

Departamento de Eletrónica, Telecomunicações e  
Informática

# **MODEL SELECTION AND VALIDATION – BIAS VS. VARIANCE**

**Author: Petia Georgieva**

**Edited by: Susana Brás ( [Susana.bras@ua.pt](mailto:Susana.bras@ua.pt) )**

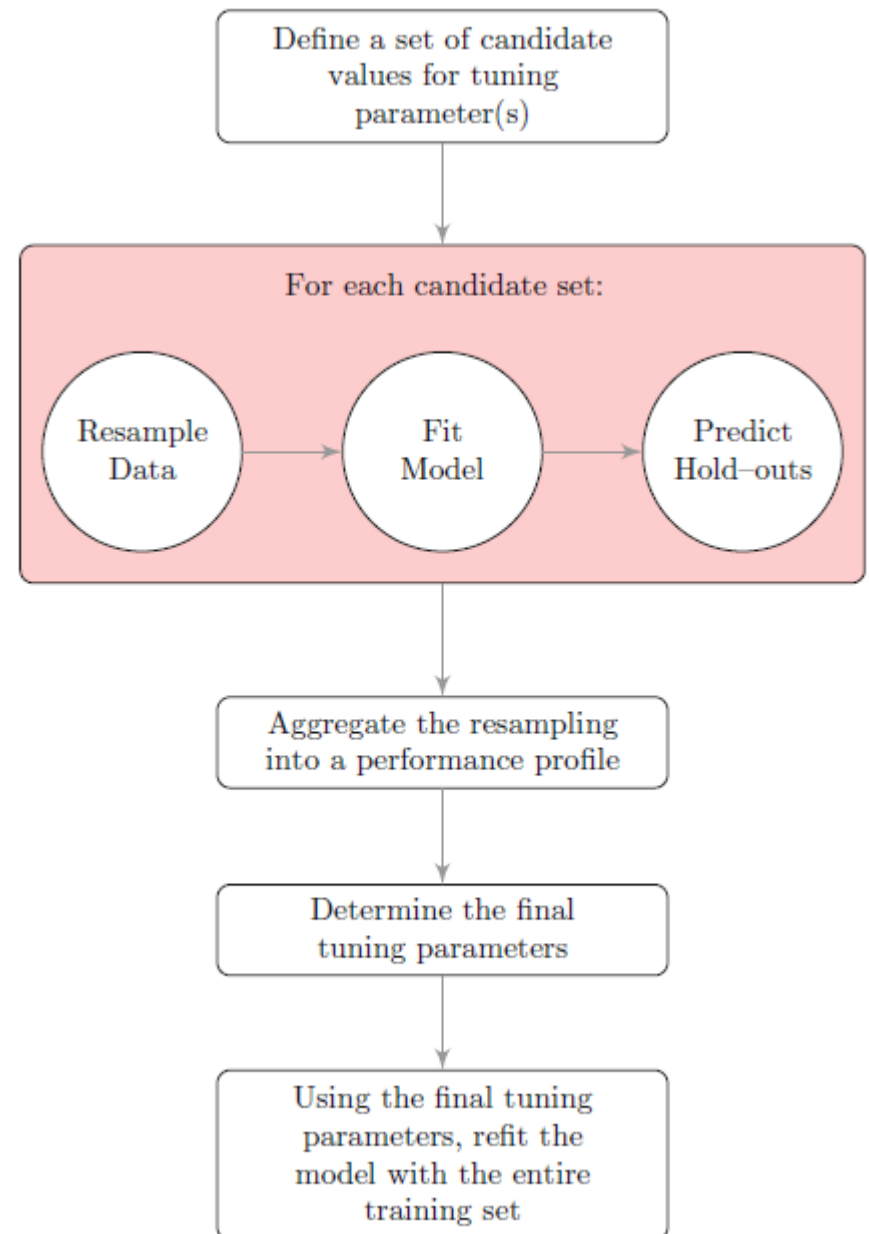
# **Lecture Outline**

- 1. Cross Validation**
- 2. Bias vs. variance**
- 3. Learning curves**
- 4. Ensemble classifiers**
- 5. Model-centric vs Data-centric ML**

# Cross Validation

# Model Tuning

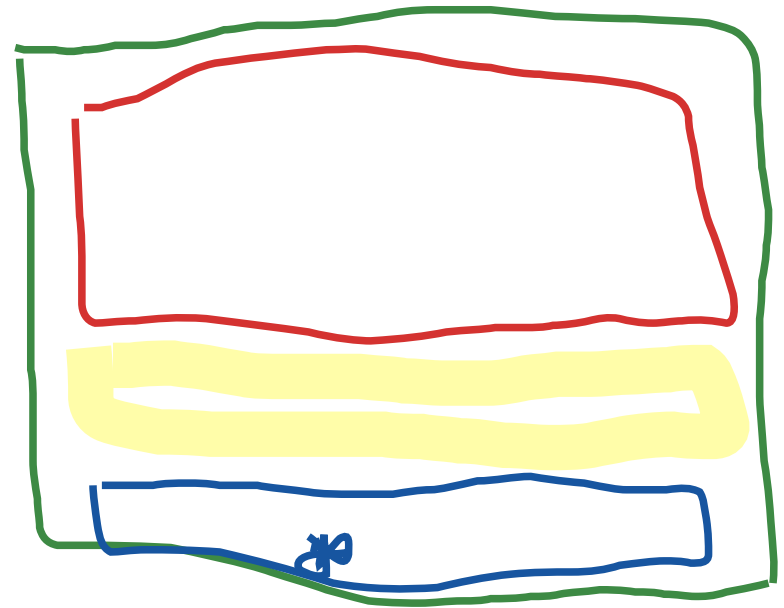
- Model parameters may be estimated directly by data.
- Nevertheless, there are parameters that cannot be estimated by data.
- There are different approaches to searching for the best parameters. A general approach that can be applied to almost any model is to define a set of candidate values, generate reliable estimates of model utility across the candidates values, then choose the optimal settings.
- A more difficult problem is obtaining trustworthy estimates of model performance for these candidate models.
- The apparent error rate can produce extremely optimistic performance estimates.
- A better approach is to test the model on samples that were not used for training.



# Data Splitting 🗨️

The correct division between train and test is essential to obtain a correct approach to the effectiveness of the model.

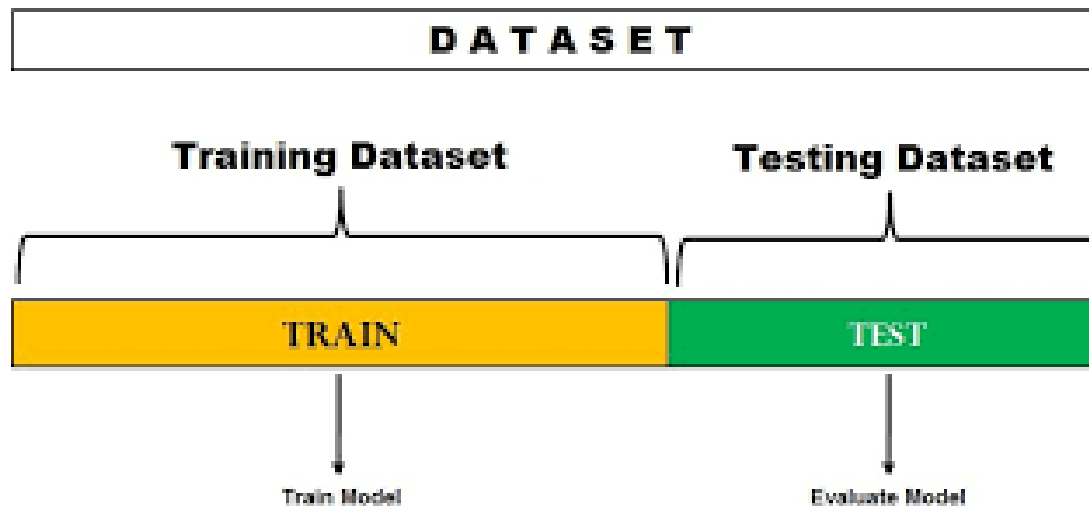
- **Holdout**
- **Random splitting**
- **Stratified random selection**
- **Leave one out**
- **K-fold**
- **Bootstrap**



# Data Splitting

The correct division between train and test is essential to obtain a correct approach to the effectiveness of the model.

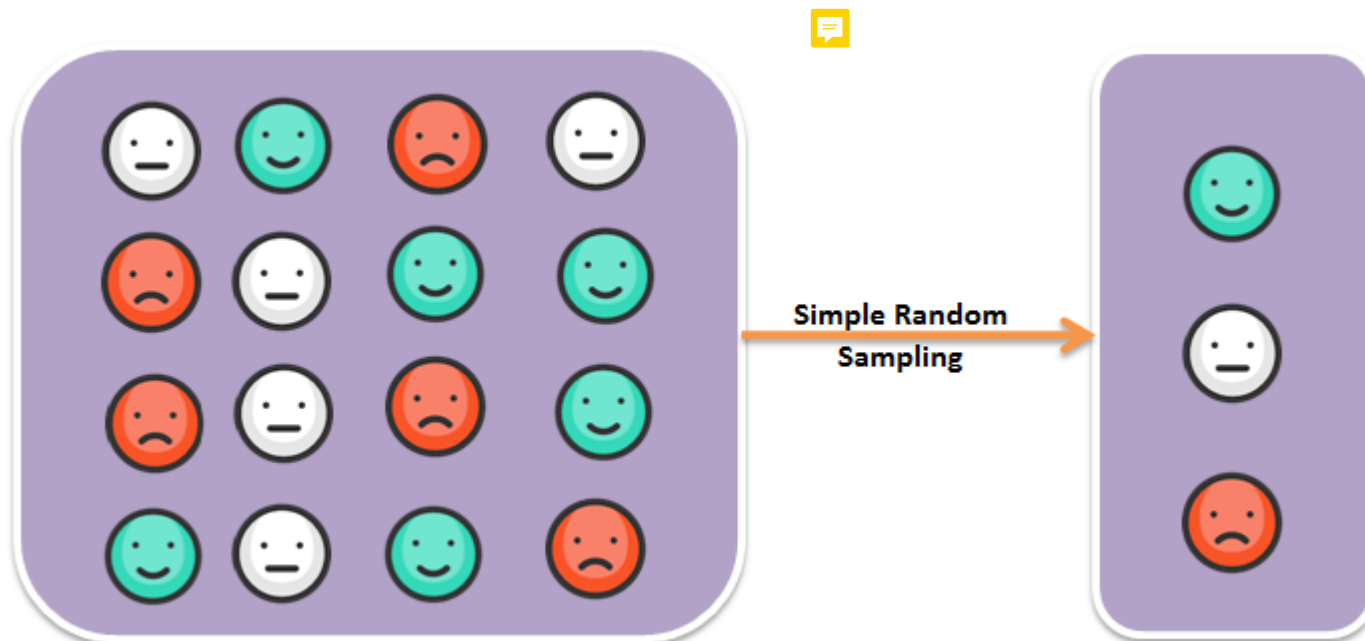
- **Holdout:** Define  $x\%$  for training and  $(100-x)\%$  for test.



