Departamento de Eletrónica, Telecomunicações e Informática

LECTURE 6:
MODEL SELECTION AND VALIDATION – BIAS VS. VARIANCE

Petia Georgieva
(petia@ua.pt)

universidade de aveiro

Lecture Outline

1. **Model selection: Bias vs. variance**
2. **K –fold Cross Validation**
3. **Learning curves**
4. **Ensemble classifiers**
5. **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.

Train & Test subsets (*holdout method*)

If you have chosen the model, split data into two sets:

- Training set (70%-80 %) : used to train the model
- Test set (30%-20%) : used to test the trained model

- **Optimize the model parameters with training data**
  (minimize some cost/loss function J)

After the training stage is over (i.e. the cost function J converged)

- Compute the MSE on test data (for regression problems)

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

or

- **Compute the model accuracy or some other metric from the confusion matrix, on test data (for classification problems)**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Mean Squared Error (MSE) is not the same as the cost function!!!

Different Cost/Loss Functions

Training data MSE

- **Linear Regression Cost Function** with L2 Regularization (Ridge Regression)

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

Ridge Regression

- **Logistic Regression Cost Function** with L2 Regularization

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

- **SVM Cost Function** with L2 Regularization

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

Mean Squared Error (MSE) is NOT the cost function!!!

3 way split: Train/Dev/Test Sets

Choose ML model: Logistic Regression, Neural Network (NN), etc. ?
Choose model hyper-parameters:
- # of layers in NN ?
- # of hidden units (neurons) in NN ?
- Which activation functions in NN ?
- What is the best learning rate ?
- What is the best regularization parameter (λ) ?
- What is the best polinomial degree ?
- ……

Devide dataset in 3 sub-sets:
- Training set
- Cross Validation (CV) set = Development set = 'dev' set
- Test set

Traditional division for Small data set (up to 10000 examples) :
60% - 20% - 20%

Big data (1 million. examples): 98% - 1% - 1%

Model /hyper parameter selection

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

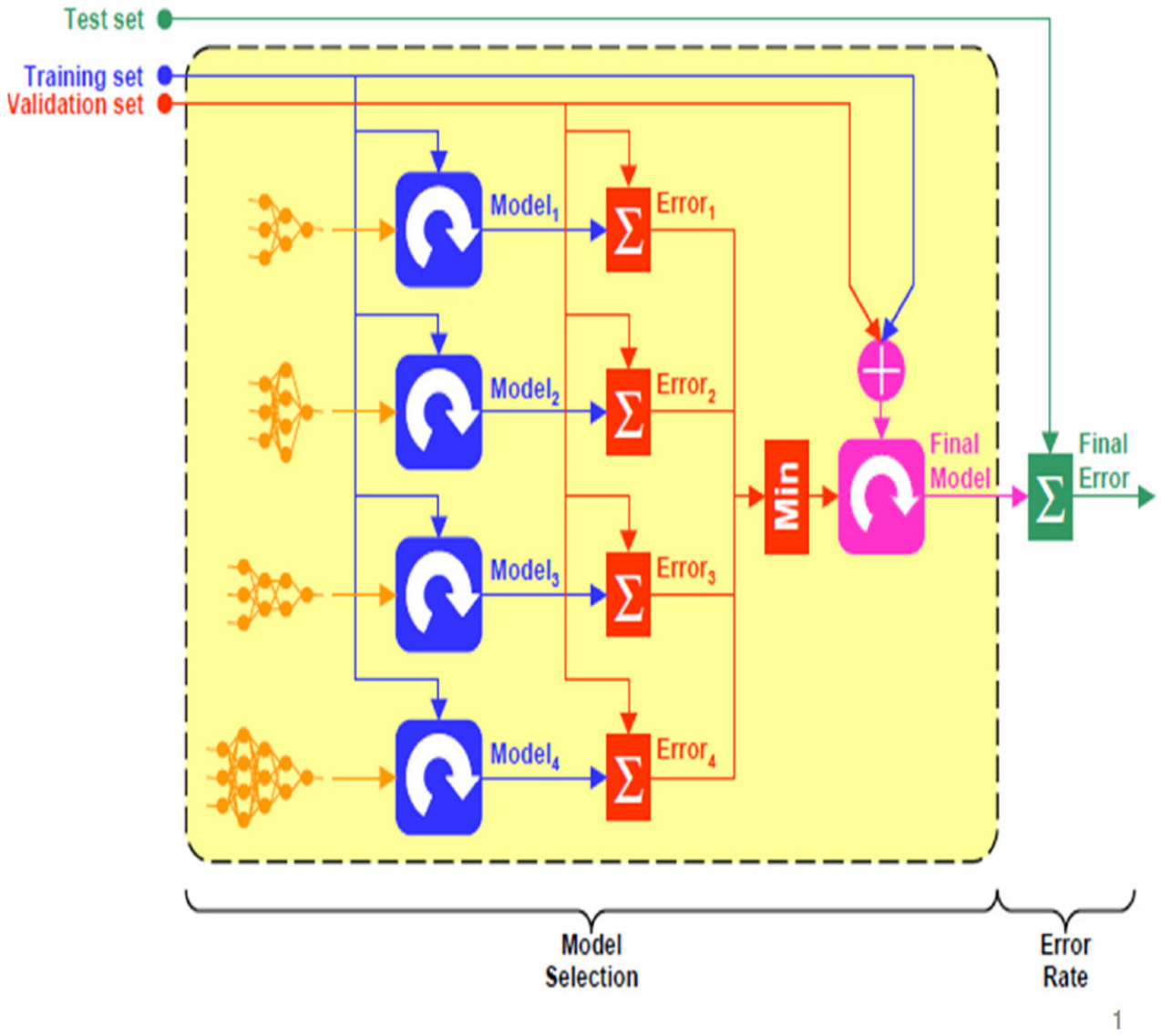
Step 2: Test the optimized models from step 1 with the CV set and choose the model with the min CV error (or other performance metric with dev data):

