3 Evaluating Recommendations of Metalearning/AutoML Systems

Summary. This chapter discusses some typical approaches that are commonly used to evaluate metalearning and AutoML systems. This helps us to establish whether we can trust the recommendations provided by a particular system, and also provides a way of comparing different competing approaches. As the performance of algorithms may vary substantially across different tasks, it is often necessary to normalize the performance values first to make comparisons meaningful. This chapter discusses some common normalization methods used. As often a given metalearning system outputs a sequence of algorithms to test, we can study how similar this sequence is from the ideal sequence. This can be determined by looking at a degree of correlation between the two sequences. This chapter provides more details on this issue. One common way of comparing systems is by considering the effect of selecting different algorithms (workflows) on base-level performance and determining how the performance evolves with time. If the ideal performance is known, it is possible to calculate the value of *performance loss*. The loss curve shows how the loss evolves with time or what its value is at the maximum available time (i.e., the time budget) given beforehand. This chapter also describes the methodology that is commonly used in comparisons involving several metalearning/AutoML systems with recourse to statistical tests.

3.1 Introduction

In this chapter we describe the methodology that can be used to assess the quality of recommendations generated by a given metalearning/AutoML system.

In many tasks it is necessary to compare performance of base-level algorithms across different tasks (datasets). As some tasks may be more difficult than others, the performance may differ substantially across tasks. Hence it may be useful to rescale the performance values so that these could be compared. Different rescaling techniques are discussed in Section 3.3.

The recommendations of typical metalearning/AutoML system can be seen as suggestions regarding which particular algorithm (workflow) should be applied to the target dataset. Quite often, the suggestions are in the form of an ordered sequence of algorithms (or workflows), which can be considered as the *recommended ranking* generated.

The strategy of recommending a ranked list of items can be compared to the strategy used in information retrieval, where normally a list of potentially useful documents is suggested to the user. The reason for this is simple: the first item in the list may not be relevant, and hence it is better to give the user other options. But let us come back to the topic of evaluation, where the following issues arise:

1. Is the performance of the particular metalearning/AutoML system satisfactory from the user's point of view?
2. How does the performance of the particular metalearning/AutoML system compare with other systems of this kind (competitors?)

The aim of this chapter is to address both issues above. As for the first issue, there are two ways of assessing the quality of the recommended ranking of algorithms (workflows). One is by comparing the recommended ranking with a golden standard. This issue is taken up in Section 3.5. The second one aims to assess the effects when the recommendations are followed. More details about this are given in Section 3.6.

3.2 Methodology for Evaluating Base-Level Algorithms

The topic of evaluation of machine learning (ML) algorithms is discussed in various textbooks on machine learning (see, e.g., Mitchell (1997); Kohavi (1995); Schaffer (1993)), so it is not discussed in great detail here. However, we need to present some basic concepts that are required to explain the methodology for evaluating metalearning/AutoML systems.

3.2.1 Generalization error

One of the important concepts in ML is the concept of *generalization error*. ML algorithms are normally designed to minimize this error. In other words, when presented with some dataset, one should not try to fit this data perfectly, but rather try to generate a model that would perform well on yet unseen data points. In the following section, we will briefly review how this can be measured.

3.2.2 Evaluation strategies

The following evaluation strategies can be used to measure the generalization on unseen data points: *holdout*, *cross-validation*, or *leave-one-out* evaluation (Kohavi, 1995; Schaffer, 1993).

Holdout. When applying the holdout evaluation, the original dataset is split into two sets, a training set, consisting of, e.g., 90%, and a holdout set consisting of the remaining 10%. The model is trained on the training set, and evaluated on the holdout set. The division into two subsets is controlled by a parameter, and its setting affects the estimate of the performance.

Cross-validation (CV) is often used to overcome the problem of the holdout method, by evaluating the model several times. In so-called 10-fold cross-validation the given algorithm would be evaluated 10 times. In each fold, it would be trained on a different portion of the original dataset containing 10% of instances, while the remaining 90% would be used for testing. So each fold would result in certain performance measures. In classification, for instance, this would normally be *accuracy, precision, recall, AUC,* etc. The values of a particular metric are often aggregated. So, for instance, from 10 accuracy values we can obtain the mean value of accuracy.