# Data Splitting

The correct division between train and test is essential to obtain a correct approach to the effectiveness of the model.

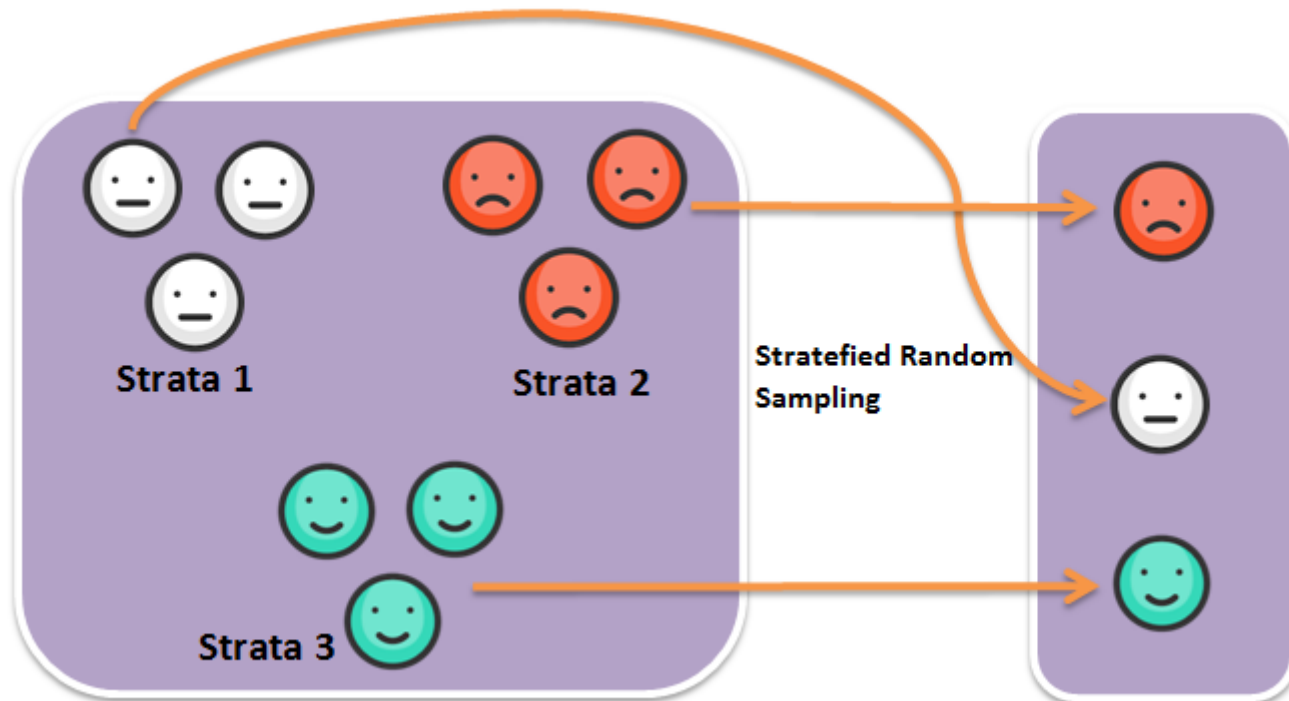
- **Random splitting:** randomly select samples to the train set.
  - If the dataset is homogeneous, it will produce an homogeneous and representative train and test sets of all classes.



# Data Splitting

The correct division between train and test is essential to obtain a correct approach to the effectiveness of the model.

- **Stratified random selection:** random selection of samples in a similar proportion of the size of the class on the amount of data. (Random selection within each class).
  - There is a higher likelihood that the outcome distributions will match.

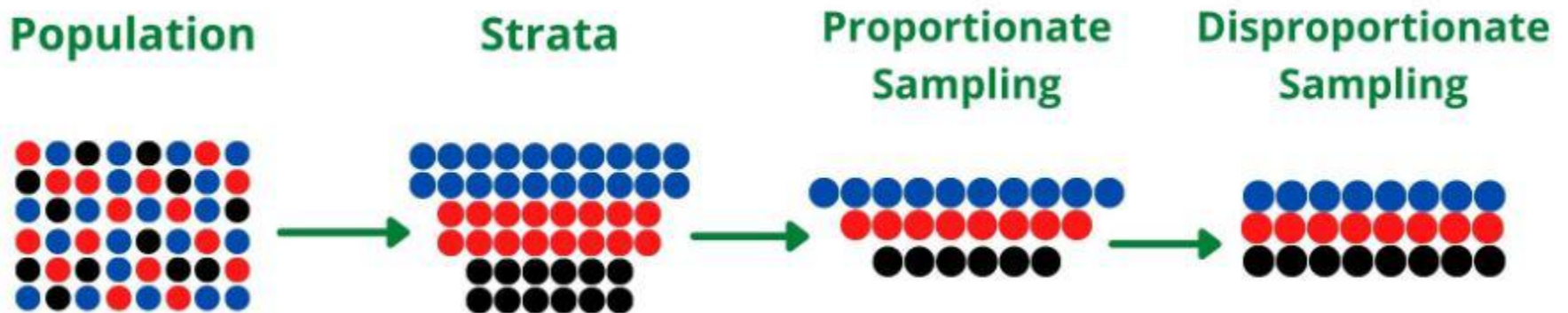




# Data Splitting 🗨️

The correct division between train and test is essential to obtain a correct approach to the effectiveness of the model.

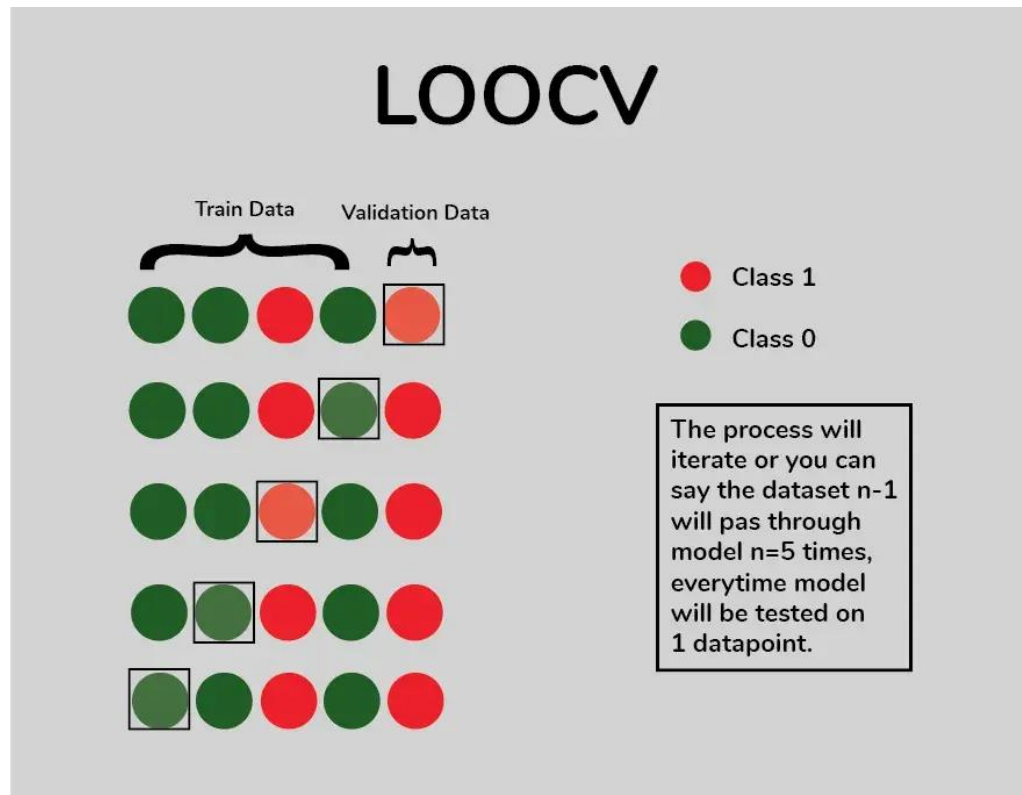
- **Stratified random selection:** random selection of samples in a similar proportion of the size of the class on the amount of data. (Random selection within each class).
  - There is a higher likelihood that the outcome distributions will match.



# Data Splitting

The correct division between train and test is essential to obtain a correct approach to the effectiveness of the model.