Cross validation (CV)/dev error => $$E_{cv}(\theta) = \frac{1}{2m_{cv}}\left[\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right) - y_{cv}^{(i)}\right)^2\right]$$

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

Training/Valid (Dev)/Test subsets



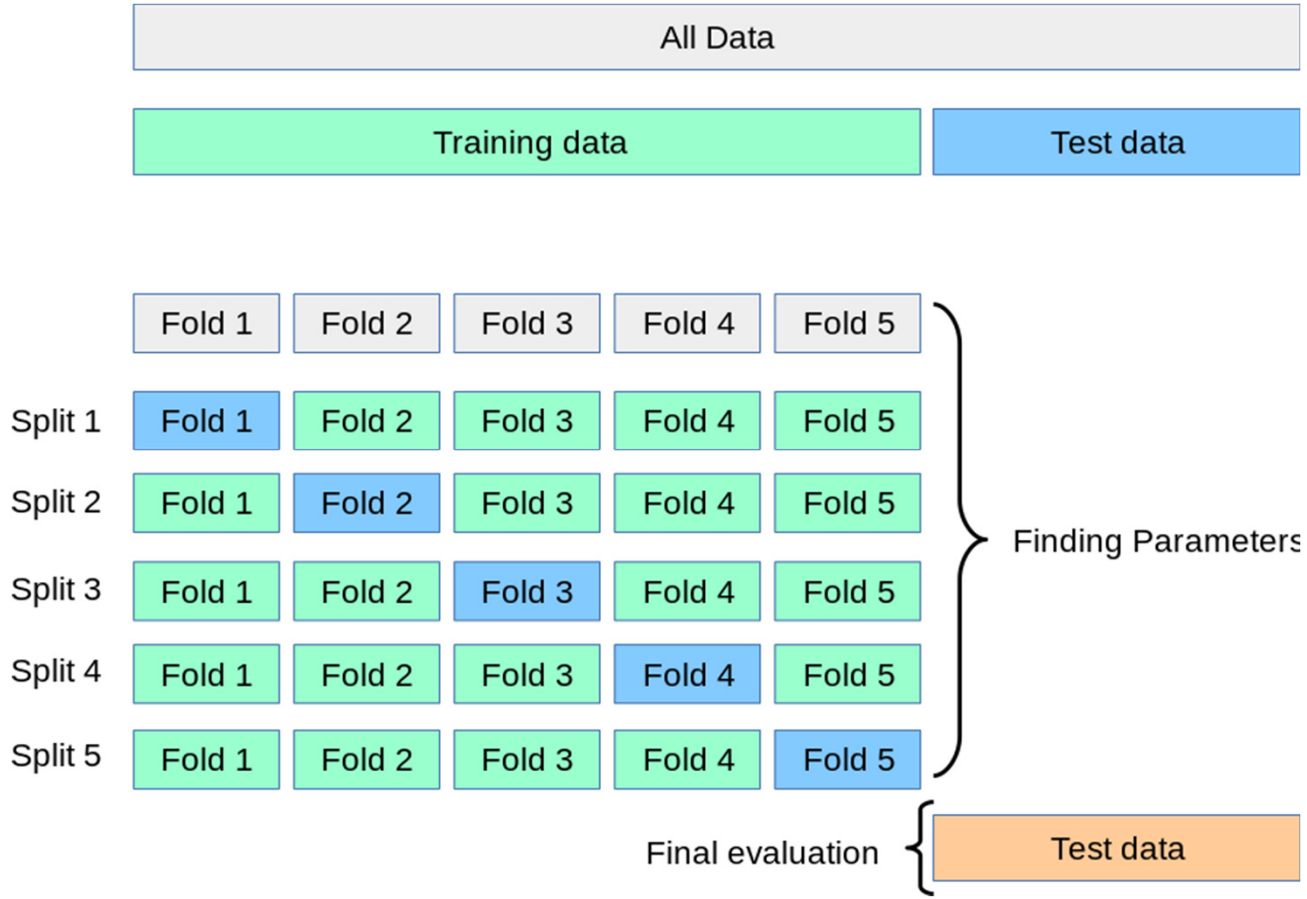
1

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.

