Cross Validation

Duke Course Notes

Cynthia Rudin

Cross validation (CV) means multiple things. Here are the contexts in which people use it:

- "We evaluated the algorithm by 10 fold cross-validation"
- "The parameters of the algorithm were tuned by 10-fold cross-validation" (part of nested cross-validation)

It is confusing that the term CV is used both for evaluation and tuning. It means different things in these two contexts. Nested CV puts them together.

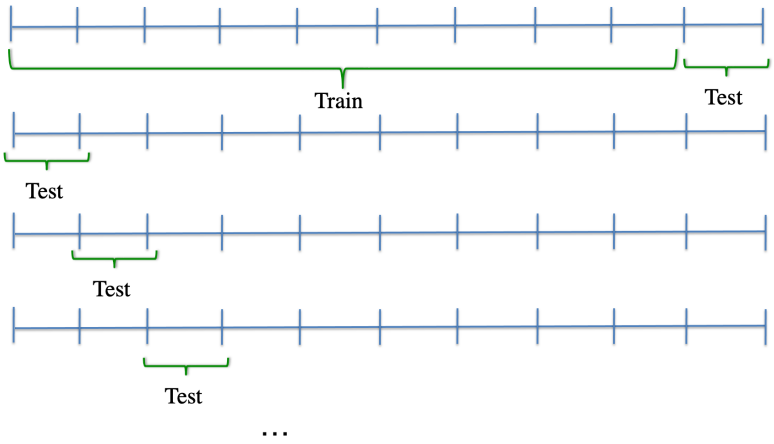
CV for Evaluation

CV is the most popular way to evaluate a machine learning algorithm on a dataset. To do it, you will need a dataset, an algorithm, and an evaluation measure. The evaluation measure might be the squared error between the predictions and the truth, or it might be misclassification error.

To do it:

- Divide the data into 10 approximately-equally-sized"folds"
- Train the algorithm on 9 folds, compute the evaluation measure on the last fold.
- Repeat this 10 times, using each fold in turn as the test fold.
- Report the mean and standard deviation of the evaluation measure over the 10 folds.

Below we illustrate the rotation of the test fold.



If you are comparing algorithms, you would report the algorithm that performed the best, which is the one with the best average out-of-sample performance across the 10 test folds. Typically, you would compute 2-sample paired t-tests (across the performance on the 10 test folds) to determine whether the best algorithm is significantly better than the second best algorithm, the third best algorithm, etc.

For instance, I might report a result like this:

Alg 1	Alg 2	Alg 3	Alg 4
.87 ±.01	.85 ±.04	.81±.03	*.86 ±.01*

This means that Algorithm 1 is the **best**, and it was significantly better than Algorithm 2 according to a paired t-test across folds. It was also better than Algorithm 3 according to a paired t-test. It was not statistically significantly better than Algorithm 4, which is why that one is in *italics*.

If you are comparing algorithms, make sure to use the same folds for each algorithm. Otherwise it is not a controlled experiment and one algorithm might get lucky because you gave it an easier dataset.

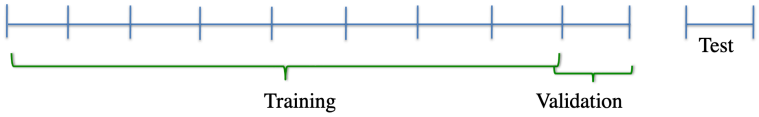
CV for Parameter Tuning

How do you tune the regularization parameter for an algorithm? The regularization term doesn't involve the training data, it only helps you generalize to the test set. Thus, tuning it requires something like test data (but of course we aren't allowed to touch the real test data for parameter tuning). That's where CV for

parameter tuning comes in. We want to use this only for parameters like the regularization parameter, that aren't tuned on the training set during minimization.

In brief, the procedure of CV for parameter tuning involves setting aside part of the training set for validation, and using that as an internal test set. Whichever parameter value performs best on the internal tests wins.