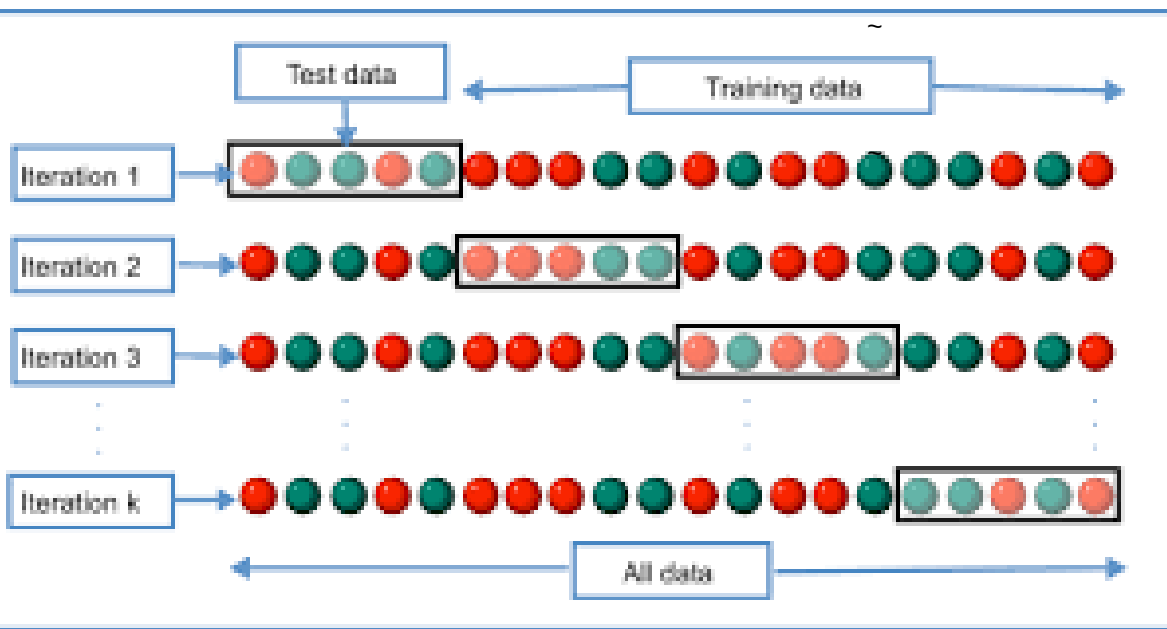
- **Leave one out:** when the dataset is small, use  $N-1$  samples to train the model, and test in the remaining one. Run the method until all samples were evaluated as test.



# Data Splitting

The correct division between train and test is essential to obtain a correct approach to the effectiveness of the model.

- **K-fold:** the training subset is defined by  $k-1$  partitions of the data, and the test is the remaining one. It should be repeated until all partitions were evaluated as test.



- The final validation error is the average error of  $K$  trainings.
- Choose the best model or the best hyper-parameter the one that minimises the validation error.

# Data Splitting

The correct division between train and test is essential to obtain a correct approach to the effectiveness of the model.

- **Bootstrap:** the training subset is defined by sampling with replacement.
  - The bootstrap sample is the same size as the original data set.
  - Some samples will be represented multiple times in the bootstrap sample while others will not be selected at all (“out-of-bag”).
  - If the training set size is small, this bias may be problematic, but will decrease as the training set sample size becomes larger.



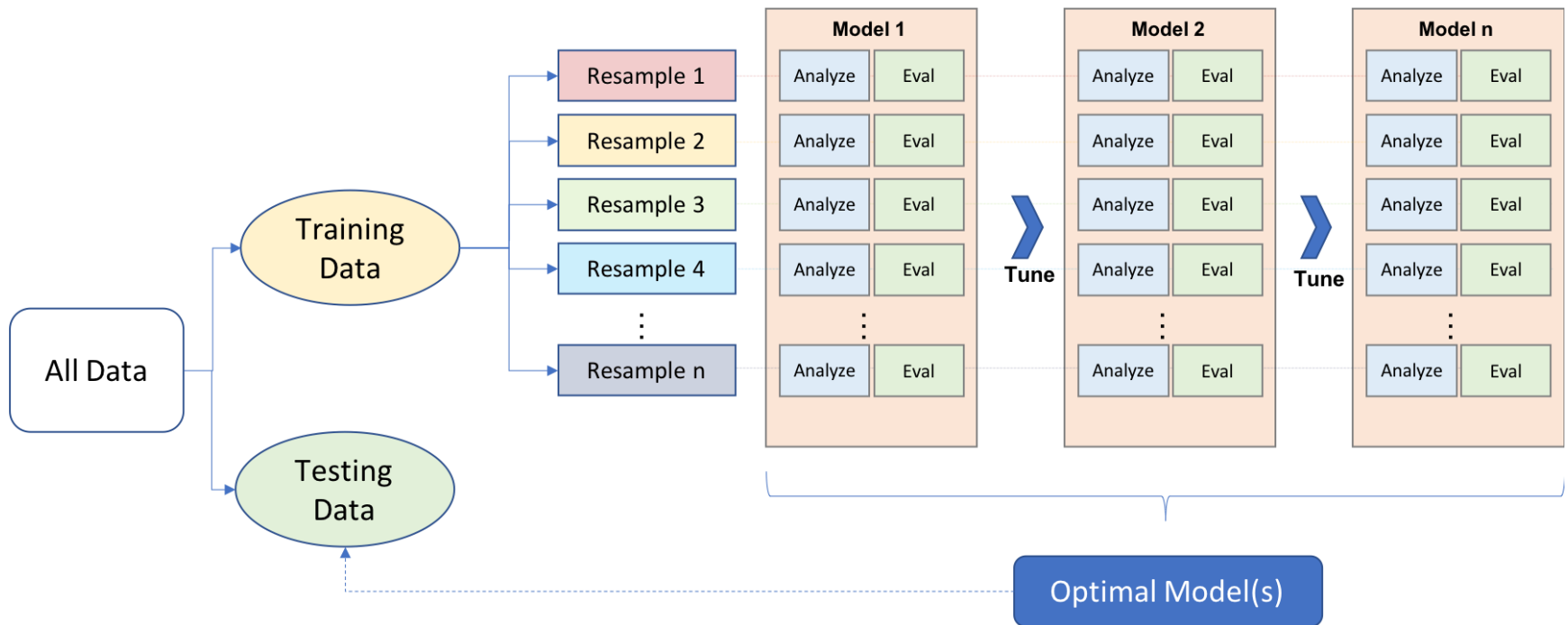
# Data Splitting

The correct division between train and test is essential to obtain a correct approach to the effectiveness of the model.

- **Holdout:** Define  $x\%$  for training and  $(100-x)\%$  for test.
- **Random splitting:** randomly select samples to the train set.
  - If the dataset is homogeneous, it will produce an homogeneous and representative train and test sets of all classes.
- **Stratified random selection:** random selection of samples in a similar proportion of the size of the class on the amount of data. (Random selection within each class).
  - There is a higher likelihood that the outcome distributions will match.
- **Leave one out:** when the dataset is small, use  $N-1$  samples to train the model, and test in the remaining one. Run the method until all samples were evaluated as test.
- **K-fold:** the training subset is defined by  $k-1$  partitions of the data, and the test is the remaining one. It should be repeated until all partitions were evaluated as test.
- **Bootstrap:** the training subset is defined by sampling with replacement.

Another important aspect of a data splitting technique is the uncertainty (i.e., variance or noise). An unbiased method may be estimating the correct value (e.g., the true theoretical performance) but may pay a high price in uncertainty. This means that repeating the data splitting procedure may produce a very different value (but done enough times, it will estimate the true value).

# The modeling process



# Model / hyper parameter selection

**Step 1:** Optimize parameters  $\theta$  (to minimize some cost function  $J$ ) using the same training set for all models. Compute some perf. metrics with the training data (i.e. error, accuracy) :

Training error =>

$$E_{train}(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2 \right]$$

**Step 2:** Test the optimized models from step 1 with the CV set and choose the model with the min CV error (or other performance metric with dev data):

Cross validation (CV)/dev error =>

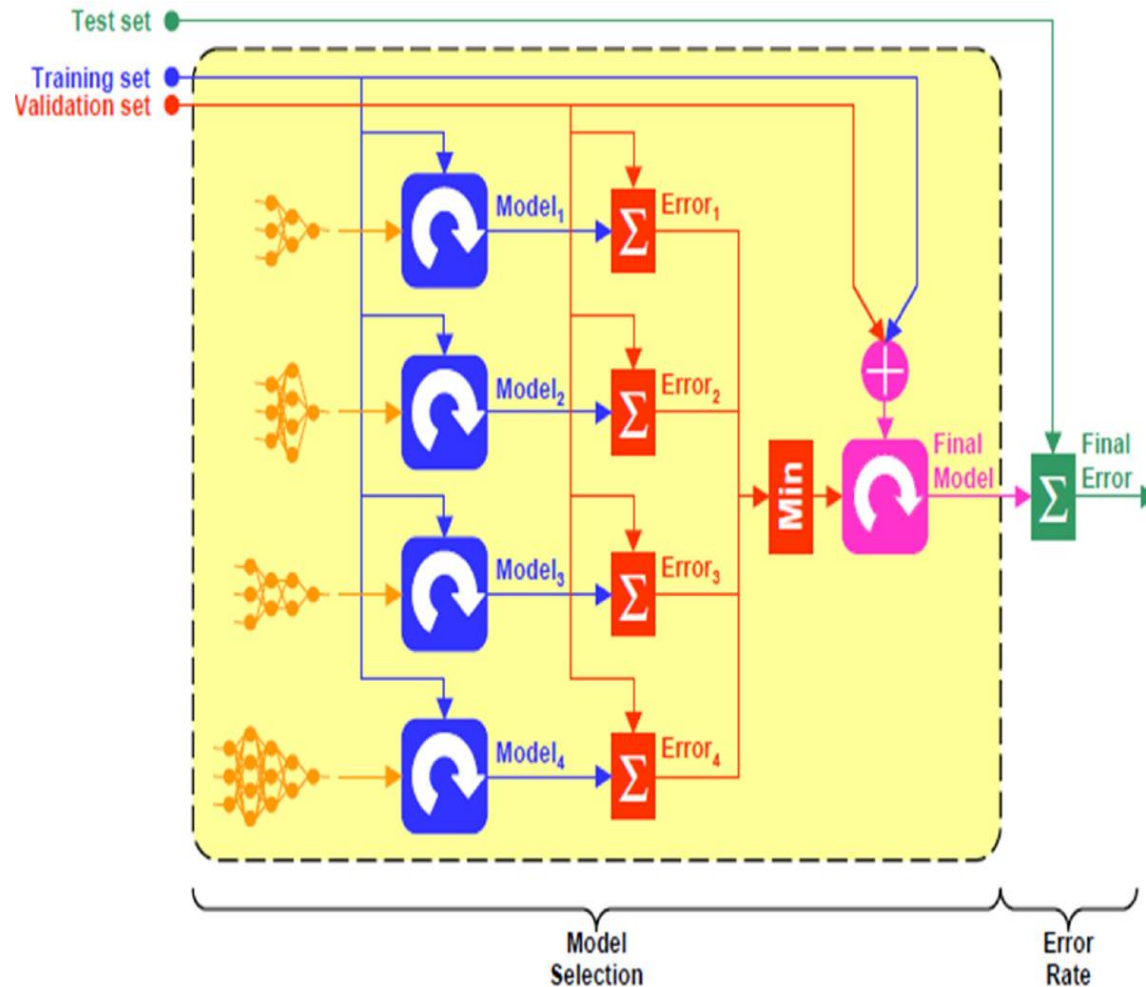
$$E_{cv}(\theta) = \frac{1}{2m_{cv}} \left[ \sum_{i=1}^{m_{cv}} \left( h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2 \right]$$

**Step 3:** Retrain the best model from step 2 with both train and CV sets starting from the parameters got at step 2. Test the retrained model with test set and compute test data perf. metric (**the real model performance !!!**):

Test error =>

$$E_{test}(\theta) = \frac{1}{2m_{test}} \left[ \sum_{i=1}^{m_{test}} \left( h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)} \right)^2 \right]$$

# Training/Valid (Dev)/Test subsets



The most credible outcome of the model training is the performance metric with the test data, not used for training or validation of the model.



