Evaluation & Hyperparameter Tuning

Ata Kaban

With worked examples adapted from Andrew Moore's tutorial slides
https://sites.astro.caltech.edu/~george/aybi199/AMooreTutorials/

Recap (1)

Each supervised learning method consists of 3 ingredients:

- Model: form of function we want to learn (has free parameters)
- Cost function: given a training set, it measures the misfit of any particular function from the model
- Training algorithm: gradient descent minimisation of the cost function

Running the training algorithm on some training data learns "best" values of the free parameters, giving us a predictor.

Recap (2)

Hyperparameters are “higher-level” free parameters

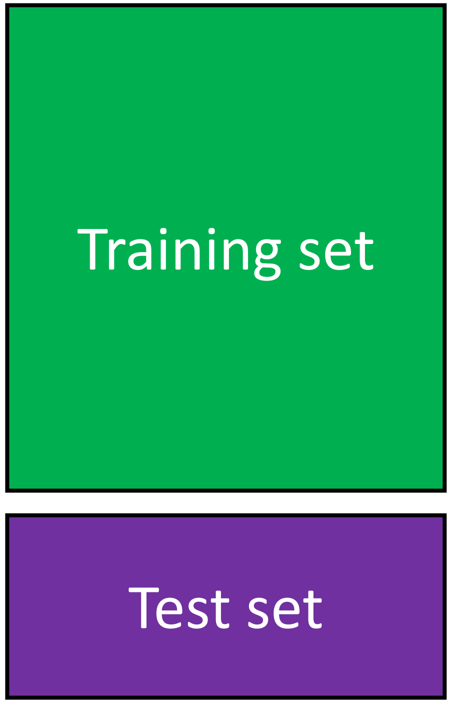
- In Neural Networks:
  - Depth (number of hidden layers)
  - Width (number of hidden neurons in a hidden layer)
  - Activation function (choice of nonlinearity in non-input nodes)
  - Regularisation parameter (way to trade off simplicity vs. fit to the data)
- In polynomial regression
  - Order of the polynomial (use of $x, x^2, x^3, \dots, x^m$)
- In general
  - Model choice

Evaluation of a predictor before deployment

Always split the available annotated data randomly into:

- A training set- to be used for training – i.e. estimating all the free parameters
- A test set - to be used to evaluate the trained predictor before deploying it

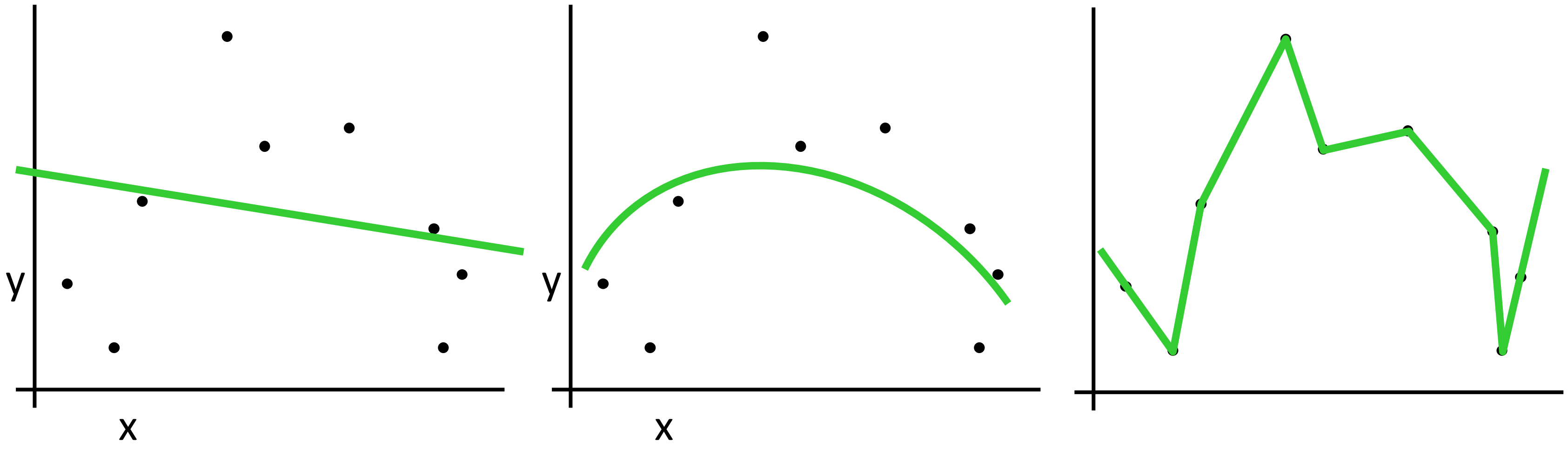
Evaluation of a predictor serves to estimate its future performance, before deploying it in the real world.



Which model? How to set hyperparameters?

- Each hyperparameter value corresponds to a different model
- We need methods that evaluate each candidate model
- For this evaluation we can no longer use our cost function computed on training set – why?
  - The more complex (flexible) the model, the better it will fit the training data
  - But the goal is to predict well on future data
  - A model that has capacity to fit any training data will overfit.
- To choose between models (including hyperparameters) we need a criterion to estimate future performance

Which model to choose?



Fit with Model 1 | Fit with Model 2 | Fit with Model 3

Remember: Even if the models only differ by one hyperparameter, they are different models.
Choosing a particular value of a hyperparameter requires evaluating each model.

