CRITICAL POINTS IN ASSESSING LEARNING PERFORMANCE VIA CROSS-VALIDATION

Hanna Suominen, Tapio Pahikkala, Tapio Salakoski

Turku Centre for Computer Science (TUCS) and
University of Turku, Department of Information Technology
20014 University of Turku, FINLAND
firstname.lastname@utu.fi

ABSTRACT

Quality assessment of learning methods is essential when adapting them to different tasks, introducing new algorithms, developing the existing ones, or tracking the learning improvements over time. Obtaining realistic results is, however, complicated. In this paper, we clarify this by addressing performance evaluation measure and method selection, with a main focus on combining the AUC measure with the cross-validation method. We conclude that it is crucial to choose both the measure and method that reflect the evaluation aspects, learning task and data in question. Furthermore, the evaluation setting must correspond to the intended use environment and the test set has to be completely independent from the creation of the learner. Finally, special caution is needed when forming the cross-validation folds.

1. INTRODUCTION

In *supervised learning*, a machine is taught with a set of training data instances with preferred outputs to perform a specific task. By a task, we mean the prediction of an output for an unseen data instance. For example, the aim in automated *classification* is to build a system, which assigns for each input instance according to their content the class or classes to which it belongs based on absorbing information from previously observed instance-class pairs. An inherent question is how well the learning method performs in its task from the output quality perspective. Answering this is essential, for example, when adapting a learner to a given task, introducing a new algorithm, developing existing systems further or tracking the learning improvements over time.

Performance evaluation can be divided into three step process [Spärck Jones and Galliers, 1996, pp. 19–20], [Hirschman and Thompson, 1997] (Figure 1): The first step is to define the aspect, or *criterion*, of the system performance to be assessed. At the second one, a suitable performance evaluation *measure* is selected to reflect the criterion of interest. The third step is to specify a *method* to determine the value of the measure illustrating the system performance typically in a numerical form.

Let us consider again classification as an example learning problem. Now, classification speed and quality are possible criteria. Further, a measure reflecting the quality of the classification could focus on the number of correctly classified instances and the method to define the measure value to ask a human expert to classify a set of data, divide the set into two parts, train the classifier with the first half, use the resulting system to classify the second half, and compare the output to the expert's opinions.

Even though performance evaluation may sound easy, there are, however, many pitfalls on the way. On the one hand, details of performance evaluation, such as established measures, are to a large extent textbook knowledge. But on the other hand, quality assessment problems are very complicated and inherent to a learning task in question. In practice, a great number of published papers today suffer from serious defects in the evaluation of system performance.

In this paper, we address the measure and method steps of performance assessment by using the output quality as a criterion. As we have now specified the first evaluation step, we will later use the term *performance* to refer to this aspect. Our goal is to provide helpful guidelines to avoid pitfalls in performance evaluation by explaining the essence of selecting a measure appropriate for the task and data as well as a method taking into account both the measure and data. To concretize discussion, we focus on classification even though a majority of general principles covered can be applied to other learning tasks as well.

We begin with the measure step of assessing the quality of learners in Section 2. We illustrate the key questions required to judge by considering classification as an example learning task. We describe several established classification measures and discuss their similarities, differences, strengths and drawbacks. We start with simple, intuitive and easy-to-interpret measures based on comparing the numbers of correctly and incorrectly classified instances. Through these examples, we notice the essence of taking class sizes, class distributions and dependences between different measures into account when selecting the measure. Step by step, we proceed to established measures that are commonly considered to depict more reliably classification performance. An additional aspect of the section is generalizing the measures from the basic case of identifying instances belonging to a given class

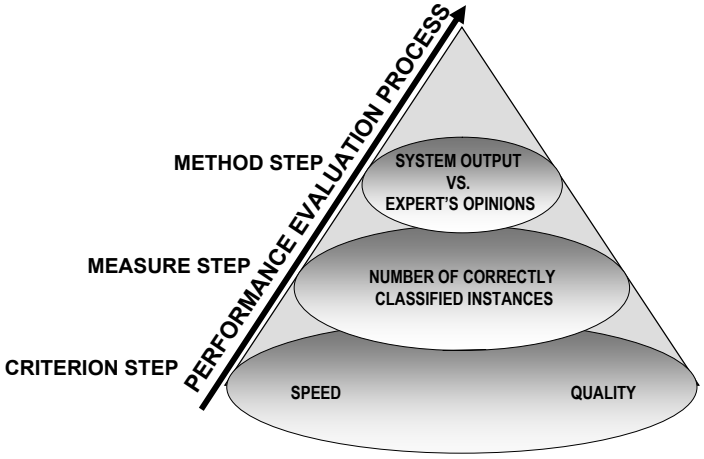

Figure 1. Steps of learning performance evaluation by [Spärck Jones and Galliers, 1996, pp. 19–20] and [Hirschman and Thompson, 1997].

to more complex learning tasks.

We then continue in Sections 3 and 4 with the method step by refining it into sequential method selection and implementation stages. Section 3 is about the method selection step: We explain techniques to compute the performance of a given learner in terms of the chosen measure and discuss their relationships to each other. After recalling the basic principles of supervised learners, we portray the importance of assessing performance with data independent from the learning process. But, on the other hand, evaluation data must be representative and in line with the amount of available data.

Section 4 focuses on the step, when performance evaluation method is implemented, and it contains the main contribution of this paper. We discuss ways to follow in practice the crucial principles of having a realistic test setting and test set completely independent from the creation of the learner. Firstly, we demonstrate dependences in the data, which complicate performance evaluation. Secondly, we explain approaches to reflect the special characteristics of the data, performance measure and learning task at the method step. Thirdly, we concretize this by considering a particular measure and method.

We conclude the study in Section 5 by summarizing the content of the paper in five general principles. We also update Figure 1 to correspond to the refined performance evaluation process and supplement it with the criteria, measures, methods, and implementation aspects discussed.

2. PERFORMANCE EVALUATION MEASURES

We next clarify the perspectives needed to judge, when selecting a performance measure and interpreting its values. As performance measures depend on a learning task and numerous performance measures exists, we consider classification as an illustrative example. (See, e.g., [Spärck Jones and Galliers, 1996, Suominen et al., 2008] for a broader discussion about performance evaluation measures for various learning tasks.) The measures considered evaluate the learning ability numerically by comparing system output to a *gold standard* (aka *reference standard*), which defines the correct output.

To lay groundwork for deeper aspects and discussion, we start with easy-to-interpret measures and their drawbacks in Section 2.1. In Section 2.2, we focus with further detail on a particularly prevalent performance measure called the *area under receiver operating characteristic curve* (*AUC*). In both sections, we begin with *binary classification*, where the task is to decide for each input instance whether or not it belongs to a given class. Then, we explain how the measures can be extended to more general learning tasks. Section 2.3 summarizes the aspects of classification measure selection discussed.

2.1. Accuracy, Precision, Recall and Related Measures

Perhaps the most intuitive classification performance evaluation measures are the proportions of correctly and incorrectly identified instances, that is, the classification *accuracy* and *error*. Their values are between zero and one, but with accuracy, one means the best performance whereas with error, being one minus accuracy, zero corresponds to the optimum. However, these simple measures may give misleading results: When the classes are strongly of an unequal size, accuracy and error may give considerably over-optimistic or pessimistic results.

As an example, in a binary classification task, where 95% of the instances belong to the class (*positive instances*), a classifier that always assigns an instance to the class will have 5% error. This misleadingly suggests that the classifier is good. Similar problems often occur in tasks, where the gold standard is defined by using a Likert scale opinion poll, because it is relatively common that the answers to a given claim pool to one option.

Another problematic situation is that the proportion of instances left correctly outside the class (*true negatives*, TN), is excessively large when compared to the numbers of *true positives* (TP), *false positives* (FP) and *false negatives* (FN). In addition, the number of true negatives may even be unknown, as in applying classification to identify web documents relevant to a query given to a web search engine.

