Table of Contents

[Section – Accidental Contamination 1](#_Toc166570963)

[Situation 1:  Contamination through Normalization (15 mins) 2](#_Toc166570964)

[Task 1: The cross-validated test errors with a normalization before the CV (“wrong”) or inside (“nested”). 4](#_Toc166570965)

[Situation 2:  Contamination through Feature Selection (30 mins) 6](#_Toc166570966)

[Task 2: Using evolutionary search for optimized feature selection. 6](#_Toc166570967)

[Key Takeaways 8](#_Toc166570968)

# 

**INSTRUCTIONS:**

* Run the processes (provided) on the specified datasets for each task.
* Updated this document your results in the tables provided
* Enter your Name and CWID in space provided.
* Submit this document by **May 19th, 2024 @ 12 am.**
* **No Extensions will be granted.**
* **DATASETS and PROCESSES available on CANVAS**
  + Navigate to FILES > GRAD STUDENT FINAL PROJECTS > DATASETS
  + Navigate to FILES > GRAD STUDENT FINAL PROJECTS > PROCESSES

**NAME: Yead Rahman**

**CWID: 10406508**

# Section – Accidental Contamination

In the last exercise we learned to calculate the predictive accuracy of a model by applying the model on data points it has not previously been trained on and to compare the model’s predictions with the known true result for this test set. If you repeat these multiple times for non-overlapping test cases, just like you do in case of the cross-validation, you end up with a reliable estimation about how well your model will perform on new cases.

This approach in general is a good basis for doing model selection, i.e., answering the question which type of model (think: “random forest or K-NN?”) is going to perform best on my data in the future.

**However,** the problem is that it is still very easy to leak information about the testing data into the training data if you perform a cross-validation in the wrong way.

We call this phenomenon **contamination** of the training data. Contamination provides access to information the machine learning method should not have access to during training.  With contamination, the model will perform better than you expect it to perform when compared to situations when this information is not available.  This is exactly why accidental contamination leads to an over-optimistic estimation about how well the model will perform in the future.

This effect can be as drastic as the example of the k-Nearest Neighbor classifier (refer section 1 in previous exercise), where the training error (all information is available in this case) was 0% while the testing error was ~50% and hence no better than randomly guessing the class of the data points.  As we will see below, the same effect happens if you leak information about the test data into the training phase.  The test error becomes effectively a training error then and is lower than you would expect without the contamination. Therefore, all the efforts you did by using a cross-validation to avoid leakage of information is pointless if you are doing it wrong.

Let’s talk show a few typical situations where accidental contamination of training data by leaking information about the test data can happen.

# Situation 1:  Contamination through Normalization (15 mins)

Let’s start with a very common situation- you would like to normalize data so that all columns have a similar range and no column overshadows others. This is particularly important before using any similarity-based models, e.g., k-Nearest Neighbors.

This seems to be a simple task: just normalize the data and then train the model and validate it with a cross-validation. This is how this would look like in RapidMiner:

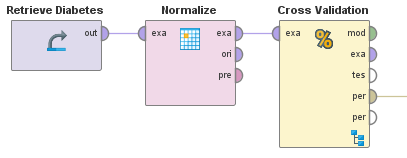


Figure 1: This RapidMiner Studio process simply performs a normalization (z-transformation) on the data before the model is validated.

**Discussion:** **Do you think the above process would cause any bias in the model being evaluated?**

This is an **incorrect approach**, as this will lead to a wrong error estimation. If you normalize the data before the cross-validation, you **leak information** about the distribution of the test data into the way you normalize the training data. Although you are not using the test data for training the model, you nevertheless instill some information about it into the training data. This is exactly why we call this effect contamination.

What you should do instead is perform the normalization inside of the cross-validation and only on the training data. **You then take the information you gathered about the training data and use it for the transformation of the test data.** This is where the RapidMiner Studio visual interface comes in handy since you can easily see the correct setup:

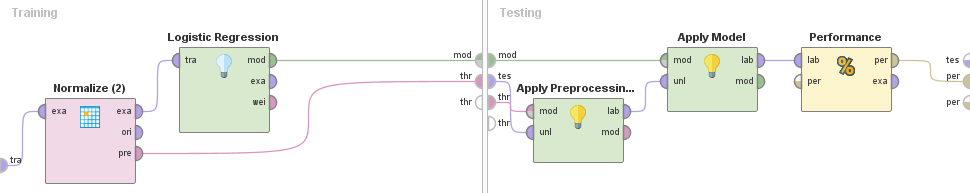


Figure 2: This is how you properly evaluate the impact of the normalization on the model accuracy. The parameters derived from the normalization on the training data are delivered into the test phase and used there instead of the other way around.

Note that we are feeding data directly into the cross-validation operator and performing all necessary steps inside. The first thing you do in the training phase on the left is to normalize only the training data. The transformed data is then delivered to the machine learning method (Logistic Regression in this example). Both the predictive model and the transformation parameters from the normalization are then delivered into the testing phase on the right. Here we first transform the test data based on the distributions found in the training data before we apply the predictive model on the transformed test data. Finally, we calculate the error as usual.

This table summarizes the cross-validated test errors, once with the normalization performed before the cross-validation like in Figure 1 and the second with the normalization inside of the cross-validation like in Figure 2. It clearly shows the effect on the error if you validate the model without measuring the impact of the data pre-processing itself:

## Task 1: The cross-validated test errors with a normalization before the CV (“wrong”) or inside (“nested”).

Processes required:

* 07 Normalization before cross-validation, and
* 08 Normalization inside cross-validation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Sets | Random Forest | | 5-NN | |
| Wrong (07) | Nested (08) | Wrong (07) | Nested (08) |
| Diabetes |  |  |  |  |
| Ionosphere |  |  |  |  |
| Sonar |  |  |  |  |
| Wine |  |  |  |  |
| Average Dev |  |  |  |  |

**Observation:** If the normalization is done before the cross-validation, the calculated error is overly optimistic in all cases and the data scientist would run into a negative surprise when going into production. On average, the actual error is 3.9% higher for Logistic Regression and 3.4% higher for a k-Nearest Neighbors with k=5. The difference is 0% for Random Forest simply because the models do not change at all if the data is normalized or not. In general, you can see that the effect is not as drastic as just using the training error, but still there are differences of more than 8% in some of the cases, higher than most people would expect. This is caused by only validating the model itself and not the impact of the data pre-processing.