# The modeling process

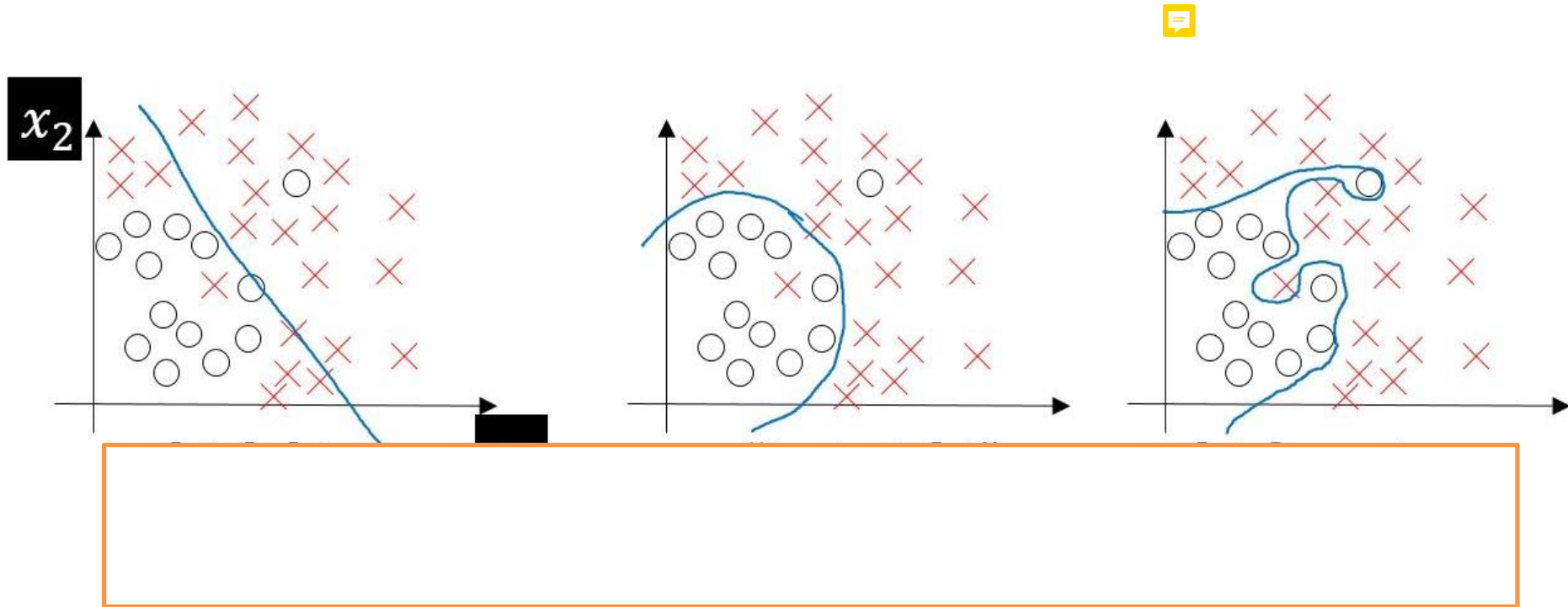
- The optimal model:
  - The simplest approach is to pick the settings associated with the numerically best performance estimates.
  - In general, it may be a good idea to favor simpler models over more complex ones and choosing the tuning parameters based on the numerically optimal value may lead to models that are overly complicated.
- Considerations:
  - Understand what tuning machine learning model is
  - Find Your Score Metric
  - Obtain Accurate Forecasting Score (data split)
  - Diagnose Best Parameter Value Using Validation Curves (cost vs parameter value or cost vs performance metric)
  - Use Grid Search To Optimise Hyperparameter Combination

# The modeling process

- Data Splitting Recommendations
  - A test set is a single evaluation of the model and has limited ability to characterize the uncertainty in the results
  - Proportionally large test sets divide the data in a way that increases bias in the performance estimates.
  - With small sample sizes:
    - The model may need every possible data point to adequately determine model values.
    - The uncertainty of the test set can be considerably large to the point where different test sets may produce very different results.
  - Data Splitting techniques can produce reasonable predictions of how well the model will perform on future samples.
  - No Data Splitting technique is uniformly better than another; the choice should be made while considering several factors:
    - If the samples size is small: repeated 10-fold cross-validation for several reasons: the bias and variance properties are good and, given the sample size, the computational costs are not large.
    - If the goal is to choose between models, as opposed to getting the best indicator of performance, a strong case can be made for using one of the bootstrap procedures since these have very low variance.
    - For large sample sizes, the differences between Data Splitting techniques become less pronounced, and computational efficiency increases in importance. Here, simple 10-fold cross-validation should provide acceptable variance, low bias, and is relatively quick to compute.

# **Bias vs Variance**

# Bias vs. Variance ?

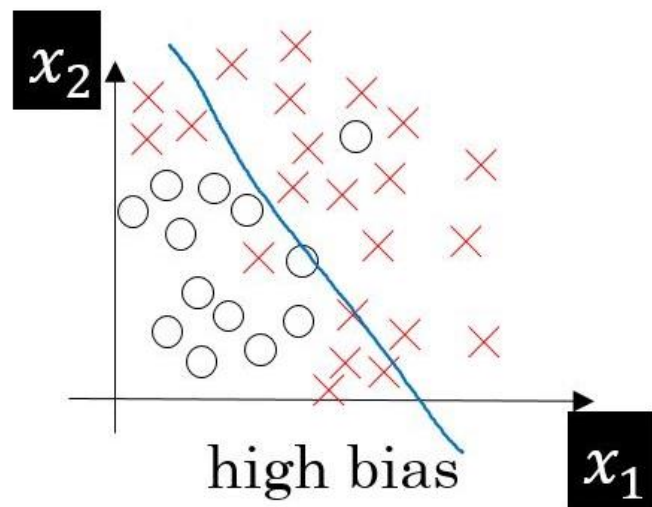


# Bias vs. Variance

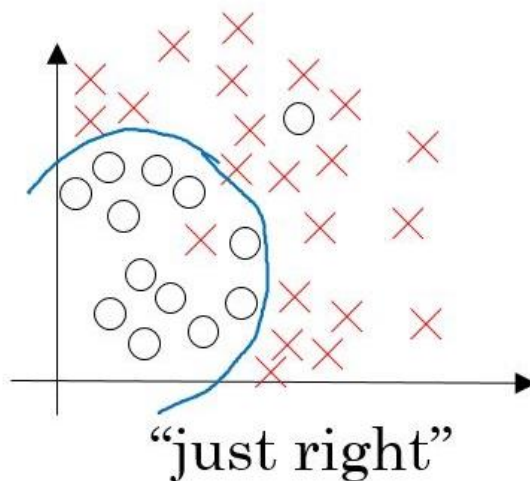
An important concept in ML is the bias-variance tradeoff.

Models with **high bias** are not complex enough and **underfit** the training data.

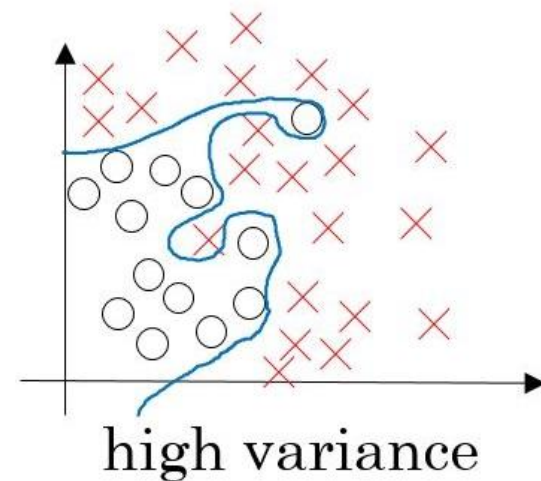
Models with **high variance** are too complex and **overfit** the training data.



**underfitting data**  
(very simple model)



(good model)



**overfitting data**  
(very complex model)

# Diagnosing Bias vs. Variance

How to diagnose if we have a high bias problem or high variance problem ?

## **High Bias (underfitting) problem:**

Training error ( $E_{train}$ ) and Validation/dev error ( $E_{cv}$ ) are both high

## **High Variance (overfitting) problem:**

Training error ( $E_{train}$ ) is low  
and Validation/dev error ( $E_{cv}$ ) is much higher than  $E_{train}$

# Bias vs. Variance

## High Bias (underfitting) :

- Increasing model complexity allows your model to capture more intricate patterns in the data.
- Adding relevant features gives your model more information to learn from.
- Reducing regularization provides your model more freedom to learn complex patterns.
- Implementing boosting algorithms like XGBoost or AdaBoost can systematically improve your model's performance.

## High Variance (overfitting) :

- Increasing your training data helps your model learn more generalizable patterns.
- Implementing stronger regularization constrains your model's complexity.
- Reducing model complexity helps prevent memorization of training data.
- Using bagging algorithms like Random Forests can effectively reduce variance through ensemble learning.

