Model Evaluation and Hyperparameter Tuning:

Learning Outcomes:

- ***Understand the significance of model evaluation***
- ***Learn different methods of model evaluation***
- ***Understand the pros and cons of the methods***

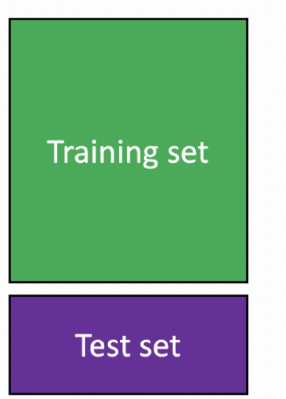
Hyperparameters are "higher-level" free parameters:

- *In neural networks:*
  - *Depth :*
    - *number of hidden layers*
  - *Width :*
    - *number of hidden neurons in a hidden layer*
  - *Activation function :*
    - *Refers to the non-linearity applied to neurons in a neural network*
    - *Choice of non-linearity in non-input nodes*
      - *Explains that activation functions are applied in layers other than the input layer, affecting how the network learns complex patterns*
  - *Regularisation parameter :*
    - *way to trade off simplicity vs. fit to the data*
- *In polynomial regression :*
  - *Order of the polynomial (i.e. use of* $x, x^2, x^3, ..., x^m$)
- *In general*
  - *Model Choice*

Evaluation of a Predictor Before Deployment

- Recall: A predictor is obtained by training the free parameters of the considered model using the available annotated data.
- Evaluation of a predictor serves to estimate its future performance, before deploying it in the real world.

- To do so, we always split the available annotated data randomly into:
  - A **training set**, used to estimate the free parameters, and
  - A **test set**, used to evaluate the performance of the trained predictor before deploying it.



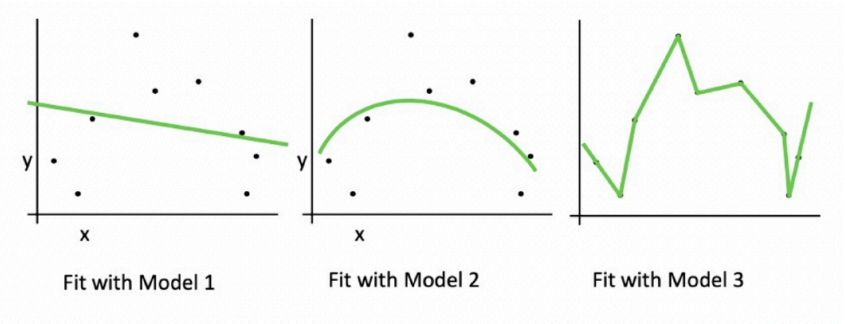
Evaluation of a Predictor Before Deployment

- *A predictor is obtained by training the free parameters of the considered model using the available annotated data.*
- *Evaluation of a predictor serves to estimate its future performance, before deploying it in the real world.*
- *To do so, we always split the available annotated data randomly into:*
  - *A training set*
    - *used to estimate the free parameters*
  - *And a test set*
    - *used to evaluate the performance of the trained predictor before deploying it*

🔥 Which Model? How to set hyperparameters?

- Each hyperparameter value corresponds to a different model.
- We need methods that evaluate each model.
- For this evaluation, we can no longer use the cost function computed on the 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/any model will overfit

Which Model to Choose?



Note here:

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

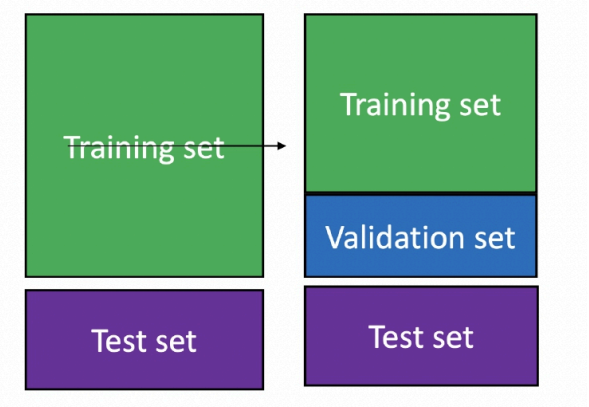
- Evaluating Models for Model Choice
  - Do not confuse this with evaluating a predictor
    - Because this means a model is already chosen
  - The training set is annotated data used for training within a chosen model.
  - The test set is also annotated data but used for evaluating the trained predictor before deploying it
  - None of these can be used to choose the model!
    - Tempting to use the test set,
      - but then, we no longer have an independent data set to evaluate the final predictor before deployment.