K –fold Cross Validation

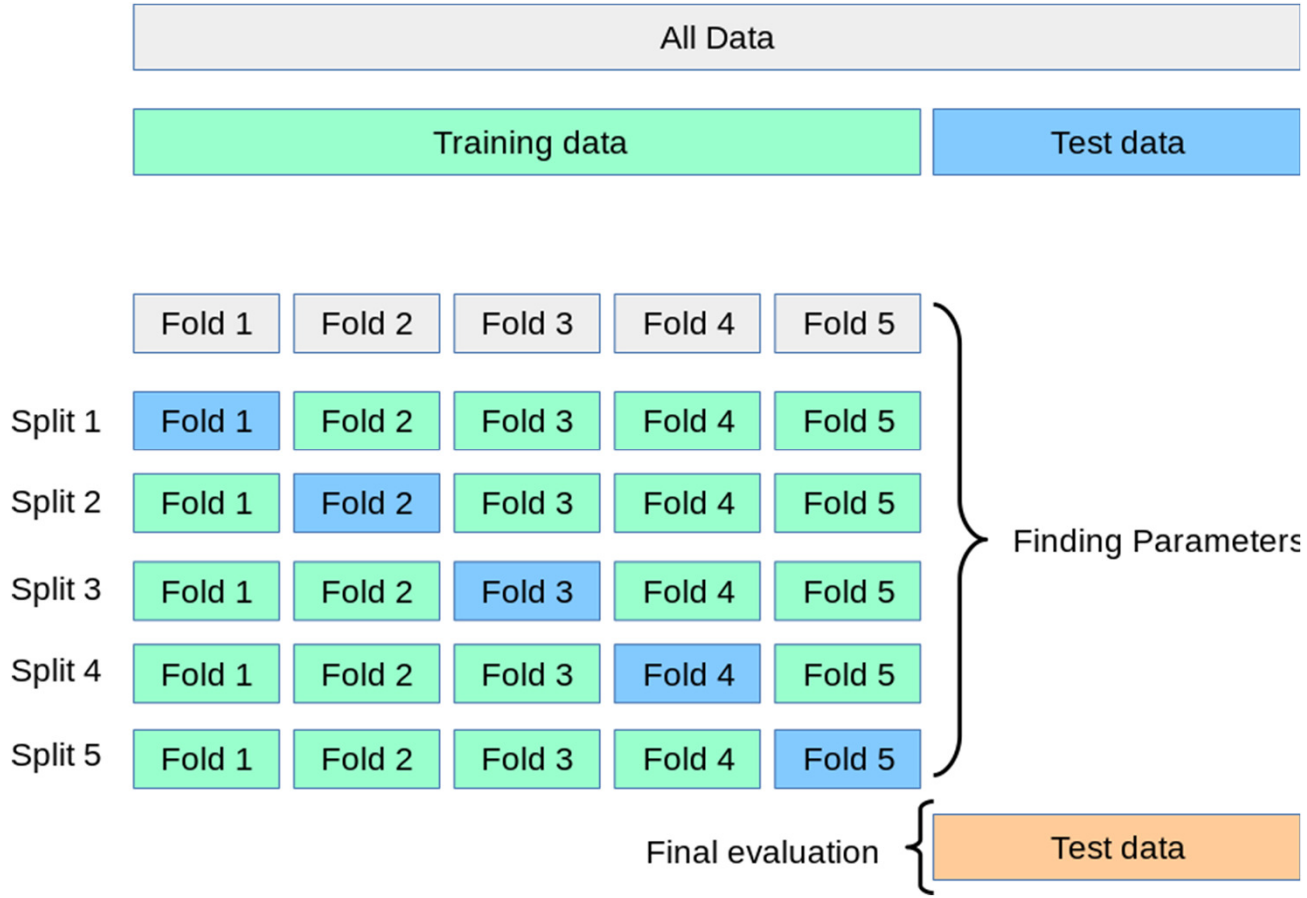
- Divide data into Training and Test subsets.
- Split Training data into K subsets (K folds).
- Use K-1 folds for training and the remaining ford for validation.
- The final validation error is the average error of K trainings.
- Choose the best model or the best hyper-parameter

the one that minimises the validation error.



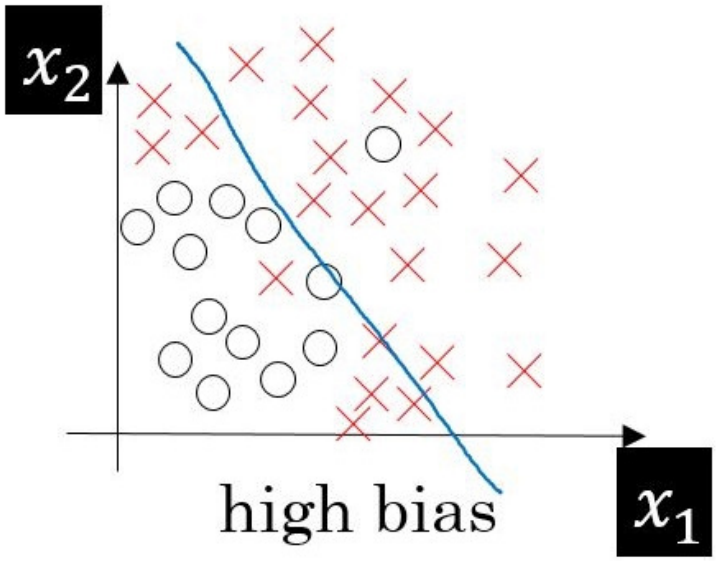
Leave-one-out Cross Validation

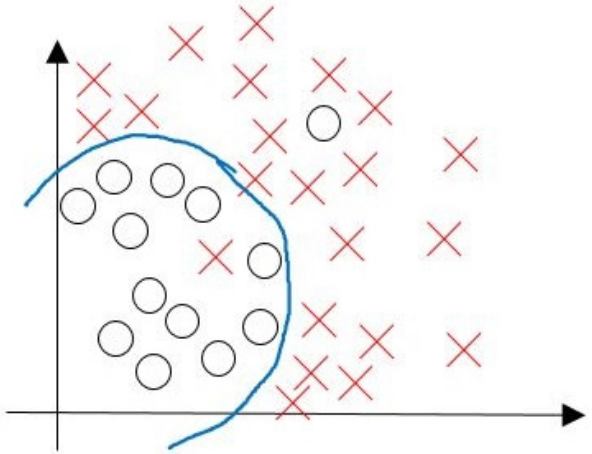
- Leave-one-out is a special case of K-fold CV
- 1 Fold = 1 example from the training data
- Useful for small data sets.

