Data Analysis and Model Classification
Guidesheet I-3: Model selection and nested cross-validation

Fumiaki Iwane   Ping-Keng Jao   Bastien Orset   Julien Rechenmann
Ricardo Chavarriaga   José del R. Millán

This guidesheet should help you understand better how to select the best features for your classifier and crucially, how to correctly estimate the performance of a model (classifier) on an unseen and independent dataset (`features`). In the previous guidesheet, we put into practice the theory behind LDA/QDA classifiers, and also had a brief introduction to model selection. We saw that a model has a certain number of parameters that are automatically optimized using the training set. However, you might have noticed that we had an additional parameter which was not automatically tuned: the choice of the classifier type (*diaglinear*, *diagquadratic*, *linear* or *quadratic*). We call it a *hyperparameter*, and the way it is chosen is called *hyperparameter optimisation* or *model selection*. Other hyperparameters are for example the specific features and the total number of features used. We will do the feature selection by ranking features according to a metric and keeping only the first N_{sel} best features.

Cross-validation for hyperparameter optimization

We will use cross-validation for model selection. This means that each classifier trained with a different N_{sel} will be considered as a different *model* and our goal is to select the best one. For this, we will analyse how the class-averaged classification error evolves with N_{sel}. To rank the features, use the `rankfeat()` function with the *Fisher* method (you need to download the function from moodle) . This function takes features and labels as inputs, and outputs a list of feature ordered by discriminative power.[1] In each fold, test your classifier with the single best feature, save the training and test error. Repeat these steps with the two best features, save the training and test error, and so on. With every feature that you add, you change the model you are testing. This means that you will end up with a matrix of test errors of size $N \times K$, where N is the number of models you tried out and K is the number of folds. The decision about which number of features is good for your model will be based on the **average of all folds**.

Remember the golden rule: the cross-validation test samples must be independent from the training set and not taken into account for model fitting. You must use **only** the training folds (9/10 of the folds for 10-fold cross-validation). Otherwise, you will **overfit**![2]

Design questions

- How would you implement your cross-validation? Write the implementation on a sheet of paper in pseudo-code.

- What is the role of your train set and test set?

- Where will you put the feature selection? *Inside* or *outside* the cross-validation loop.

Hands on

- Implement the cross-validation with *Fisher*-based feature selection and save both training and test error for each fold and each number of features. Use a *diaglinear* classifier.

---

[1] That means that if the first three values are 23,32,1, the most discriminative feature is feature #23, followed by feature #32 and feature #1

[2] https://www.youtube.com/watch?v=DQWI1kvmwRg or alternatively https://en.wikipedia.org/wiki/Overfitting

- Plot training and test error for every number of features. You can use `plot(err,'b');` to plot the error for every cross-validation fold (for `'b'`, rows = number of features, columns = folds) and then `hold on; plot(mean(err,2),'b','LineWidth',2);` to plot the mean error across folds. Do the same for the test error with another color.

- Do the two curves behave differently?

- Based on these curves, what is the optimal number of features?

- We picked the final model with the best performance. This is a cherry-picking process which means risky because we simply picked the best one with only one-time comparison without any theoretical support. To illustrate the problem in an easy way, you can implement a 4-fold cross-validation for a random classifier.[3] The theoretical error is 50%. How different the mean error across folds is from it? Repeat 1,000 times the cross-validation and save the mean test error for each repetition. Among these 1,000 values, what is the minimum test error? what do you conclude?