Evaluating Models for Model Choice

Key Idea: To choose between models or hyperparameters, separate a subset from the training set to create **validation set**.

Methods

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



So we need to take a part of the training set to see which model is best suited.

Estimating Test Performance on the Validation Set

- In practice, estimating test performance is handled differently for **regression** and **classification** models.
- Regression:
  - We compute the cost function
    - (mean squared error)
      - $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  - on the examples of the validation set (instead of the training set)
- Classification:
  - We do not compute the cross-entropy cost on the validation set.
  - Instead, we compute the 0 - 1 error metric
- $0-1 \text{ error metric } = \frac{\text{number of wrong predictions}}{\text{number of predictions}} = 1 - \text{Accuracy}$

Regression Example: Evaluating with Mean Squared Error (MSE)

For regression models, we assess the cost function using metrics like **Mean Squared Error (MSE)** on the validation set.

- **Example:**
  - Suppose we are predicting house prices based on features like square footage, number of bedrooms, and location.
  - The model predicts the house price as **$300,000**, but the actual price is **$320,000**.
  - The squared error for this prediction is **(320,000 - 300,000)² = 400,000,000**.
  - If we have multiple predictions, we calculate the **mean** of all squared errors to get MSE:

    $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  - A lower MSE indicates better model performance.

Classification Example: Evaluating with 0-1 Error Metric

For classification models, we do not compute cross-entropy cost on the validation set. Instead, we use the **0-1 error metric**, which is related to accuracy.

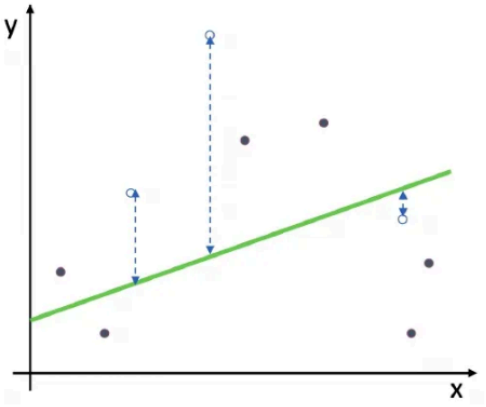
- **Example:** Suppose we are classifying emails as **Spam (1)** or **Not Spam (0)**.
  - If the model predicts **Spam**, but the actual label is **Not Spam**, that counts as an error.

- If we have 100 emails and the model misclassifies **10 emails**, then:

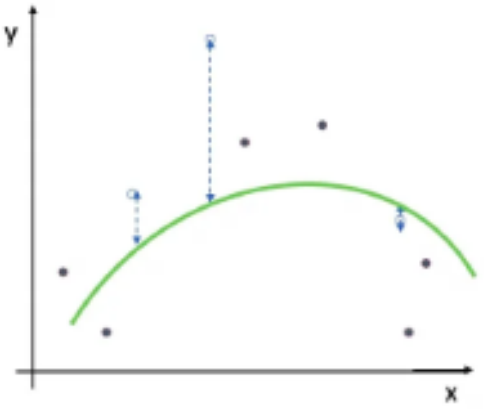
$$\text{error metric} = \frac{\text{number of wrong predictions}}{\text{number of predictions}} = \frac{10}{100} = 0.1$$

- Accuracy is:
  1 - 0.1 = 0.9
  $(\text{or } 90\%)$
- Other classification metrics include **Precision, Recall, and F1-score** for more nuanced evaluation.