Leave-one-out (LOO-CV) can be considered as a special case of cross validation procedure. Let us denote the size of the dataset by $|d|$. So in this case, the number of folds (cycles) is equal to $|d|$. It trains the model $|d|$ times on a train set of size $|d| - 1$, and evaluates it on the final data point. This way each data instance is used exactly once for testing. This method is often used when there are relatively few examples for both training and testing.

Bootstrap evaluation This strategy is used when the number of cases is small. The augmented dataset is created by generating a certain number of so-called *bootstrap samples* from the original dataset, where each original data point can be sampled with replacement (meaning that it can occur multiple times in the bootstrap). If a bootstrap of the same size as the original dataset was generated, naturally some data points have been omited, since others will occur multiple times. These data points will form the test set.

The machine learning community has adopted cross-validation as the standard for evaluating algorithm performance. However, for large datasets (e.g., image datasets) a single holdout set is often used instead. In this book, we often use the term cross-validation evaluation.

In Section 3.4 we explain how these measures are extended for evaluating metalearning and AutoML systems. In the next section we address the issue of normalization of performance values, which is needed when we want to carry out comparisons across different datasets.

3.2.3 Loss function and loss

In machine learning literature, the term *loss* is often used instead of *error*. Loss is a numeric value that is obtained by applying a *loss function* to a given algorithm a with a certain configuration of hyperparameters $\boldsymbol{\theta}$ and a given dataset d. The loss function can be defined as follows:

$$L = \mathcal{L}(a^j_{\boldsymbol{\theta}}, d_{train}, d_{valid}), \tag{3.1}$$

where d_{train} is the training portion of d and d_{valid} is the validation set, that is, a subset of d used for evaluation purposes. This distinction between validation and test set is clarified further on in Section 3.4.1.

Let us see how it can be formulated to provide more details about the processes involved. Let $M(a^j_{\boldsymbol{\theta}}, d_{train})$ represent the output of the trained model generated by $a^j_{\boldsymbol{\theta}}$ on d_{train}, the training portion of a given dataset d. Let

$$\hat{y}_{valid} = A(M(a^j_{\boldsymbol{\theta}}, d_{train}), X_{valid}) \tag{3.2}$$

represent the application of the trained model to X_{valid}, that is, a part of d_{valid} that includes just the attribute values. This function returns a prediction of the base-level performance $\hat{y}_{valid}$. Then the loss L can be determined by comparing the predictions $\hat{y}_{valid}$ with using true values y_{valid}, which is the part of d_{valid} that includes just the target variable:

$$L = \mathcal{L}(\hat{y}_{valid}, y_{valid}). \tag{3.3}$$

Sometimes it is convenient to use the short form of the loss function, shown earlier (Eq. 3.1), that includes just the input arguments.

3.3 Normalization of Performance for Base-Level Algorithms

We note that the range of performance values for different datasets may vary substantially. An accuracy of 90% may be quite high on a classification problem, but low on another one. If we want to compare the performance of systems across different datasets, it is important to rescale the values. Different approaches were proposed in the past:

- Substituting performance values by ranks
- Rescaling into 0–1 interval
- Mapping values into a normal distribution
- Rescaling into quantile values
- Normalizing by considering error margin

More details about each transformation are given in the following subsections.

Substituting performance values by ranks

This transformation is described in Chapter 2 (Section 2.2.1).

Rescaling into 0–1 interval

This transformation requires that we identify the best (maximal) performance P^d_{max} and the worst (minimal) performance P^d_{min} of an algorithm on dataset d. Regarding the worst performance, it is common to use the performance of the default classifier, which simply predicts the most frequent class. These two values determine an interval between 0 and 1, and all values are rescaled into this interval. The following equation shows how a particular performance value P^d can be rescaled into P'^d:

$$P'^d = \frac{P^d - P^d_{min}}{P^d_{max} - P^d_{min}}. \tag{3.4}$$

Values of P'^d close to 0 (1) now indicate that the performance is close to the lowest (highest) value measured for this dataset.

