Introdução à Aprendizagem Automática (IAA)

SUSANA BRÁS

SUSANA.BRAS@UA.PT

IAA – L8

Ensemble Classifiers

- Bagging
- Boosting
- Stacking
- Differences between strategies

Model centric vs Data centric ML

- Advantages and disadvantages of each one of the strategies
- What strategy to use
- Implications in dataset size
- Evaluation of models' performance
- Data and concept drift
- Model monitoring

Ensemble Classifiers

Difficult Problems

Idea

- ✓ Difficult problems are often approached through brainstorming among experts.
- ✓ Specialists from different fields contribute with different perspectives.
- ✓ These perspectives may complement each other and lead to better solutions.

In Machine Learning

- ✓ Instead of relying on a single classifier, we combine multiple classifiers.
- ✓ Each model is designed to be somewhat different from the others.

How it works

- ➢ Each classifier makes its own prediction.
- ➢ The final decision is obtained through aggregation (e.g., voting).

Why it works

- ➢ Individual models may make different errors.
- ➢ When combined, their collective decision can compensate for individual weaknesses.

A **group of diverse models** can often perform **better than a single model**.

Ensemble (Committee) Classifiers

- Learn a set of classifiers (ensemble, committee):
  - Build “weak” classifiers
  - Do not try too hard to find the best classifier
  - Choose classifiers with different natures (deterministic, probabilistic, linear, nonlinear)

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

Ensemble (Committee) Classifiers

- Combining Predictions:
  - **Voting**
    - each classifier gives its vote
    - predict the class with the majority of votes (bagging)
  - **Weighted voting**
    - make a weighted sum of the votes of the ensemble classifiers
    - weights depend on the classifier error (boosting)
  - **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

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 the majority of votes.

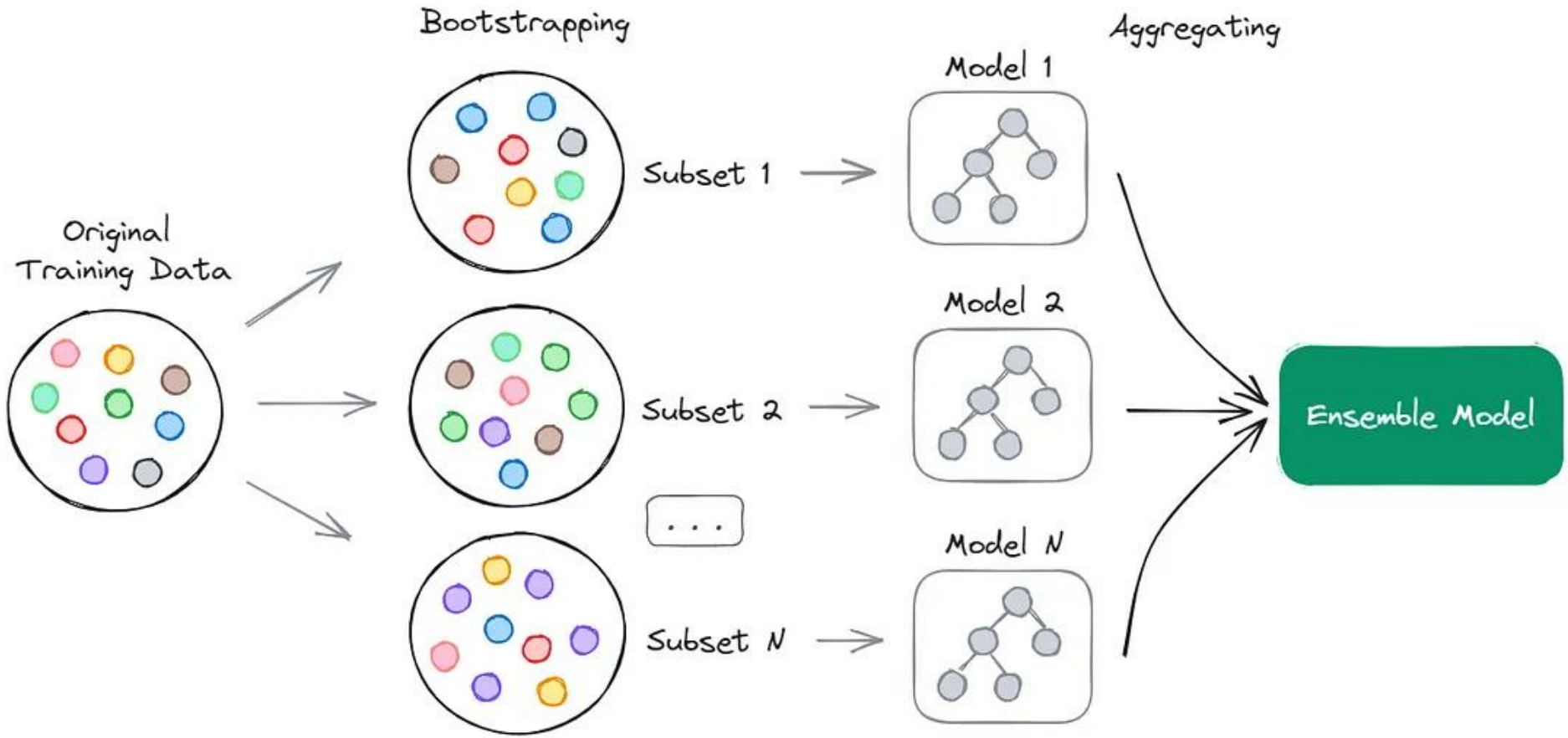
Variations are possible -e.g., size of subset, sampling w/o replacement, etc.