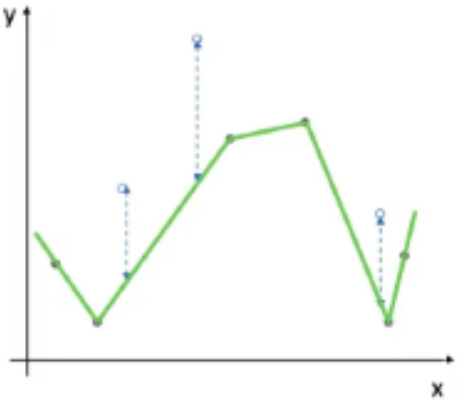
Method 1: Holdout Validation



Model 1
Mean Squared Validation Error = 2.4



Model 2
Mean Squared Validation Error = 0.9



Model 3
Mean Squared Validation Error = 2.2

- Model 1: Mean squared validation error = 2.4
- Model 2: Mean squared validation error = 0.9
- Model 3: Mean squared validation error = 2.2

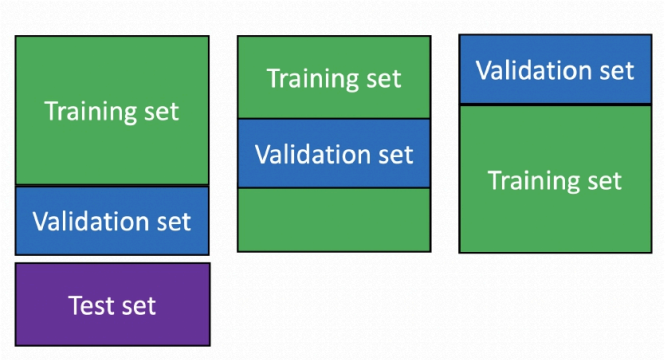
Lowest validation error: **Model 2**

Steps:

1. Randomly choose 30% of data to form a validation set
2. Remaining data forms the 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 chosen model on joined training and validation to obtain predictor
7. Estimate future performance of the obtained predictor on test set
8. Ready to deploy the predictor

Method 2: k-fold Cross-Validation

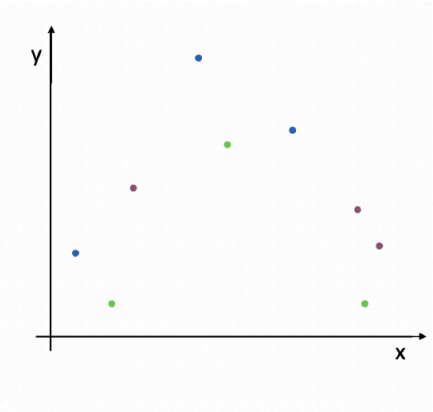
Method 2: k-Fold Cross-Validation



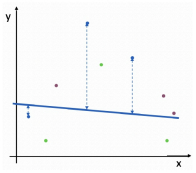
1. Split the training set randomly into k (equal-sized) disjoint sets. (In this example, $k = 3$)
2. Use $k - 1$ of those together for training
3. Use the remaining one for validation
4. Permute the k sets and repeat k times
5. Average the performances on k validation sets

Explanation of the Steps for $k = 3$-Fold Cross-Validation

Randomly split the dataset into $k = 3$ partitions, denoted by blue, green and purple points.

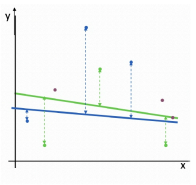


Model 1



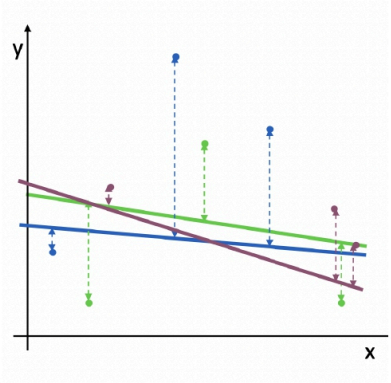
- Validation on the blue partition: Train on all the points **except** the blue partition. Compute the validation error using the points in the blue partition