Evaluating models for model choice

- Don’t confuse this with evaluating a predictor (model already chosen)

- The training set is annotated data (input, output) – use for training within a chosen model
- The test set is also annotated data (input output) – use for evaluating the performance of the trained predictor before deploying it

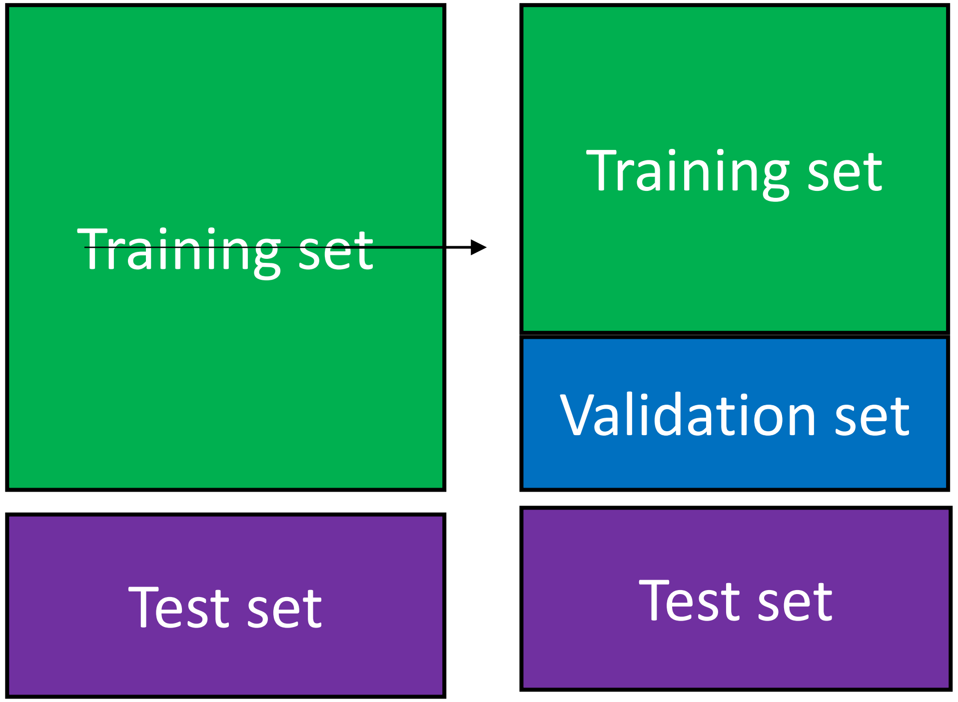
- None of these can be used to choose the model!
  - Tempting to use the test set, but if we do, we no longer have an independent data set to evaluate the final predictor before deployment!

Evaluating models for model choice

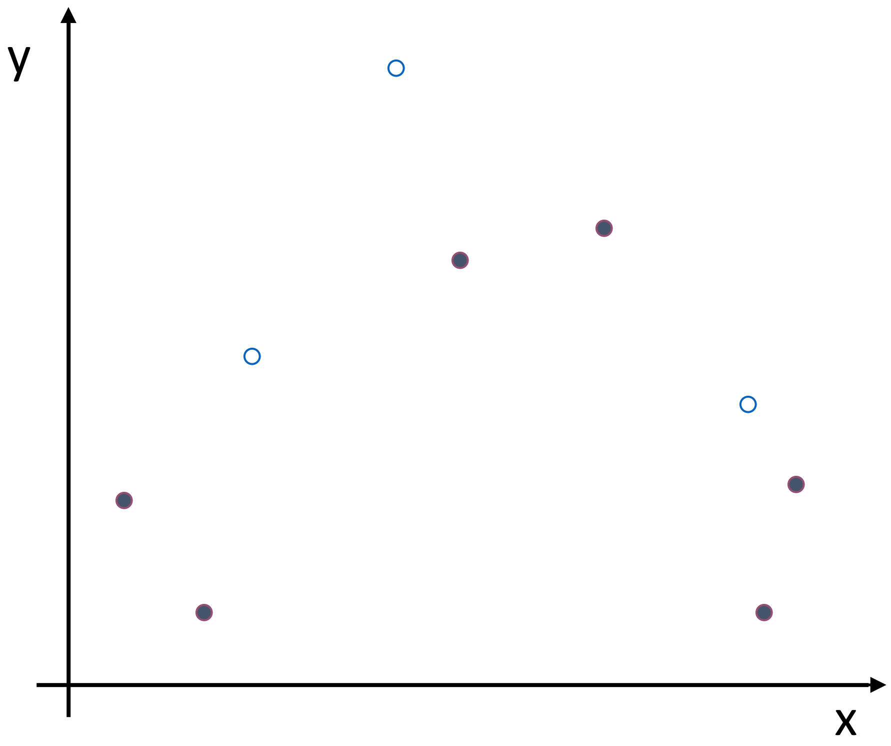
Idea: To choose between models or hyperparameters, split out a subset from the training set = validation set

Methods:

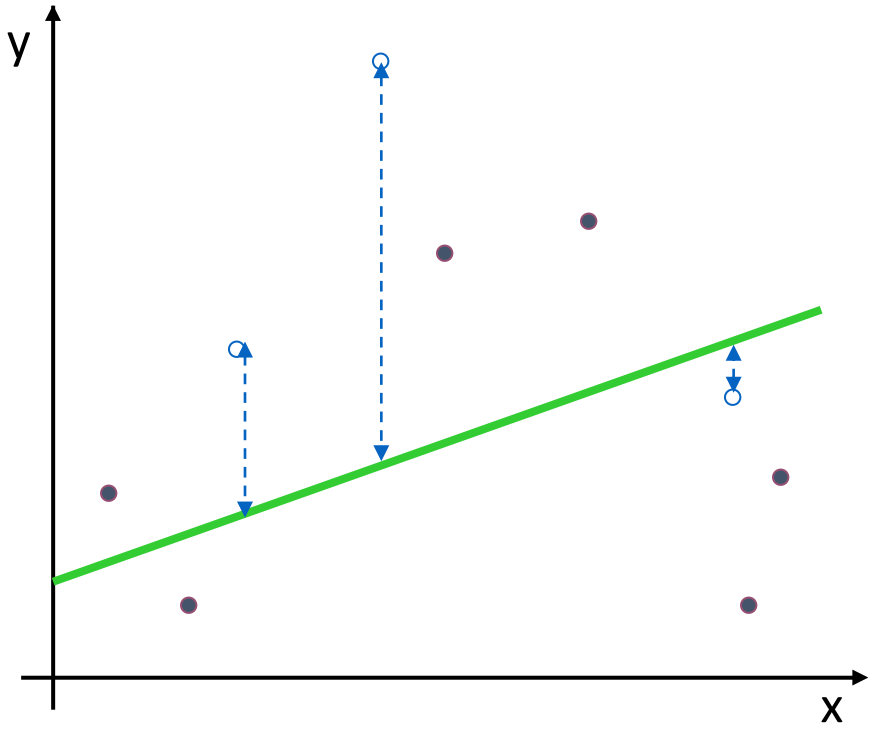
- Holdout validation
- Cross-validation
- Leave-one-out validation



Method 1: The holdout validation method



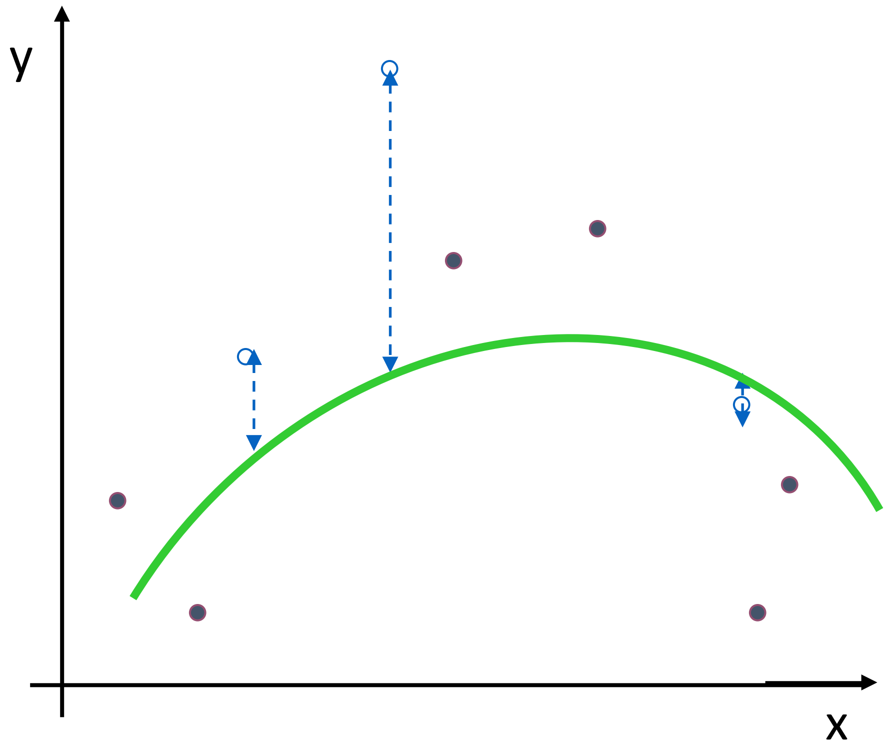
1. Randomly choose 30% of the data to form a validation set
2. The remainder is a training set
3. Train your model on the training set
4. Estimate the test performance on the validation set
5. Choose the model with lowest validation error
6. Re-train with the chosen model on joined train & validation set to obtain predictor
7. Estimate future performance of obtained predictor on the test set
8. Ready to deploy the predictor



Model 1

Mean Squared Validation Error = 2.4

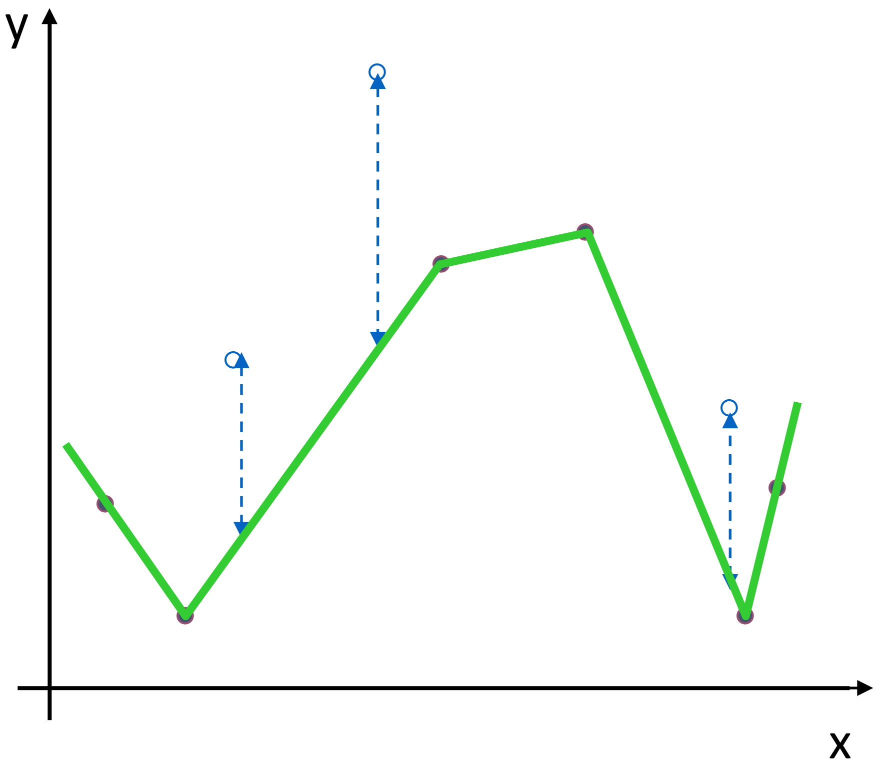
1. Randomly choose 30% of the data to form a validation set
2. The remainder is a training set
3. Train your model on the training set
4. Estimate the test performance on the validation set
5. Choose the model with lowest validation error
6. Re-train with the chosen model on joined train & validation set to obtain predictor
7. Estimate future performance of obtained predictor on the test set
8. Ready to deploy the predictor



