Contents

Document history

Version	Date	Author (Unit)	Description
v0.3			The test and evaluation framework discussed and established
v0.6			Initial draft finished and reviewed by the PSRG
v1.0			Final version of document

1 Introduction

Even though the number of algorithms designed for learning on streaming data is increasing, there is still not a unified and well accepted way for evaluating them. This is because testing and evaluating algorithms that are designed to work on streaming data are generally more complicated than those designed to work on static data. There are both statistical and computational reasons for this.

Static data is data, where each instance can be assumed to be identically and independently distributed i.i.d. If the data instances in the data stream is i.i.d., then the only new challenge compared to static data is that the data streams are open ended. It can be seen as a static data that is always growing in size. One way to deal with this is to take out a data set of fixed size, a holdout data set, and perform tests on this. Another way is to let the performance measures be "accumulating" as the data set is growing.

Various error measures related to stream data has been proposed in the papers of Gama et. al. [2], [3], [4]. A loss function is typically related to the penalty of misclassifications on classification problems or residuals in regression models. The holdout error is basically the average loss on a holdout dataset of fixed size. The predictive sequential, or *prequential* error is defined as the average loss function up to time step i, where i is the current time step.

There exist also data streams where the data points are locally i.i.d., but on a larger time scale they seem to drift. To deal with, prequental error measures with forgetting mechnisms are suggested in [4]. The forgetting mechanisms involves either using a time window or fading factors. In paper [4], convergence towards the Bayes error was shown for all these performance measures provided that the learners are consistent and data is i.i.d. Moreover, it was shown that if data was allowed to drift over time, meaning that samples are only locally i.i.d, then the prequental error measures with forgetting mechanisms were favourable.

There also exist data streams, where this local i.i.d. is far from being true. In fact it may even be so that the covariance structure between neighboring measurements is really the information carrier in the streams. In this paper we will refer to such streams as locally dependent. The main contribution of the paper is to describe new performance measures that are taylored to work on such streams.

[**TODO: Relate locally dependent streams to literature.**]

In a worst case scenario, the locally dependent streams have dependences on larger time scales as well. In this case, not much can be done a single data stream because a repetitive pattern is needed in any data driven approach. However, there are opportunities if we can assume that there exist multiple streams that can be seen as i.i.d. When we refer to a stream here, we mean an ordered list of elements at each time step. For instance, the data for a client in a bank can be described by a data stream, where at each time step, the balance on his savings account is the first element and the loan in the second place in the list. An example of multiple streams would be all the clients in the bank.

Another interesting scenario is when the locally dependent streams are stationary on a longer time scale. In this case small time windows, or substreams could be taken out for testing. Provided that these substreams are far enough apart, these substreams are i.i.d.

It is also improtant to note that the algorithms themselves are often designed to weight measurements that are close to the actual time step higher than measurements that are further back. On locally dependent streams, we must therefore not only assume that data are generated from underlying distributions that are time dependent, but also that the algorithms themselves are time dependent.

Computational challenges are related to the fact that the data come from an open-ended data stream, conceptually infinitely long, which imposes practical challenges related to restrictions on cpu-time and memory allocation.

[**TODO: Elaborate more on computational challenges.**]

In this paper we will establish formal procedures for testing and evaluating the developed models and algorithms. The applications that are covered in this paper is related to locally dependent streams. Most test and evaluation procedures in this paper, involve chopping data streams into small substreams (of fixed size) that are i.i.d and labeled as either a positive or a negative.

The test and evaluation procedure includes specification of what metrics are relevant to use to quantify the ability of the AMIDST system, such as loss functions, maximum response-times, memory limits and output format. The paper also includes considerations about quantitative improvements AMIDST should obtain over the state of the art.

In section 2, AMIDST relevant methodologies for evaluation of both batch and streaming algorithms are identified and discussed. This section forms the foundation of the next three sections, where the exact evaluation routines for each use case provider is given. These sections contains a description of the requirements related to evaluation as described in Delivery 1.2 and a short description of the algorithms and the data. At the end of these sections, the methods for evaluating predictive and runtime performances ae exposed and discussed. Section 6 concludes the report.

2 Test and evaluation methodology

As discussed in the introduction, finding appropriate performance measures on streaming data is more difficult than finding performance measures on static data. We will therefore start by discussing some relevant performance measures for classification on static data. A brief section follow where evaluation methods on locally dependent streams is exposed.

		Predicted class		
		Cat	Dog	Rabbit
Actual class	Cat	5	3	0
	Dog	2	3	1
	Rabbit	0	2	11

Table 2.1: Example of a confusion matrix. A classifier is labelling instances as either cats, dogs or rabbits. The accuracy is the sum of the diagonal elements divided by the total number (in this case 19/27).

