Hyperparameter optimization

Cross validation

see Ch. 06 in book "Python Machine Learning" by Raschka & Mirjalili

Overview

- Hyperparameters
- The holdout method (“validation” partition)
- *K*-fold cross validation
- Examples of *k*-fold cross-validation
- Interpretation of learning & validation curves. How detect:
  - Underfitting (High bias)
  - Overfitting (High variance)

Hyperparameters

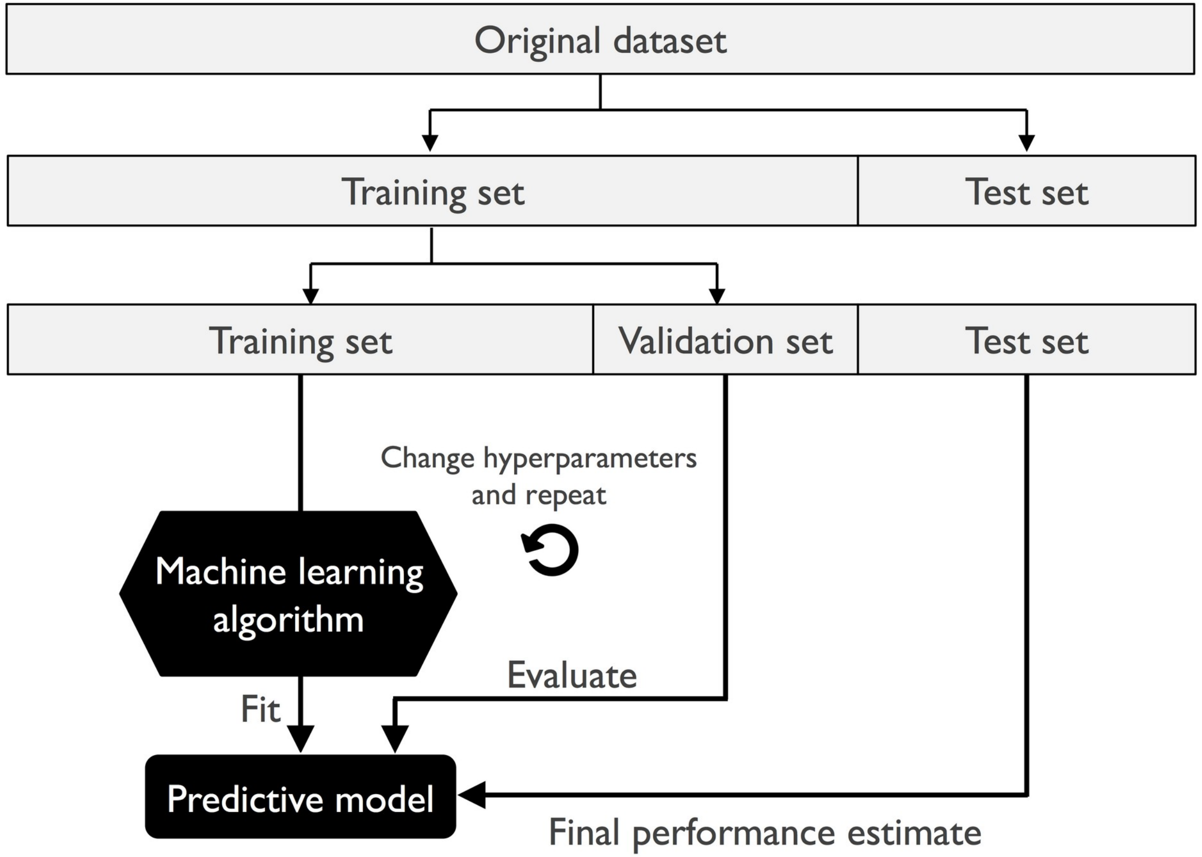
- Last time we defined **hyperparameters**
- Can anyone remember the definition?
- Hyperparameters are **non-trainable** model parameters that have an affect on a models performance
  - Examples include max depth in a decision tree, number of decision trees in a random forest model, any regularization technique, etc.
  - No clear distinction between what is considered a “hyperparameter”, and what is considered a different model architecture entirely.
- Hyperparameter optimization is also referred to as *model tuning*