Mapping values into a normal distribution

Another possibility is to use a standard normalization method, which requires that we calculate the *mean* and the *standard deviation* of all performance values. Then all success rates (or other performance values) are normalized by subtracting the mean value and by dividing the result by the standard deviation. The advantage of this method is that the values have a rather clear interpretation. Higher negative values (i.e., $<$ –0.5) indicate that the success rate is rather low. Values around 0 show that the success rate is not far from the average. High positive values (i.e., $>$ 0.5) indicate that the performance is rather good.

The disadvantage of this method is that it assumes that the values are distributed normally. This may not hold in some applications.

Rescaling into quantile values

This method transforms all values into quantiles, which are cut points that divide the range of a probability distribution into continuous intervals with equal probabilities.

Normalizing by considering error margin

Gama and Brazdil (1995) suggested that all performance values be expressed in terms of the *error margin* (Mitchell, 1997), which is calculated as follows: $EM = sqrt(ER \times (1 - ER)/NT)$, where ER represents the error rate and NT the number of examples in the test set. The errors are converted to values indicating how many EMs it is below the best error rate, or alternatively above the worst error rate. The disadvantage of this method is that it assumes that the values are distributed normally.

3.4 Methodology for Evaluating Metalearning and AutoML Systems

In this section we describe the evaluation methodology that needs to be considered when evaluating metalearning and AutoML systems. We distinguish two modes: in one, the evaluation is carried out just once, following the strategy (protocol) of hold-out. More details can be found in the next subsection. The other mode involves a cycle following the cross-validation (CV) (or leave-one-out (LOO)) strategy. More details about this are given in Subsection 3.4.2.

3.4.1 One-pass evaluation with hold-out

This scheme follows the strategy described in Section 3.2, which relies on the division of the given dataset into a training subset and test set. This division is determined by the user setting up the experiment. It is outside the scope of a given metalearning/AutoML system. When comparing the performance of various metalearning/AutoML systems, this division should be constant in all comparisons. Many metalearning/AutoML systems carry out such evaluation internally to determine which base-level algorithm should be recommended. Let us examine this in more detail.

Objective of metalearning/AutoML systems

In the Introduction we have described various metalearning/AutoML problems, including algorithm selection (AS), hyperparameter optimization (HPO) and combined algorithm selection and hyperparameter optimization (CASH). Here we will consider CASH, as it subsumes the other two.

The formal definition of the CASH problem exploits the concept of *loss* discussed in Section 3.2.3. The following definition is based on the work of Thornton et al. (2013):

$$a^*_{\boldsymbol{\theta}*} = \underset{a^j \in \mathcal{A}, \boldsymbol{\theta} \in \Theta^j}{\arg\min} \; \mathcal{L}(a^j_{\boldsymbol{\theta}}, d_{train}, d_{valid}). \tag{3.5}$$

As we see, the objective of metalearning systems is to minimize the loss by exploring different alternative algorithms and hyperparameter settings. In this process, they also carry out evaluation so that they would come up with the best recommendation. This issue is addressed in the next subsection.

Inner evaluation carried out by metalearning/AutoML systems

Many metalearning/AutoML systems include an inner loop which includes evaluation: they try out some base model, evaluate this model on a set of unseen data cases, and repeat this process again.

As these systems do not (and must not) use the test set in any way, an additional set of cases is required (Varma and Simon, 2006). So, the training set is further divided into an *inner training set*, on which the base models are trained, and a *validation set*, which is used to assess the performance of different variants (e.g., with different hyperparameter settings) and select the best one. This division can be determined by a parameter of the metalearning system, and can even be optimized without our intervention. The division of the dataset is illustrated in Figure 3.1.