2.1 Performance measures for classification on static data

On static data, we assume that the dataset has a fixed size n, where each instance is independently drawn from a joint probability distribution $P(X, Y)$, where X and Y are random variables. The X variable is known as the explanatory variable and Y is the class label. These two variables have output spaces Ω_X and Ω_Y, respectively. For instance in a binary classification problem Ω_Y can either be true or false, while Ω_X is a space of all possible explanatory vectors.

In classification, we typically consider a hypothesis function $h : \Omega_X \rightarrow \Omega_Y$. In terms of evaluating the performance of h, we use a dataset of n input-output pairs (x_i, y_i) that are independently drawn from $P(X, Y)$. The result of such an experiment can be shown in a confusion matrix. An example is shown table 2.1, where the classifier is attempting to distinguish cats, dogs and rabbits.

A global measure of the classification algorithm is the classification accuracy which is basically the diagonal elements divided by the total number (in this case 19/27). It is important to note that the accuracy is not telling the whole story of the classification rule. For instance, by looking at the table it is seen that the algorithm distinguishes distinguish cats from rabbits quite easily, but it is much harder to distinguish cats from dogs. This information can not be found from only looking at accuracy. Moreover, accuracy is also very dependent on how evenly the classes are distributed. For instance, when there are a lot more instances of one class compared to the others, a naive classification rule that always predict the majority class will get a high accuracy, even though the method is not using any of the information that is contained in the explanatory variables. These are some reasons for showing the full confusion matrix to interpret the status of the classification rule. However, there are more numbers that can be derived from the confusion matrix. To simplify this exposition we have chosen to limit the discussion to binary classification.

When the classification is a binary, such as classifying cats and non-cats, the confusion matrix becomes two dimensional as shown in 2.2. In binary classification, it is common to introduce positives and negatives, instead of the class labels. A true positive is therefore an actual cat that has been predicted to be a cat by the classifier of interest. False

		Predicted class	
		Cat	Not cat
Actual class	Cat	5	3
	Not cat	2	17

Table 2.2: Example of a confusion matrix for a classifier of cats and not cats. The accuracy is 22/27.

		Actual Condition	
		Positive	Negative
Test outcome	Positive	5	2
	Negative	3	17

Table 2.3: Example of a confusion table for a classifier of cats and not cats. The true positives and true negatives are on the diagonal, while the two other numbers are the false positives and the false negatives.

positives, true negative and false negatives are defined in an equivalent manner. The results are commonly shown in a confusion table (see table 2.3), which is not the same as a confusion matrix.

From a confusion table it is easy to calculate numerous numbers that describes the classification rule. Specifically we mention the true positive and false positive rates. True positive rates, also known as recall, is the number of true positives divided by the total number actual positives. The false positive rates, also known as fall-out, is the number of false positives divided by the total number actual positives.

[TODO: Possibly add a bit more about other measures based on the confusion table.]

By investigating various numbers that can be deduced from the confusion table, it is possible to discuss classification rules, even when the datasets are unevenly distributed. That is, when some class label have more instances than others.

However, these numbers do not take into account that some misclassifications might be more costly than others. For instance, a false positive might be more costly than a false negative, such as in the case of cancer diagnostics. It might be more costly to not threat a person that is sick, compared to threating a healthy person. Moreover, the cost of each false positive (or false negative) may not be constant either. For instance, if the classifier is predicting whether a client in a bank will default a loan or not, the cost is clearly related to the size of the loan in question. The next subsection includes a procedure to include such costs in a performance measure.

2.1.1 Empirical risk

In mathematical optimization, statistics, decision theory and machine learning, a loss function or cost function is a function that maps an event or values of one or more variables onto a real number that is intuitively representing some *cost* associated with the event. Loss functions can be used on optimization problems, where an algorithm or method is optimized by minimizing the loss function. Moreover, loss functions are frequently used to diagnose and compare various algorithms or methods.

In this paper, we define the *loss function* as a real and lower-bounded function L on $\Omega_X \times \Omega_Y \times \Omega_Y$. The value of the loss function at an arbitrary point $(x, h(x), y)$ is interpreted as the loss, or cost, of taking the decision $h(x)$ at x, when the right decision is y. Notice that in this paper, the loss function is dependent on x as well. This is of high practical use, because a certain misclassification might be more expensive than another.

In the frequentist perspective, the expected loss is often referred to as the risk function. It is obtained by taking the expected value over the loss function with respect to the probability distribution $P(X, Y) : \Omega_X \times \Omega_Y \to \mathbb{R}^+$. The *risk function* is given by

$$R(h) = \int_{\Omega_X, \Omega_Y} L(x, h(x), y) dP(x, y). \tag{2.1}$$

In the case when the costs are independent of x and also that there is no cost related to correct classification, the risk function reduces to the well known expected cost of misclassification (ECM)

$$ECM = c(1|0)p(1|0)p_0 + c(0|1)p(0|1)p_1. \tag{2.2}$$

Here, $c(1|0)$ is the cost for misclassifying an item of class zero as class one and $p(1|0)$ is the misclassification probability given class zero. The quantities $c(0|1)$ and $p(0|1)$ are defined equivalently, while p_0 and p_1 are the priors.

