Cross-validation: the illusion of reliable performance estimation

Zoltán Prekopcsák, Tamás Henk, Csaba Gáspár-Papanek

Budapest University of Technology and Economics
Magyar tudósok körútja 2., H-1117, Budapest, Hungary
prekopcsak@tmit.bme.hu
http://prekopcsak.hu

Abstract. In data mining, we are often faced with the task of estimating model performance from training data. This estimation is supposed to express the expectation of the performance on future, previously unseen data and it is very much needed for business decisions and also for the analyst to compare different models. One of the most widely used performance estimation technique is cross-validation which has more and more misuse in these days. This paper describes common mistakes in using cross-validation that significantly obfuscate the estimations, presents several numerical examples on how misleading the estimation can be, and propose a data mining process for ensuring valid performance estimations.

1 Introduction

Data mining is a mature field of research and is increasingly used in business decision making, intelligent data-driven applications and also by other scientific fields. The two main branches of data mining are descriptive and predictive data mining [5]. Descriptive data mining only deals with understanding past data and extracting actionable knowledge, whereas predictive modeling has the goal of predicting future outcomes. There has been plenty of learning algorithms proposed for predictive modeling tasks, but theoretical results show that there is no best algorithm in general [10].

In practice, we need to compare different algorithms and decide which one to use for the task at hand. There exist some methods for estimating model performance on future data under the assumption that future data has the same distribution as the training set. The most widely used technique is cross-validation that we will introduce in the following section. Section 3 will present typical mistakes when using cross-validation and provide numerical examples on the effect of these mistakes. In Section 4, we propose a data mining process for ensuring meaningful performance estimations and summarize our findings. The examples for this paper are created with the open-source data mining suite RapidMiner [1], because it has built-in support for cross-validation, but the underlying principles and results apply for any data mining project realized with any data mining software or programming language.

2 Estimating model performance

The performance of a predictive model can be measured in many ways for different tasks. For classification, the usual performance measure is the error rate, but other measures (e.g. recall, precision, AUC) are also proposed in the data mining literature [5]. For regression, the most widely used measures are the mean absolute error and the mean squared error. In this paper, we do not specify the actual performance measure, because the ideas apply to all of them.

The simplest way for the estimation is holdout validation. It means that we split the original data set to a training set and a test set. The models learn on the training set and give prediction on the test set, so we can measure the performance on the test set. The splitting is usually done in the ratio of 70-30 and stratified sampling is used to keep distributions the same in the two sets. The main problem with simple holdout validation is that the test set is usually too small, so the variance of the estimation is high, and we are using only 70 percent of the data for training, so the estimate has a pessimistic bias.

In k-fold cross-validation, we are splitting the data into k stratified sets and run the holdout validation k times with each set being once the test set and all the others being the training set. The k performance measures are then averaged to give the overall performance estimate. There is a special case when k equals the number of examples in the whole dataset, which means that in each fold we test on only one example and it is called leave-one-out cross-validation. We also have to mention bootstrapping [4] which is a similar estimation technique that is based on probabilistic sampling, and most of our results also apply for that, but for the sake of simplicity we are only describing cross-validation examples.

There has been a lot of theoretical and experimental work on the accuracy of cross-validation. Some authors state that 10-fold cross validation is the best of the above methods [6], but propose a more difficult way for model selection [7]. There are also some probabilistic bounds on the accuracy of the leave-one-out estimation [3], but other authors state that leave-one-out validation does not converge to the correct performance as the size of the training set grows to infinity [11]. In spite of all these arguments, cross-validation is still the most widely used performance estimation technique in practice.

3 Typical mistakes with cross-validation

We are practicioning data miners ourselves and we are experiencing a growing number of cross-validation misuse at various places. As lecturers at university courses on data mining, we tutor students during their first steps with data mining. They have practical assignments with data mining tools like SPSS Modeler, SAS Enterprise Miner or RapidMiner, and they usually make a lot of mistakes during the learning process. Furthermore, as researchers, we attend conferences and read papers that state extraordinary model performances, but sometimes we fail to reproduce the results or with further investigations it turns out that the reported performances are optimistically biased. As a final note, we are

frequent contestants in data mining competitions, where model performance is usually measured after the deadline on a hidden test set, and we often see contestants being surprised by their bad final performance compared to the online leaderboard or their own estimations. All these events indicate a problem in the process design that makes performance estimation unreliable. In the following, we present some of the most likely causes for these problems.