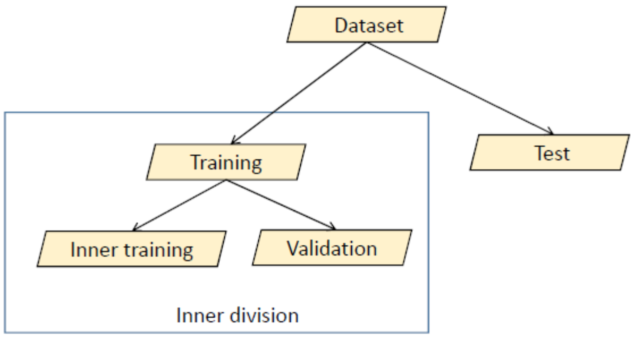


Fig. 3.1: Training, test, and validation set

The model that performs best on the validation set is selected by the metalearning/AutoML system. It is possible to retrain it on the whole training set (i.e., the inner training set plus the validation set), so that it would include as much data as possible to enhance its performance. Finally, the model is evaluated on the test set and its performance is reported to the user.

Note that the internal division can also be upgraded to include a cross-validation procedure in the inner evaluation concerned with selection. This has the advantage that different performance values obtained on different folds can be aggregated into a single value (e.g., mean) with lower dispersion. Consequently, the decisions made on the basis of this value tend to be more reliable. The disadvantage is that using CV takes more time than using hold-out, and hence the system takes more time to come up with a recommendation.

We note that this evaluation method provides a single result, as, in fact, it uses the methodology of hold-out on the meta-level. The use of cross-validation (CV) (or leave-one-out (LOO-CV), on the meta-level is discussed in the next subsection.

Avoiding biased evaluation

When evaluating an AutoML system that includes metalearning capabilities, it is important to ensure that the dataset used for evaluation is not included in the meta-dataset accompanying the metalearning system, as this would lead to a biased estimate of performance. Some systems, such as Auto-sklearn, are shipped with a meta-dataset consisting of many common benchmark datasets. If the aim is to evaluate how this system performs on new datasets, obviously these should not be included in the meta-dataset accompanying the system.

However, if the aim is to simply use the metalearning system to solve practical problems, it does not matter if the new dataset belongs to the pool of existing meta-datasets, or if it is similar to the one in this pool. If this happens, one might expect that the system would have used its metalearning capability to do well in such situations.

3.4.2 Meta-level evaluation with cross-validation

The multiple pass evaluation with cross-validation requires that the base-level datasets are considered as *instances*. Then these can be divided into folds, as outlined in the description of the cross-validation (CV) strategy in Section 3.2. As the process is repeated for all folds, we obtain as many test results as there are folds. This information can be aggregated into aggregated values (e.g., averages of individual performance values).

We note that in many domains the meta-dataset can be relatively small and include a limited number of base-level datasets. Hence, the leave-one-out (LOO-CV) methodology is often adopted, as it is appropriate in such situations. As the case left out is, in this case, a base-level dataset, this strategy could be referred to as *leave-one-dataset-out*.

Meta-level evaluation with table lookups

As evaluating base-level algorithms can be computationally expensive, it is common to use the following strategy: the performance of each algorithm on every dataset is recorded and stored in the corresponding meta-dataset. Whenever it is necessary to conduct the same evaluation again, previous results are retrieved using simply a table lookup.

The advantage of such an evaluation methodology is that it facilitates the task of conducting large-scale evaluation of metalearning systems. As the results of the experiments are pre-computed, the experiments can be carried out multiple times without computational overhead. Table lookups can be extended by surrogate models; an empirical performance model is used to predict the performance of configurations that were not explicitly examined. NAS-Bench-101 is a recent large-scale meta-dataset that can be used to retrieve experimental results.[1]

3.5 Evaluating Recommendations by Correlation

The quality of the recommended ranking of algorithms (workflows) is typically established through comparison with the *golden standard*, that is, the ideal ranking on the

[1]NASBench: A neural architecture search dataset and benchmark, https://github.com/google-research/nasbench

new (test) dataset(s). Sometimes, the ideal ranking is also referred to as the *true ranking*.

