts of consecutive intervals till F.

The two-layer architecture divides the histogram problem into two phases. In the first phase, the algorithm traverses the data stream and incrementally maintains an equal-width discretization. The second phase constructs the final histogram using only the discretization of the first phase. The computational costs of this phase can be ignored: it traverses once the discretization obtained in the first phase. We can construct several histograms using different number of intervals and different strategies: equal-width or equal-frequency. This is the main advantage of PiD in exploratory data analysis.

2.2 Analysis of the Algorithm

The histograms generated by PiD are not exact. There are two sources of error:

1. The set of boundaries. The breaks of the histogram generated in the second layer are restricted to the set of breaks defined in the first layer.
2. The frequency counters. The counters of the second layer are aggregations of the frequency counters in the first layer. If the splitting operator does not trigger, counters in first layer are exact, and also counters in second layer. The splitting operator can produce inexact counters. If the merge operation of the second layer aggregate those intervals, final counters are correct.

A comparative analysis of the histograms produced by PiD and histograms produced by exact algorithms using all the data reveals some properties of the PiD algorithm. Assuming a equal-width discretization (that is the split operator did not trigger) for the first layer and any method for the second layer, the error of PiD boundaries (that is the sum of absolute differences between boundaries between PiD and batch discretization) is bound, in the worst case, by: $R * N_2/(2 * N_1)$, where N_1 denotes the number of intervals of $layer_1$, N_2 the number of intervals of $layer_2$, and R is the range of the random variable. This indicates that when N_1 increases, the error decreases. The algorithm guarantees that frequencies at second layer are exact (for the second layer' boundaries). We should note that the splitting operator will always decrease the error.

The time complexity of PiD depends on the discretization methods used in each layer. The time complexity of the second layer is constant because its input is the first layer that has a (almost) fixed number of intervals. The time complexity for the first layer is linear in the number of examples.

3 Change Detection

The algorithm described in the previous section assumes that the observations came from a stationary distribution. When data flows over time, and at least for large periods of time, it is unlikely the assumption that the observations are generated at random according to a stationary probability distribution. At least in complex systems and for large time periods, we should expect changes in the distribution of the data.

3.1 Related Work

A fundamental question when monitoring a stream of values is: *Are the observations we are receiving now from the same distribution we have observed in the past?*.

There are several methods in machine learning to deal with changing concepts [3,4,5,6]. In machine learning drifting concepts are often handled by time windows or weighted examples according to their age or utility. In general, approaches to cope with concept drift can be classified into two categories: *i*) approaches that adapt a learner at regular intervals without considering whether changes have really occurred; *ii*) approaches that first detect concept changes, and next, the learner is adapted to these changes. Examples of the former approaches are *weighted examples* and *time windows* of fixed size. Weighted examples are based on the simple idea that the importance of an example should decrease with time (references about this approach can be found in [4,6,7]). When a time window is used, at each time step the learner is induced only from the examples that are included in the window. Here, the key difficulty is how to select the appropriate window size: a small window can assure a fast adaptability in phases with concept changes but in more stable phases it can affect the learner performance, while a large window would produce good and stable learning results in stable phases but can not react quickly to concept changes. In the latter approaches, with the aim of detecting concept changes, some indicators (e.g. performance measures, properties of the data, etc.) are monitored over time (see [5] for a good classification of these indicators). If during the monitoring process a concept drift is detected, some actions to adapt the learner to these changes can be taken. When a time window of adaptive size is used these actions usually lead to adjusting the window size according to the extent of concept drift [5]. As a general rule, if a concept drift is detected the window size decreases, otherwise the window size increases.

3.2 Monitoring Distributions on Two Different Time Windows

Most of the methods in this approach monitor the evolution of a distance function between two distributions: from past data in a *reference window* and in a current window of the most recent data points. An example of this approach, in the context of learning from Data Streams, has been present by [8]. The author proposes algorithms (statistical tests based on Chernoff bound) that examine samples drawn from two probability distributions and decide whether these distributions are different.