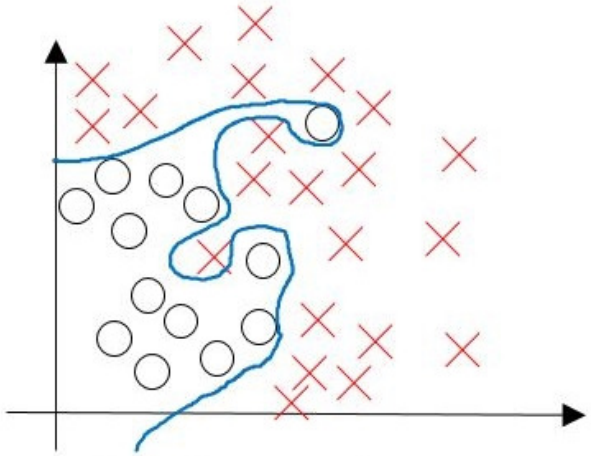


Bias vs. Variance

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**.**







underfiting data
(very simple model)

(good model)

overfiting data
(very complex model)

Diagnosing Bias vs. Variance

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

High Bias (underfiting) problem:

Training error (*Etrain)* and Validation/dev error (*Ecv)* are both high

High Variance (overfiting) problem:

Training error (*Etrain)* is low
and Validation/dev error (*Ecv)* is much higher than *Etrain*

Choose regularization parameter λ

For a given model, try different values of λ =[0, 0.01, 0.1, 1,....]

Step 1: For each λ, optimize parameters θ using the training set

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

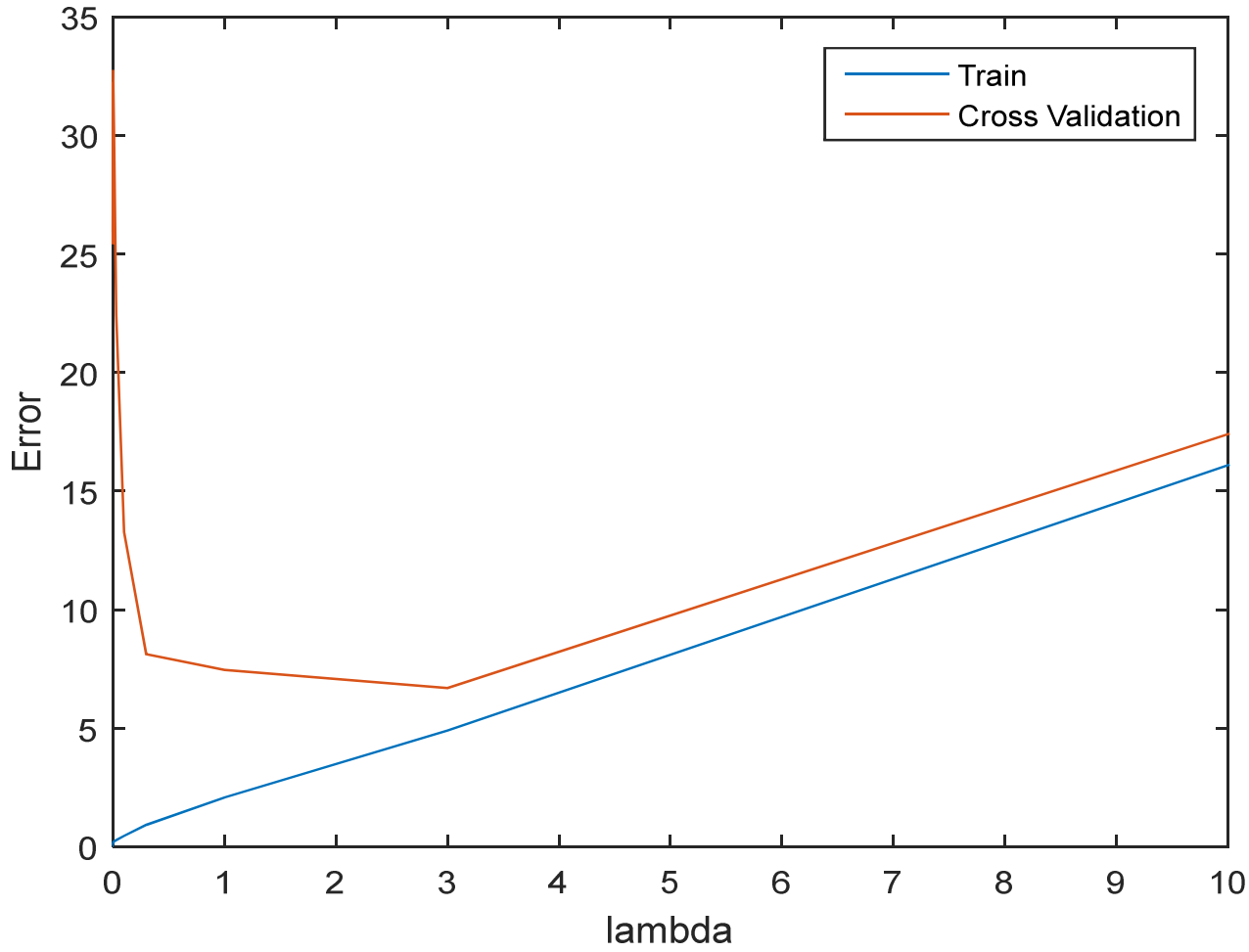
Step 2: Test the optimized models from step 1 with the CV set and choose the model with λ that gets min CV error:

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

Step 3: Retrain the model with best λ from step 2 with both train and CV sets starting from the parameters θ got at step 2. Test the retrained model with the test set and compute the error:

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

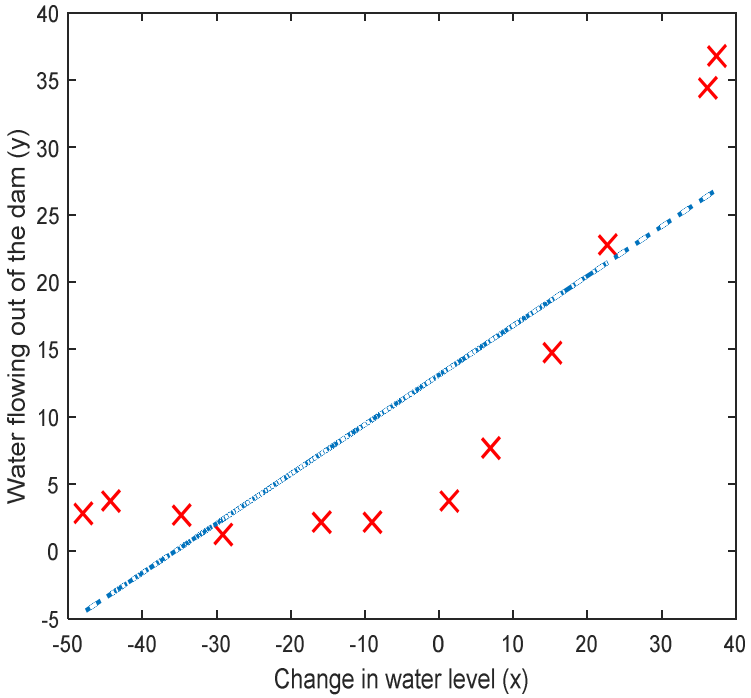
Select λ using CV(dev) set

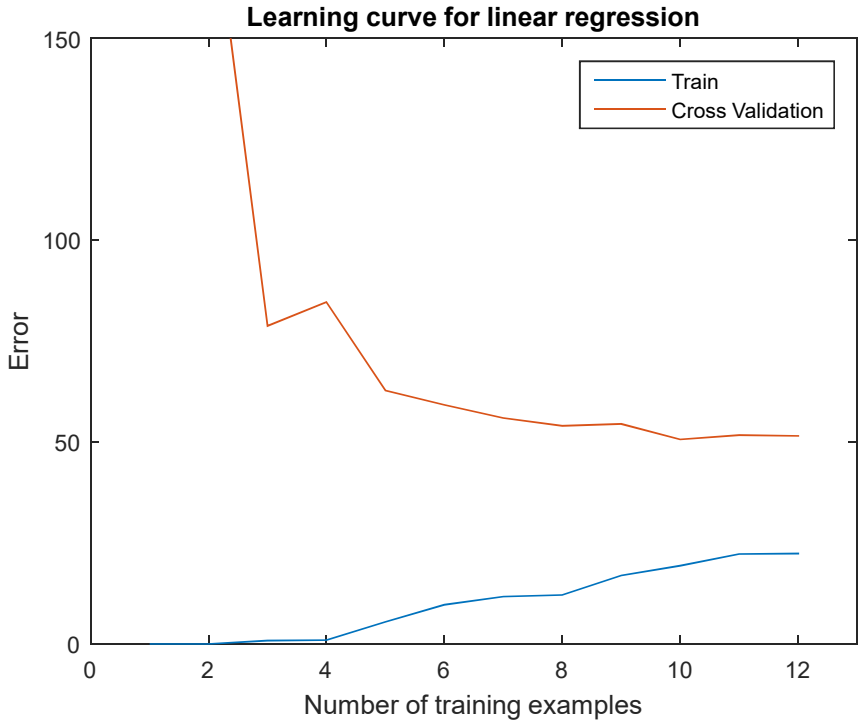


Best λ=3

Learning Curves

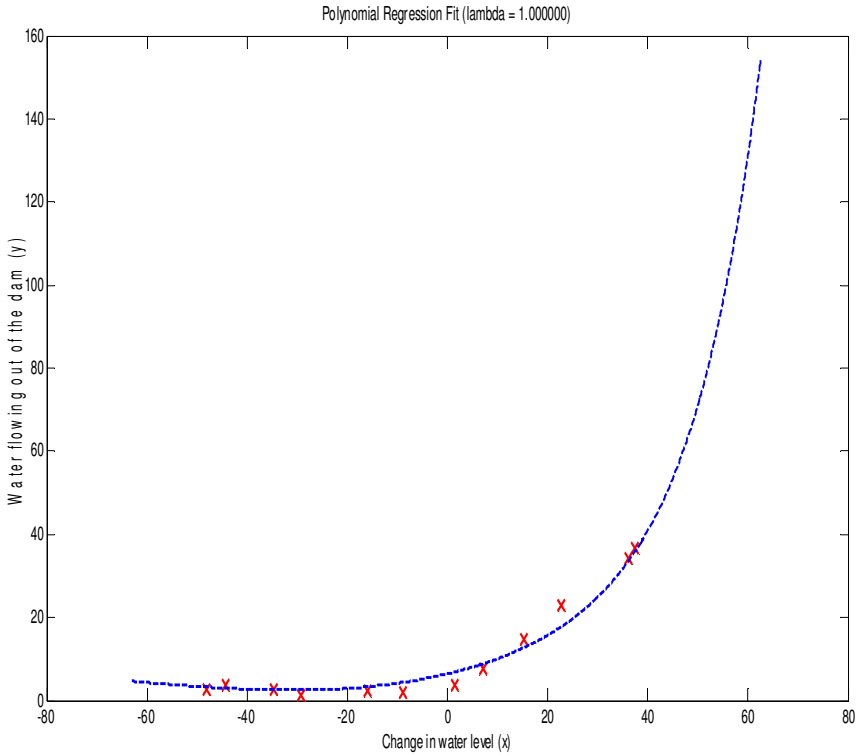
$$h_\theta(x) = \theta_0 + \theta_1 x$$

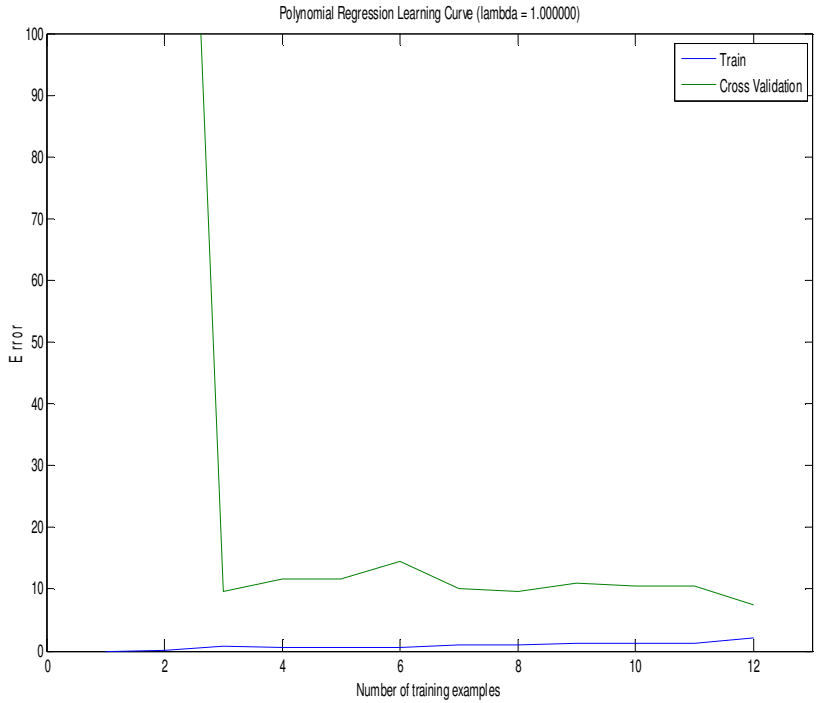




If a learning algorithm is suffering from high bias, getting more training data will not help much