The relationship  
between bias and  
variance

bias-variance trade-off

# Optimize the Bias/ Variance balance

- Start Simple: Begin with a simple model to establish a baseline.
- Monitor Performance: Use cross-validation to track training and validation metrics.
- Adjust Complexity: Gradually increase complexity while observing changes in bias and variance.
- Use Regularization: Apply techniques like dropout or L2 regularization.
- Analyze Learning Curves: Identify whether adding more data or complexity improves performance.

Choose your optimization strategy based on (suggestion):

- Use boosting techniques when dealing with high bias
- Apply bagging methods when fighting high variance
- Consider ensemble approaches for complex scenarios



# Learning Curves

# Choose regularization parameter $\lambda$

**For a given model, try different values of  $\lambda = [0, 0.01, 0.1, 1, \dots]$**

**Step 1:** For each  $\lambda$ , optimize parameters  $\theta$  using the training set

$$E_{train}(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2 \right]$$

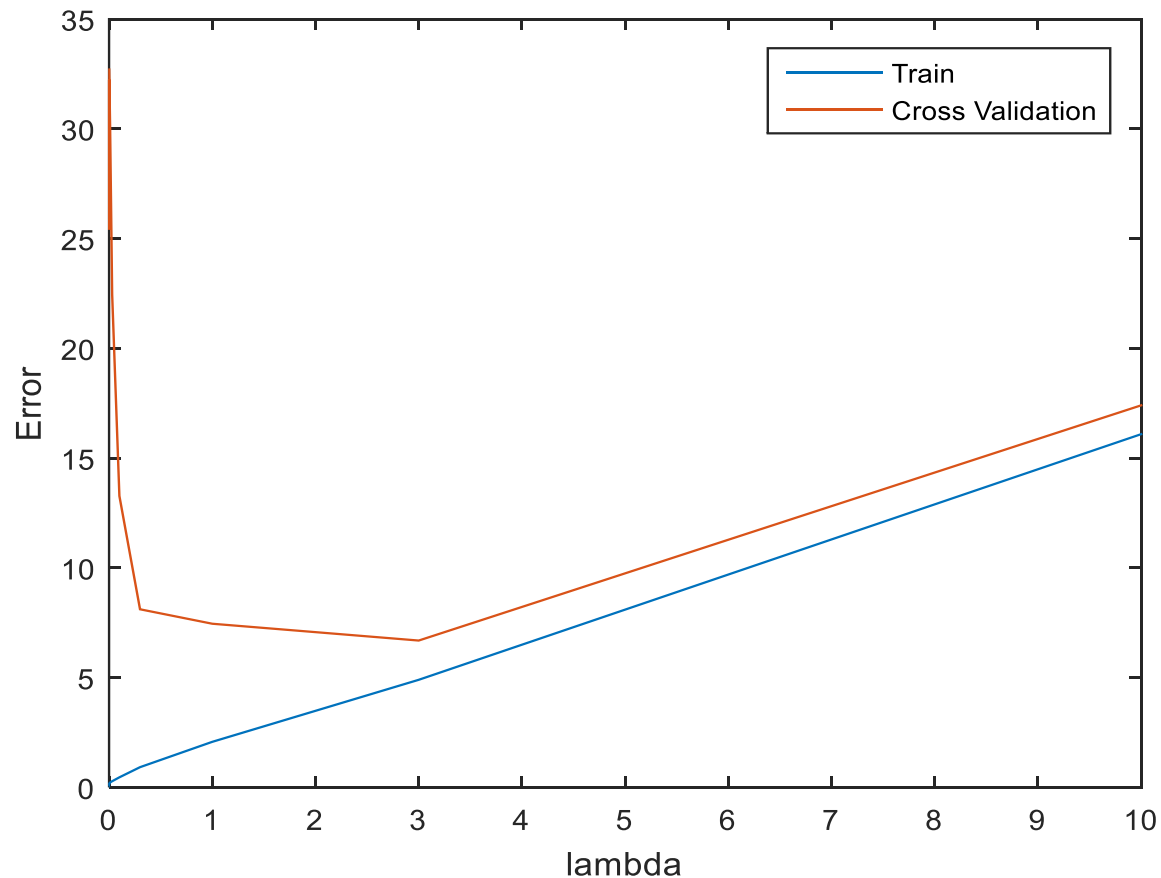
**Step 2:** Test the optimized models from step 1 with the CV set and choose the model with  $\lambda$  that gets min CV error:

$$E_{cv}(\theta) = \frac{1}{2m_{cv}} \left[ \sum_{i=1}^{m_{cv}} \left( h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2 \right]$$

**Step 3:** Retrain the model with best  $\lambda$  from step 2 with both train and CV sets starting from the parameters  $\theta$  got at step 2. Test the retrained model with the test set and compute the error:

$$E_{test}(\theta) = \frac{1}{2m_{test}} \left[ \sum_{i=1}^{m_{test}} \left( h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)} \right)^2 \right]$$

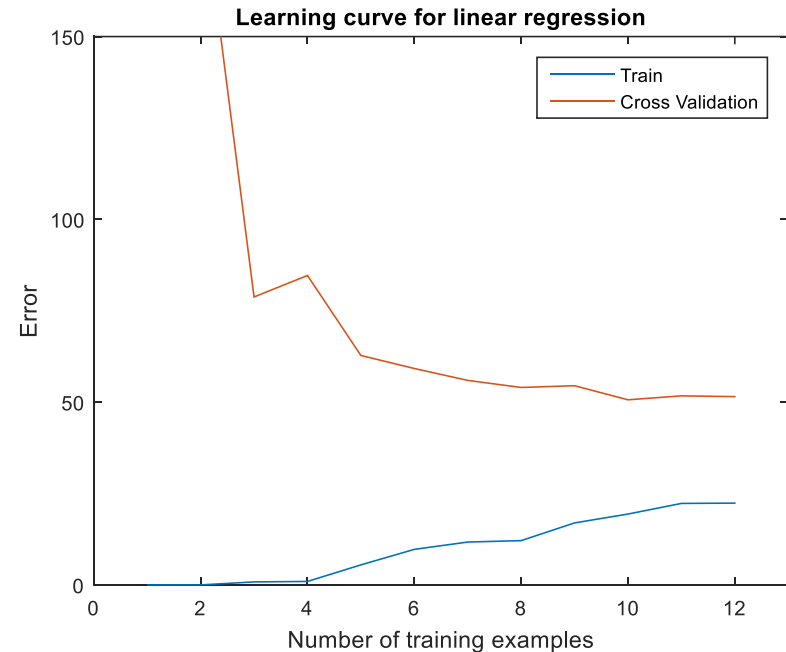
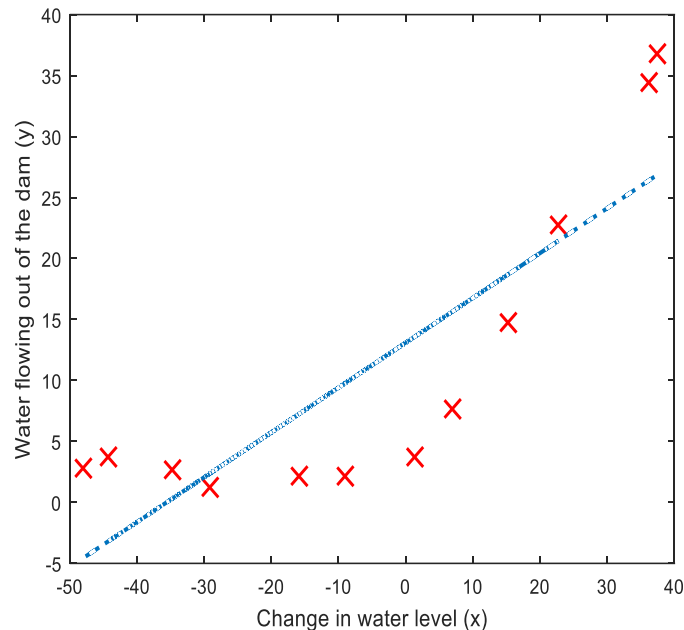
# Select $\lambda$ using CV(dev) set



**Best  $\lambda = 3$**

# Learning Curves

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

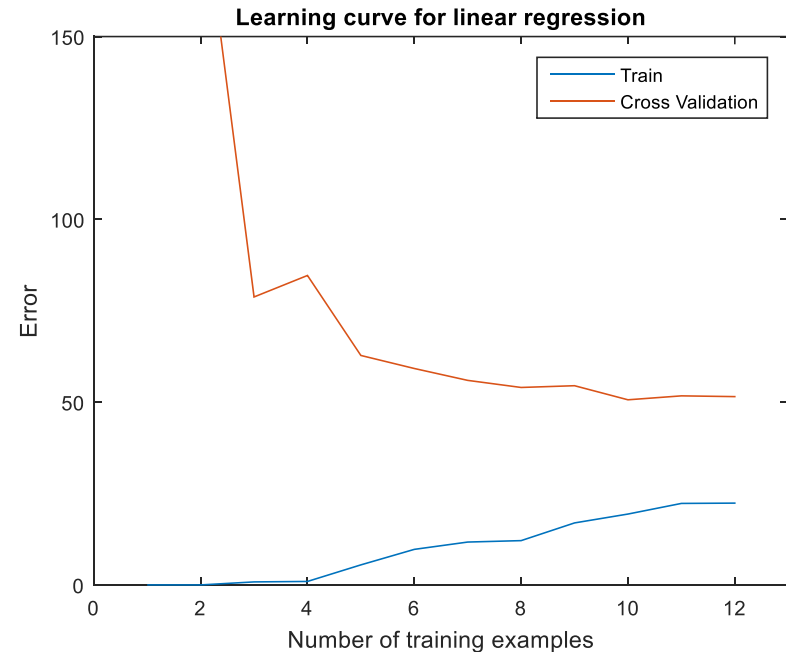
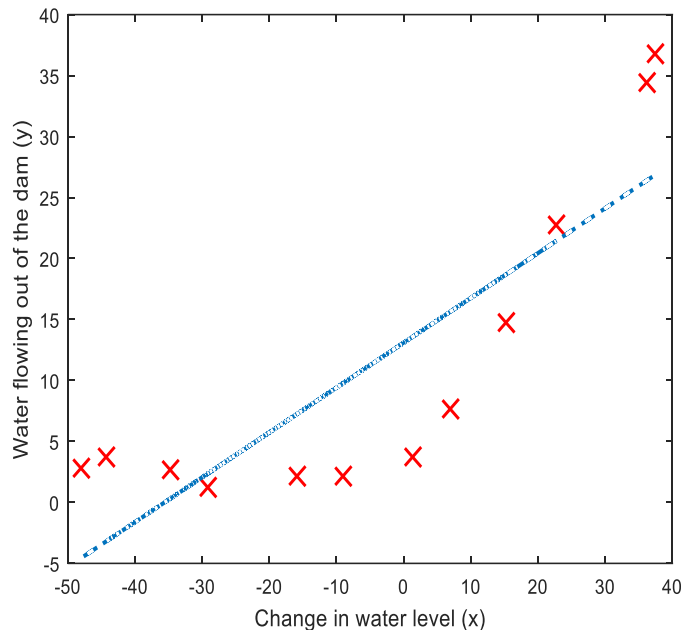


**What is the problem: high bias or high variance ?**

Learning curves are plots that display the performance of a machine learning model as the number of training samples increases. We use them to evaluate the model's ability to generalize to new, unseen data, and to identify issues, such as overfitting and underfitting.