Precision, *recall* and many other classification measures based on these are independent of true negatives and hence also applicable when the number of true negatives is unknown or excessively large. Precision is defined as the proportion of correctly classified positive instances from all positive instances in the system output. Recall is the proportion of correctly identified positive instances from all instances that should have been identified as positive. Both these measures get values between zero and one, higher values indicating the better the performance.

There is a tradeoff between precision and recall, and therefore one must consider the two measures together if choosing to use them in performance evaluation. A pitfall is to, for example, maximize recall without taking precision into account simultaneously, because a perfect recall can be trivially accomplished by classifying all instances as positives. More generally a system can increase recall at the cost of decreased precision by assigning more instances to the class, and vice versa.

It is often desirable to have only one value that characterizes performance, and for precision p_{P} and recall p_{R}, this is popularly done by taking their weighted harmonic mean

$$\begin{aligned} p_{\mathrm{F}}(Y, f(X), \alpha) &= \frac{1}{\alpha \frac{1}{p_{\mathrm{P}}} + (1-\alpha)\frac{1}{p_{\mathrm{R}}}} \\ &= \frac{p_{\mathrm{P}}\, p_{\mathrm{R}}}{\alpha p_{\mathrm{R}} + (1-\alpha) p_{\mathrm{P}}}, \end{aligned} \quad (1)$$

where f specifies the classifier and $f(X) = (f(x_1), \ldots, f(x_m))^{\mathrm{T}}$ is its output for an input sequence $X = (x_1, \ldots, x_m)$. $Y = (y_1, \ldots, y_m)^{\mathrm{T}}$ is the respective gold standard and $\alpha \in [0, 1]$ a factor determining the weighting of precision and recall. The values of (1) are between zero and one, higher values indicating the better the performance.

The most common choice in (1) is to weight precision and recall evenly, that is, to select $\alpha = 0.5$ giving

$$p_{\mathrm{F1}}(Y, f(X)) = \frac{2 p_{\mathrm{P}} p_{\mathrm{R}}}{p_{\mathrm{P}} + p_{\mathrm{R}}} = \frac{2TP}{2TP + FP + FN}.$$

This is known as $F1$, *F measure* or *balanced F score*.

We now extend the previous measures to more general tasks of *multi-class classification*, where there are multiple classes and each instance belongs to one of them, and to *multi-label classification*, where each instance can belong to several classes at the same time. With multi-class classification, this is straightforward: an instance is correctly classified if it belongs to the same class both in the system output and gold standard. As above, accuracy (resp. error) is the proportion of correctly (resp. incorrectly) identified instances to the total number of instances. Precision and recall must be, however, be defined separately for each class. With respect to a certain class i, an instance is false positive if it is assigned to this class in the system output but not in the gold standard. Similarly, it is false negative if it belongs to the class i in the gold standard but not in the system output. Now, precision and recall can be defined separately for each class as above. We denote them as p_{P_i} and p_{R_i}, respectively. Generalization for multi-label classification is similar, as it can be decomposed into distinct binary classification problems, except accuracy and error must also be computed separately for each class.

If we want to measure the overall performance with one measure in the multi-class or label case, *micro* or *macro-averaging* can be used. For example, if N_C is the number of classes and $p_{\mathrm{F}_i}, i \in \{1, \ldots, N_C\}$, are the values of (1) calculated separately for each class by using p_{P_i} and p_{R_i}, the macro-averaged variant of (1) is the average of $p_{\mathrm{F}_1}, \ldots$ and $p_{\mathrm{F}_{N_C}}$. Further, the micro-averaged variant of (1) is defined by replacing p_{P} and p_{R} with their micro-averaged forms

$$\begin{aligned} p_{P_{\text{micro}}}(Y, f(X)) &= \frac{\sum_{i=1}^{N_C} TP_i}{\sum_{i=1}^{N_C} (TP_i + FP_i)}, \\ p_{R_{\text{micro}}}(Y, f(X)) &= \frac{\sum_{i=1}^{N_C} TP_i}{\sum_{i=1}^{N_C} (TP_i + FN_i)}, \end{aligned}$$

where TP_i, FP_i and FN_i are the class-specific TP, FP and FN, respectively.

Macro and micro-averaged variants of (1) reflect, by definition, different performance aspects. The former emphasizes the significance of performing well in all classes, including also those with relatively rare occurrence frequency. The latter weights each code assignment decision equally resulting the dominance of the performance in the common classes.

2.2. AUC

The measures defined in Section 2.1 are sensitive to the relative number of positive and negative instances. This class distribution dependence can be problematic if, for example, the distributions of the gold standard and real data differ. AUC is a measure invariant to class distribution (see, e.g., [Fawcett and Flach, 2005]), and for this reason, its use has been recommended instead, or in addition to, F (see, e.g., [Airola et al., 2008] in the task of extracting protein-protein interactions). On the other hand, unlike precision, recall and F, it incorporates the number of true negatives. Further, unlike previously discussed measures, AUC can be applied in preference learning and ranking tasks as well, because only the pairwise order of the instances with respect to the classification topic, that is, information defining which of the two instances is larger, is needed. For these reasons AUC has gained a substantial popularity in machine learning community.

We define AUC for binary classification in a proba-

bilistic fashion as follows:

$$p_{\mathrm{AUC}}(Y, f(X)) = \frac{\sum_{y_i=+1, y_j=-1} \delta(f(x_i) > f(x_j))}{y_+ y_-}, \quad (2)$$

where y_+ and y_- are the numbers of positive ($y_i = +1$) and negative ($y_i = -1$) examples in the gold standard and

$$\delta(b) = \begin{cases} 1 & \text{if } b = \text{TRUE} \\ 0 & \text{if } b = \text{FALSE} \end{cases}$$

The interpretation of formula (2) is that AUC is equivalent to the *Wilcoxon-Mann-Whitney statistic*, which is the probability that, given a randomly chosen positive example and a randomly chosen negative example, the classifier will correctly distinguish them [Cortes and Mohri, 2004].

As being a probabilistic measure, the values of AUC are between zero and one, larger values indicating better performance. When interpreting the values, one must notice that the AUC value of 0.5 denotes random performance and one perfect performance. This means that in order to outperform the random baseline, the learner must achieve AUC value larger than 0.5.

Alternatively, AUC can be obtained in a traditional way by computing the *receiver operating characteristic* (*ROC*) curve and then calculating the area under the curve. From this definition, we can easily see the relation to precision, recall and F: To define the ROC curve, let us consider the binary classification problem again. Instead of assuming the system output to be a positive or a negative label for each input instance, as we have done previously, many classification algorithms produce real-value output. The magnitude of the output value reflects the classification confidence; the largest positive values are assigned to instances that are most strongly positive, and the largest negative values to the most strongly negative instances. In order to compare a real-value output to a binary gold standard, it must, however, be divided into positive and negative instances. This can be done by simply defining a threshold value θ, which divides the real-value interval into two parts so that instances with $f(x_i) \leq \theta$ are classified as negatives and with $f(x_i) > \theta$ as positives. The ROC curve refers to plotting the recall (aka *true positive rate*) at certain levels of the *false positive rate* $FP/(FP + TN)$ obtained by varying the values of θ from the minimum threshold value (i.e., the one resulting all instances to be positives) to the maximum (i.e., the one resulting all instances to be negatives). In addition to one value characterizing performance (i.e., the area under the curve (AUC) here), the ROC curve provides supplementary information about the relationship between recall and false positive rate: analyzing the curve from left to right corresponds to trading off false positives for false negatives.

A straightforward generalization of the Wilcoxon-Mann-Whitney statistic for ranking tasks is given, for example, by [Fung et al., 2006]. There, the labels of the data points are not necessarily binary, but they can be ordinal classes or even real valued. In this case, the performance measure corresponds to the probability that, given randomly chosen two examples with different labels, the learning method will correctly predict which of them has a larger label value. Because of this, the measure is, in fact, closely related to the Kendall's τ correlation coefficient [Kendall, 1970]. For further discussion about AUC and its extension to tasks with multiple classes that are not necessarily ordinal, we refer to [Hand and Till, 2001, Lachiche and Flach, 2003, Lachiche et al., 2006].