3.3 Entropy Based Change Detection

As a measurement of information (from Information Theory [9]), we conveniently adapted the entropy as a measurement of change detection. Entropy between the probabilities absolute difference measures the dispersion of the Distributions Differences and is defined by the following equation:

$$H(p||q) = -\sum_i |q(i) - p(i)| * log_2(|q(i) - p(i)|)$$

were q_i and p_i represents the probability of a point belongs to the bin i of the current window and the probability of belongs to the correspondent reference bin. From the properties of the above equation, we can derive the bounds: $0 \leq H(q||p) \leq 2$. Smaller values of $H(p||q)$, corresponds to smaller dispersion between the distributions of the two variables.

3.4 Kullback-Leibler Based Change Detection

From information theory [9], the Relative Entropy is one of the most general ways of representing the distance between two distributions [10]. Also known as the Kullback-Leibler's distance or divergence, it measures the distance between two probability distributions and so it can be used to test for change. Given a reference window with empirical probabilities p_i, and a sliding window with probabilities q_i, the KL distance is:

$$KL(p||q) = \sum_i p(i) log_2 p(i)/q(i).$$

The KL divergence is non negative and asymmetric and as higher is his value, the more distinct the distribution of the two variables. A higher value of the distance represents distributions that are further apart.

3.5 Cosine Distance Based Change Detection

Instead of comparing the distance between two probability distributions we compare the angle between them. The cosine distance [11] is derived from the dot product of two vectors and is given by the following equation:

$$C(p||q) = 1 - \frac{\sum_i p(i)q(i)}{||p|| ||q||}$$

We considered this definition in order to have non-negative values and guarantee that $0 \leq C(q||p) \leq 2$. This measure is symmetric and a lower value means that two distributions are closer.

4 Experimental Evaluation

The advantage of the two-layer architecture of PiD is that after generating the $layer_1$, the computational costs, in terms of memory and time, to generate the final histogram (the $layer_2$) is low: only depends on the number of intervals of the $layer_1$. From $layer_1$, we can generate histograms with different number of intervals and using different strategies (equal-width or equal-frequency). We should note that the standard algorithm to generate equal-frequency histograms requires a sort operation, which could be costly for large n. This is not the case of PiD. Generation of equal-frequency histograms from the $layer_1$ is straightforward.

4.1 Methodology and Design of Experiments

In order to analyze the capacity to detect changes using different kinds of distributions we design three different datasets:

– 100000 points of a random variable from a Normal distribution. This dataset is composed by two samples from a normal distribution (with different parameters) of 50000 values each one. The initial 50000 points are generated from the standard normal distribution and then we force an abrupt change using parameters distinct enough to guarantee that in spite of the similar shape of distributions, the algorithm should detect changes.
– 100000 points of a random variable from a Normal distribution. This dataset is composed by two samples from a normal distribution (with different parameters) of 50000 values each one. The initial 50000 points are generated from the standard normal distribution and then we force a smooth change, using similar parameters expecting that the algorithm should not detect changes.
– 100000 points of a random variable from LogNormal and Normal distributions of 50000 points each one.

The data is received at any time producing an equal-with histogram. The number of bins were defined according to the equation on section 2.1, setting both the variables ϵ and δ as 5%. We considered that the initial data points should be used as a representation of data and that the number of initial points should be chosen according to the number of intervals of the $layer_1$. So we decided that the first $30 * nBins1$ data points are part of a stabilization process and that no change occurs in this range. The windows size was also defined as dependent on the number of intervals of the $layer_1$, being half of these ones: $\frac{nBins1}{2}$.

Assuming that sample in the stabilization set has distribution P and that the current windows has distribution Q, we use as a measure to detect whether has occurred a change in the distribution the measures described above. For all the 3 measures the more distinct the distributions P and Q the higher is the distance between them. According to this, we define that had occurred a change in the distribution of the current window relatively to the reference distribution if the distance computed based on those distributions is greater that $\mu + z_{1-\frac{\alpha}{2}}\sigma$, where μ and σ represents the sample mean and the estimate standard deviation, and $z_{1-\frac{\alpha}{2}}$ represents the point on the standard normal density curve such that the probability of observing a value lower than $z_{1-\frac{\alpha}{2}}$ is equal to $1-\frac{\alpha}{2}$. For instance, we established a confidence level of 95% ($\alpha = 0.05$), so $z_{1-\frac{\alpha}{2}} = 1.96$. If no change occurs we maintain the reference distribution and consider more $\frac{nBins1}{2}$ data points in the current window, and start a new comparison. If we detect a change we clean the reference data set and initialize the process of search for changes.