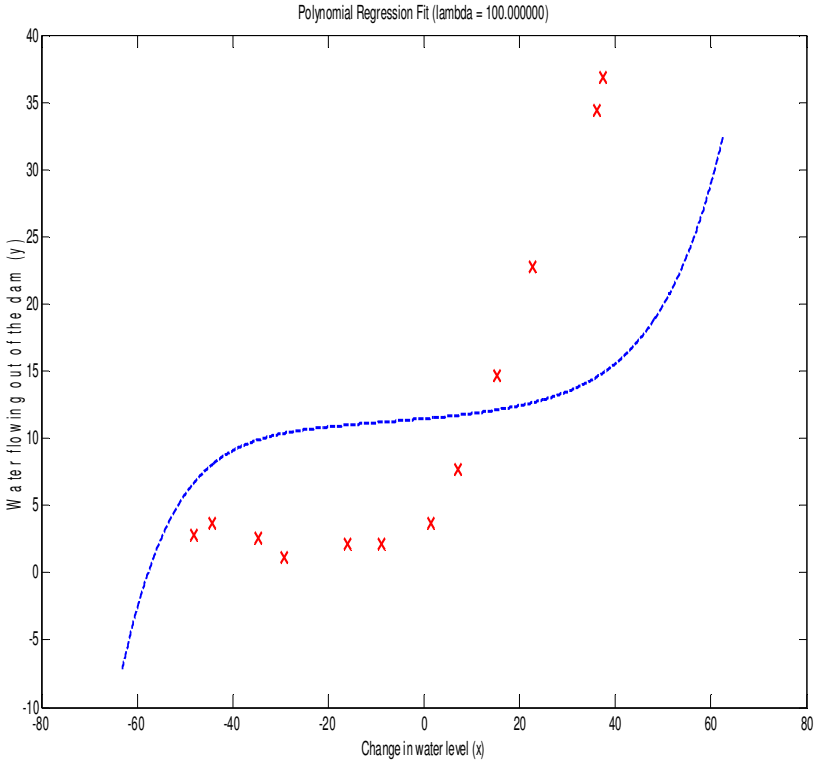
Learning Curves

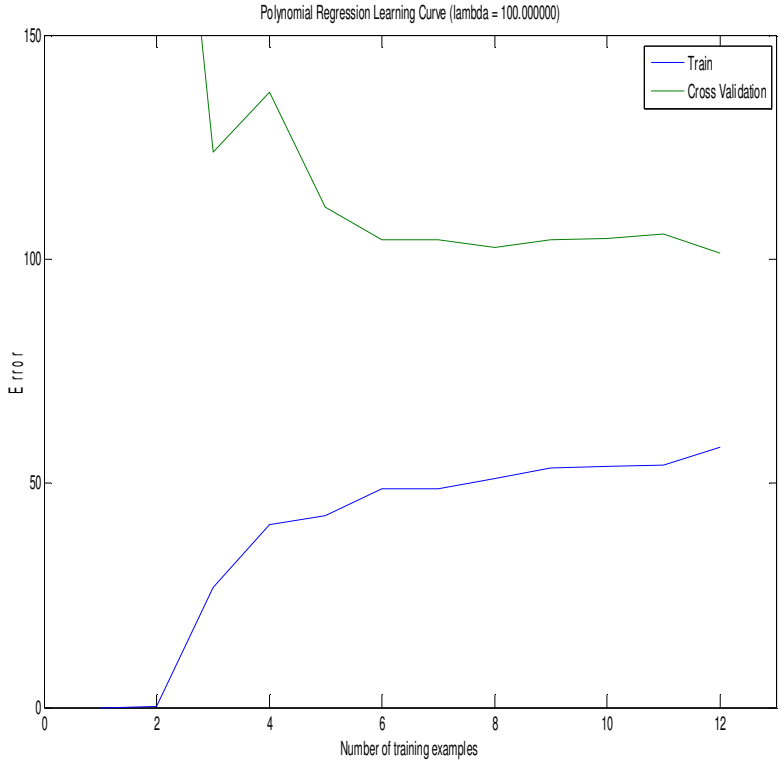




If a learning algorithm is suffering from high variance, getting more training data is likely to help

Regularization and Learning Curves





Polynomial regression, λ=100

Learning curve, λ=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 λ – fixes high bias

- Try increasing λ – fixes 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 ensemble classifier votes for one of the classes
- predict the class with the majority of votes (e.g., bagging)

2. Weighted voting
- make a *weighted* sum of the votes of the ensemble classifiers
- weights depend on the classifier error (e.g., 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

a) **Bagging:** 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.

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging(Round1)	7	8	10	8	2	5	10	10	5	9
Bagging(Round2)	1	4	9	1	2	3	2	7	3	2
Bagging(Round3)	1	8	5	10	5	5	9	6	3	7

Ensemble classification-Boosting

- **Basic Idea:**

- next classifiers focus on examples that were misclassified by earlier classifiers

- increase the weight of incorrectly classified examples, this ensures that they will become more important