# Learning Curves

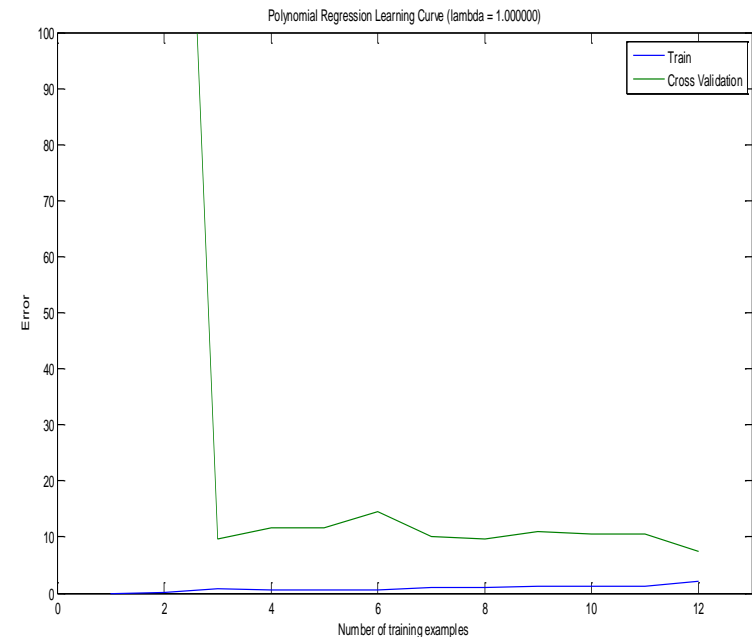
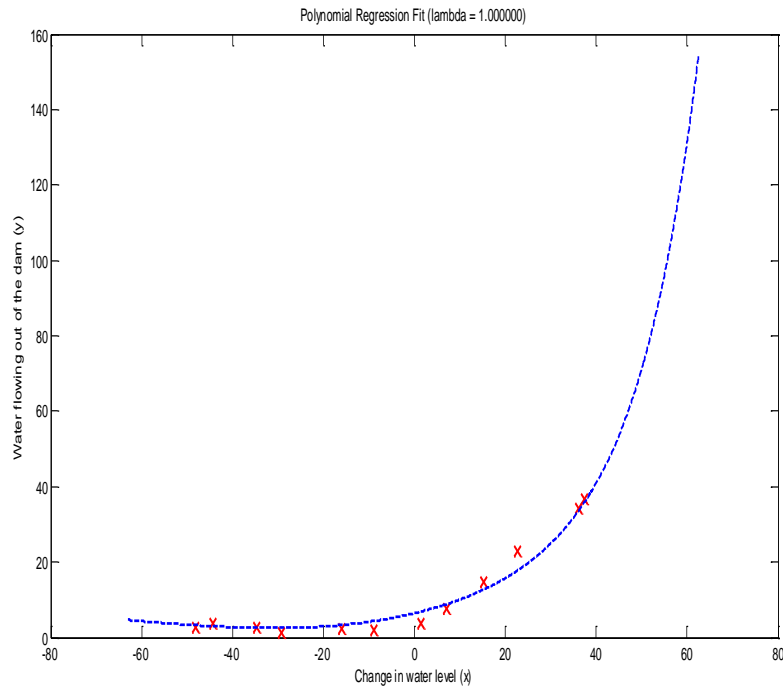
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



**If a learning algorithm is suffering from high bias, getting more training data will not help much**

Underfitting. In this case, the training and validation curves can even converge to a plateau, but the plateau is far from the desired value of the cost function.

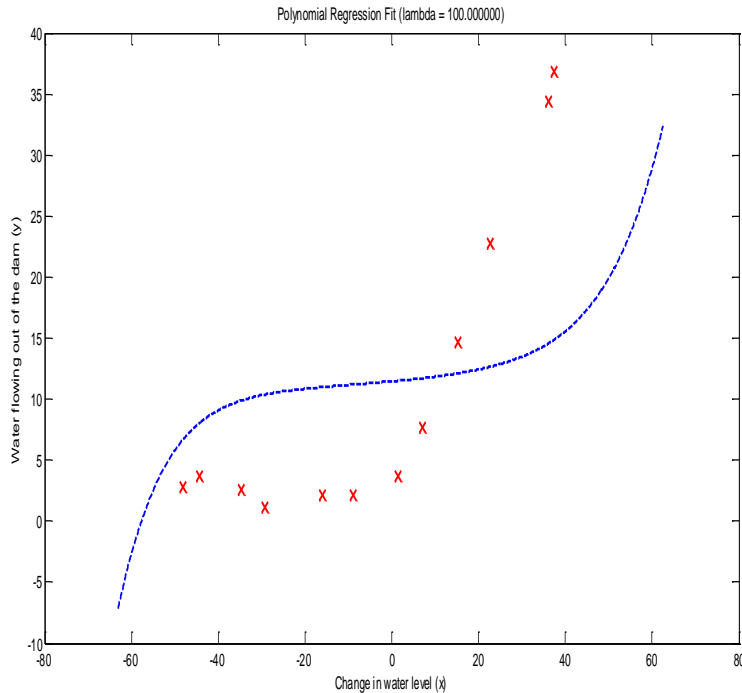
# Learning Curves



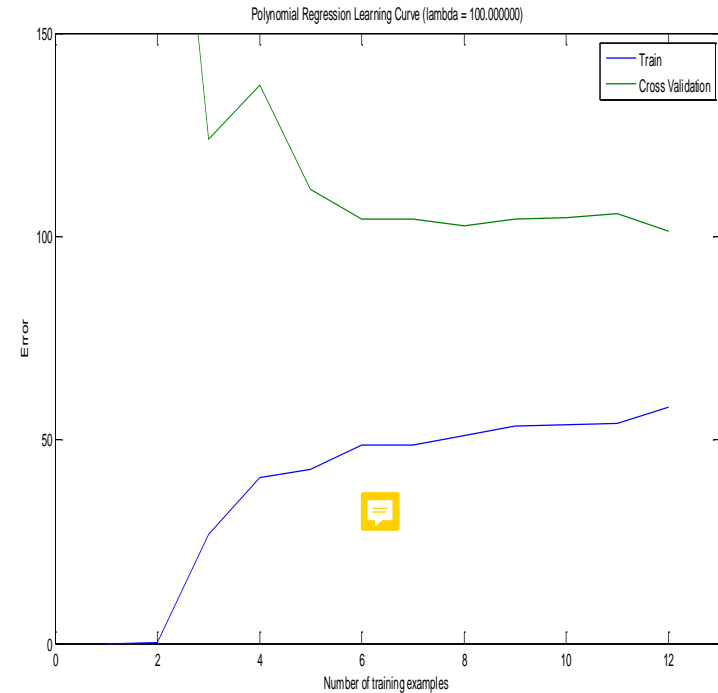
**If a learning algorithm is suffering from high variance, getting more training data is likely to help**

Overfitting. The two curves can even converge to a plateau, but there is a huge gap between them showing that there is a high difference in the value of the cost function chosen.

# Regularization and Learning Curves




**Polynomial regression,  $\lambda = 100$**



**Learning curve,  $\lambda = 100$**

# Hints to improve ML model

Suppose you have learned a data model (hypothesis). However, when you test your hypothesis on a new set of data, you find that it makes unacceptably large errors in its prediction (regression or classification). What should you try next? 

- **Get more training examples – fixes high variance**
- **Try smaller sets of features – fixes high variance**
- **Try getting additional features – fixes high bias**
- **Try adding polynomial features - fixes high bias**
- **Try decreasing  $\lambda$  – fixes high bias**
- **Try increasing  $\lambda$  – fixes high variance**



# Hints to improve ML model

Suppose you have learned a data model (hypothesis). However, when you test your hypothesis on a new set of data, you find that it makes unacceptably large errors in its prediction (regression or classification). What should you try next?

Underfit -> high bias

A regression problem where the model is experiencing unacceptably large errors on new data.

Probably the model is underfitting (high bias):

- the model isn't complex enough to capture the underlying patterns in the data
- resulting in high bias and poor performance both on the training set and new data
- In the case of regression, this would manifest as large prediction errors

# Hints to improve ML model

Suppose you have learned a data model (hypothesis). However, when you test your hypothesis on a new set of data, you find that it makes unacceptably large errors in its prediction (regression or classification). What should you try next?


Overfit -> high variance

A regression problem where the model is experiencing unacceptably large errors in prediction when testing on new data.

In an overfitting scenario, a model learns the noise and specific details in the training data so well that it doesn't generalize to new, unseen data. This is often characterized by:

- Low error on the training data (because the model fits it too closely)
- High error on new test data (because the model overfits to the training set's peculiarities and doesn't generalize well).

# Hints to improve ML model

Suppose you have learned a data model (hypothesis). However, when you test your hypothesis on a new set of data, you find that it makes unacceptably large errors in its prediction (regression or classification). What should you try next? 

The description primarily suggests underfitting (high bias), but it could also suggest overfitting (high variance).