The holdout method ("validation" partition)

- Last time talked about the need for a third partition when splitting the dataset
- Does anybody remember **why** we wanted the additional "validation" partition?
  - If we use the test-partition to adjust a models hyperparameters, we are indirectly training the model on the test-partition.
  - Thus the test-partition is **no longer "unseen"** by the model and we risk **overfitting** the model **on the test-set** if we also use the test-partition for final evaluation of the model
- Despite this, many still use the test-partition for both model tuning and final model evaluation. This is **bad practice**

The holdout method ("validation" partition)

- The method of splitting the full dataset into training partition and an evaluation partition is referred to as the *hold out method*



Stop here for holdout validation

When splitting training set again it's called **holdout cross-validation**

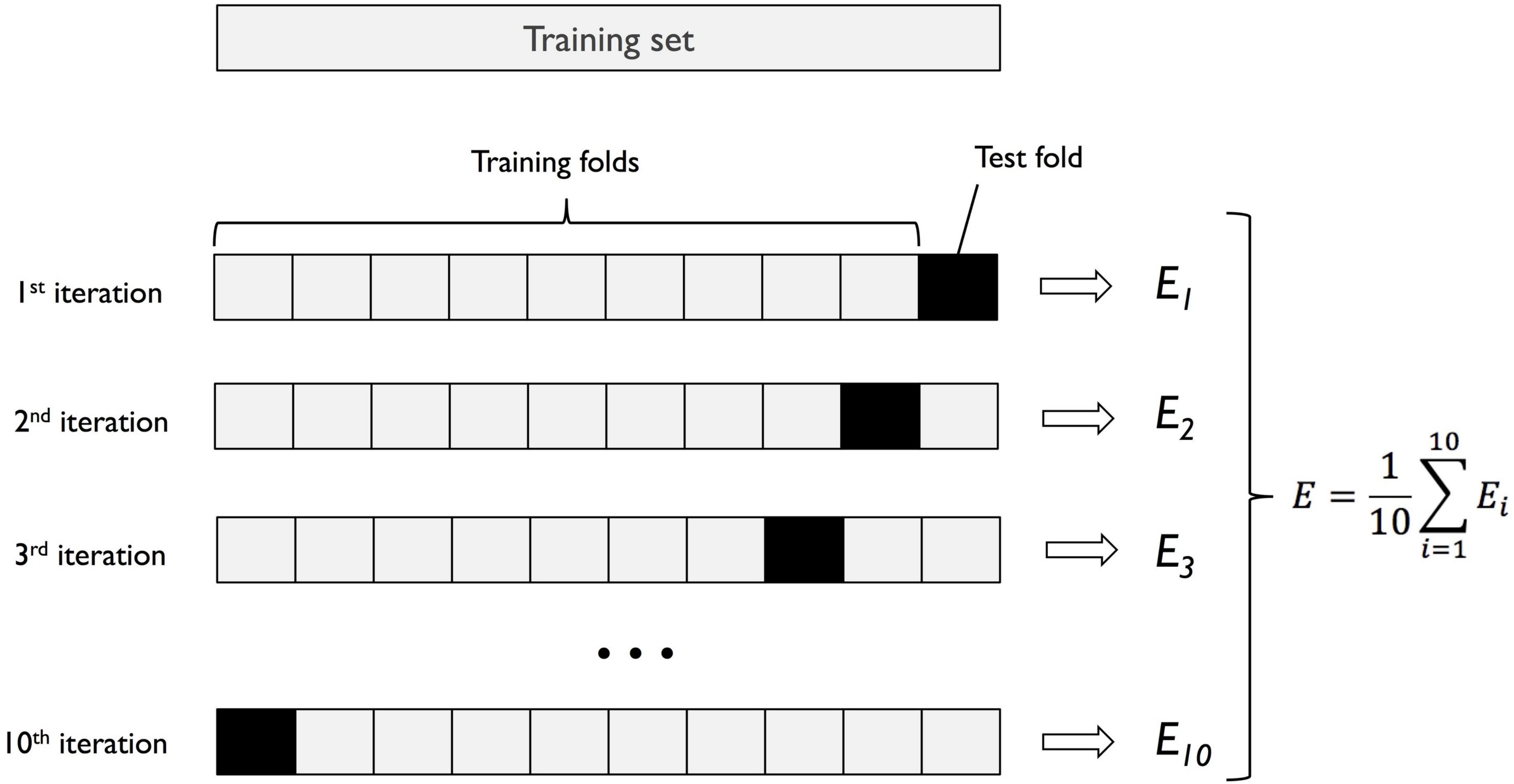
The holdout method (“validation” partition)