- weight the predictions of the classifiers with their error

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 classfiers

- Train the lower level classifiers *C1...Cnc*
  t- true; f- false

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

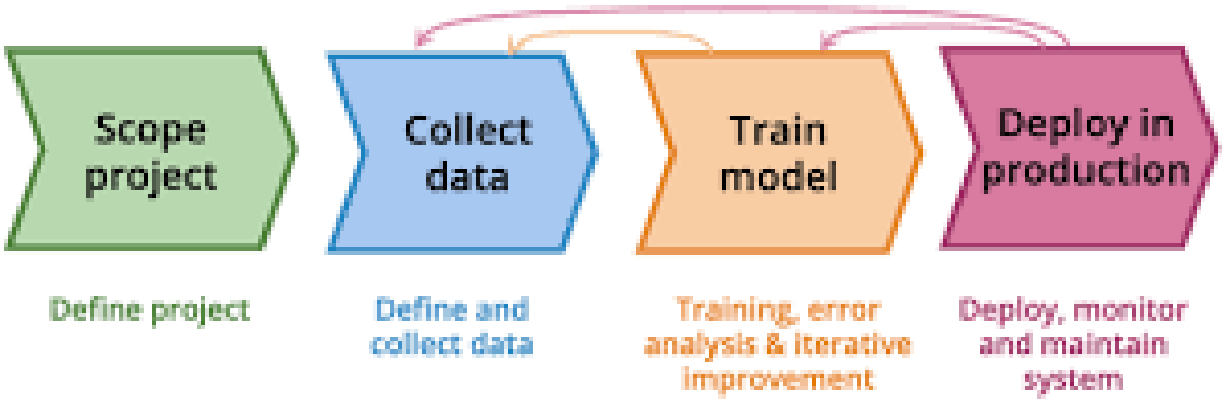
(d) training set for stacking

- Form a feature vector consisting of their predictions

- Train the meta classifier with these feature vectors

Model-centric vs Data-centric ML

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

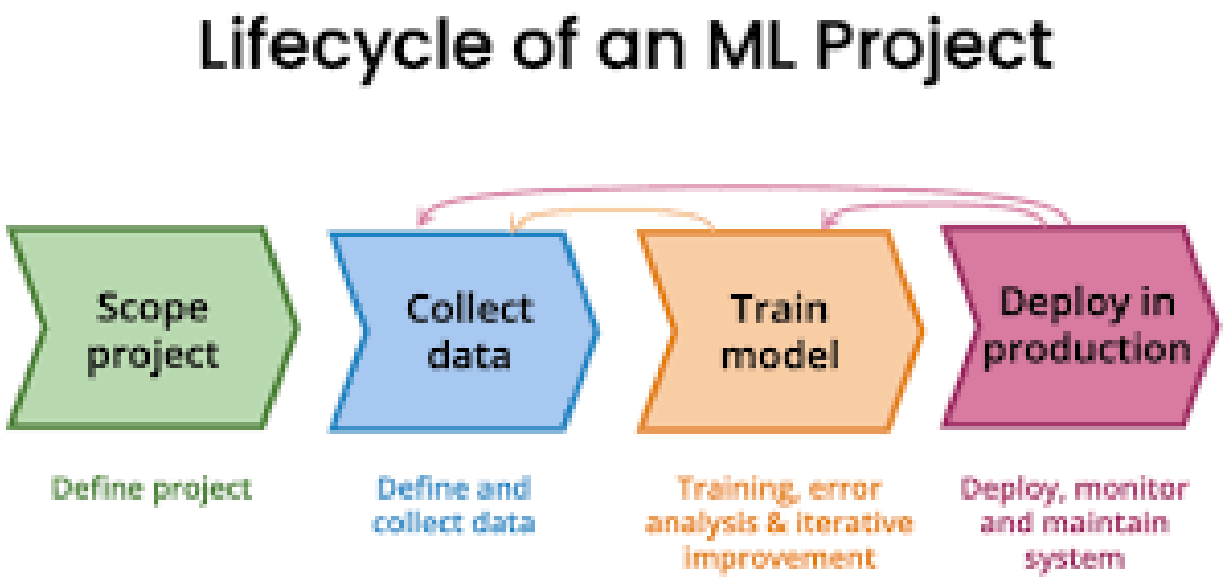
Data-centric

- 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

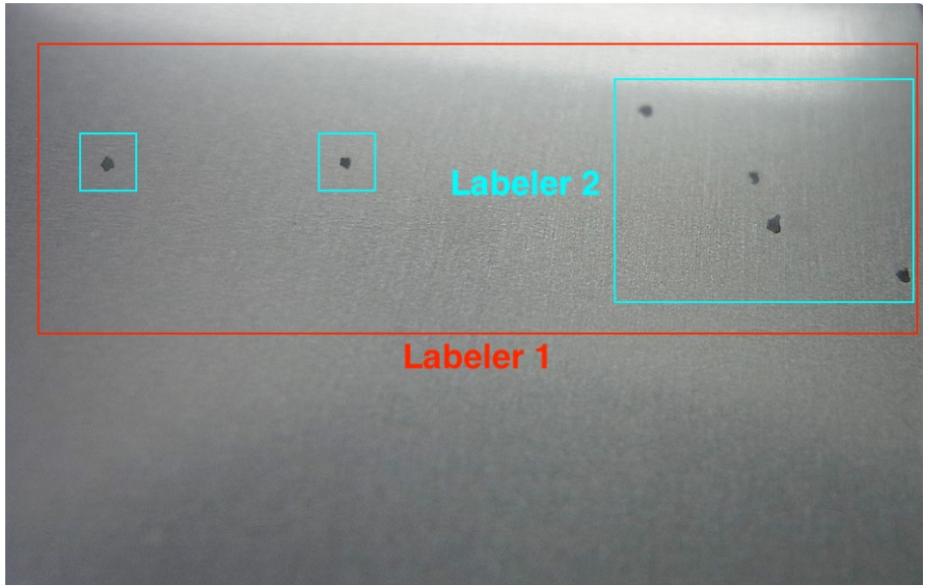


Error analysis shows your algorithm does poorly in speech with car noise in the background. What to do?