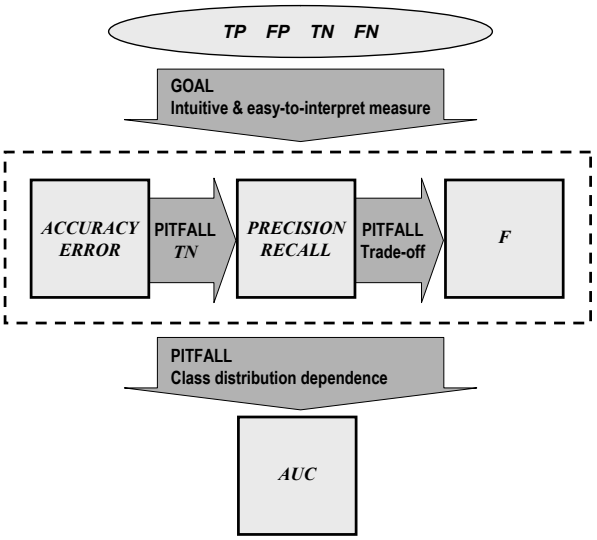

Figure 2. A schematic diagram of the discussed classification performance measures.

2.3. Summary

The section discussed critical points in performance measures with a focus on classification tasks. To summarize, the appropriate choice of a performance measure depends on both the task and data in question. The measures described in Sections 2.1 and 2.2 portray the differing aspects of various well-established measures and the critical decisions related to measure selection (Figure 2). However, many alternative measures exist and new ones are introduced all the time. Hence, it is essential to review the prevailing practices in similar type of tasks.

3. PERFORMANCE EVALUATION METHOD SELECTION

In this section, we first recall in Section 3.1 the concept of supervised learning. Then, we proceed in Sections 3.2–3.5 to describing four techniques that can be chosen at the method step to specify the performance of a given learner (e.g., classifier) in terms of the chosen measure: resubstitution, hold-out, cross-validation, and bootstrap. Section 3.6 summarizes the method selection discussion.

3.1. Supervised Learning

A *supervised learner* is a machine that is taught with a set of training data instances with preferred outputs to per-

form a specific task. By a task, we mean the prediction of an output for an unseen data instance. Formally, let again

$$X = (x_1, \ldots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$$

be a sequence of inputs and

$$Y = (y_1, \ldots, y_m)^{\mathrm{T}} \in \mathbb{R}^m$$

a sequence of outputs, where $\mathcal{X}$, called the input space, is the set of possible inputs. Here, $(\mathcal{X}^m)^{\mathrm{T}}$ denotes the sets of row vectors of size m whose elements belong to the set $\mathcal{X}$ while $\mathbb{R}^m$ denotes the set of real valued column vectors of size m. Further, let

$$S = ((x_1, y_1), \ldots, (x_m, y_m))^{\mathrm{T}} \in (\mathcal{X} \times \mathbb{R})^m$$

be a *training set* of m training examples, where $(\mathcal{X} \times \mathbb{R})^m$ denotes the set of all possible training sets of size m. Notice that while we call S a training set, we consider it as an ordered sequence of data instances.

Training can be considered as a process of selecting a function among a set of candidates that best performs the task in question. Following the notation of [Herbrich, 2002], we formalize the training algorithm as follows: An algorithm $\mathcal{A}$, that selects the function given the training set S, can be considered as a mapping

$$\mathcal{A} : \bigcup_{m \in \mathbb{N}} (\mathcal{X} \times \mathbb{R})^m \to \mathcal{H},\ S \mapsto f,$$

where $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}}$, called the hypothesis space, is a set of functions among which the algorithm selects an appropriate hypothesis $f \in \mathcal{H}$. With $\mathbb{R}^{\mathcal{X}} = \{f \mid \mathcal{X} \to \mathbb{R}\}$ and $\bigcup_{m \in \mathbb{N}} (\mathcal{X} \times \mathbb{R})^m$ we denote the set of all functions from $\mathcal{X}$ to $\mathbb{R}$ and the set of all possible training sets, respectively.

Learning algorithms have often parameters that must be selected before the learners are trained. For example, regularized kernel methods usually have a so-called regularization parameter that controls the trade-off between the empirical error and the complexity of the learned function. Moreover, the kernel function determining the space of functions that the method is able to learn has typically also parameters to be selected. We do not usually know in advance which values of the parameters provide us the best performance for unseen data. We can fix the values of these parameters by evaluating performance with a spectrum of various values, and choosing the one best one. This supervised learning phase is known as *validation* or *model selection*. For doing this validation, we have to have not only a measure but also a method to specify its value. In addition, the methods are needed to evaluate the final performance. This phase is called *testing* or *model assessment*.

3.2. Resubstitution

Techniques that evaluate the performance of the learner on the training set are called *resubstitution* methods. More formally, if we adopt the same notation as previously, the *resubstitution performance* of f is

$$p(Y, f(X)), \tag{3}$$

where, p is a function defining the performance evaluation measure. For classification tasks, for example, F measure or AUC can be chosen.

The resubstitution performance does not reliably predict the true performance of the learner on new data; the performance estimates are bound to be over-optimistic when same data is used both for training and testing. This does not, however, mean that it would not be useful to apply resubstitution too. For example, if the learner performs on the training set weakly, it is likely have an even weaker performance on unseen data. Again, if the performance on the training set is very high, problems related to over-fitting may be confronted in new data. Further, if the size of the training set is especially small, the resubstitution estimate may have smaller variance than the other less optimistically biased performance estimation methods, such as cross-validation or bootstrap (see [Braga-Neto and Dougherty, 2004] for a more thorough discussion about performance estimation in a small-sample setting). We will discuss these methods below. In summary, for reliable performance evaluation, it is essential that the data used for training, validation and testing are completely independent of each other in order to avoid over-optimistic bias.

3.3. Hold-out

One of the most popular ways to estimate the performance of the learner on an unseen data is to use a *hold-out* estimation technique. By hold-out, we mean that a part of the available data is set aside to form a hold-out set for performance estimation. This method can be used both with and without the validation phase, but hold-out set must always be independent of the data used at the previous phases. To simplify the notation, we next consider only a situation without validation.

Let

$$E = \{1, \ldots, m\}$$

be an index set, where the indices refer to training examples. Moreover, let

$$H \subseteq E$$

denote a set of indices referring to the examples in the training set that belong to the *hold-out set*, and let

$$\overline{H} = \{1, \ldots, m\} \setminus H = E \setminus H$$

be the set indexing the rest of the training examples. Further, let

$$\begin{aligned} X_H &\in (\mathcal{X}^{|H|})^{\mathrm{T}}, \\ Y_H &\in \mathbb{R}^{|H|}, \text{ and} \\ S_H &\in (\mathcal{X} \times \mathbb{R})^{|H|} \end{aligned}$$

denote, respectively, the sequence containing only the inputs, outputs, and training examples that are indexed by H. With this notation, S_H is the hold-out set. Further, let

$$f_{\overline{H}} = \mathcal{A}(S_{\overline{H}})$$

denote a learner that is trained using only the training examples $S_{\overline{H}}$, where $S_{\overline{H}}$ is defined analogously to S_H. By overloading our notation,

$$f_{\overline{H}}(X_H) \in \mathbb{R}^{|H|}$$

is a vector consisting of the output values for the hold-out examples X_H that are predicted by $f_{\overline{H}}$. The outcome of

$$p(Y_H, f_{\overline{H}}(X_H)), \tag{4}$$

where p is a performance measure, of a learner $f_{\overline{H}}$ on the hold-out set S_H is called the *hold-out performance*. In practice, it is common to hold one-third of the data out for testing and use the remaining two-thirds for training, and if validation is needed the training data is divided further equally between training and validation.

