There are a number of ways to validate second level models (meta-models). In this reading material you will find a description for the most popular ones. If not specified, we assume that the data does not have a time component. We also assume we already validated and fixed hyperparameters for the first level models (models).

a) Simple holdout scheme

Split train data into three parts: partA and partB and partC.

Fit N diverse models on partA, predict for partB, partC, test\_data getting meta-features partB\_meta, partC\_meta and test\_meta respectively.

Fit a metamodel to a partB\_meta while validating its hyperparameters on partC\_meta.

When the metamodel is validated, fit it to [partB\_meta, partC\_meta] and predict for test\_meta.

b) Meta holdout scheme with OOF meta-features

Split train data into K folds. Iterate though each fold: retrain N diverse models on all folds except current fold, predict for the current fold. After this step for each object in train\_data we will have N meta-features (also known as out-of-fold predictions, OOF). Let's call them train\_meta.

Fit models to whole train data and predict for test data. Let's call these features test\_meta.

Split train\_meta into two parts: train\_metaA and train\_metaB. Fit a meta-model to train\_metaA while validating its hyperparameters on train\_metaB.

When the meta-model is validated, fit it to train\_meta and predict for test\_meta.

c) Meta KFold scheme with OOF meta-features

Obtain OOF predictions train\_meta and test metafeatures test\_meta using b.1 and b.2.

Use KFold scheme on train\_meta to validate hyperparameters for meta-model. A common practice to fix seed for this KFold to be the same as seed for KFold used to get OOF predictions.

When the meta-model is validated, fit it to train\_meta and predict for test\_meta.

d) Holdout scheme with OOF meta-features

Split train data into two parts: partA and partB.

Split partA into K folds. Iterate though each fold: retrain N diverse models on all folds except current fold, predict for the current fold. After this step for each object in partA we will have N meta-features (also known as out-of-fold predictions, OOF). Let's call them partA\_meta.

Fit models to whole partA and predict for partB and test\_data, getting partB\_meta and test\_meta respectively.

Fit a meta-model to a partA\_meta, using partB\_meta to validate its hyperparameters.

When the meta-model is validated basically do 2. and 3. without dividing train\_data into parts and then train a meta-model. That is, first get out-of-fold predictions train\_meta for the train\_data using models. Then train models on train\_data, predict for test\_data, getting test\_meta. Train meta-model on the train\_meta and predict for test\_meta.

e) KFold scheme with OOF meta-features

To validate the model we basically do d.1 -- d.4 but we divide train data into parts partA and partB M times using KFold strategy with M folds.

When the meta-model is validated do d.5.

Validation in presence of time component

f) KFold scheme in time series

In time-series task we usually have a fixed period of time we are asked to predict. Like day, week, month or arbitrary period with duration of T.

Split the train data into chunks of duration T. Select first M chunks.

Fit N diverse models on those M chunks and predict for the chunk M+1. Then fit those models on first M+1 chunks and predict for chunk M+2 and so on, until you hit the end. After that use all train data to fit models and get predictions for test. Now we will have meta-features for the chunks starting from number M+1 as well as meta-features for the test.

Now we can use meta-features from first K chunks [M+1,M+2,..,M+K] to fit level 2 models and validate them on chunk M+K+1. Essentially we are back to step 1. with the lesser amount of chunks and meta-features instead of features.

g) KFold scheme in time series with limited amount of data

We may often encounter a situation, where scheme f) is not applicable, especially with limited amount of data. For example, when we have only years 2014, 2015, 2016 in train and we need to predict for a whole year 2017 in test. In such cases scheme c) could be of help, but with one constraint: KFold split should be done with the respect to the time component. For example, in case of data with several years we would treat each year as a fold.

The data leakage might be due to the way the test set was sampled. Instead of constructing the set out of random, a few pictures out of the same class are chosen (should be the reason of the clustering effect) and combined into pair to save time. As a result of no free lunch theorem, this process causes data leakage and provides extra information to the data set.