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 do 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.

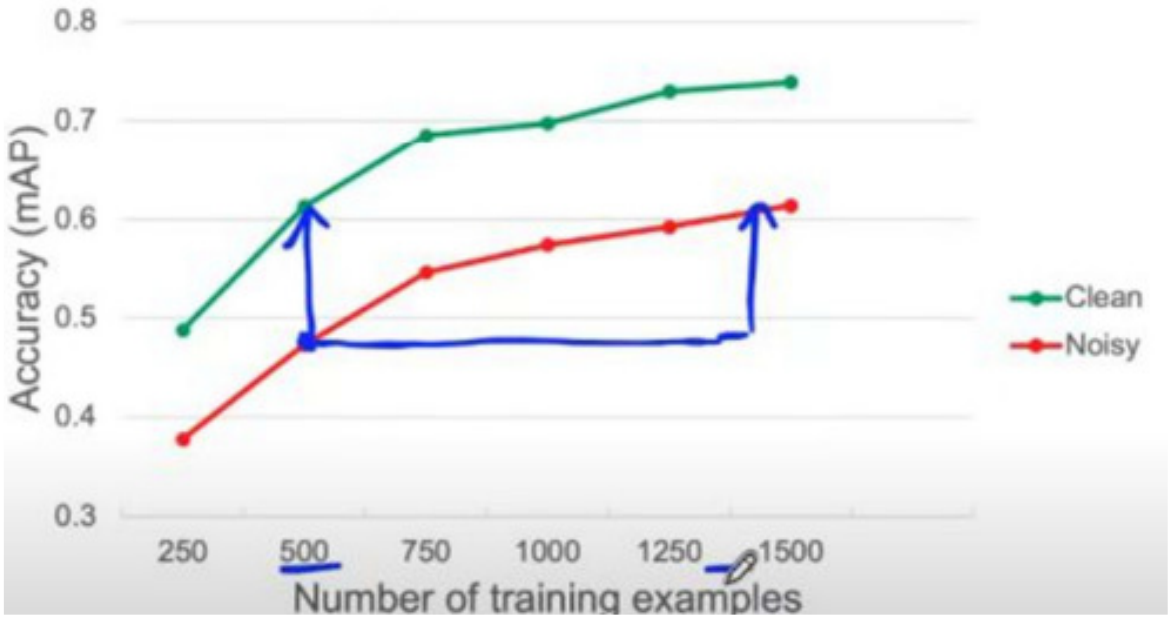
For classes where the labellers disagree, revise the labelling instructions until they become consistent.

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 examples (double the training set)

Apply data centric view => significant improvements



Takeaways: Data-centric AI

For ML engineer /Data scientist

Model-centric AI	Data-centric AI
How can you change the model (code) to improve performance	How can you systematically change the data (input X or labels y) to improve performance

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).