The problem in the basic hold-out technique is that the sets selected for testing and training may not be representative samples of the underlying problem. In *stratified hold-out*, the training and test sets are constituted from the full data set so that each class is properly represented in both sets. Another approach to secure the representative samples is to repeat the training and testing phases several times with different random divisions of the full dataset into training and test sets. In this *repeated hold-out technique*, the overall value of the performance measure is derived by averaging the values of (4) of the different iterations, and we next focus on it more carefully.

3.4. Cross-validation

From the idea in repeated hold-out, we can derive a performance evaluation technique known as *N-fold cross-validation*. Here, the data is first partitioned into N approximately equal folds. The folds are usually mutually exclusive. Then, each fold in turn is used for testing while the reminder is used for training (note that we again consider the simpler case without a validation phase). The N-fold cross-validation performance evaluation estimate is the average of the hold out estimates obtained with N cross-validation folds.

Note, however, that the cross-validation folds do not necessarily have to form a partition of the data set. Instead, we can perform cross-validation also by repeatedly selecting a random hold-out set of a certain size so that the hold-out sets in different cross-validation rounds may overlap with each other.

We now present formalizations for the cross-validation methods. In N-fold cross-validation, we have a sequence of hold-out sets $H_1, \ldots, H_N$, where $N \in \mathbb{N}$ and $H_j \subseteq E$. The overall performance is obtained by averaging (4) over the hold-out sets:

$$\frac{1}{N} \sum_{j=1}^{N} p(Y_{H_j}, f_{\overline{H_j}}(X_{H_j})). \tag{5}$$

The N-fold cross-validation performance estimates, of course, depend on the number N of the folds, since the smaller is their size, the closer the learner trained with the rest of the data set should be to the learner trained with the whole data set. The estimates are also dependent on the sequence of hold-out sets used. There are, in fact, $\binom{m}{m/N}$ possibilities to choose a hold-out set H of size $|H| = m/N$ out of m examples and the performance differences between the different possibilities may be large.

If the amount of available computational resources is sufficiently large or if the used learning method has efficient computational shortcuts for cross-validation such as certain types of nearest neighbor classifiers [Mullin and Sukthankar, 2000], one can perform so-called *complete cross-validation* [Kohavi, 1995] in which each of the $\binom{m}{m/N}$ hold-out splits are used at a time. Formally,

$$\binom{m}{m/N}^{-1} \sum_{H \subset E} p(Y_H, f_{\overline{H}}(X_H)). \tag{6}$$

A widely used and analyzed special case of N-fold cross-validation, in which $N = m$, and hence $|H| = 1$, is so-called *leave-one-out cross-validation*. By definition, leave-one-out cross-validation is also complete, since it uses every possible hold-out split of size one. A thorough review of the use of leave-one-out cross-validation in machine learning is made by [Elisseeff and Pontil, 2003].

We also note that some performance measures, F measure for example, may have to be evaluated with predictions originating from different cross-validation folds. This is the case especially if the amount of examples in each fold is too small for the measure to provide sensible results. For example, it does not make sense to compute F measure or AUC for one example only, and hence the measures cannot be evaluated using (5) together with the leave-one-out cross-validation. To formalize this, we first define a function

$$\kappa : \{1, \ldots, m\} \to 2^{\{1,\ldots,m\}},\ i \mapsto H,$$

where $2^{\{1,\ldots,m\}}$ denotes the powerset of $\{1, \ldots, m\}$, that maps each example index i to the set H consisting of the indices of other training examples that belong into the same cross-validation fold as the ith example. Note that κ is well-defined only if the cross-validation folds do not overlap with each other. Further, let

$$F = (f_{\overline{\kappa(1)}}(x_1), \ldots, f_{\overline{\kappa(m)}}(x_m))^{\mathrm{T}}.$$

Then, instead of using (5), the cross-validation performance can also be evaluated through

$$p(Y, F)$$

However, this approach, sometimes called the *micro-average*, may have some problems. We discuss these issues in the context of AUC in Section 4.3.

In order to assure that the hold-out splits in cross-validation estimates represent the underlying problem, stratification can also be used in forming the cross-validation folds. This results in a technique known as *stratified cross-validation*.

Cross-validation techniques are, in practice, the most often used ones to measure classification performance. In particular, (stratified) ten-fold cross-validation and ten times ten-fold cross-validation have been recommended [Kohavi, 1995]. Also the leave-one-out cross-validation is an attractive and prevalent technique; by following this method, the greatest possible amount of data is used for training in each step and random sampling to divide the data for training and testing is unnecessary. Further, leave-one-out cross-validation is known to be an almost unbiased estimator of the learning performance. However, the variance of leave-one-out cross-validation may be larger than that of N-fold cross-validation. This was experimentally demonstrated in [Kohavi, 1995], and the parameter selection method recommended there was ten-fold cross-validation repeated ten times with different fold partitions. We refer to the previous reference too for further discussion on the statistical properties of the cross-validation estimators.

The computational cost can be seen as a limitation for cross-validation techniques in general, and in particular for the leave-one-out form. If the learner has to be retrained each time a hold-out estimate is computed, the calculation of the average performance may be too expensive from a computational point of view. This is especially true when repetition with different partitions or leave-one-out cross-validation is used. Fortunately, with some learning algorithms, the training examples in the hold-out set can be efficiently "unlearned" and the retraining does not have to be performed. This is the case, for example, with the basic regularized least-squares learners (see, e.g., [Zhang, 1993, Pahikkala et al., 2006, Rifkin and Lippert, 2007, An et al., 2007, Pahikkala, 2008]). In addition, we have also shown that similar types of efficient cross-validation algorithms can also be constructed for the AUC maximizing regularized least-squares [Pahikkala et al., 2008a, Pahikkala et al., 2008b] and for the label ranking regularized least-squares [Pahikkala et al., 2007a, Pahikkala et al., 2007b].

3.5. Bootstrap

In *bootstrap*, a training set is formed by sampling the dataset with replacement. That is, the training set may contain multiplicated instances. Let us assume that a data set of m examples is sampled m times to give the training data set, and the examples of the original data set that are not selected for training are used for testing. Now, the probability of any given example not being chosen after m samples is $(1 - \frac{1}{m})^m \approx e^{-1} \approx 0.368$ and consequently, the expected number of distinct instances from the original dataset appearing in the test set is $0.632m$. Hence, this technique is called *0.632 bootstrap*.

The 0.632 bootstrap performance of f is counted by unifying the performance for the bootstrap sample i with the resubstitution performance in a following manner [Efron, 1983]: Let p^{BS_i} be the performance for the bootstrap sample i that is counted analogously to equation (4) but the indexes of the variables x and y are chosen according to the test set in question. Similarly, resubstitution performance for the sample i, p^{RS_i}, is defined by using the equation (3). Given the number N of bootstrap samples, the 0.632 bootstrap performance is

$$\frac{1}{N}\sum_{i=1}^{N}\left(0.632p^{BS_i} + 0.368p^{RS_i}\right).$$

The advantages of bootstrap are especially tangible for very small data sets. This was demonstrated, for example, in a comparative study [Braga-Neto and Dougherty, 2004] using both synthetic and real breast-cancer patient data: When compared to cross-validation error estimation, bootstrap was better method in terms of variance, but at the cost of computational demands and often increased bias, albeit much less than with resubstitution.

The smaller variance but the higher bias of the bootstrap error estimator has been proved more generally for example, in [Efron, 1983, Davison and Hall, 1992]). However, the 0.632 bootstrap has also shown to give an over-optimistic performance estimate when the classifier is a perfect memorizer and the data set is completely random [Kohavi, 1995]. To address these challenges, a so-called *0.632+ bootstrap* method has been introduced [Efron, 1997]. As [Ambroise and McLachlan, 2002] explain, the 0.632+ version weights p^{BS_i} more in situations with relatively large over-fitting measured by the difference $p^{BS_i} - p^{RS_i}$. Hence, they recommend using it in cases, where the learner is likely to be over-fitted.