To distinguish between the two:

- If your model performs poorly on both training and test data, it is underfitting (high bias).
- If your model performs well on training data but poorly on new data, it is more likely overfitting (high variance).

# Ensemble Classifiers

# Ensemble (Committee) Classifiers

- Learn a set of classifiers (ensemble, committee) and combine (somehow) their predictions.

Build “weak” classifiers, do not try too hard to find the best classifier. Choose classifiers with different nature (deterministic, probabilistic, linear, nonlinear)

- **MOTIVATION:**

- reduce the variance ( results are less dependent on specificity of training data)
- reduce the bias (multiple classifiers means different models)

# Combining Predictions

## 1. Voting

- each classifier gives its vote
- predict the class with the majority of votes (bagging)

## 2. Weighted voting

- make a *weighted* sum of the votes of the ensemble classifiers
- weights depend on the classifier error (boosting)

## 3. Stacking

- use a higher level (meta) classifier to make the final decision.
- the meta classifier has as inputs the predictions of the ensemble classifiers and the outputs are the class labels

# Ensemble classification- Bagging

## Algorithm Idea:

- a) Take some of the examples from the original training data (with replacement) but keep the size equal to the original data set.
- b) Train M classifiers (preferably different type of models e.g. Log Regr., SVM, DTree etc. ) with different Bags.
- c) Test all classifiers with the test examples.
- d) Assign the class that receives majority of votes.

Variations are possible -e.g., size of subset, sampling w/o replacement, etc.

The models in bagging are independent and run in parallel. Once each model is trained, their results are aggregated to produce the final outcome of the ensemble algorithm.

## Advantages:

- Reduces overfitting
- Highly robust against noise in class labels
- Generally improves the performance of the base classifier
- Easily parallelized

# Ensemble classification- Boosting

## Algorithm Idea:

### 1.Sequential Training:

- In the first iteration, all data points are assigned equal weights, and a model is created.
- Based on this model's performance, the weights are adjusted:
  - the weights of misclassified samples are increased (this ensures that they will become more important)
  - the weights of correctly classified points are decreased
  - weight the predictions of the classifiers with their error
- This process repeats for a specified number of iterations or until the model reaches a desired level of accuracy.

**2.Final Output:** The predictions from all models are combined to produce a final output for the ensemble.

Examples of Ensemble Classifiers : Random Forest ( set of Decision Trees)

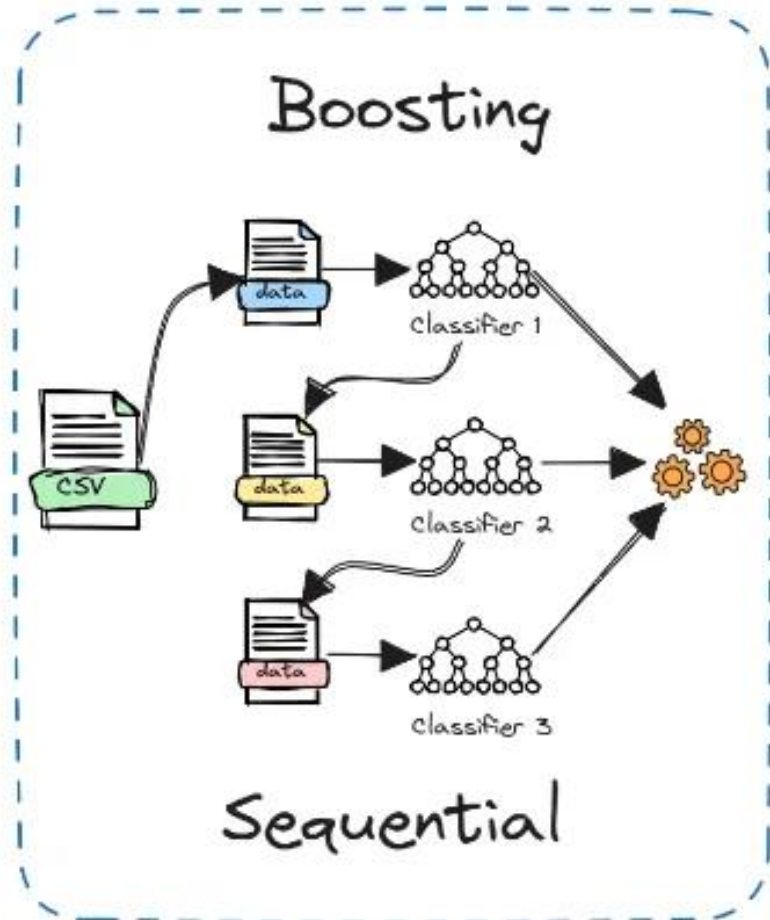
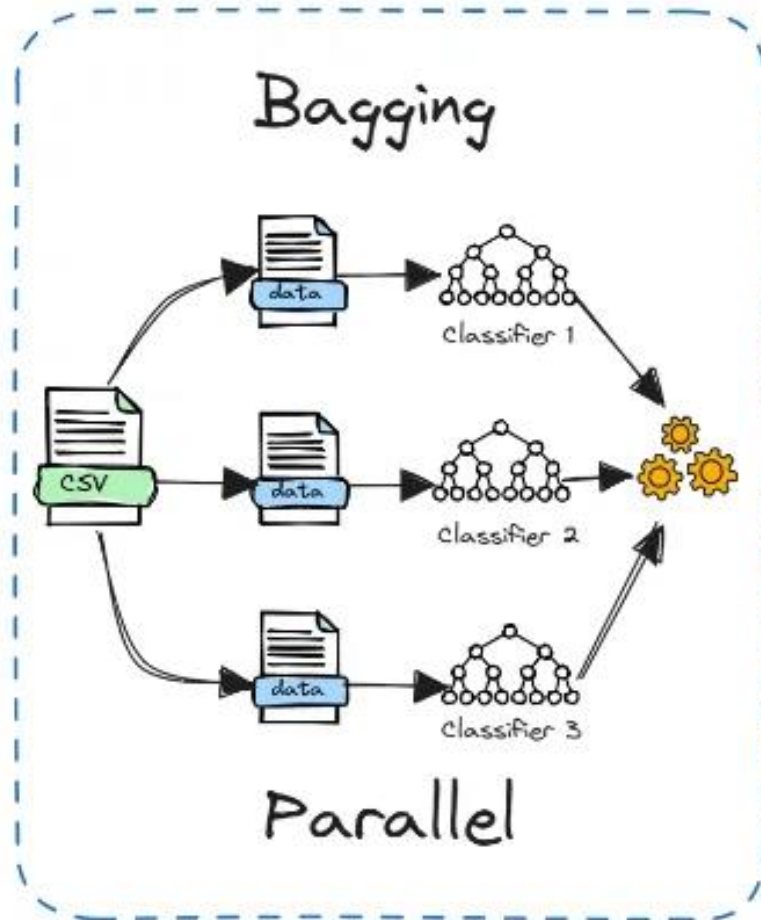
Boosting follows a sequential process where each model attempts to correct the errors of the previous one.

### Considerations:

- Generally achieves very high accuracy.
- Not robust against noise in class labels.
- May increase the error of the base classifier if poorly tuned.
- Difficult to parallelize due to its sequential nature.



# Bagging vs Boosting



# STACKING (hierarchical) - Example

Attributes			Class
$x_{11}$	...	$x_{1n_a}$	$t$
$x_{21}$	...	$x_{2n_a}$	$f$
...	...	...	...
$x_{n_c1}$	...	$x_{n_cn_a}$	$t$

(a) training set

$C_1$	$C_2$	...	$C_{n_c}$
$t$	$t$	...	$f$
$f$	$t$	...	$t$
...	...	...	...
$f$	$f$	...	$t$

(b) predictions of the classifiers

- Train the lower level classifiers  $C_1...C_{n_c}$   
 $t$ - true;  $f$ - false

$C_1$	$C_2$	...	$C_{n_c}$	Class
$t$	$t$	...	$f$	$t$
$f$	$t$	...	$t$	$f$
...	...	...	...	...
$f$	$f$	...	$t$	$t$

(d) training set for stacking

- Form a feature vector consisting of their predictions
- Train the meta classifier with these feature vectors

# **Model-centric vs Data-centric ML**

# Deciding what to do next ?

Suppose you have trained a ML model on some data. When you test the trained model on a new set of data, it makes unacceptably large errors. What should you do ?

- **Get more training examples ?**
- **Try smaller sets of features (feature selection) ?**
- **Try getting additional features (feature engineering) ?**
- **Try using different/nonlinear kernels ?**
- **Try other values of the hyper parameters (e.g. regul. parameter) ?**

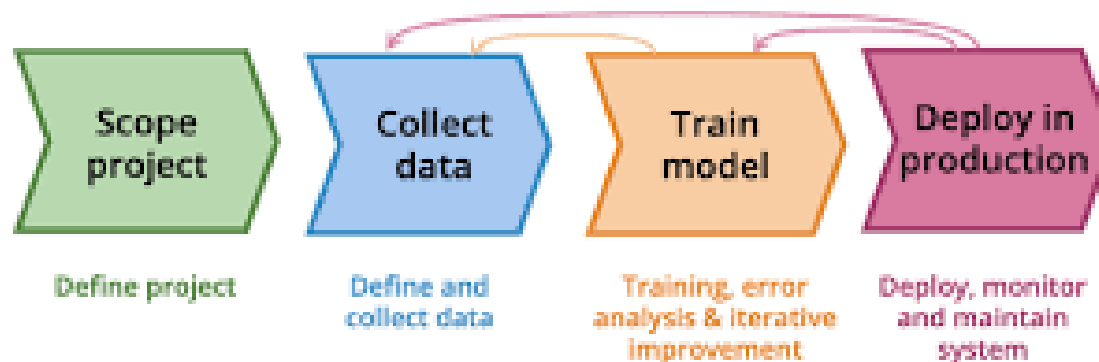
## **Machine learning diagnostics = Model-centric approach**

Run tests to gain insight what isn't working with the learning algorithm and how to improve its performance.

Diagnostics is time consuming, but can be a very good use of your time.