4.2 Controlled Experiments with Artificial Data

In this Section we evaluate the 3 change detection methods in controlled experiments using artificial data. The goals of these experiments are:

1. Capacity to Detect and React to drift.
2. Resilience to False Alarms when there is no drift, that is not detect drift when there is no change in the target concept.
3. The number of examples required to detect a change after the occurrence of a change.

Table 1. Results of the 3 change detection methods using artificial data

Datasets	**TP**			**FP**			**Ne (mean)**			**Ne (std)**		
	Cos	Ent	KLD	Cos	Ent	KLD	Cos	Ent	KLD	Cos	Ent	KLD
D1	1	1	1	0.1	0	0	1964	707	258	189.3	0	0
D2	1	0.9	1	0	0	0	7038	12400	5691	1427	5228	933.5
D3	0.95	0.95	0.95	0.65	0	0	9285	801	258	8906	188	0

Table 1 shows the results of the 3 change detection methods using the described artificial data. It can be observed that the use of the Kullback-Leibler Distance as a measure to decide if there is a drift achieve better results and reaches those requiring a smaller number of examples. It also shows that all the measures have a good resilience to false alarms when there are no drifts. Notice that even in the second dataset, where the change was very smooth, the cosine and the Kullback-Leibler distances detected it in all the experiments. The entropy of the absolute differences also detected it almost the times. Although the cosine distance detects changes when there not existing, the performance of other measures were very consistent and precise. The results obtained with those data sets clearly show that the cosine distance presents the worse results.

To evaluate the performance of the 3 algorithms we also use quality metrics such as *Precision* and *Recall*. The *Precision* gives a ratio between the correct detected changes and all the detected changes and *Recall* is defined as a ratio between the correct detected changes and all the occurred changes:

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

Table 2 shows the precision and recall achieved by the 3 change detection methods using artificial data. For both quality metrics, the closest to one, the better are the results. Those results sustain the above observations, showing that Kullback-Leibler Distance is the algorithm that reaches better results and that the algorithm based on the cosine distance is the worst one.

Table 2. The *Precision* and the *Recall* obtained using the 3 algorithms to detect changes

Datasets	**Precision**			**Recall**		
	Cos	Ent	KLD	Cos	Ent	KLD
D1	0.9091	1	1	1	1	1
D2	1	1	1	1	0.9000	1
D3	0.5938	1	1	1	1	1

4.3 Data and Experimental Setup

To obtain data, tests were carried out in a Kondia HS1000 machining centre equipped with a Siemens 840D open-architecture CNC. The blank material used for the tests was a 170 mm profile of Aluminum with different hardness. The maximum radial depth of cut was 4.5 mm using Sandvik end-mill tools with two flutes and diameter 8, 12 and 20 mm. Tests were done with different cutting parameters, using sensors for registry vibration and cutting forces.

A multi-component dynamometer with an upper plate was used to measure the in-process cutting forces and piezoelectric accelerometers in the X and Y axis for vibrations measure. A Karl Zeiss model Surfcom 130 digital surface roughness instrument was used to measure surface roughness.

Each record includes information on the following seven main variables used in a cutting process:

– ***Fz*** - feed per tooth
– ***Diam*** - tool diameter
– ***ae*** - radial depth of cut
– ***HB*** - Hardness on the type of material
– ***Geom*** - tools geometry
– ***rpm*** - spindle speed
– ***Ra*** - average surface roughness

This factors was proceeding the Design of Experiment explained in [12] was used for validation a bayesian model for prediction of surface roughness. A multi-component dynamometer with an upper plate was used to measure the in-process cutting forces (X and Y axis). The cutting speed on X and Y axis was also measured.

We use these sensors measures to detect when a change had occurred in the experiments. We must point out that a change in the activity does not mean a change in the data coming from sensors.