3.6. Summary

The focus of the section was on alternative learning performance evaluation methods, and Figure 3 illustrates associations between the discussed approaches. We emphasized the importance of having completely independent, but representative training, validation and test sets. Furthermore, we concluded that cross-validation techniques, specifically ten-fold, ten times ten-fold and leave-one-out forms, are particularly suitable methods for classification tasks. With very small data sets, bootstrap is an attractive alternative.

4. PERFORMANCE EVALUATION METHOD IMPLEMENTATION

After selecting a performance method, follows an implementation step, when the value for a given measure and system is computed in practice. This step contains many fundamental and critical decisions: For reliable performance evaluation, the test set has to be completely independent from the creation of the learner, and test settings must correspond to the situation, where the classifier will be actually used. For example, if a machine learning system is tested with a material somehow dependent on the training or validation data, or in significantly different from the real input, the calculated performance level will not describe the true quality. However, it is not easy

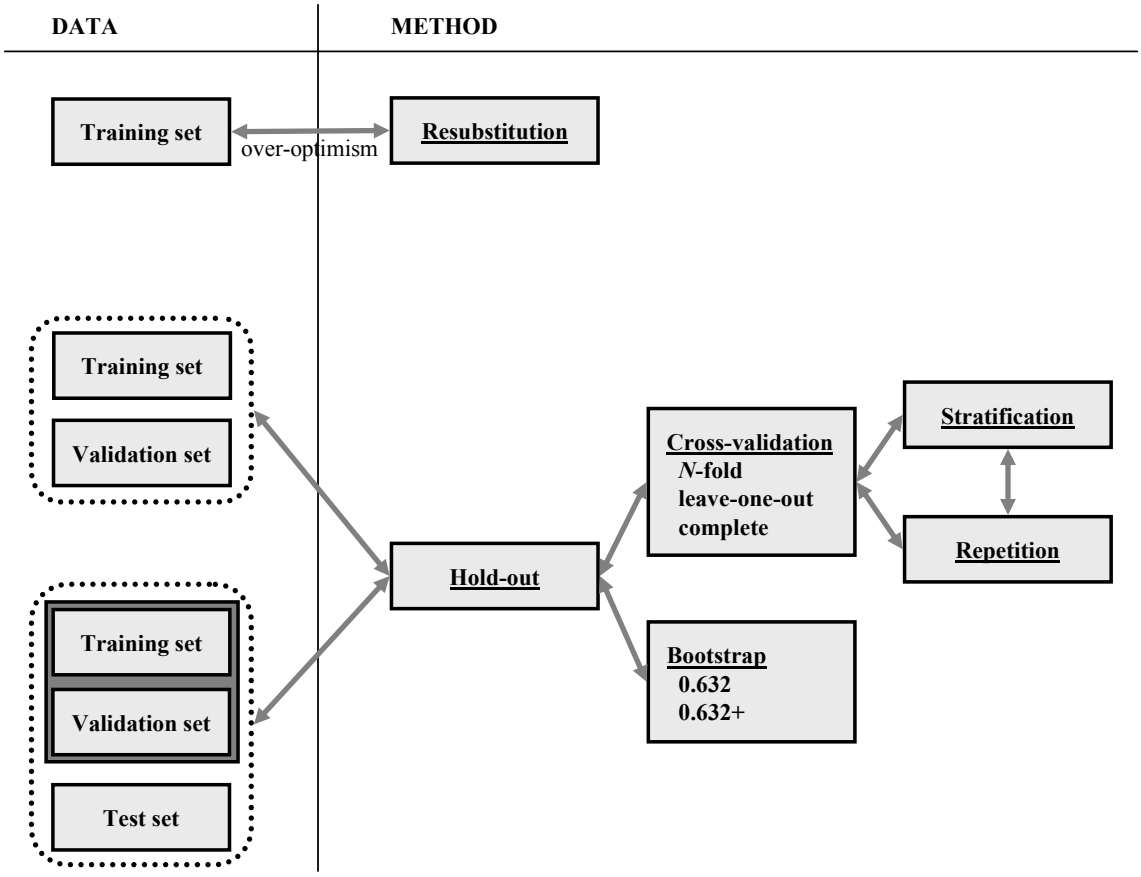

Figure 3. A schematic diagram of the discussed performance evaluation methods for supervised learners.

in practice to notice all data dependences that have to be taken into account when holding out data.

We next concretize the crucial principles of having a realistic test setting and no information leak between the phases of training, validation and testing. We begin by giving intuitive examples about dependence-related difficulties in Section 4.1. The examples are based on our previous experiences with text classification tasks, and the difficulties they portray may occur especially when evaluating classification performance with real-world data sets. We continue in Section 4.2 with techniques to avoid the problems caused by the dependences by reflecting the learning task in question when forming the cross-validation folds. Section 4.3 considers the trickiness of taking special characteristics of the data, learning task and measure into account in the cross-validation folds by focusing on measuring AUC during cross-validation. Finally, Section 4.4 synthesizes the lessons learnt.

4.1. Dependences in the Data

In classification performance evaluation, the aim is to assess the quality of class predictions on unseen data. Consequently, it is crucial that information from training does not leak into the hold-out set. With text classification, we see at least three types of task-specific semantic dependences that must be taken into account when dividing data into training and hold-out sets: *author*, *community* and *time*.

The aim in text classification is to categorize the documents based on the similarity of their content, and typically the dividing factor should not be the author, the community in which the text was written nor its writing time: Because of the individual differences in writing style, the author has a great impact on the written text, and as a result, texts written by the same person are very likely to be more similar than those written by two different people. Similarly, texts reflect the surrounding community and time, when it was written.

For example, every hospital ward has their own style of writing patient records due to documentation practices, jargon and abbreviations developed during time, and hence if multiple reports about the same patient case were written, those with authors from the same ward would probably be more similar than those written by people from different wards. Further, because, for example, treatments, medicines, hospital equipments etcetera change in time, also the language changes. Therefore, in a task of separating the documents of cardiology patients from those of fracture patients, problems may occur, if the document collection contains documents from a long time interval; reports written at the same time about different patient types may be more alike than those about the same patient type but written at different times.

Because of the author related dependences, one should

consider in performance evaluation whether it is necessary to avoid assigning texts written by the same author to training and test sets. This kind of avoidance may be needed, for example, in context sensitive spelling error detection because it does not happen in practice that a classifier performing the task is trained and used with instances originating from the same document. Another kind of dependence was present in [Suominen et al., 2006], where we studied automated identification of notes about a given topic in the domain of intensive care patient records. We did not use text about the same patient both for training and testing, because otherwise we could have got an over-optimistic impression about the performance; notes about the same patient both in training and testing can be seen as a potential information leak, as data about one patient is likely to be very homogeneous. Hence, it is easier for a learner to recognize notes relevant to a given topic if both sets contain data about same patients. Similarly, if a classifier will be used in many organizations and it is trained with data originating from all these places, its tested performance will be over-optimistic. Finally, our guideline for time-dependence is to test the performance regularly and re-train the classifier if necessary.

As an example of taking data dependences into account at the method step of performance evaluation, we focus on the study [Sætre et al., 2007]. The authors discuss differences between two commonly used alternatives of doing ten-fold cross-validation using a corpus containing information about protein-protein interactions. The first approach is to divide the data examples into ten groups before doing any analysis of the data, and the second one is to perform pre-processing and feature extraction parts of the analysis on the whole corpus only after that divide data into cross-validation folds. The previously mentioned reference shows evidence of a serious information leak from training to testing in the latter approach because it gave substantially better impression of the system performance than the former alternative.

Another study emphasizing unreliability of performance evaluation if it contains steps not in line with cross-validation is [Simon et al., 2003]. The authors' experiments are related to genetics and include measuring miss-classifications when using two types of leave-one-out cross-validation: one, where the left-out instance is removed before any processing and the other, where data is analyzed and pre-processed before the removal. As expected, the results underscore better reliability of the former approach. Authors also stress validation on independent data, which is large enough to demonstrate statistically learning performance.