3.1 Using global information

There are certain application areas where it is very common to have hundreds or thousands of attributes. In many data mining software suites, it is very hard to handle so many attributes, so the first step is usually some kind of feature selection to reduce the number of attributes to a few dozen. If the feature selection is performed on the whole dataset and cross-validation comes afterwards, then the selected features already incorporate some information on the test set.

Most people would consider this effect negligable, but we did an experiment with a dataset of 100 rows, 500 attributes and random binary labels. We did a feature selection based on information gain, kept the top 10 features, and in the cross-validation, we have used k-nearest neighbor which was able to learn the random labels with the average accuracy of 63%. We repeated the experiment with a dataset of 1000 rows and the accuracy was 54%, still significantly better than random prediction.

The lesson to be learnt here is that we have to put every supervised method inside the training block of the cross-validation and cannot learn global information before that.

3.2 Using future data

Let us consider a time series prediction problem when we are given a time series of length N and the task is to create a predictive model for the next value, given all the past values. This is a regression problem and is usually solved by assuming that the future value only depends on the k previous values, so we can transform the long time series segment to a table having $N - k$ rows and $k + 1$ columns (k for the previous values and one for the next value to predict).

We can use nearest neighbor algorithm with distance metric learning, which is a very popular technique nowadays [9]. For example, we can learn a Mahalanobis distance metric for the training vectors of size k and use that for the calculation of the nearest neighbor. If we use this method, we might get excellent results with cross-validation with almost zero error, but it will not generalize for future data from the same time series. The problem here is that in case of leave-one-out validation, we have all past and future data in the training set. When we try to predict the test example t_i from the vector $t_{i-k} \dots t_{i-1}$, we have a training vector $t_{i-k+1} \dots t_i$ with label t_{i+1}, so the correct answer for t_i is included in the training set. Some algorithms, like nearest neighbor with distance metric learning, can explicitly learn this rule, but other algorithms can also use this information in a hidden way which produces an unreliable performance estimate.

A related problem is when the dataset has some duplicated rows due to errors in joining tables or other preprocessing steps. When using leave-one-out, the simple nearest neighbor algorithm will always have zero error for these rows.

3.3 Picking the best model

On small and medium sized datasets, even correctly used cross-validation can be very misleading when run multiple times. The variance of the performance estimate is quite high in these cases, so if we run the estimation multiple times with different model parameters and pick the best, we can get a result that is far from the real performance.

Let us introduce a simple experiment that will make this problem easier to understand. We have a binary classification task with 100 training examples which is quite small but not very unusual in practice. We are considering 100 different models from the family of COIN models. The COIN model is something that everyone has used before: toss a coin and vote for positive class if heads is up and vote for negative class when tails is up. It needs no explanation that the expected error rate will be 50%.

So we take 100 different coins and measure their accuracy on all the 100 training examples (as we would do in the case of cross-validation). From the 100 coins, we choose the one with the lowest error rate. If the coins are fair coins, then we can be almost sure (with 95% confidence) that our best coin produces an error rate at most 40%, so we will pick that coin and expect it to behave the same on future data. This argument might seem a bit unnatural, but we can get similar results with bigger training sets and higher number of COIN models, so it shows how the *picking the best* approach might fail. A more statistical and precise argument can be found in [8].

We have seen RapidMiner processes which did an iteration over thousands of differently parameterized models, selected the best, reported great error rates, but on a hidden test set they finally performed 5-10% worse than expected. This problem often appears in research papers and data mining competitions and there is no straightforward solution to avoid it. However, in the following section, we will try to provide some rules of thumb.

4 Recommended approach

The first step in avoiding the above mistakes is that we should be aware of them and treat cross-validation results with a bit of mistrust. Second, we can check for specific errors in the process design. Here we provide a short checklist for the mistakes that are easy to recognize.

- ☐ Do I use the label attribute before cross-validation?
- ☐ Do I get a different preprocessing result if I remove the label attribute from the dataset?

☐ Is there any explicit dependency between distinct rows?

☐ Are there any duplicate rows in the dataset?

If your answer is *no* for the above questions then the cross-validation is well defended from the first two mistakes, but not from *picking the best*. If you have a very large dataset, then you can select a holdout dataset for final testing that you will test only once after the algorithm is ready and all the parameters are set to optimal. Be aware that if you see a bad result and start running new tests on that dataset then you are cheating again and the performance estimate will be unreliable.

In case of small or medium sized datasets, you need a bit of statistics to make sure that your results are significant and not just due to the variance of the estimation. You can find the details in the paper of Salzberg [8], but there is a simple trick that might solve the problem in most of the cases.