- The big disadvantage with holdout cross-validation is that the **performance** of particular model hyperparameter configurations could be **highly sensitive** to **how** the **training set is partitioned** into train and validation sets.
- This leads us to the second variation of cross-validation ***k*-fold cross-validation**

K-fold cross-validation (CV)

- For *k*-fold CV we still perform the initial holdout split into training set and test set
- We then partition the training set into *k* folds
- *K*-fold cross-validation is then performed in *k* iterations. For each iteration (*i*):
  - Set the validation partition equal to fold *i*
  - Reset the models trainable parameters
  - Train the model on the remaining $k - 1$ folds
  - Evaluate the model on the validation partition and store the accuracy (or other metric)
- After all iterations are complete → Compute average accuracy
- Additional plus is that we can compute the uncertainty of the models performance

K-fold cross-validation (CV)



K-fold cross-validation (CV)

- The higher we set *k*
  - Model validation becomes **more thorough**
  - Model validation becomes more **computationally expensive**
- Empirical evidence shows that *k*=10 is a good starting point
  - A compromise between bias & model variance and computational cost [see book page 198]
- If working with a relatively **small dataset *k* > 10**
- If working with a relatively **large dataset *k* < 10**

K-fold cross-validation (CV)

- Important to maintain distributions of dataset in folds
  - *StratifiedKFold* from *sklearn.model_selection*
  - *cross_val_score* from *sklearn.model_selection*
- Leave-one-out-cross-validation (LOOCV)
  - Special case of k-fold CV where one sets $k = n$ (number of total samples)
- *cross_val_score* has a parameter *n_jobs* which specifies the number of CPU cores used for CV. Can be used to speed up the computations

Examples K-fold CV

- `K_fold_CV_with_sklearn.ipynb`
  - Example of how to perform k-fold CV and pipelines to evaluate a model

Hyperparameter optimization

Learning & validation curve interpretation

see Ch. 06 in book “Python Machine Learning” by Raschka & Mirjalili

Repitition bias/variance (underfitting/overfitting)