As our general guideline, dependences and their handling in performance evaluation must be analyzed for each data set and task separately. Usually the procedure is similar to previous examples: One must define a unit, such as one author, patient, organization, or time interval, and avoid breaking these units when forming the hold-out sets. This hold-out strategy is known as *leave-cluster-out*, and when used in a cross-validation way, *leave-cluster-out cross-validation*, in which a whole *dependence cluster* of examples is held out in each cross-validation round, it has been empirically shown to be more suitable method than leave-one-out cross-validation [Pahikkala et al., 2006]. The key question is, however, which of many dependences present at the same time should be taken into account. A possible solution is to try to avoid as many of them as possible at the same time, or to replicate leave-cluster-out performance evaluation one dependence in turn and in this way assure that the system works in an acceptable level with all dependences. We will next address leave-cluster-out cross-validation with greater depth.

4.2. Reflecting the Learning Task in Cross-validation Folds

In [Pahikkala et al., 2006], we made an experiment in which we demonstrated the "clustered training set effect" by comparing the leave-one-out cross-validation performance of a trained regularized least-squares to a leave-cluster-out cross-validation performance so that each fold in the leave-cluster-out cross-validation consists of the training examples that form a cluster in the training set. The problem we considered was dependency parse ranking of sentences extracted from biomedical texts.

In the experiment, we obtained a training data set by generating a set of parse candidates for one hundred sentences. Each sentence had a known "correct" parse that a parser is supposed to output for the sentence. For each candidate parse, we calculated a score value indicating how close to the correct parse it is (see [Tsivtsivadze et al., 2005] for a more thorough description of the experimental setting).

The task of the learning machine was, for each of the one hundred sentences, to rank its candidate parses in the order of their score values. For this purpose, we trained a regularized least-squares regressor using all of the generated parses and their score values as training data. The ranking performance was measured for each sentence using Kendall's τ correlation coefficient, which can be considered as generalization of the AUC performance measures for more than two ordinal classes. The coefficient was, of course, calculated for each sentence separately, since we were not interested of the mutual order of the parses originating from different sentences. The overall performance for the whole data set was obtained by taking the average of the correlation coefficients of the sentences in the data set.

Due to the feature representation of the parses (see [Tsivtsivadze et al., 2005]), two parses originating from a same sentence have almost always larger mutual similarity than two parses originating from different sentences, and hence the data set consisting of the parses is heavily clustered according to the sentences the parses were generated from. Therefore, the clustered structure of the data had a strong effect on the performance estimates obtained by cross-validation, because data instances that are in the same cluster as a held out instance have a dominant effect

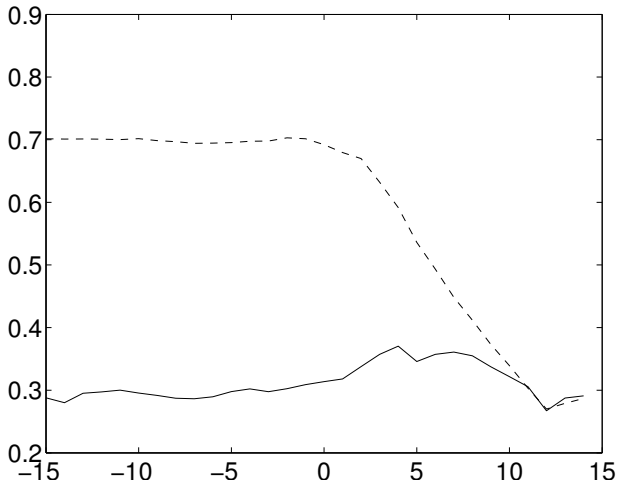

Figure 4. The ranking performance of the regularized least-squares algorithm computed with leave-one-out cross-validation (dashed line) and leave-cluster-out cross-validation (solid line) [Pahikkala et al., 2006]. The x-axis denotes the value of the regularization parameter in a logarithmic scale. The y-axis is the ranking performance measured with τ correlation coefficient.

on the predicted output of the held out instance. This does not, however, model the real world, since a parse ranker is usually not trained with parses originating from the sentence from which the new parse with an unknown score value is originated. The problem can be solved by performing the cross-validation on the sentence level so that all the parses generated from a sentence would always be either in the training set or in the test set.

We compared the performance estimates given by both leave-one-out cross-validation and leave-cluster-out cross-validation in which the fold partition was formed according to the sentences so that no parses originating from the same sentence ended up into the same cross-validation fold. Regularized least-squares has a regularization parameter that controls the trade-off between the training error and the complexity of the learned prediction function. We also made a grid search for the value of the regularization parameter to test the performances of the two cross-validation approaches also in model selection. The tested grid points were $2^{-15}, 2^{-14}, \ldots, 2^{14}$.

The results of the comparison are illustrated in Figure 4. From the figure, we observe that the performance difference between the leave-one-out and leave-cluster-out cross-validation is, with the lower values of the regularization parameter corresponding to the least regularized cases, over 0.3 correlation points. Thus, leave-one-out cross-validation clearly overestimates the ranking performance, especially with the smaller values of the regularization parameter. Increasing the regularization seems to correct this over-optimistic bias to some extent but the large values of the regularization parameter may not be optimal for the learning task. Indeed, this was confirmed when we tested the ranking performance of regularized least-squares with one hundred test sentences unseen to the regularized least-squares. The test performance was very closely following the performance of leave-cluster-out cross-validation for every tested value of the regularization parameter, and hence the optimal parameter values were found around 2^4, while the leave-one-out cross-validation preferred very low values.

4.3. Measuring AUC during Cross-validation

When aiming for a maximal AUC with biological data as considered by [Parker et al., 2007], a common practice for performance evaluation is to use a ten-fold cross-validation. The following two techniques for obtaining a single AUC value when performing cross-validation were considered by [Bradley, 1997]:

In *pooling*, the predictions made for the data instances in each cross-validation round are pooled into a one set and one AUC score common to all folds is calculated from it. When leave-one-out cross-validation is used, this is the only way to obtain the AUC score. The assumption made when the pooling approach is used is that each of the classifiers produced on each of the cross-validation rounds comes from the same population. This assumption may make sense when using performance measures such as classification accuracy, but it is more dubious when computing AUC, since some of the positive-negative pairs are constructed using data instances from different folds. Indeed, [Parker et al., 2007] show that this assumption is generally not valid for cross-validation and can lead to large pessimistic biases.

An alternative approach, *averaging*, is to calculate the AUC score separately for each cross-validation fold and average them to obtain one common estimate for the classification performance in cross-validation. However, the number of positive-negative example pairs used in computing the AUC scores may be too small to get reliable scores if the overall number of examples used in the cross-validation process is too small or if the number of cross-validation folds is too large. Further problems of the averaging approach are discussed by [Parker et al., 2007].

There is a clear fundamental similarity between these two cross-validation approaches and micro and macro-averaging discussed in Section 2.1. Note, however, that the purpose is different: before measuring the overall performance with one measure in the presence of multiple classes or labels was addressed, whereas in this section the aim is to construct one value representing performance over all cross-validation rounds.

To formalize pooling and averaging, recall that

$$\kappa : \{1, \ldots, m\} \rightarrow 2^{\{1,\ldots,m\}},\ i \mapsto H,$$

where $2^{\{1,\ldots,m\}}$ denotes the powerset of $\{1, \ldots, m\}$, is a function that maps each example index i to the set H consisting of the indices of other training examples that belong into the same cross-validation fold as the ith example. Note that κ is well-defined only if the cross-validation folds do not overlap. This is the case with both pooling and averaging. Formally, the pooling approach can

be written as

$$\frac{1}{y_+ y_-} \sum_{y_i=+1, y_j=-1} \delta(f_{\overline{\kappa(i)}}(x_i) > f_{\overline{\kappa(j)}}(x_j)). \quad (7)$$

In (7), the sum is taken over every possible positive-negative pair of training examples. However, $\overline{\kappa(i)} \neq \overline{\kappa(j)}$ for most of the pairs, and hence the pooling estimate can be biased. The averaging estimate can be written as

$$\frac{1}{N} \sum_{j=1}^{N} p_{\text{AUC}}(Y_{H_j}, f_{\overline{H_j}}(X_{H_j})), \quad (8)$$

Here, the p_{AUC} estimate for the individual folds is not biased in the way it is in the pooling approach. However, only a small subset of the positive-negative pairs of training examples are taken to account.