In a typical data mining process, you want to optimize the parameters of a specific algorithm and find the parameters with the best performance. If you take many possible parameter settings, run a cross-validation with each of them and report the best result, then you commit the fault of *picking the best*. Instead, you can create a single cross-validation and in each training phase you take the current training set, optimize the parameters on that dataset, and use the best parameters for the test. Of course, in the optimization part, you will always pick the best model, but you will test it only once on a previously unseen test set. You can see the difference in terms of RapidMiner process trees on Figure 1.

There are several drawbacks of this approach. First, you will not receive the best parameter setting at the end, but the best parameters in each of the cross-validation folds. Second, it will multiply the runtime with the number of folds done in the outer cross-validation, but it might be acceptable in the case of small datasets. On the other hand, you will receive a reliable performance estimate that you can trust to work the same on future data.

The recommended approach presented above is implemented as a RapidMiner process and it can be downloaded from our website [2].

5 Summary

We addressed the problem of producing a reliable performance estimation for classification and regression tasks. We described different performance estimation methods and presented common mistakes in using them through examples with cross-validation. Finally, we proposed a recommended approach with a checklist and a RapidMiner process template to help data miners produce valid performance estimations. We welcome any comments in email concerning the topic and the examples presented in this paper.

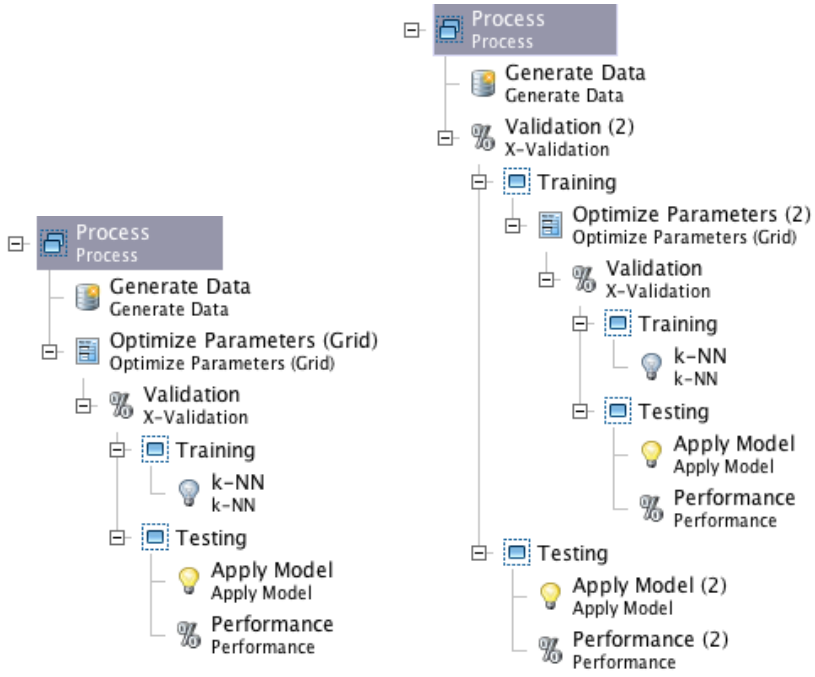

Fig. 1: RapidMiner process with *picking the best* (a) and the corrected process (b)

References

1. RapidMiner homepage, `http://www.rapidminer.com`
2. Supporting website with downloadable process template, `http://prekopcsak.hu`
3. Devroye, L. and Györfi, L. and Lugosi, G.: A probabilistic theory of pattern recognition. Springer (1996)
4. Efron, B. and Tibshirani, R.: An introduction to the bootstrap (1993)
5. Han, J. and Kamber, M.: Data mining: concepts and techniques. (2006)
6. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. Proceedings of the International Joint Conference on Artificial Intelligence, 1137–1145 (1995)
7. Provost, F. and Fawcett, T. and Kohavi, R.: The case against accuracy estimation for comparing induction algorithms. Proceedings of the Fifteenth International Conference on Machine Learning (1998)
8. Salzberg, S.L.: On comparing classifiers: Pitfalls to avoid and a recommended approach. Data Mining and Knowledge Discovery, 1 (3), 317–328 (1997)
9. Weinberger, K.Q. and Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. Journal of Machine Learning Research, 10, 207–244 (2009)
10. Wolpert, D.H. and Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1 (1), 67–82 (1997)
11. Zhang, P.: On the distributional properties of model selection criteria. Journal of the American Statistical Association, 87 (419), 732–737 (1992)