# ML Project

## Lifecycle of an ML Project



### Model-centric

- Collect as much data as we can
- Optimize the model so it can deal with the noise in the data

#### Approach:

- Data is fixed after standard preprocessing
- Model is improved iteratively

### Data-centric

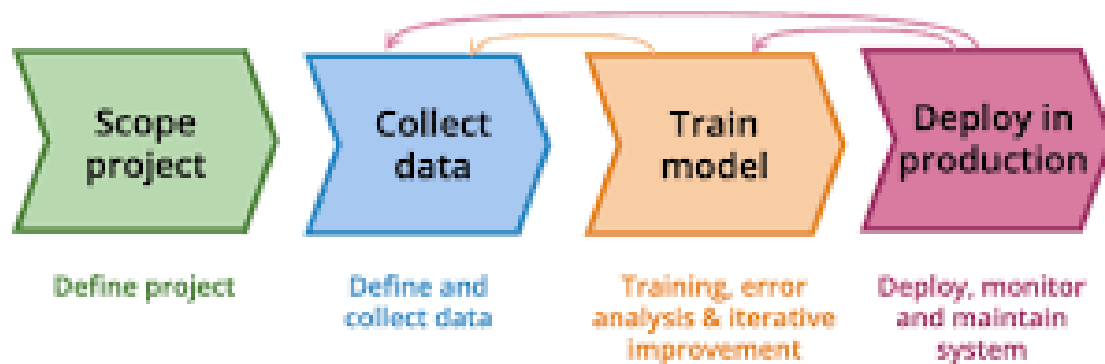
- Data consistency is key
- Higher investment in data quality tools rather collecting more data
- Allows more models to do well

#### Approach:

- Hold the code/algorithms fixed
- Iterated the data quality

# Train ML model for Speech Recognition

## Lifecycle of an ML Project



Error analysis shows your algorithm does poorly in speech with car noise in the background. What to do?

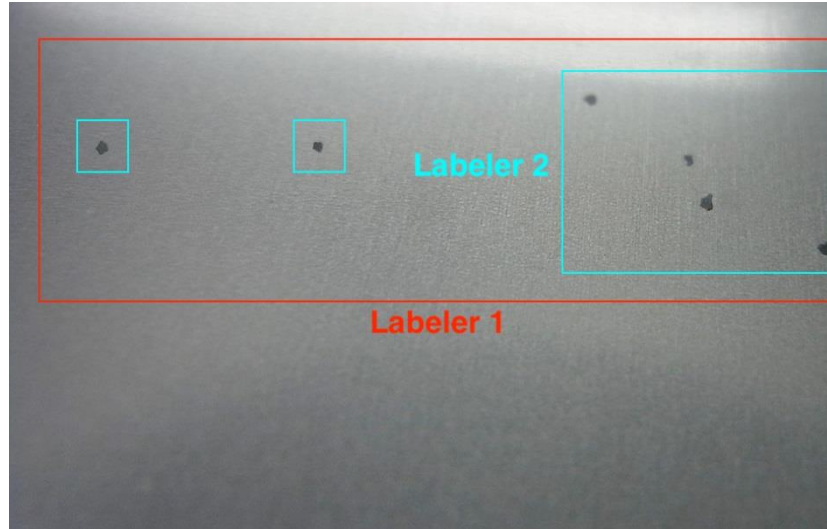
### **Model centric view:**

Try hard to tune the model architecture to improve performance

### **Data-centric view:**

Modify the data (get new examples relevant for the car noise scenario) to improve performance.

# Data centric approach - label consistency



## Labelling instructions:

Use bounding boxes to indicate the positions of the defects.

Is the data labelled consistently ?

Ask two independent labellers to label a sample of images.

Measure consistency between labellers to discover where they disagree.

For classes where the labellers disagree, revise the labelling instructions until they become consistent.

# Clean vs. noisy data

You have 500 examples, and 12% of them are noisy (incorrectly or inconsistently labelled)

The following are **about equally effective**:

- Clean up the noise
- Collect another 500 new **noisy** examples (double the **noisy** training set)

Apply data centric view => significant improvements





# Takeaways: Data-centric AI

For ML engineer /Data scientist

## Model-centric AI

How can you change the model (code) to improve performance

## Data-centric AI

How can you systematically change the data (input X or labels y) to improve performance

Data centric **doesn't only** mean using the **right high quality data to train your model**, but it also means:

1. using data to measure the model's success
2. perform error analysis driven iteration on that data to improve the model
3. maintain the model by monitoring the incoming data and the models performance on it.

# Takeaways: Data-centric AI

Make data-centric AI an efficient and systematic process.

**from BIG Data to Good Data** - ensure high quality data in all stages of the ML project lifecycle

## **Good data is:**

- Defined consistently (definition of labels  $y$  is unambiguous);
- Cover of important cases (good coverage of inputs  $X$ );
- Has timely feedback from production data (distribution covers data and concept drift).

WE'VE DECIDED  
TO TAKE BIG  
DATA TO THE  
NEXT LEVEL...

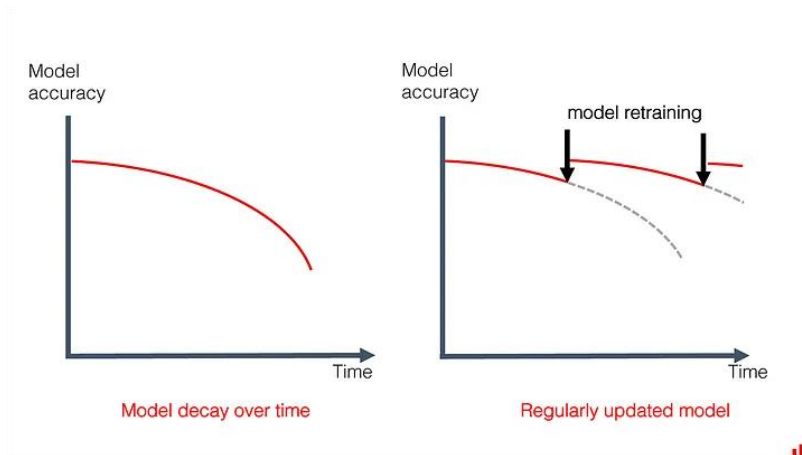


HUMONGOUS  
DATA

# Data and Concept Drift

No model lasts forever. Even if the data quality is fine, the model itself can start degrading.

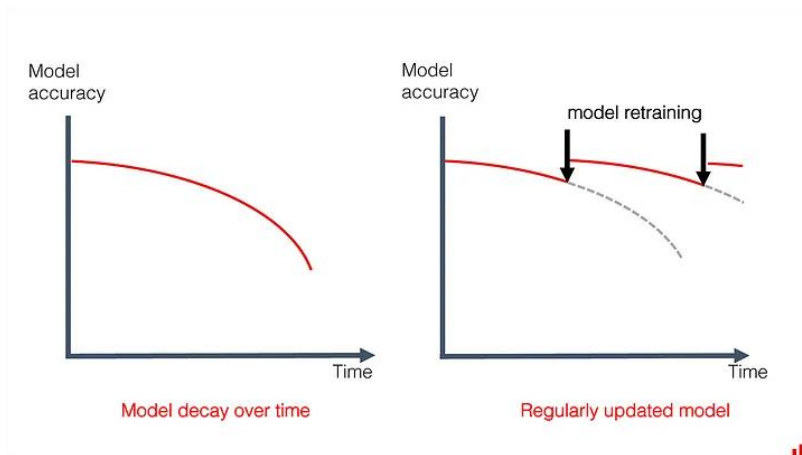
Some models can last years without an update. For example, certain computer vision or language models. Or any decision system in an isolated, stable environment. Others might need daily retraining on the fresh data.



# Data and Concept Drift

No model lasts forever. Even if the data quality is fine, the model itself can start degrading.

Some models can last years without an update. For example, certain computer vision or language models. Or any decision system in an isolated, stable environment. Others might need daily retraining on the fresh data.



**Data drift, feature drift, population, or covariate shift.**

Quite a few names to describe essentially the same thing. Which is: the input data has changed. The distribution of the variables is meaningfully different. As a result, the trained model is not relevant for this new data.

**Concept drift** occurs when the patterns the model learned no longer hold.

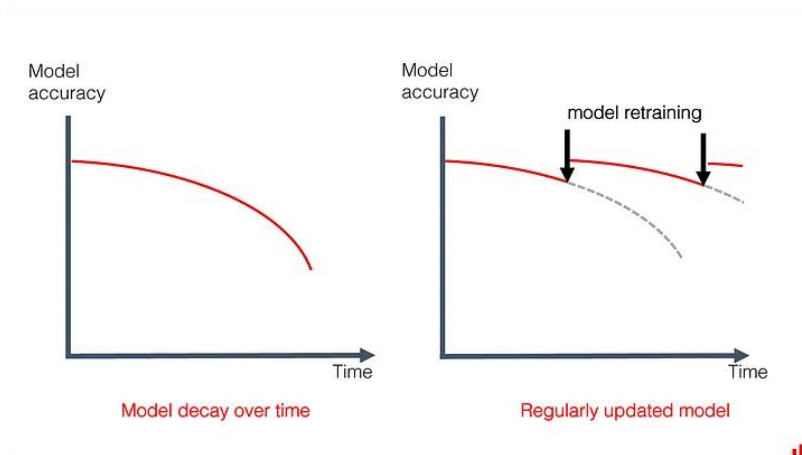
In contrast to the data drift, the distributions (such as user demographics, frequency of words, etc.) might even remain the same. Instead, the relationships between the model inputs and outputs change.

In essence, the meaning of what we are trying to predict evolves. Depending on the scale, this will make the model less accurate or even obsolete.

# Data and Concept Drift

No model lasts forever. Even if the data quality is fine, the model itself can start degrading.

Some models can last years without an update. For example, certain computer vision or language models. Or any decision system in an isolated, stable environment. Others might need daily retraining on the fresh data.



## All models degrade.

Sometimes, the performance drop is due to low data quality, broken pipelines, or technical bugs.

If not, consider:

- **Data drift:** change in data distributions. The model performs worse on unknown data regions.
- **Concept drift:** change in relationships. The world has changed, and the model needs an update. It can be gradual (expected), sudden (you get it), and recurring (seasonal).

In practice, the semantic distinction makes little difference. More often than not, the drift will be combined and subtle.

What matters is how it impacts the model performance, if retraining is justified, and how to catch this on time.