Model 2

Mean Squared Validation Error = 0.9

1. Randomly choose 30% of the data to form a validation set
2. The remainder is a training set
3. Train your model on the training set
4. Estimate the test performance on the validation set
5. Choose the model with lowest validation error
6. Re-train with the chosen model on joined train & validation set to obtain predictor
7. Estimate future performance of obtained predictor on the test set
8. Ready to deploy the predictor



Model 3

Mean Squared Validation Error = 2.2

1. Randomly choose 30% of the data to form a validation set
2. The remainder is a training set
3. Train your model on the training set
4. Estimate the test performance on the validation set
5. Choose the model with lowest validation error
6. Re-train with the chosen model on joined train & validation set to obtain predictor
7. Estimate future performance of obtained predictor on the test set
8. Ready to deploy the predictor

Choose the model with the lowest validation error

Model 1
Mean Squared Validation Error = 2.4

Model 2
Mean Squared Validation Error = 0.9

Model 3
Mean Squared Validation Error = 2.2

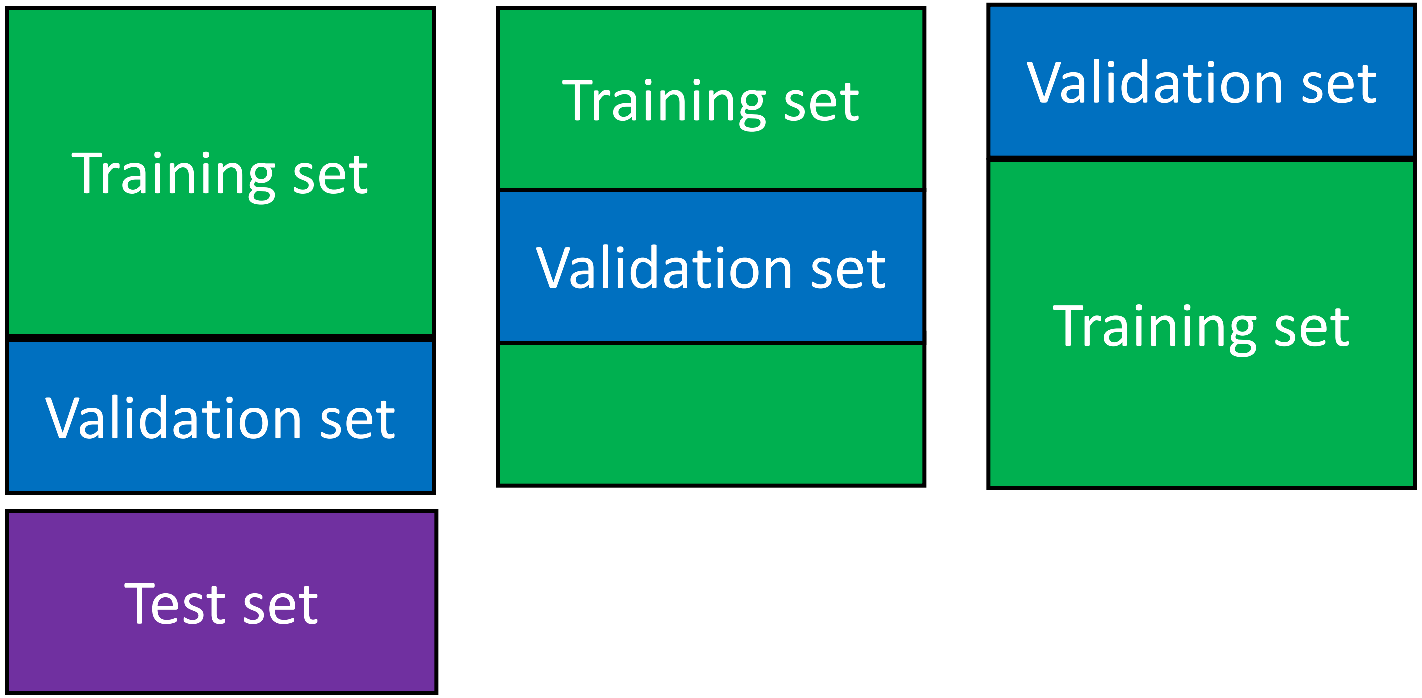
A practical detail on point 4

"4. Estimate the test performance on the validation set"

This is done differently in regression and in classification:

- In regression, we compute the cost function (mean square error) on the examples of the validation set (instead of the training set)
- In classification, we don't compute the cross-entropy cost on the validation set, instead on validation set we compute the 0-1 error metric: $\frac{number\ of\ wrong\ predictions}{number\ of\ predictions}$ = 1 - Accuracy
    - There are also other metrics, besides Accuracy, that take account of the 2 types of error specific to classification (false positives and false negatives)