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

Bagging


Bootstrapping
Aggregating
Original
Training Data
Subset 1
Subset 2
. . .
Subset N
Model 1
Model 2
Model N
Ensemble Model

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.

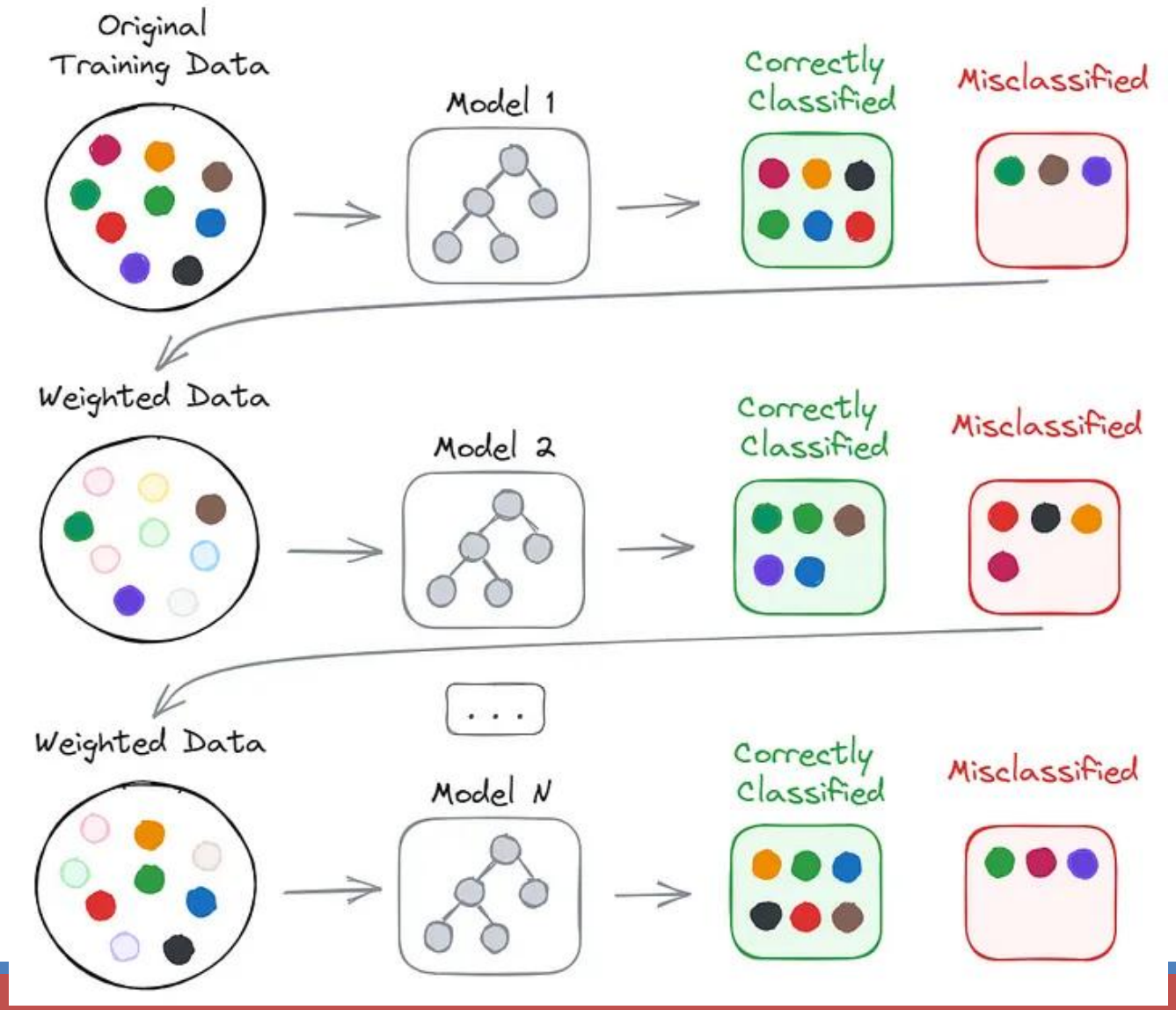
Examples of Ensemble Classifiers : Random Forest (set of Decision Trees)