So-called *leave-pair-out cross-validation* was considered by [Cortes et al., 2007] in context of magnitude preserving ranking algorithms and by us in [Pahikkala et al., 2008a]. Here, we propose its use for AUC calculation, since it avoids many of the pitfalls associated to the pooling and averaging techniques. Analogously to leave-one-out cross-validation or complete cross-validation (6), each possible positive-negative pair of training instances is left out of at a time from the training set. In fact, leave-pair-out cross-validation can be considered as a stratified complete cross-validation with hold-out sets of size two, since each of the folds has the same amount of positive and negative examples. Formally, the AUC performance is calculated with leave-pair-out cross-validation as

$$\frac{1}{y_+ y_-} \sum_{y_i=+1, y_j=-1} \delta(f_{\overline{\{i,j\}}}(x_i) > f_{\overline{\{i,j\}}}(x_j)),$$

where $f_{\overline{\{i,j\}}}$ denotes a classifier trained without the ith and jth training example. Being an extreme form of averaging where each positive-negative pair of training examples forms an individual hold-out set, this approach is natural when AUC is used as a performance measure, since it guarantees the maximal use of available training data. Further, the leave-pair-out cross-validation estimate, taken over a training set of m examples, is an unbiased estimate of the true error over a sample of $m - 2$ examples (for a proof, see [Cortes et al., 2007]).

Similarly to other types of complete cross-validation, a naive implementation of leave-pair-out cross-validation would require training a number of models that is quadratic with respect to the number of training instances. However, for certain learning algorithms, such as regularized least-squares [Pahikkala et al., 2006, Pahikkala, 2008, An et al., 2007] or AUC maximizing regularized least-squares [Pahikkala et al., 2008b, Pahikkala et al., 2008a], using techniques based on matrix calculus the closed form solution of regularized least-squares can be manipulated to derive efficient algorithms for leave-pair-out cross-validation.

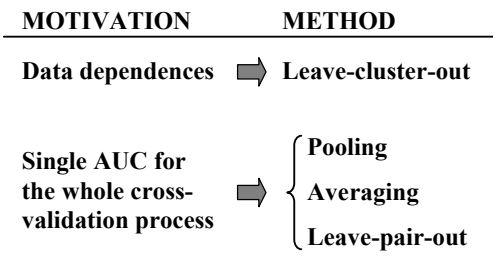

Figure 5. A schematic diagram of the discussed methods to take the preceding evaluation process into account at the method implementation step and their motivation.

4.4. Summary

In this section, we have addressed the step following method selection, that is, implementing the method for a given measure and learner. In addition to having independent and representative training, validation and test sets, the setting must be realistic. Fulfilling these principles in practice is, however, challenging and requires special caution. We explained a method to overcome data dependence-related difficulties (Figure 5). The dependences we considered in text classification tasks encompass author, community and time. Moreover, we described techniques to compute a single performance evaluation measure value for the whole cross-validation process which are particularly useful when the combination of AUC and leave-one-out cross-validation have been chosen.

In short, at the performance evaluation method implementation step it is essential to reflect the entire preceding evaluation process starting from the characteristics of the learning task and ending with the specifics of the chosen performance evaluation method. Instead of a divide and conquer methodology, implementation should be based on the analysis of the process in its entirety.

5. CONCLUSION

In this paper, we considered the measure and method selection steps of learning performance evaluation, with a main focus on combining AUC with cross-validation. Our goal was to provide helpful guidelines in order to obtain realistic assessment results.

We summarize the study with five general principles:

1. It is crucial to choose both the measure and method that reflects the evaluation aspects, learning task and data in question.

2. Testing must be completely independent from the creation of the learner. This can be assured by not having an information leak between training, validation and test sets.

3. Measure and method steps are intertwined. Therefore, the specifics of the measure should affect cross-validation folds.

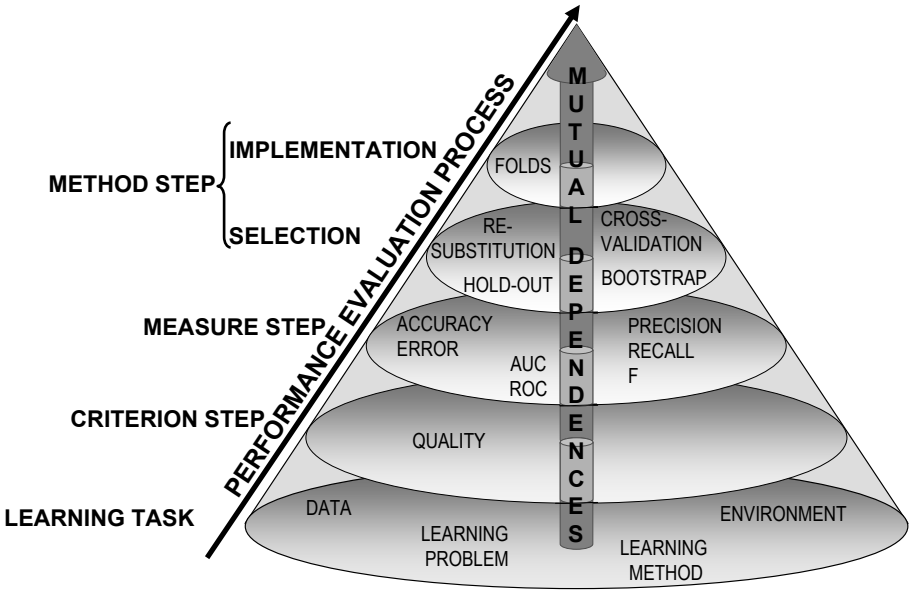

Figure 6. Steps and aspects of learning performance evaluation.

4. Data dependences may complicate performance evaluation. Their handling must be analyzed for each data set and task separately, but we encourage using leave-cluster-out cross-validation.

5. The evaluation setting must correspond to the environment. This has to be taken into consideration when forming the cross-validation folds.

We illustrate the principles in Figure 6 as the steps and aspects of learning performance evaluation: The first item is included as the stone base of the performance evaluation cone or pyramid. *Folds* at the method implementation step portray on the one hand the distinct phases of training, validation and testing (i.e., the second item) and on the other hand taking the specifics of the learning task into account when forming cross-validation folds (i.e., the items three, four, and five). The third item clarifies the dependent steps of choosing the measure, selecting the method and implementing it, that is, the top layer of the cone. The mutual dependence arrow connecting all the cone layers reflects the fourth item. This arrow connecting also the stone base with performance evaluation steps takes the fifth item into account.

Figure 6 is a refinement of the model discussed in [Spärck Jones and Galliers, 1996, pp. 19–20] and [Hirschman and Thompson, 1997] (Figure 1). The difference is in the base and top layers: we have added the stone base presenting the learning task, divided the method step at the ceiling level into the phases of selection and implementation, as well as concretized the mutual dependence of the steps as an arrow connecting all the layers. Our primary message is that all the steps are dependent of each other, and hence the previous steps has to be carefully considered in the process of performance evaluation.

6. ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of the Academy of Finland and the Finnish Funding Agency for Technology and Innovation, Tekes. We also thank anonymous reviewers for constructive feedback.

7. REFERENCES

[Airola et al., 2008] Airola, A., Pyysalo, S., Björne, J., Pahikkala, T., Ginter, F., and Salakoski, T. (2008). A graph kernel for protein-protein interaction extraction. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, pages 1–9, Columbus, Ohio, USA. Association for Computational Linguistics.

[Ambroise and McLachlan, 2002] Ambroise, C. and McLachlan, G. J. (2002). Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences of the United States of America*, 99(10):6562–6566.

[An et al., 2007] An, S., Liu, W., and Venkatesh, S. (2007). Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression. *Pattern Recognition*, 40(8):2154–2162.