Model 1



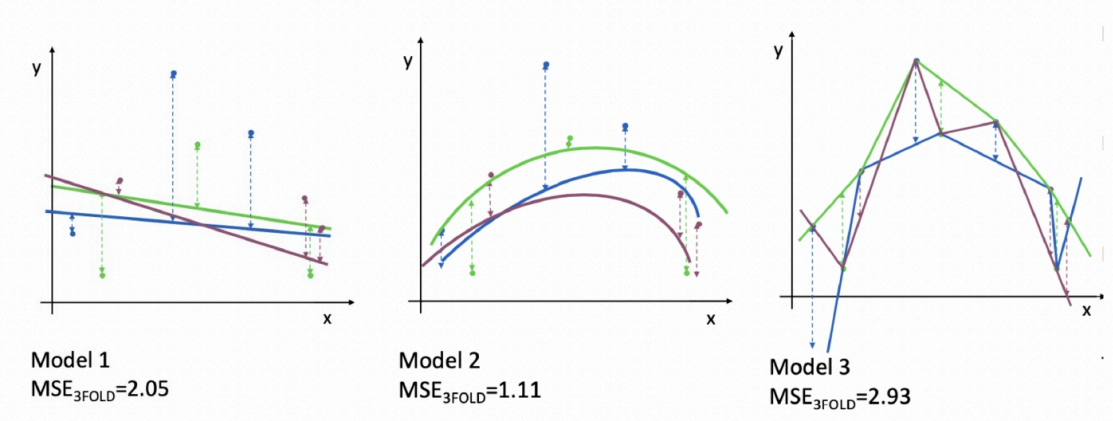
- Validation on the blue partition: Train on all the points **except** the blue partition. Compute the validation error using the points in the blue partition
- Validation on the the green partition: Train on all the points **except** the green partition. Compute the validation error using the points in the green partition

Model 1



- Validation on the the blue partition: Train on all the points **except** the blue partition. Compute the validation error using the points in the blue partition
- Validation on the the green partition: Train on all the points **except** the green partition. Compute the validation error using the points in the green partition
- Validation on the 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
  - $\mathrm{MSE}_{\mathrm{3FOLD}} = 2.05$

Repeat for other models:



- Choose the model with the smallest average 3-fold cross validation error. Here, this is Model 2.
- Re-train with chosen model on joined training and validation to obtain the predictor
- Estimate future performance of the obtained predictor on test set
- Deploy the predictor in real-world

Method 3: Leave-one-out Cross-Validation

Very similar to method 2, k-fold cross validation just k is locked in at 1

🔥

Method 3: Leave-one-out Cross-Validation

1. We leave out a single example for validation, and train on all the rest of the annotated data
2. For a total of N examples, we repeat this N times, each time leaving out a single example
3. Take the average of the validation errors as measured on the left-out points
4. Same as N-fold cross-validation where N is the number of labelled points

🔥

Leave-One-Out Cross-Validation

It is a technique where we train the model on **all but one** data point and test on the **remaining single** data point.

This process is repeated for each data point in the dataset.

Example: Predicting Student Exam Scores (Regression Task)

Suppose we have a dataset of **5 students**, where each student's **hours studied** is used to predict their **exam score**.

Student	Hours Studied	Exam Score
A	2	50
B	4	70
C	6	80
D	8	90
E	10	95

Using Leave-One-Out Cross-Validation, we do the following:

1. **Iteration 1**: Use students **B, C, D, and E** to train the model and predict student **A's** score.
2. **Iteration 2**: Use students **A, C, D, and E** to train and predict student **B's** score.
3. **Iteration 3**: Use students **A, B, D, and E** to train and predict student **C's** score.
4. **Iteration 4**: Use students **A, B, C, and E** to train and predict student **D's** score.
5. **Iteration 5**: Use students **A, B, C, and D** to train and predict student **E's** score.

At the end, we calculate an overall error metric (e.g., **Mean Squared Error**) by averaging the errors from each iteration.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

	Advantages	***Disadvantages***
Holdout Validation	*Computationally cheapest*	*Most unreliable if sample size is not large enough*
3-fold	*Slightly more reliable than holdout*	• $Wastes\ \frac{1}{3}rd\ annotated\ data$ • $Computationally\ 3\ times\ as\ expensive\ as\ holdout$
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*