This evaluation protocol is applicable for evaluating metalearning systems that produce a ranking, such as the one presented in Chapter 2. In each leave-one-dataset-out cycle, the recommended ranking is compared against the ideal (true) ranking on the left-out dataset, and then the results are averaged for all cycles.

Different predicted rankings have different degrees of accuracy. For instance, given the golden standard (true ranking) is $(1, 2, 3, 4)$, the ordering $(2, 1, 3, 4)$ is intuitively a better prediction (i.e., is more accurate) than the ordering $(4, 3, 2, 1)$. This is because the former ordering is more similar to the true ranking than the latter one.

Different measures can be used to evaluate how close (or how similar or how accurate) the recommended ranking is to the golden standard. This distance represents in effect a measure of *ranking accuracy*. A very common one is based on rank correlation used earlier (Sohn, 1999; Brazdil and Soares, 2000; Brazdil et al., 2003). Here we have opted for *Spearman's rank correlation* (Neave and Worthington, 1992), but Kendall's Tau correlation could have been used as well. Obviously, we want to obtain rankings that are highly correlated with the golden standard. This will enable us to assess the accuracy of a given ranking method.

Metalearning/AutoML method M_A will be considered *more accurate* than method M_B if it generates a ranked list of recommendations that is more similar to the true ranking than those obtained by method M_B.

Spearman's rank correlation

The similarity between predicted and true rankings can be measured using Spearman's rank correlation coefficient (Spearman, 1904; Neave and Worthington, 1992) (see Eq. 3.6):

$$r_s(\hat{R}, R)) = \frac{\sum_{i=1}^{n}(\hat{R}_i - \bar{\hat{R}}_i)(R_i - \bar{R}_i)}{\sqrt{(}\sum_{i=1}^{n}(\hat{R}_i - \bar{\hat{R}}_i)^2 \sum_{i=1}^{n}(R_i - \bar{R}_i)^2)}, \tag{3.6}$$

where $\hat{R}_i$ represents the predicted ranks, $\bar{\hat{R}}_i$ their mean value, R_i the true rank of item i, and n the number of items. In situations where there are no ties, the following formula (Eq. 3.7) can be used

$$r_s(\hat{R}, R)) = 1 - \frac{6\sum_{i=1}^{n}(\hat{R}_i - R_i)^2}{n^3 - n}. \tag{3.7}$$

An interesting property of Spearman's coefficient is that it is basically the sum of squared rank errors, which can be related to the normalized mean squared error measure, commonly used in regression (Torgo, 1999).

The sum is rescaled to yield more meaningful values: the value of 1 represents perfect agreement, and –1 represents complete disagreement. A correlation of 0 means that the rankings are not related, which would be the expected score of a random ranking method.

The statistical significance of the values of r_S can be obtained from the corresponding table of critical values, which can be found in many textbooks on statistics (e.g., Neave and Worthington (1992)).

Table 3.1: Recommended and true ranking for the `letter` dataset

Algorithm	C5b	C5r	C5t	MLP	RBFN	LD	Lt	IB1	NB	RIP
Recommended	1.5	5.5	7	9	10	4	3	1.5	5.5	8
True	1	3	5	7	10	8	4	2	9	6

The use of Spearman's rank correlation coefficient (Equation 3.6) to evaluate ranking accuracy is illustrated in Table 3.1.[2] The reader can verify that the value of Spearman's correlation r_s is 0.707. According to the table of critical values of r_S, the value obtained is significant at a level of 2.5% (one-sided test).[3] Therefore, Spearman's coefficient confirms that the recommended ranking is a good approximation to the true ranking.

Weighted rank measure of correlation

Spearman's rank correlation coefficient has the disadvantage that it treats all ranks equally. An alternative measure is a weighted rank measure of correlation which gives more importance to higher ranks than lower ones, following da Costa and Soares (2005) and da Costa (2015). This measure weighs the distance between two ranks using a linear function of those ranks:

$$r_w(\hat{R}, R)) = 1 - \frac{6\sum_{i=1}^{n}(\hat{R}_i - R_i)^2((n - \hat{R}_i + 1) + (n - R_i + 1))}{n^4 + n^3 - n^2 - n}. \tag{3.8}$$

The authors provide a table of critical values permitting to test whether a given value of the coefficient is significantly different from zero.