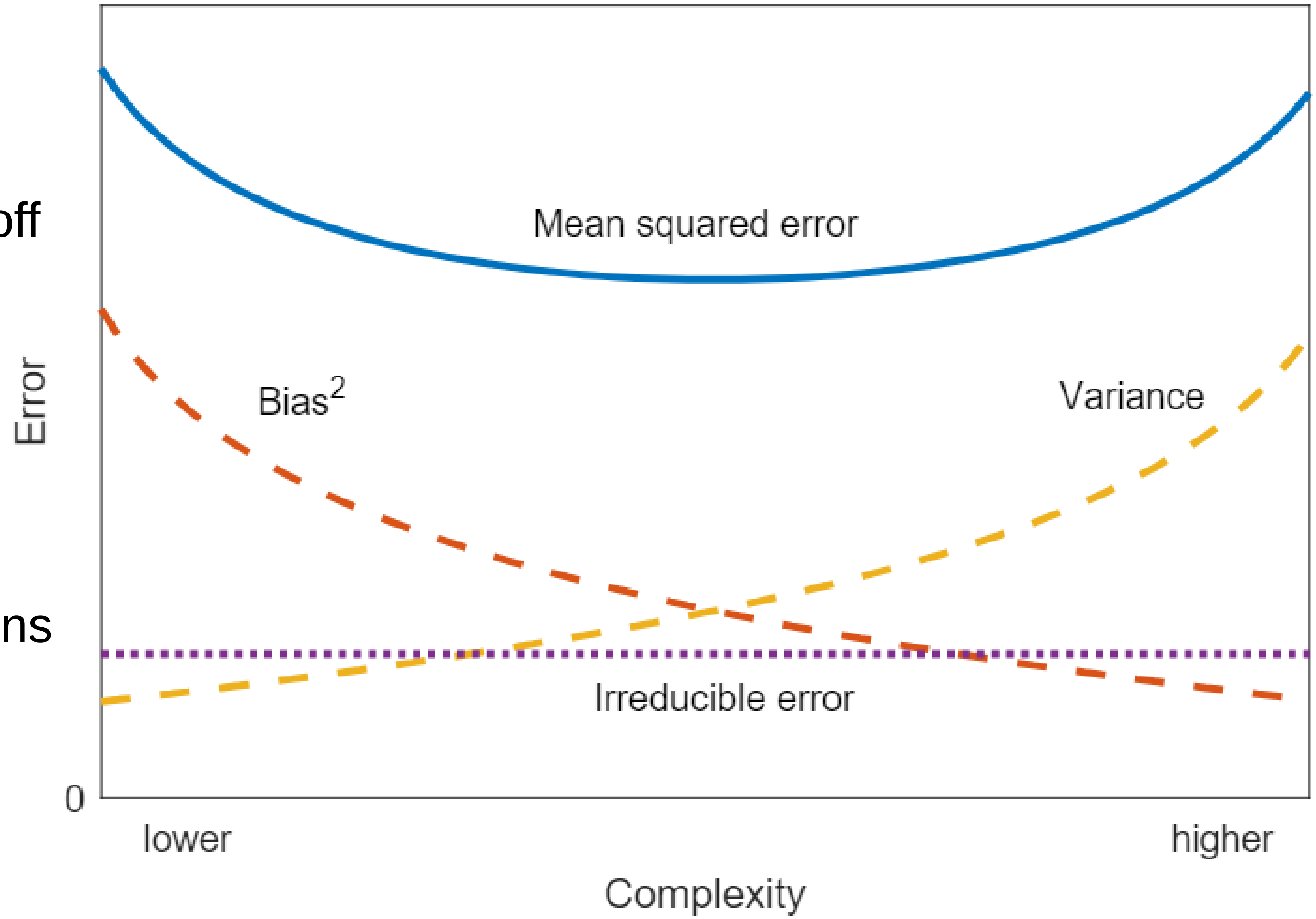
- High variance: Model is too complex → too many degrees of freedom or trainable parameters in this model
  - Model tends to do well (overfit) on the training data
  - Does not generalize well on unseen data
  - **Big gap between training accuracy and validation accuracy**
- High bias: Model is not complex enough → too few trainable parameters for this model, or too strict regularization constraints
  - Model performs poorly on training data and unseen data