[Bradley, 1997] Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159.

[Braga-Neto and Dougherty, 2004] Braga-Neto, U. M. and Dougherty, E. R. (2004). Is cross-validation valid for small-sample microarray classification? *Bioinformatics*, 20(3):374–380.

[Cortes and Mohri, 2004] Cortes, C. and Mohri, M. (2004). AUC optimization vs. error rate minimization. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, Massachusetts, USA.

[Cortes et al., 2007] Cortes, C., Mohri, M., and Rastogi, A. (2007). An alternative ranking problem for search engines. In *Proceedings of the 6th Workshop on Experimental Algorithms*, volume 4525 of *Lecture Notes in Computer Science*, pages 1–21. Springer, Berlin / Heidelberg, Germany.

[Davison and Hall, 1992] Davison, A. and Hall, P. (1992). On the bias and variability of bootstrap and cross-validation estimates of error rate in discrimination problems. *Biometrika*, 79(2):279–284.

[Efron, 1983] Efron, B. (1983). Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331.

[Efron, 1997] Efron, B. (1997). Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560.

[Elisseeff and Pontil, 2003] Elisseeff, A. and Pontil, M. (2003). Leave-one-out error and stability of learning algorithms with applications. In Suykens, J., Horvath, G., Basu, S., Micchelli, C., and Vandewalle, J., editors, *Advances in Learning Theory: Methods, Models and Applications*, volume 190 of *NATO Science Series III: Computer and Systems Sciences*, chapter 6, pages 111–130. IOS Press, Amsterdam, Netherlands.

[Fawcett and Flach, 2005] Fawcett, T. and Flach, P. (2005). A response to Webb and Ting's On the application of ROC analysis to predict classification performance under varying class distributions. *Machine Learning*, 58(1):33–38.

[Fung et al., 2006] Fung, G., Rosales, R., and Krishnapuram, B. (2006). Learning rankings via convex hull separation. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems 18*, pages 395–402. MIT Press, Cambridge, Massachusetts, USA.

[Hand and Till, 2001] Hand, D. and Till, R. (2001). A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186.

[Herbrich, 2002] Herbrich, R. (2002). *Learning kernel classifiers: theory and algorithms*. MIT Press, Cambridge, Massachusetts, USA.

[Hirschman and Thompson, 1997] Hirschman, L. and Thompson, H. S. (1997). Overview of evaluation in speech and natural language processing. In Cole, R., editor, *Survey of the State of the Art in Human Language Technology*, pages 409–414. Cambridge University Press, New York, NY.

[Kendall, 1970] Kendall, M. G. (1970). *Rank Correlation Methods*. Griffin, London, UK, 4. edition.

[Kohavi, 1995] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Mellish, C., editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1137–1143, San Mateo, California, USA. Morgan Kaufmann.

[Lachiche et al., 2006] Lachiche, N., Ferri, C., and Macskassy, S., editors (2006). *Proceedings of the Third Workshop on ROC Analysis in Machine Learning (ROCML'06)*. http://www.dsic.upv.es/%7Eflip/ROCML2006/.

[Lachiche and Flach, 2003] Lachiche, N. and Flach, P. (2003). Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC curves. In Fawcett, T. and Mishra, N., editors, *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, pages 416–423. AAAI Press.

[Mullin and Sukthankar, 2000] Mullin, M. and Sukthankar, R. (2000). Complete cross-validation for nearest neighbor classifiers. In Langley, P., editor, *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 639–646, San Francisco, California, USA. Morgan Kaufmann.

[Pahikkala, 2008] Pahikkala, T. (2008). *New Kernel Functions and Learning Methods for Text and Data Mining*. TUCS Dissertations No 103. Turku Centre for Computer Science, Turku, Finland.

[Pahikkala et al., 2008a] Pahikkala, T., Airola, A., Boberg, J., and Salakoski, T. (2008a). Exact and efficient leave-pair-out cross-validation for ranking RLS. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'08)*. To appear.

[Pahikkala et al., 2008b] Pahikkala, T., Airola, A., Suominen, H., Boberg, J., and Salakoski, T. (2008b). Efficient AUC maximization with regularized least-squares. In Holst, A., Kreuger, P., and Funk, P., editors, *Proceedings of the 10th Scandinavian Conference on Artificial Intelligence (SCAI 2008)*, volume 173 of *Frontiers in Artificial Intelligence and Applications*, pages 76–83. IOS Press, Amsterdam, Netherlands.

[Pahikkala et al., 2006] Pahikkala, T., Boberg, J., and Salakoski, T. (2006). Fast n-fold cross-validation for regularized least-squares. In Honkela, T., Raiko, T., Kortela, J., and Valpola, H., editors, *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)*, pages 83–90, Espoo, Finland. Otamedia.

[Pahikkala et al., 2007a] Pahikkala, T., Suominen, H., Boberg, J., and Salakoski, T. (2007a). Transductive ranking via pairwise regularized least-squares. In Frasconi, P., Kersting, K., and Tsuda, K., editors, *Workshop on Mining and Learning with Graphs (MLG'07)*, pages 175–178.

[Pahikkala et al., 2007b] Pahikkala, T., Tsivtsivadze, E., Airola, A., Boberg, J., and Salakoski, T. (2007b). Learning to rank with pairwise regularized least-squares. In Joachims, T., Li, H., Liu, T.-Y., and Zhai, C., editors, *SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, pages 27–33. http://research.microsoft.com/users/LR4IR-2007 /#a2.

[Parker et al., 2007] Parker, B. J., Gunter, S., and Bedo, J. (2007). Stratification bias in low signal microarray studies. *BMC Bioinformatics*, 8(326).

[Rifkin and Lippert, 2007] Rifkin, R. and Lippert, R. (2007). Notes on regularized least squares. Technical Report MIT-CSAIL-TR-2007-025, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.

[Sætre et al., 2007] Sætre, R., Sagae, K., and ichi Tsujii, J. (2007). Syntactic features for protein-protein interaction extraction. In Baker, C. and Su, J., editors, *Short Paper Proceedings of the 2nd International Symposium on Languages in Biology and Medicine (LBM 2007)*, volume 319 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Simon et al., 2003] Simon, R., Radmacher, M. D., Dobbin, K., and McShane, L. M. (2003). Pitfalls in the use of dna microarray data for diagnostic and prognostic classification. *Journal of the National Cancer Institute*, 95(1):14–18.

[Spärck Jones and Galliers, 1996] Spärck Jones, K. and Galliers, J. R. (1996). *Evaluating natural language processing systems: an analysis and review*, volume 1083 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany.

[Suominen et al., 2006] Suominen, H., Pahikkala, T., Hiissa, M., Lehtikunnas, T., Back, B., Karsten, H., Salanterä, S., and Salakoski, T. (2006). Relevance ranking of intensive care nursing narratives. In Gabrys, B., Howlett, R., and Jain, L., editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 4251 of *Lecture Notes in Computer Science*, pages 720–727, Berlin / Heidelberg, Germany. Springer.

[Suominen et al., 2008] Suominen, H., Pyysalo, S., Hiissa, M., Ginter, F., Liu, S., Marghescu, D., Pahikkala, T., Back, B., Karsten, H., and Salakoski, T. (2008). *Handbook of Research on Text and Web Mining Technologies*, chapter Performance Evaluation Measures for Text Mining. IGI Global, Hershey, Pennsylvania, USA. To appear.

[Tsivtsivadze et al., 2005] Tsivtsivadze, E., Pahikkala, T., Pyysalo, S., Boberg, J., Mylläri, A., and Salakoski, T. (2005). Regularized least-squares for parse ranking. In Famili, A., Kok, J., Peña, J., Siebes, A., and Feelders, A., editors, *Proceedings of the 6th International Symposium on Intelligent Data Analysis*, volume 3646 of *Lecture Notes in Computer Science*, pages 464–474, Heidelberg, Germany. Springer.

[Zhang, 1993] Zhang, P. (1993). Model selection via multifold cross validation. *The Annals of Statistics*, 21(1):299–313.