The weighted measure of correlation is useful in many practical applications including, besides algorithm recommendation, information retrieval, stock trading, and recommender systems. In all these systems the output is in the form of a ranking.

3.6 Evaluating the Effects of Recommendations

A disadvantage of using correlation to evaluate rankings is that it does not show directly what the user is gaining or losing when following the ranked list of recommendations. As such, many researchers have adopted an approach which simulates the sequential evaluation of algorithms on the new dataset, as the ranked list of recommended algorithms is followed.

3.6.1 Performance loss and loss curves

The measure that is used is the *performance loss*, defined as the difference in accuracy between $\hat{a*}$ and $a*$, where $\hat{a*}$ represents the best algorithm identified by the system at a particular time and $a*$ the truly best algorithm that is known to us (Leite et al., 2012). Note that in many cases the truly best algorithm is not known to us. However, it is common to use some proxy, such as the algorithm found after some rather long search.

[2]The recommended ranking is the same as the one presented in Table 2.3 in Chapter 2.

[3]The significance level is $1-$ confidence level.

Many loss curves used in the literature show how the loss depends on the number of tests carried out. An example of such a curve is shown in Figure 3.2.

As tests may take different times, it is useful to show how the loss depends on time. Let us use the term *loss–time curves* to refer to such curves. Figure 3.3 shows the result.

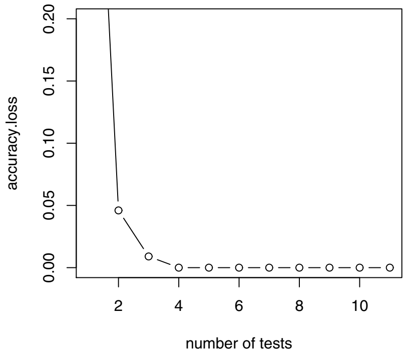


Fig. 3.2: Loss curves depending on the number of tests

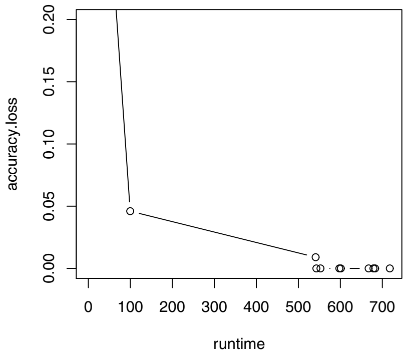


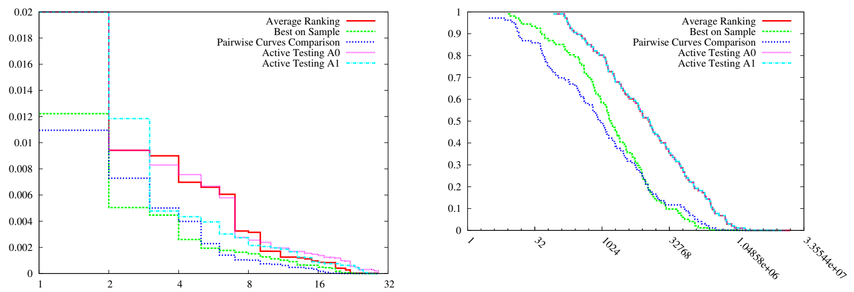
Fig. 3.3: Loss curves depending on runtime

3.6.2 Characterizing loss curves by AUC

Each loss–time curve can be characterized by a value representing the *mean loss* in a given interval, corresponding to the area under the loss curve. This characteristic is similar to the *area under the curve (AUC)*, but there is one important difference. When talking about AUCs, the *x*-axis values span between 0 and 1, while the loss curves span between some T_{min} and T_{max} defined by the user. Typically, the user searching for a suitable algorithm would not worry about very short times where the loss could still be rather high. In the experiments carried out by Abdulrahman et al. (2018), T_{min} was set to 10 seconds. In an on-line setting, however, we might need a much smaller value. The value of T_{max} was set to 10^4 seconds, corresponding to about 2.78 hours. We assume that most users would be willing to wait a few hours, but not days, for the answer. Also, many loss curves reach 0, or values very near 0, at this time. Note that this is an arbitrary setting representing a kind of default. Suitable values should be sought in accordance with the requirements of a particular domain.