In general, the risk $R(h)$ cannot be computed because the distribution $P(x, y)$ is unknown. However, we can compute an approximation, called empirical risk, by averaging the loss function on the training set $\mathbf{x} \times \mathbf{y}$ of size n, where each element is (x_i, y_i). The empirical risk is given by

$$R_{emp}(h, \mathbf{x} \times \mathbf{y}) = n^{-1} \sum_{i=1}^{n} L(x_i, h(x_i), y_i). \tag{2.3}$$

Notice that L is an array of $n \times 2 \times 2$ elements. Many supervised learning algorithms are optimized by finding the h in a hypothesis space $\mathcal{H}$ that minimizes the empirical risk. This paper will not focus on empirical risk minimization, but rather focus on using the empirical risk to compare methods.

2.1.2 Evaluation of families of classification rules

So far we have discussed how to evaluate a single classification rule. However, most classification rules in the AMIDST framework is based on comparing an estimated probability to a certain threshold. We call this estimated probability the output function $q : \Omega_X \to \mathbb{R}^+$. In the AMIDST framework the range of q is usually $[0, 1]$, but this restriction is not necessary for this theory. The potential classification rules are the family of hypothesis functions $\mathcal{H}$, where each element $h_T : \Omega_X \to \Omega_Y$ has the form

$$h_T(x) = \begin{cases} 0 & \text{for} \quad q(x) \leq T \\ 1 & \text{else.} \end{cases} \tag{2.4}$$

It is of interest to evaluate all these classification rules. The receiver operating characteristic ROC is a plot of the true positive rate as a function of the false positive rate, as T vary over all relevant variables. ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from (and prior to specifying) the cost context or the class distribution. ROC analysis is related in a direct and natural way to cost/benefit analysis of diagnostic decision making.

[TODO: Add a ROC plot.]

The ROC curve allows a visual exposition of the family of classifiers $\mathcal{H}$. However, it is also of interest to reduce all this information into a single number. The area under the ROC curve, also known as AUROC, is such a number. AUROC is independent of T and also the priors. It may be difficult to comprehend what AUROC is at this point, but we will return to it afterward.

First, let X_0 and X_1 be random variables with probability distributions $P(X|Y = 0)$ and $P(X|Y = 1)$, respectively. We define the random variables $Q_0 = q(X_0)$ and $Q_1 = q(X_1)$. To summarize, Q_0 is basically a random variable that is related to when you pick a random sample of class zero and run the system and then you get an output value (usually a probability of something happening). The same thing can be said for Q_1.

The probability $P(Q_1 > Q_0)$ is of interest, because this says what is the probability that if you take one sample from each of the populations, what is the chance that the output value from sample one is higher than the output value of sample zero. This question is independent of T, meaning that the discussion about priors and costs are not needed. This probability is called the concordance probability.

Without exposing the details, it is possible to show that

$$\text{AUROC} = P(Q_1 > Q_0). \tag{2.5}$$

It is important to note that nothing need to be assumed about the probability distributions of Q_0 and Q_1. Furthermore, it is worth noting that the concordance probability is exactly equal the common language effect size of the Mann-Whitney U test. We have therefore chosen to add a section of the Mann-Whitney U test.

2.1.3 Mann-Whitney U test

In statistics, the Mann-Whitney U test (also called the Mann-Whitney-Wilcoxon (MWW), Wilcoxon rank-sum test, or Wilcoxon-Mann-Whitney test) is a nonparametric test of the null hypothesis that two populations are the same against an alternative hypothesis, especially that a particular population tends to have larger values than the other. It has greater efficiency than the t-test on non-normal distributions and it is nearly as efficient as the t-test on normal distributions.

We define a training set $\mathbf{x} \times \mathbf{y}$ with n input-output pairs (x_i, y_i), independently drawn from $P(X, Y)$. From the training set we have two populations $\mathbf{q_0} = \{q(x_i), | \, y_i = 0\}$ and $\mathbf{q_1} = \{q(x_i), | \, y_i = 1\}$. Their sizes are n_0 and n_1 so that $n_0 + n_1 = n$. Calculating the U statistics is straightforward, where these two values are obtained

$$U_0 = \sum_{i=1}^{n_0} \sum_{j=1}^{n_1} H(\, q_j - q_i \,) \quad \text{and} \quad U_1 = \sum_{i=1}^{n_0} \sum_{j=1}^{n_1} H(\, q_i - q_j \,). \tag{2.6}$$