A final thing which is important to notice: the model will perform roughly the same way in production, no matter if you perform the normalization outside or inside the cross-validation. When building a production model, you would typically use the complete dataset anyway and hence also apply the normalization on the complete data. But the performance of the model will be the lower one, pretty much in the range of the one shown in the column “Nested”. So, the correct validation is not helping you create better models, instead it is telling you the truth about how well (or poorly) the model will work in production without letting you run into a catastrophic failure later on.

# Situation 2:  Contamination through Feature Selection (30 mins)

As a last example, let’s look at another pre-processing step which is frequently performed for optimizing the accuracy of machine learning models, namely feature selection. The goal is to select an optimal subset of features or data columns used by the model. In theory, many models can make this selection themselves, but in reality, irrelevant columns can throw models off and it’s better to keep only those columns which provide meaningful information.

Data scientists could manually select a subset of features and evaluate the model’s accuracy by calculating a cross-validated test error using only the data of the reduced feature set. Since the number of combinations grows exponentially with the number of columns in the input data, this isn’t often a feasible approach; the reason why automated optimization techniques are widely used. But just as in the case of parameter optimization, picking the optimal feature set is basically an extension of training the predictive model, and it needs to be properly validated as well.

Of course, you could use an extra hold-out set for evaluating any built model at the end. But as before, you end up with the disadvantage of only using a single validation set. The better alternative is to **nest the automatic feature selection** (using an inner cross-validation to guide the search for the optimal set of features) into an outer cross-validation again, just as we did in case of the parameter optimization.

The correct setup starts with an outer cross-validation just as described above where we used an outer cross-validation to evaluate the performance of the parameter optimization. The inner setup of this cross-validation looks slightly different though, as follows:

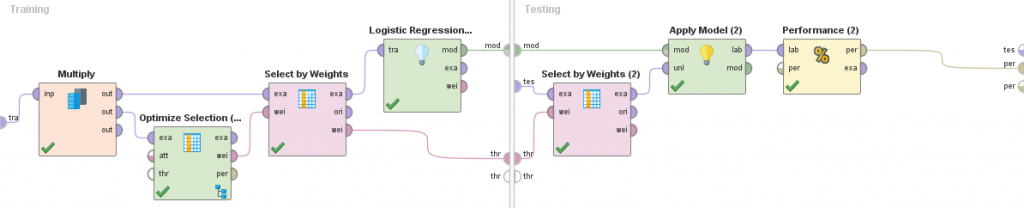


Figure 3: An automatic evolutionary feature selection is used on the training data of the outer cross validation. The optimal feature set is selected before the model is trained and delivered to the test phase where it is applied again before prediction.

## Task 2: Using evolutionary search for optimized feature selection.

Processes Required:

* 15 Feature selection Logistic Regression without outer validation
* 16 Feature selection Logistic Regression with outer validation
* 19 Feature selection k-NN without outer validation
* 20 Feature selection k-NN with outer validation

Document your results for the wrong version of the cross-validation where you don’t properly validate the effect of the feature selection and only use the performance value of the optimal feature set as found during the optimization runs. The value in the columns “Nested” are determined with the correct approach using an outer cross-validation as depicted above:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Sets | Logistic Regression | | k-NN | |
| Wrong (15) | Nested (16) | Wrong (19) | Nested (20) |
| Diabetes |  |  |  |  |
| Ionosphere |  |  |  |  |
| Sonar |  |  |  |  |
| Wine |  |  |  |  |
| Average Dev |  |  |  |  |

**Observation:** Here you see that the amount by which your expectations would be off is on average up to 4.2%. The largest deviation was even as high as 11% (5-NN on Sonar) – which is even higher than the maximum of 8% we saw in the normalization case above. Think about this: your error in production would be double (22% vs. 11%) what you expected; just because you did not the right kind of validation which considers the effect of the data pre-processing. This can be avoided.

It is worth pointing out that the calculated average and maximum deviations greatly **depend on the data sets and model types you use**. We have seen above that the effect of the wrong validation for normalization was zero for Random Forest since this model type does not care about the different scales of the data columns. On the other hand, the effect was larger for feature selection using a Random Forest. And this is the problem: you never know in advance how large the effect will really be. **The only way to be prepared about what is coming in the future is to properly validate the effects of the model itself as well as all other optimizations with a correct form of modular, nested cross-validations.**

# Key Takeaways

* You can *contaminate* your training data set by applying data transformations before the cross-validation which leads to information leakage about the test data into the complete data set.
* This *“bigger” machine learning needs to be evaluated* just like the actual machine learning method itself.
* Most data science products do *not* allow you to perform the model validation in a correct way, i.e. considering the effect of pre-processing or model optimizations.
* We have seen three examples showing *how large the effect on model validation* can be when done incorrectly.
* For an un-validated *normalization*, the effect is on average up to 3.9% with a maximum of 8%.
* For an un-validated *feature selection*, the effect is on average up to 4.2% with a maximum of 11%.
* It is important to understand that these effects can be *even higher* for different data sets or different forms of un-validated pre-processing.
* The only way to avoid surprises when going into production is to *properly validate before*.