3.6.3 Aggregating loss curves from multiple passes of CV

In Subsection 3.4.2 we discussed multiple-pass evaluation of metalearning/AutoML systems with recourse to either a CV or LOO-CV strategy. In each pass of the evaluation the system generates one loss/loss–time curve. It is thus useful to aggregate the individual loss into a *mean loss curve* in order to obtain an overall picture. An alternative to this would be to construct a *median loss curve*. Figure 3.4 shows the loss/loss–time curves of five different metalearning systems obtained on tests on 105 datasets. It is often useful



(a) Mean loss curve (b) Mean loss–time curve

Fig. 3.4: Mean loss and loss–time curves of five metalearning systems obtained on 105 datasets. Image taken from van Rijn (2016)

to elaborate percentile bands (e.g., 25% and 75%), showing where the most frequent values lie.

3.6.4 Statistical tests at a given time budget

The need for statistical comparisons is motivated by the fact that showing that a metalearning/AutoML method generates more accurate predictions than another one on average does not provide a sufficiently convincing argument. The values used in the comparison are *estimates* of the corresponding true values obtained on a sample of datasets. These estimates, like the estimates of accuracy of algorithms in the learning tasks, have a certain variance, which may imply that the differences between two methods may not be statistically significant. Therefore, we need a methodology to assess the statistical significance of the differences between metalearning/AutoML methods.

Statistical tests can be applied whenever we have two (or more) series of numeric values capturing some aspect of the performance of two (or more) metalearning/AutoML methods. The numbers can be of two types. The first one involves some kind of correlation coefficients between the recommended ranking and true ranking (see Section 3.5). The second involves values that characterize two loss curves. This can be performance at a given time budget, or alternatively AUC values that characterize performance in a given interval of time. In situations when two or more methods have been used on multiple datasets, it is usual to follow the multiple comparison procedure described by Demšar (2006). It involves Friedman's test, and if this test indicates that there are significant differences among the methods, it is possible to proceed with a post hoc test, which can be either Nemenyi test or Dunn's test.

Systems can be ranked according to their performance at a given time budget. If we do not know beforehand the value of the budget, but know the time interval where the budget could lie, it is possible to carry out aggregation across all the values in this interval. One way to do this is to estimate the *area under the loss–time curve* in this interval. Figure 3.5 shows an example. All systems are ranked according to their performance, measured by the area under the loss–time curve in the given interval. It would of course be possible to create an alternative ranking relative to a specific time budget.

These ranks are represented in 1D linear space. If the distance between two ranks is greater than the so-called *critical distance*, the performance difference between the two systems is statistically significant. This critical distance depends on the number of algorithms and the number of datasets. Systems for which no statistical difference was found are connected by a thick black line.

In our example the test did not find a statistical difference between "Pairwise Curves Comparisons" and "Best on Sample". Indeed, as Figure 3.4(b) already showed, the two loss–time curves are near one another, so one might be inclined to come to a similar conclusion by just looking at the earlier figure. On the other hand, the figure shows that there is statistical difference between "Pairwise Curves Comparisons" and the "Average Rank". Note that these conclusions depend on the selected time budget, selected datasets, and, of course, the metalearning systems under scrutiny.

3.7 Some Useful Measures

3.7.1 Loose accuracy

The loose accuracy measure ($LA@X$) uses the ranking accuracy of the top X elements in the ranked list (Kalousis, 2002; Sun and Pfahringer, 2013). $LA@X$ returns the value of 1 if one of the X top predicted elements includes the top element in the correct ranking, and otherwise returns 0. $LA@1$ is a special case and is sometimes referred to as *restricted accuracy*. It returns the value of 1 if the top prediction is correct, and 0 otherwise.