Method 2: k-fold Cross-validation



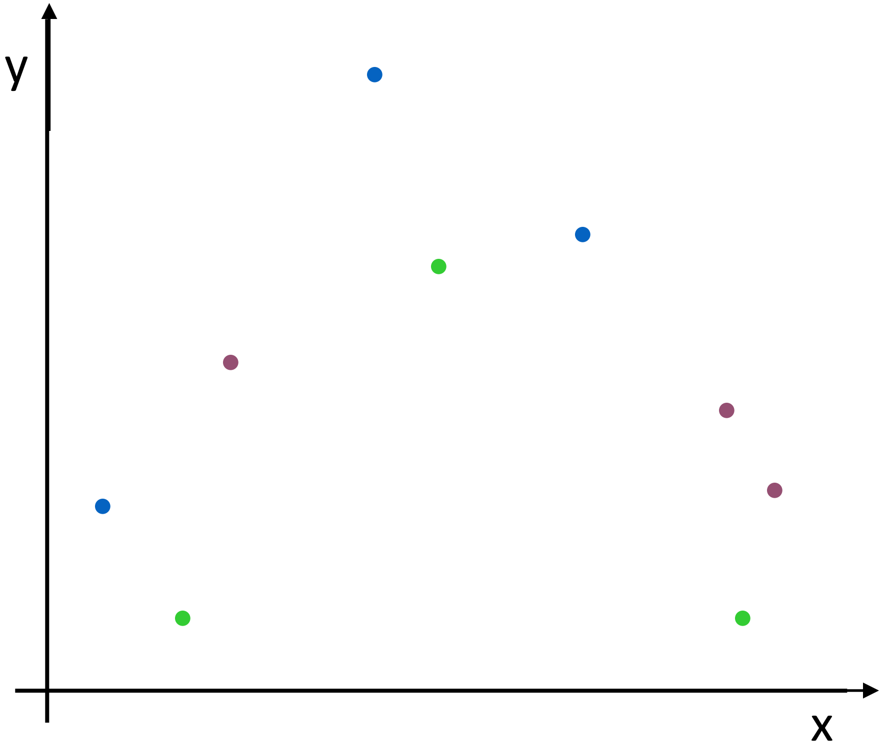
Split the training set randomly into k (equal sized) disjoint sets.
(In this example, k=3)

Use k-1 of those together for training

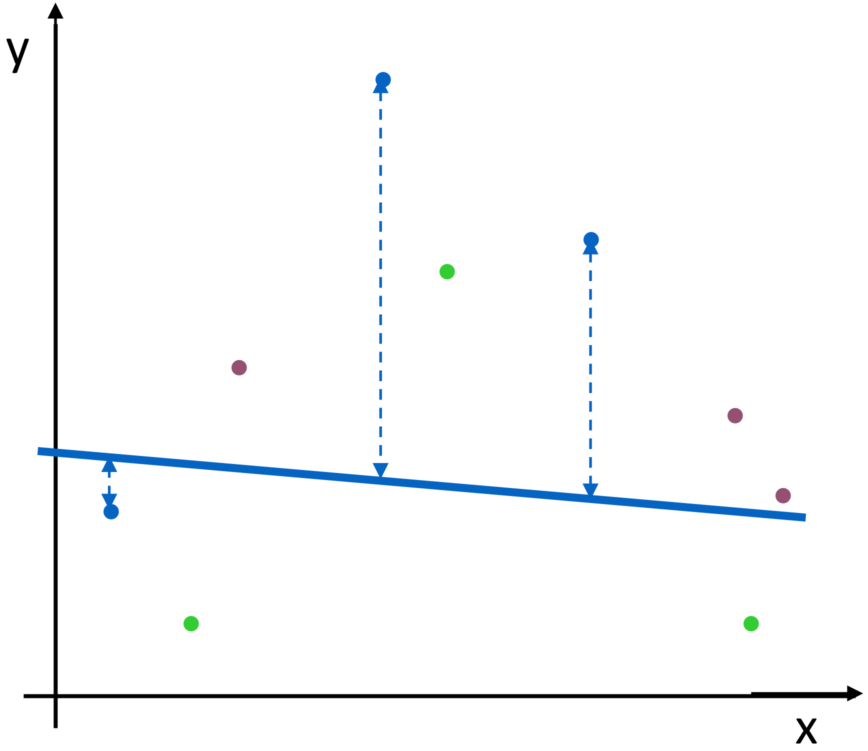
Use the remaining one for validation.

Permute the k sets and repeat k times.

Average the performances on the k validation sets.