Note that the validation data is part of the training set, so we are allowed to use it to tune parameters (we must never look at the test set to tune parameters). Below we illustrate the validation set.

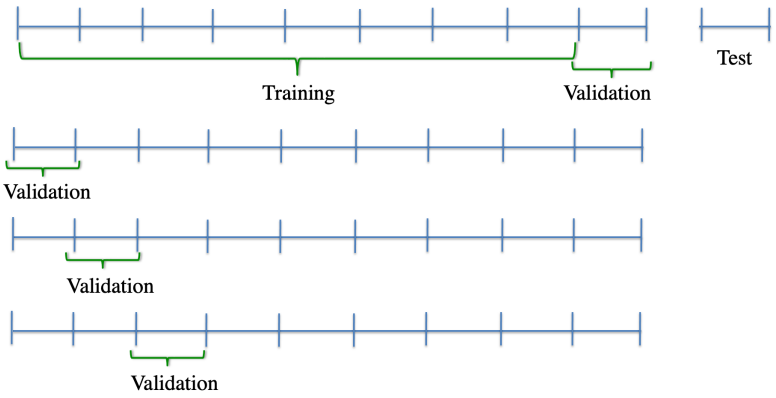


This procedure is computationally expensive, because we will train the algorithm many times for each possible value of the parameter. So, as an example, we'll call the parameter κ and we allow it to take one of four possible values: 1, 10, 100, 1000, or 10000.

The procedure is as follows:

- Set aside the test set (if you have one yet).
- Reserve a validation set from the training set.
- Train the algorithm on the rest of the training set for each possible κ value, evaluate on the validation set. Rotate the validation fold and repeat.
- Report the mean of the evaluation measure for each κ value over the validation folds. Choose the best κ value.
- Train on the full training set (training + validation) with best κ, use the model on the test set.

The illustration below shows the rotation of the validation fold.



If κ=100 had the best performance averaged over the validation folds, we train the model on the full training set with κ=100. We use that model on the test set.

In practice, this computation becomes expensive so people often choose fewer folds and only a small number of choices for κ. Usually we tune only the regularization parameter this way, because if you have two parameters, we would need to choose possible values for the combination of these variables (a grid search, among the grid of values for these variables).

Nested CV

Nested CV combines both types of CV. It uses CV for evaluation as the the outer loop, and CV for tuning parameters as an inner loop. That is, nested CV *evaluates an algorithm, including its parameter tuning.*

The outer loop rotates over the test set. The inner loop rotates over the validation set.

It is entirely possible that nested CV will choose a different "best" value of parameter κ for each test set. Remember, here, the parameter tuning is part of the algorithm, and you are evaluating the whole algorithm, including all of the parameter tuning (parameters tuned by minimization of the loss and those tuned in any other way). So the algorithm you are evaluating now includes the CV for tuning parameters.

So what is the final model?

People often get confused and ask what the "final" model is, given that each rotation of nested CV could give a different κ.

Remember that nested CV evaluates an algorithm, including tuning. If you just want to know what the final model is, you would just use the procedure above for tuning.

Since here you are not interested in evaluation, you would not set aside part of your data for testing. Instead, you would use the full dataset for training. (As discussed above, you would create a validation set, which rotates within the training set. We choose the "best" parameter value for κ based on the validation folds. Then we train on the full training set, using the best value for parameter κ.)

CV tips

If you are working with an imbalanced dataset, it is common to take the minority class data points and split them evenly between the test folds.

If training the model is really expensive, it is easier not to tune the regularization parameter, so that you only need to do CV for evaluation. In that case, it is helpful if the algorithm has some guidelines on how to tune that regularization parameter. (The GOSDT algorithm for decision trees is an example of an algorithm with this kind of guideline.) Also, in the case that training is expensive, you might want to do 5-fold CV instead of 10-fold CV so you only need to train it 5 times. Sometimes people even do 3-fold CV.

An interesting issue with CV for parameter tuning is that a parameter that performs best for a fraction (say 9/10ths) of the training set may not actually be the best parameter for the whole training set. But this is another issue for another day.