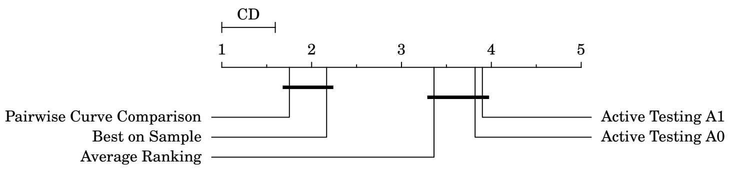


Fig. 3.5: A typical result obtained with the Nemenyi test. Image taken from van Rijn (2016)

3.7.2 Normalized discounted cumulative gain (DCG)

Discounted cumulative gain (DCG) has often been used to evaluate the effectiveness of search engines (Järvelin and Kekäläinen, 2002) and has been adopted in some work on algorithm selection (e.g., Sun and Pfahringer (2013)). This function uses a graded relevance scale in accordance with the corresponding position in the ranked list:

$$NDCG@X = DCG@X \times (IDCG@X)^{-1}, \tag{3.9}$$

where $IDCG@X$ represents the ideal DCG at X. The term $DCG@X$ is defined as

$$DCG@X = \sum_{i=1}^{X} (\frac{2^{g_i - 1}}{log_2(i+1)}), \tag{3.10}$$

where g_i is the value of the position in the ranked list.

References

Abdulrahman, S., Brazdil, P., van Rijn, J. N., and Vanschoren, J. (2018). Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine Learning*, 107(1):79–108.

Brazdil, P. and Soares, C. (2000). A comparison of ranking methods for classification algorithm selection. In de Mántaras, R. L. and Plaza, E., editors, *Machine Learning: Proceedings of the 11th European Conference on Machine Learning ECML 2000*, pages 63–74. Springer.

Brazdil, P., Soares, C., and da Costa, J. P. (2003). Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277.

da Costa, J. P. (2015). *Rankings and Preferences: New Results in Weighted Correlation and Weighted Principal Component Analysis with Applications*. Springer.

da Costa, J. P. and Soares, C. (2005). A weighted rank measure of correlation. *Aust. N.Z. J. Stat.*, 47(4):515–529.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30.

Gama, J. and Brazdil, P. (1995). Characterization of classification algorithms. In Pinto-Ferreira, C. and Mamede, N. J., editors, *Progress in Artificial Intelligence, Proceedings of the Seventh Portuguese Conference on Artificial Intelligence*, pages 189–200. Springer-Verlag.

Järvelin, K. and Kekäläinen, J. (2002). Cumulative gain-based evaluation of IR techniques. *IEEE Transactions on Information Systems*, 20(4).

Kalousis, A. (2002). *Algorithm Selection via Meta-Learning*. PhD thesis, University of Geneva, Department of Computer Science.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of Int. Joint Conference on Articial Intelligence (IJCAI)*, volume 2, pages 1137–1145. Montreal, Canada.

Leite, R., Brazdil, P., and Vanschoren, J. (2012). Selecting classification algorithms with active testing. In *Machine Learning and Data Mining in Pattern Recognition*, pages 117–131. Springer.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Neave, H. R. and Worthington, P. L. (1992). *Distribution-Free Tests*. Routledge.

Schaffer, C. (1993). Selecting a classification method by cross-validation. *Machine Learning*, 13(1):135–143.

Sohn, S. Y. (1999). Meta analysis of classification algorithms for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1137–1144.

Spearman, C. (1904). The proof and measurement of association between two things. *American Journal of Psychology*, 15:72–101.

Sun, Q. and Pfahringer, B. (2013). Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine Learning*, 93(1):141–161.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 847–855. ACM.

Torgo, L. (1999). *Inductive Learning of Tree-Based Regression Models*. PhD thesis, Faculty of Sciences, Univ. of Porto.

van Rijn, J. N. (2016). *Massively collaborative machine learning*. PhD thesis, Leiden University.

Varma, S. and Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. *BMC Bioinformatics*, 7(1):91.