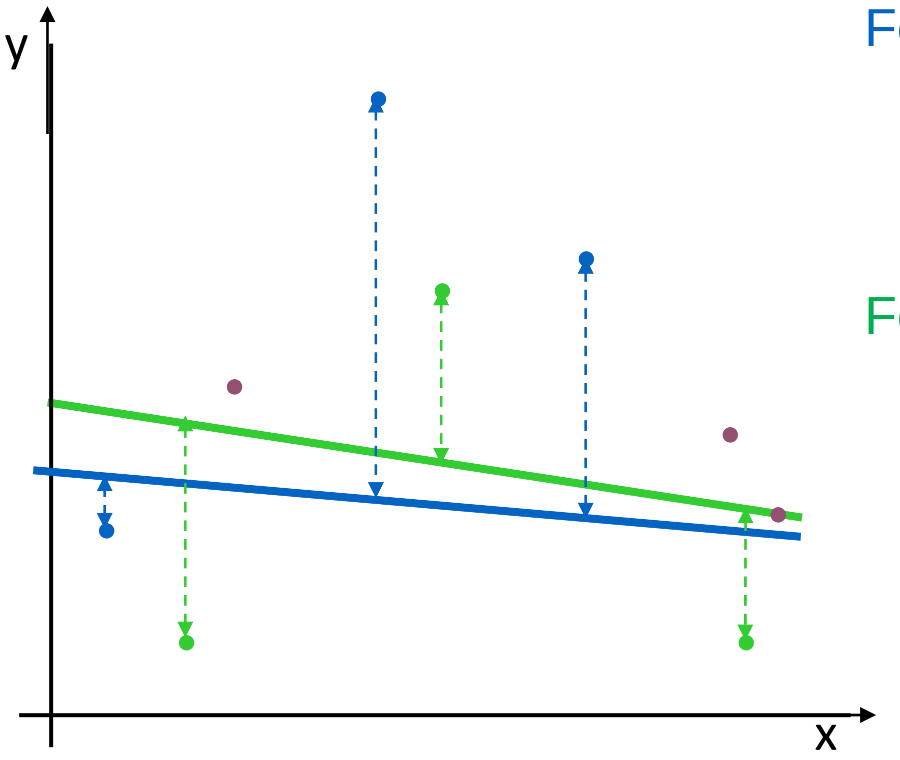


Randomly break the dataset into k partitions (here k=3)



Randomly break the dataset into k partitions (here k=3)

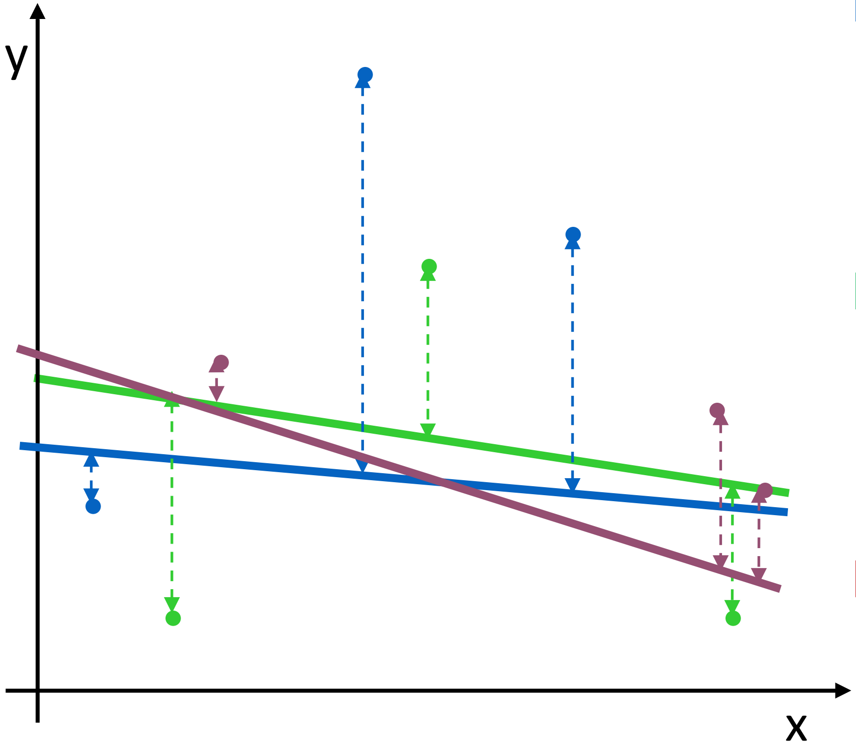
For the blue partition: Train on all the points <u>except</u> the blue partition. Compute the validation error using the points in the blue partition.



Randomly break the dataset into k partitions (here k=3)

For the blue partition: Train on all the points except the blue partition. Compute the validation error using the points in the blue partition.

For the green partition: Train on all the points except the green partition. Compute the validation error using the points in the green partition.



Randomly break the dataset into k partitions (here k=3)

For the blue partition: Train on all the points except the blue partition. Compute the validation error using the points in the blue partition.

For the green partition: Train on all the points except the green partition. Compute the validation error using the points in the green partition.

For the purple partition: Train on all the points except the purple partition. Compute the validation error using the points in the purple partition.

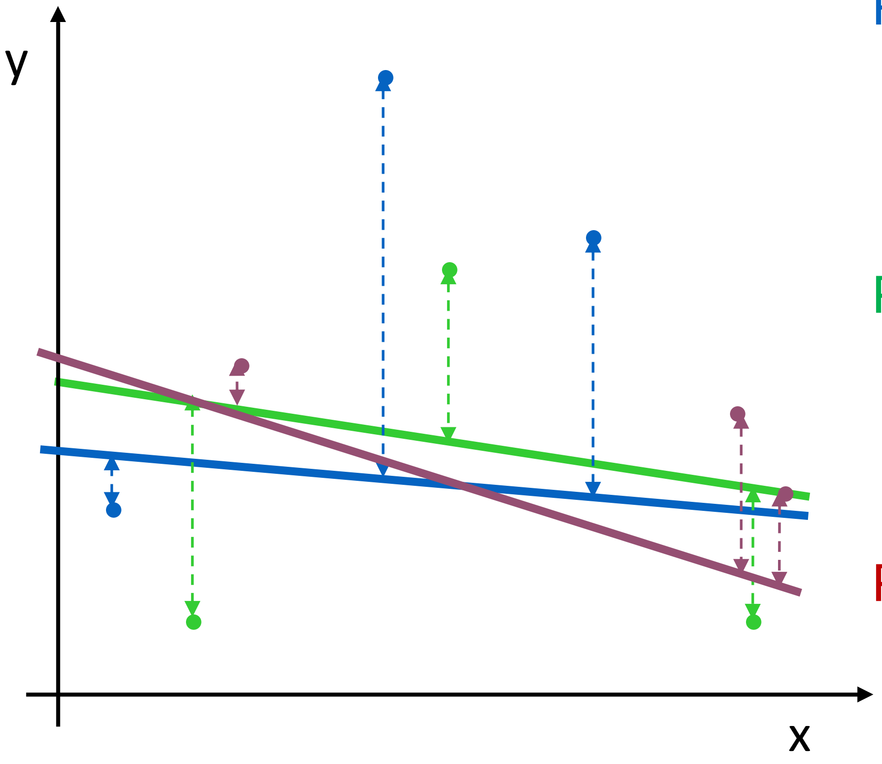
Randomly break the dataset into k partitions (here k=3)