Learning & validation curves

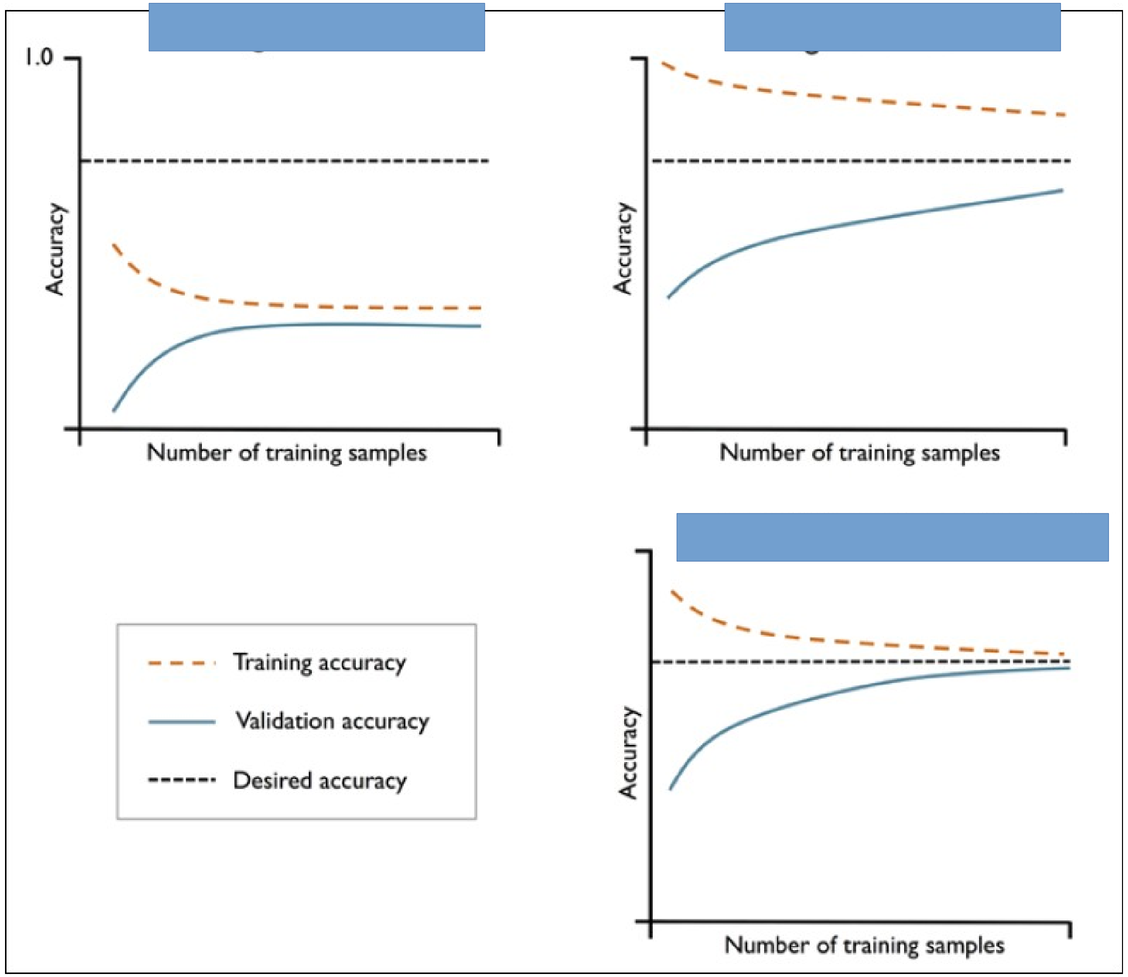
- A learning curve is a plot of a models **accuracy vs. # samples** in the training set
  - Train accuracy vs number of training samples
  - Validation accuracy vs number of training samples
- Validation curves are related to learning curves, but instead of varying the number of training samples, we vary the value of model parameters

Learning & validation curves

- There will always be a level of **irreducible error**
- The aim is to find the optimal trade-off between bias and variance that **minimizes overall error**
- Since there is a level of irreducible error, desired accuracy is set to less than 1 in the following illustrations