Nested cross-validation for performance estimation

The error obtained with **simple** cross-validation is a **biased** estimation of the true error. This means, that although our **simple** cross-validation loop allows us to choose the best hyperparameters of our model (e.g. number of features), we are unsure about the true performance of our model, and we would need a new unseen dataset to test and to accurately estimate the test error. To do this optimally on a small dataset, we will do an *outer* cross-validation to test the performance of the classifier while the model selection will be performed in an *inner* cross-validation inside of each fold of the *outer* cross-validation.
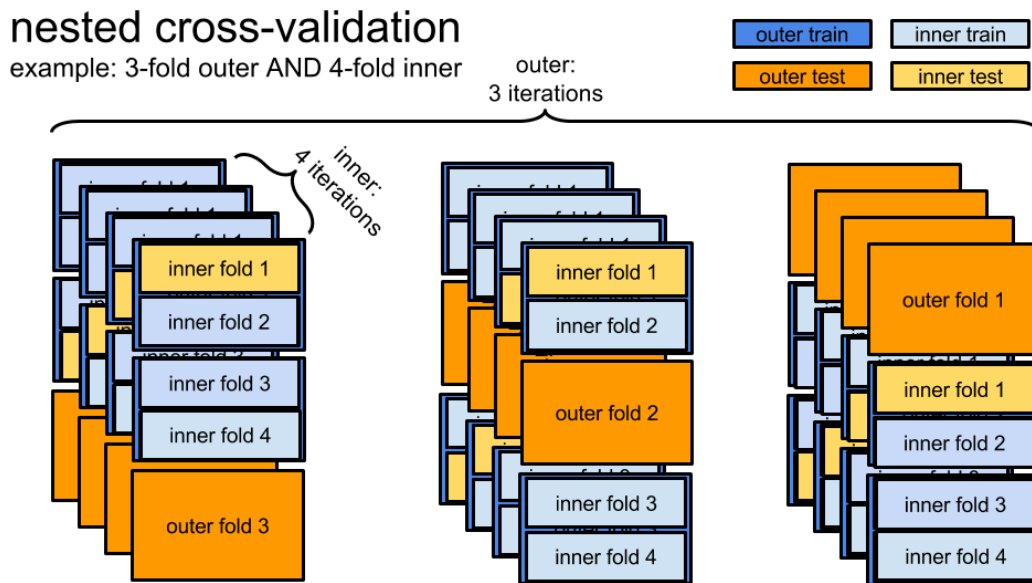

Figure 1: Graphical representation of nested cross-validation: three outer folds and four inner folds. Note that you need to have labels for all the data that you are using for cross-validation.

From now on, we will call the error computed on a test fold of the outer cross-validation *test* error, while the "test" error computed for each hyperparameter value during the *inner* cross-validation will be called the *validation* error. The *validation* error, which helps us assess the hyperparameters, can be computed as many times as we want (once per model and inner fold). Therefore it will be saved in a $N \times K_{in}$ matrix. Where N is the number of models that you try and K_{in} is the number of inner folds. The *test* error can be computed only once per outer fold, to assess the final performance of the model. Therefore it will be saved in a $1 \times K_{out}$ vector, where K_{out} is the number of outer loop folds.

---

[3]A random classifier is very stupid: to classify a sample, it simply flips a coin to decide on whether the output will be 0 or 1.

Design questions

- How would you implement your inner and outer cross validation loops ? Write the implementation on a sheet of paper in pseudo-code.

- What is the role of your inner train set, inner test set, outer train set, outer test set?

- Where will you put the feature selection? Inside the inner cross-validation? Inside the outer cross-validation? Outside the outer cross-validation? Or any combination of the preceding?

Hands on

- Implement the nested cross-validation with *Fisher*-based feature selection. In each *outer* cross-validation fold, select the best model. The best model is the one that has the lowest mean validation error across inner cross-validation folds. Save the optimal hyperparameter (N_{sel}) and the corresponding minimum validation error (*optimal validation* error) and associated training error (*optimal training* error). Use a *diaglinear* classifier.

- How does the *hyperparameter* differ across folds? Compare the median error of the *hyperparameters* to the *hyperparameter* found in the simple cross-validation of the last section?

- Make a `boxplot` of the distribution of *optimal training*,*optimal validation* and *test* error. How do they differ and why?

- Do the same but vary the classifier type as a hyperparameter on top of the number of features. How does the classifier type change from one fold to another?

- Does your final model change?

- Test on Kaggle your final model.