For the blue partition: Train on all the points except the blue partition. Compute the validation error using the points in the blue partition.

For the green partition: Train on all the points except the green partition. Compute the validation error using the points in the green partition.

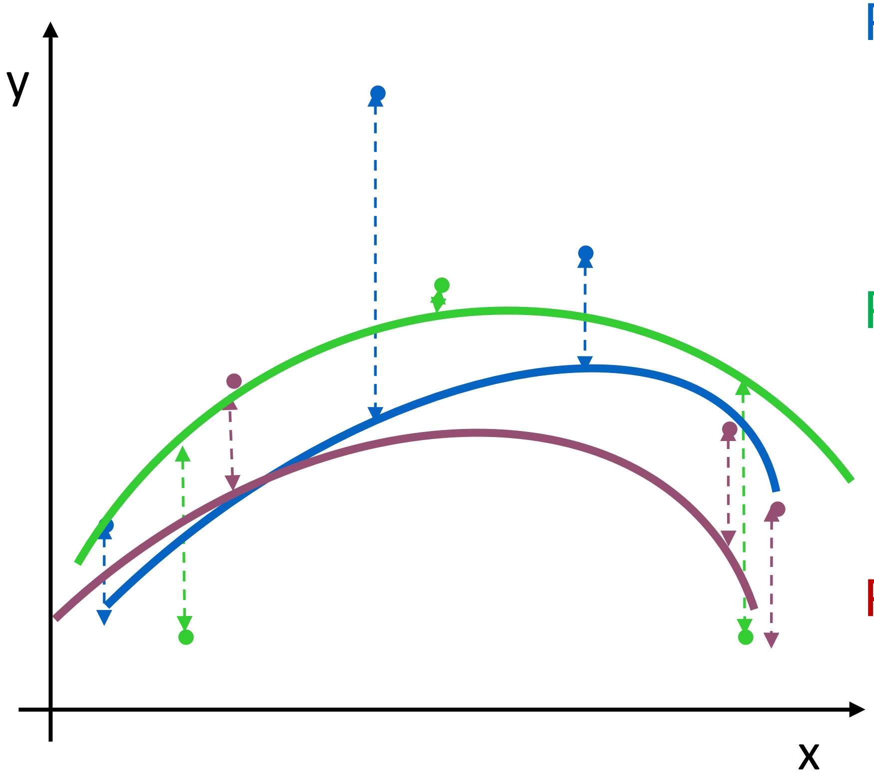
For the purple partition: Train on all the points except the purple partition. Compute the validation error using the points in the purple partition.

Take the mean of these errors



Model 1

MSE_{3FOLD}=2.05



Model 2

MSE_{3FOLD}=1.11

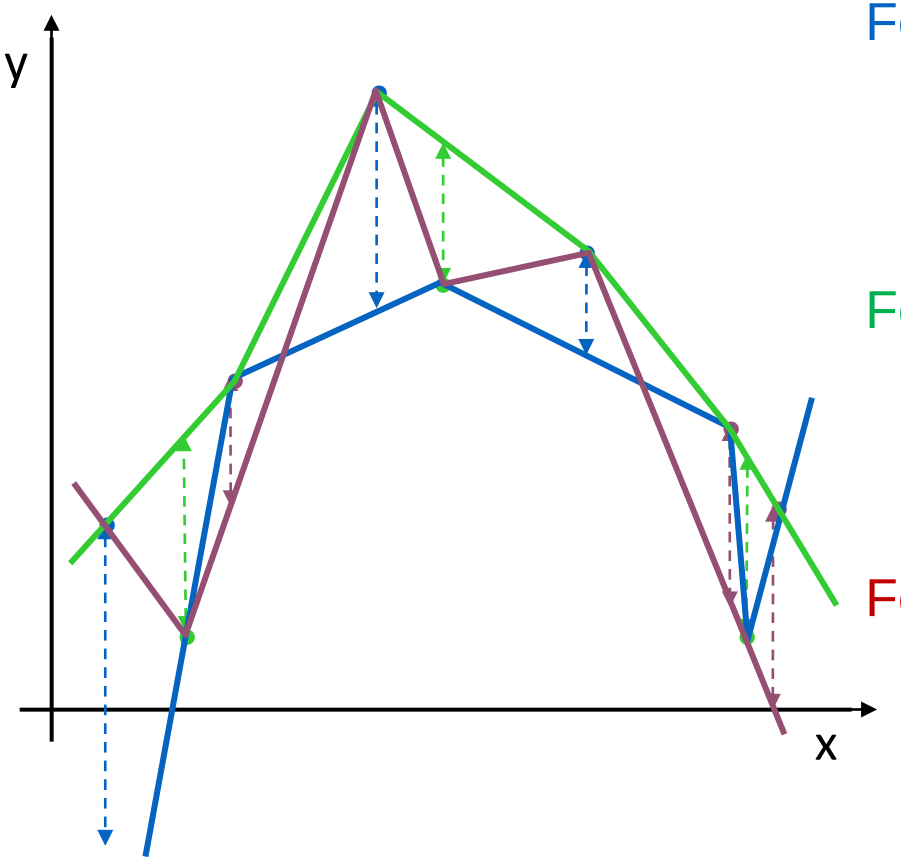
Randomly break the dataset into k partitions (here k=3)

For the blue partition: Train on all the points except the blue partition. Compute the validation error using the points in the blue partition.