Here $H(\cdot)$ is the heaviside step function and notice that $U_0 + U_1 = n_0 n_1$. For large samples, each U is approximately normally distributed. In that case, the standardized value

$$z = \frac{U_0 - m_U}{\sigma_U}, \tag{2.7}$$

where m_U and σ_U are the mean and standard deviation of U given by

$$m_U = \frac{n_0 n_1}{2} \quad \text{and} \quad \sigma_U = \sqrt{\frac{n_0 n_1 (n_0 + n_1 + 1)}{12}}. \tag{2.8}$$

Significance of test can be checked in tables of the normal distribution. Although, such an hypothesis test is interesting by itself, we are more interested in the concordance probability $P(Q_1 > Q_0)$ which is defined by

$$P(Q_1 > Q_0) = \frac{U_1}{n_0 n_1}. \tag{2.9}$$

It is important to note that even though the Mann-Whitney U test is dependent on wether the shapes of the probability distributions of Q_0 and Q_1 are similar, this requirement is not necessary for the concordance probability. Also. equation (2.9) shows a simple way of calculating $P(Q_1 > Q_0)$ and AUROC.

2.2 Performance measures for classification on streaming data

In the use case scenarios in the Amidst software there are generally two paths for evaluating the methods. The multi class classifiers are generally evaluated by accuracy, while the binary classifiers involve dividing data into substreams.

In contrast to for instance [4], we can generally not assume that data is locally i.i.d. A data instance at a certain time step has to be seen in context of the preceding time steps. Our approach for testing is to manually select a number of time windows or substreams that are labeled as either negatives or positives. There are two important approaches here.

The first approach deals with multiple streams and the global time dependence is removed by sampling all these streams into multiple substreams, which are starting and ending at the same time steps. Each substream is labeled as either a positive or negative based on the data at the end time. An evaluation time step that is a fixed between the start and the end is also defined. The explanatory variables are derived from the data between the start and the evaluation time step. In this sense, this problem is reduced to a problem of static data, where each i.i.d. instance is labeled.

The second approach is when there is basically one stream where one can assume stationarity on a global scale. Substreams with large enough distance to other substreams are taken out of the stream. These substreams are labeled as either a positive or a negative. For instance, a positive is when something is happening towards the end of the substream, while a negative is a substream where nothing happens. When the AMIDST software run, the output function is evaluated at each time step. Depending of the problem, a heuristic algorithm is chosen to output a single number for each substream based on the output values. This can for instance be the maximum value of the output function on an interval that is prior to an event.

3 Cajamar: test and evaluation

3.1 Use case requirements

Summarize the use case requirements for the different application scenarios. This information should be derived from Deliverable 1.2.

3.2 Model and data characteristics

Describe aspects of the model and data relevant for the ensuing test and evaluation discussion. Much of this information can be synthesized from the existing documents, and should serve to make the document more self-contained.

3.3 Predictive performance: test and evaluation

3.3.1 Application scenario 1

3.3.2 Application scenario 2

3.4 Run-time performance: test and evaluation

3.4.1 Application scenario 1

3.4.2 Application scenario 2

4 Daimler: test and evaluation

4.1 Use case requirements

Summarize the use case requirements for the different application scenarios. This information should be derived from Deliverable 1.2.

4.2 Model and data characteristics

Describe aspects of the model and data relevant for the ensuing test and evaluation discussion. Much of this information can be synthesized from the existing documents, and should serve to make the document more self-contained.

4.3 Predictive performance: test and evaluation

4.3.1 Application scenario 1

4.3.2 Application scenario 2

4.4 Run-time performance: test and evaluation

4.4.1 Application scenario 1

4.4.2 Application scenario 2

5 Verdande: test and evaluation

5.1 Use case requirements

Summarize the use case requirements for the different application scenarios. This information should be derived from Deliverable 1.2.

5.2 Model and data characteristics

Describe aspects of the model and data relevant for the ensuing test and evaluation discussion. Much of this information can be synthesized from the existing documents, and should serve to make the document more self-contained.

5.3 Predictive performance: test and evaluation

5.3.1 Application scenario 1

5.3.2 Application scenario 2

5.4 Run-time performance: test and evaluation

5.4.1 Application scenario 1

5.4.2 Application scenario 2

6 Conclusion

References

[1] Kaptein, M.: Rstorm: Developing and testing streaming algorithms in r. The R Journal **6**(1) (2014) 123–132

[2] Gama, J.a., Rodrigues, P.P., Sebastião, R.: Evaluating algorithms that learn from data streams. In: Proceedings of the 2009 ACM Symposium on Applied Computing. SAC '09, New York, NY, USA, ACM (2009) 1496–1500

[3] Gama, J.a., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. In: 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). (2009) 329–337

[4] Gama, J., Sebastião, R., Rodrigues, P.P.: On evaluating stream learning algorithms. Machine Learning **90**(3) (2012) 317–346