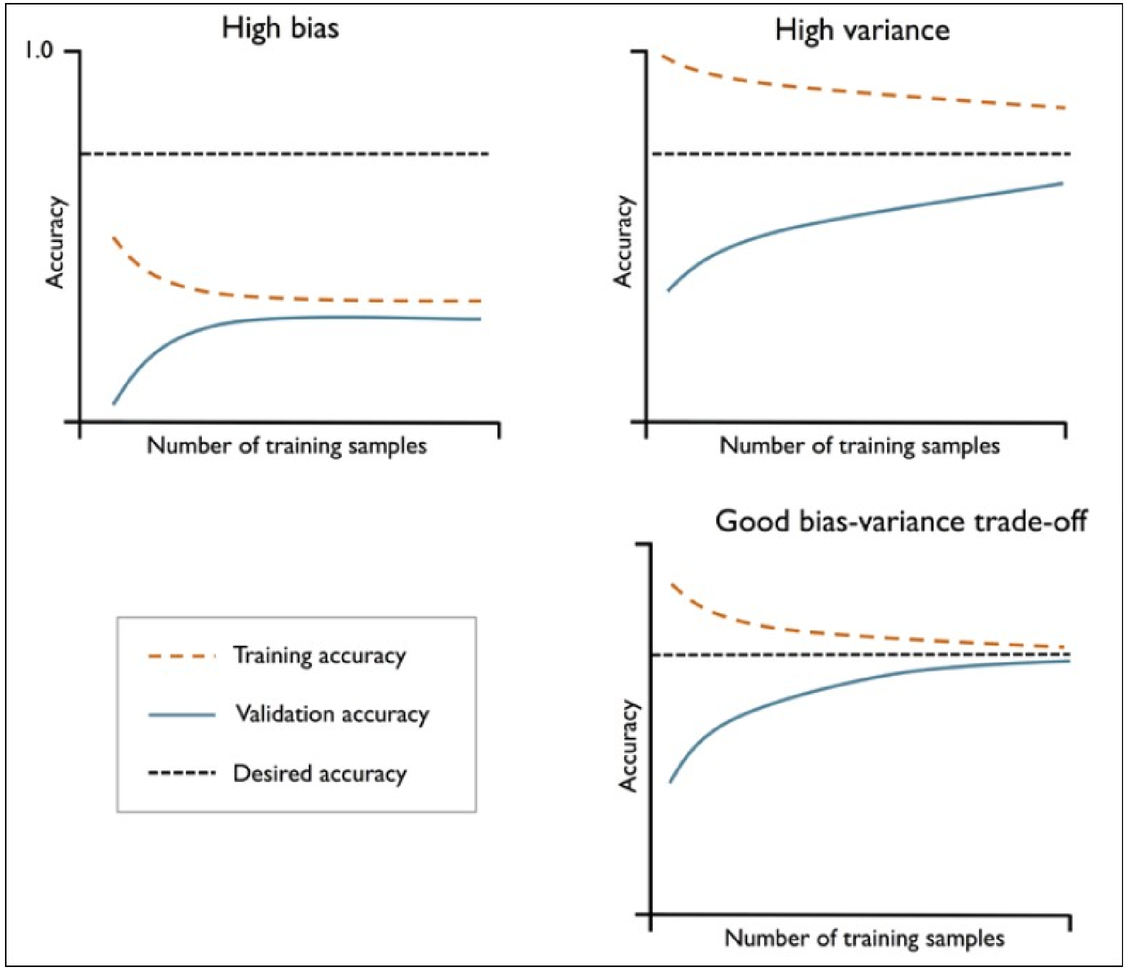


Learning & validation curves



Learning & validation curves

<u>Underfitting</u>, try:
- Construct or collect more features
- More flexible modelling
- Less regularisation



<u>Overfitting</u>, try:
- More data / fewer features
- Less flexible modelling
- More regularization

<u>Balanced fitting</u>, try:
- Pat yourself on the back
- Perform victory dance
- Brag about it

Examples Learning/Validation curves

- `learning_n_validation_curves.ipynb`
  - Example learning/validation curves can be applied for model development

Thank you for listening