For the green partition: Train on all the points except the green partition. Compute the validation error using the points in the green partition.

For the purple partition: Train on all the points except the purple partition. Compute the validation error using the points in the purple partition.

Take the mean of these errors



Model 3

$MSE_{3FOLD}=2.93$

Randomly break the dataset into k partitions (here k=3)

For the blue partition: Train on all the points <u>except</u> the blue partition. Compute the validation error using the points in the blue partition.

For the green partition: Train on all the points <u>except</u> the green partition. Compute the validation error using the points in the green partition.

For the purple partition: Train on all the points <u>except</u> the purple partition. Compute the validation error using the points in the purple partition.

Take the mean of these errors

Method 3: Leave-one-out validation

- We leave out a single example for validation, and train on all the rest of the annotated data
- For a total of N examples, we repeat this N times, each time leaving out a single example
- Take the average of the validation errors as measured on the left-out points

- Same as N-fold cross-validation where N is the number of labelled points

Advantages & Disadvantages

	Advantages	Disadvantages
Holdout validation	Computationally cheapest	Most unreliable if sample size is not large enough
3-fold	Slightly more reliable than holdout	• Wastes 1/3-rd annotated data. • Computationally 3-times as expensive as holdout
10-fold	• Only wastes 10% • Fairly reliable	• Wastes 10% annotated data • Computationally 10-times as expensive as holdout
Leave-one-out	Doesn't waste data	Computationally most expensive

Large sample

Small sample

Using model validation to tune hyperparameters

Example 1: Choosing number of hidden units in a Multi-Layer Perceptron

- Step 1: Compute 10-fold CV error for six different model classes:

Candidates	Train ERR	10-FOLD-CV-ERR	Choice
0 hidden units			
1 hidden units			
2 hidden units			☺
3 hidden units			
4 hidden units			
5 hidden units			

- Step 2: Whichever candidate choice gave best CV score: train it with all the data, and that's the predictor you'll use.

Example 2: Choosing number of hidden layers in a neural nets

- Step 1: Compute 10-fold CV error for six different model classes:

Candidates	Train ERR	10-FOLD-CV-ERR	Choice
0 hidden layer			
1 hidden layer			
2 hidden layers			☺
3 hidden layers			
4 hidden layers			
5 hidden layers			

- Step 2: Whichever model class gave best CV score: train it with all the data, and that's the predictor you'll use.

Example 3: Choosing the activation function is (deep) neural net

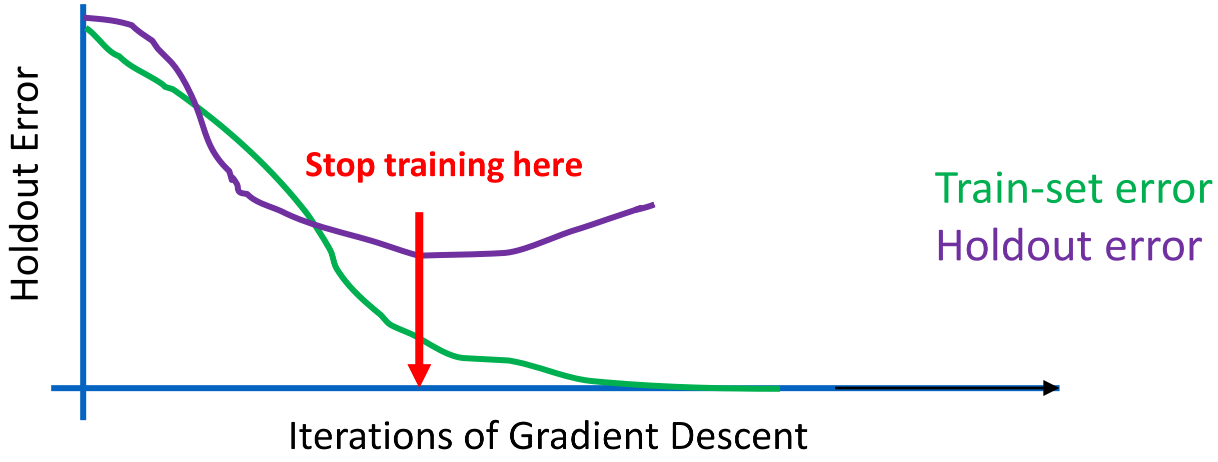
- Step 1: Compute 10-fold CV error for six different model classes:

Candidates	Train ERR	10-FOLD-CV-ERR	Choice
σ_1			
σ_2			
σ_3			☺
σ_4			
σ_5			
σ_6			

- Step 2: Whichever candidate choice gave best CV score: train it with all the data, and that's the predictor you'll use.

Example 4: Early Stopping using Holdout validation

- Suppose you have a neural net with too many hidden units. It will overfit.
- As Backprop (gradient descent) progresses, monitor the error on a holdout set



What you should know

- Why you can’t use “training-set-error” to choose between models
- Why you need model validation methods to tune hyperparameters
- Methods for model validation and how they work
    - Holdout validation
    - k-fold cross-validation
    - Leave-one-out validation
- Advantages & disadvantages of each model validation method