Our goal is to study the three detection methods in a real data problem. For each sensor we record the points where each method signals a change. We study concordances in change points for the different methods. As the sensors response to different incentives, the agreement in change points for the sensors was not carried out.

For the second sensor, neither the entropy of the absolute differences nor cosine distances detected any change.

The obtained results shows that the changes detected by Kullback-Leibler distance for the data points from the second sensor agree with the results obtained with the first sensor, supporting that this measure is consistent.

Comparing the results obtained with the three detected methods on the data points from the first sensor, we found out 21 conformities in changes points between the Kullback-Leibler and the cosine distances. The entropy of the absolute differences reached 23 and 21 accordance's with the Kullback-Leibler and the cosine distances, respectively.

5 Conclusion and Future Work

The main conclusion we found is that the Kullback-Leibler Distance reach better performances than the other algorithms for all the kinds of artificial datasets we had study and that the cosine distance were the worst algorithm. From the artificial data results we can also conclude that both entropy of the absolute differences algorithm and Kullback-Leibler Distance has a good resilience to false alarms when there are no drifts on the datasets.

From the experiments with real data, we must take into consideration that we don't have information if there are changes and when they occur; and so on we can't extract strong or well based conclusions from the results obtained with that data. Although this lack, we can conclude that the concordances in changes points between the detection methods supports their capacity to detect changes. As a final conclusion, one can say that the results achieved so far are quite encouraging and motivate the continuation of the work. As an improvement of this initial work we intend to apply the described algorithms into dataset collect in a medical and in an industrial context. We also intend to improve the algorithms in order to reduce the number of points that are needed to detect a change.

Acknowledgments

Thanks to the financial support given by the FEDER, the Plurianual support attributed to LIAAD, project ALES II (POSI/EIA/55340/2004), and project RETINAE.

References

1. Gama, J., Pinto, C.: Discretization from Data Streams: applications to Histograms and Data Mining. In: ACM Symposium on Applied Computing, pp. 662–667. ACM Press, New York (2006)
2. Pestana, D.D., Velosa, S.F.: Introdução á Probabilidade e á Estatística. Fundação Calouste Gulbenkian (2002)
3. Klinkenberg, R.: Learning drifting concepts: Example selection vs. example weighting. Intelligent Data Analysis 8(3), 281–300 (2004)

4. Klinkenberg, R., Joachims, T.: Detecting concept drift with support vector machines. In: Langley, P. (ed.) Proceedings of ICML 2000. 17th International Conference on Machine Learning, Stanford, US, pp. 487–494. Morgan Kaufmann Publishers, San Francisco (2000)
5. Klinkenberg, R., Renz, I.: Adaptive information filtering: Learning in the presence of concept drifts. In: Learning for Text Categorization, pp. 33–40. AAAI Press, Stanford, California, USA (1998)
6. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning 23, 69–101 (1996)
7. Maloof, M., Michalski, R.: Selecting examples for partial memory learning. Machine Learning 41, 27–52 (2000)
8. Kifer, D., Ben-David, S., Gehrke, J.: Detecting change in data streams. In: VLDB 2004: Proceedings of the 30th International Conference on Very Large Data Bases, pp. 180–191. Morgan Kaufmann Publishers Inc., San Francisco (2004)
9. Berthold, M., Hand, D.: Intelligent Data Analysis - An Introduction. Springer, Heidelberg (1999)
10. Dasu, T., Krishnan, S., Venkatasubramanian, S., Yi, K.: An Information-Theoretic Approach to Detecting Changes in Multi-Dimensional Data Streams. In: Interface 2006 (Pasadena, CA) Report (2006)
11. Tan, P.-N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison-Wesley, Reading (2006)
12. Correa, M., de Ramírez, M.J., Bielza, C., Pamies, J., Alique, J.R.: Prediction of surface quality using probabilistic models. In: 7th Congress of the Colombian Association of Automatic, Cali, Colombia, March 21–24, 2007 (2007) (in Spanish) Domingos, P., Hulten, G.: Learning from infinite data in finite time. In: Advances in Neural Information Processing Systems 14. MIT Press, Cambridge, MA (2002)