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

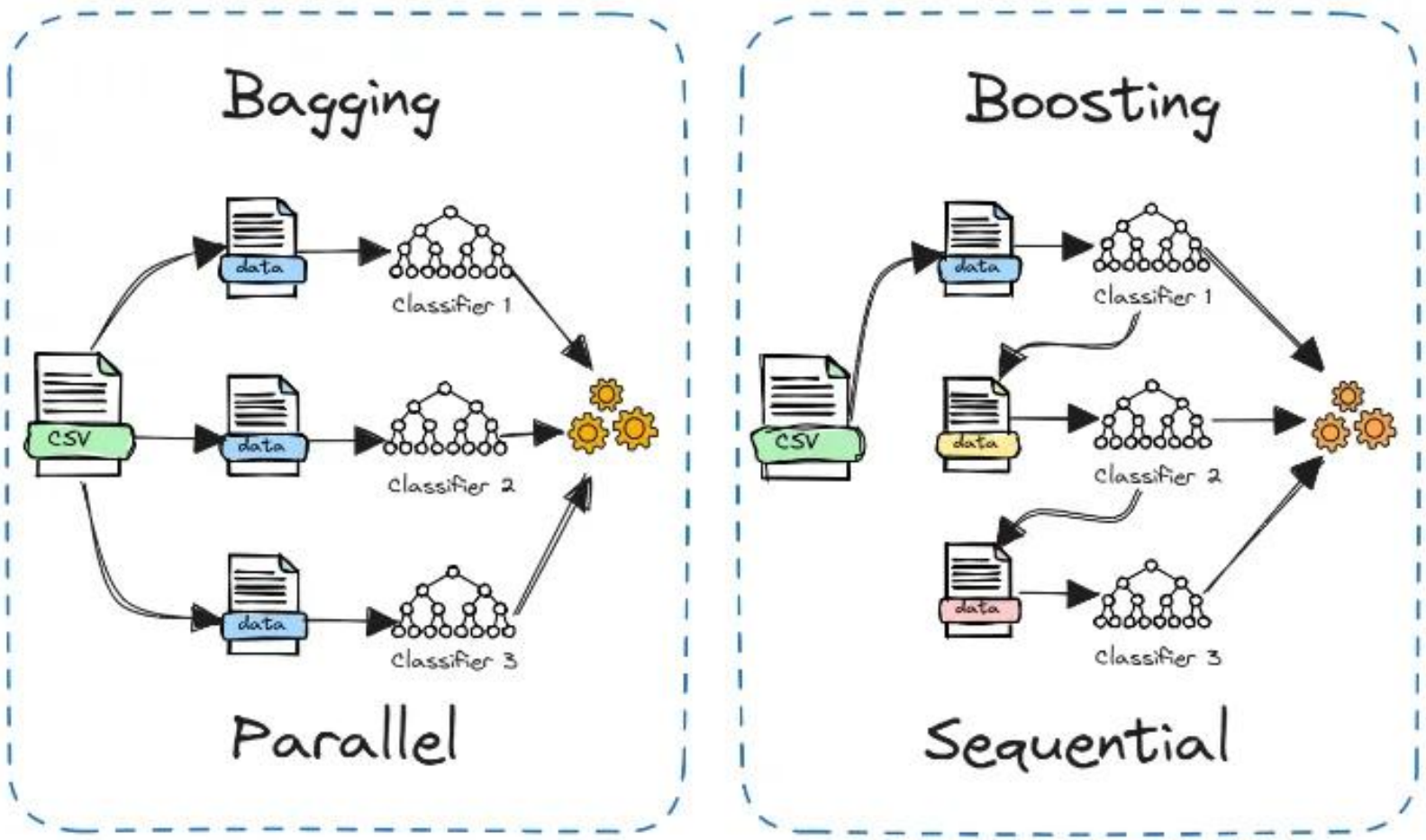
Considerations:

- ✓ 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.

Boosting


Original Training Data
Model 1
Correctly Classified
Misclassified
Weighted Data
Model 2
Correctly Classified
Misclassified
. . .
Weighted Data
Model N
Correctly Classified
Misclassified

Bagging vs Boosting



Stacking

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

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

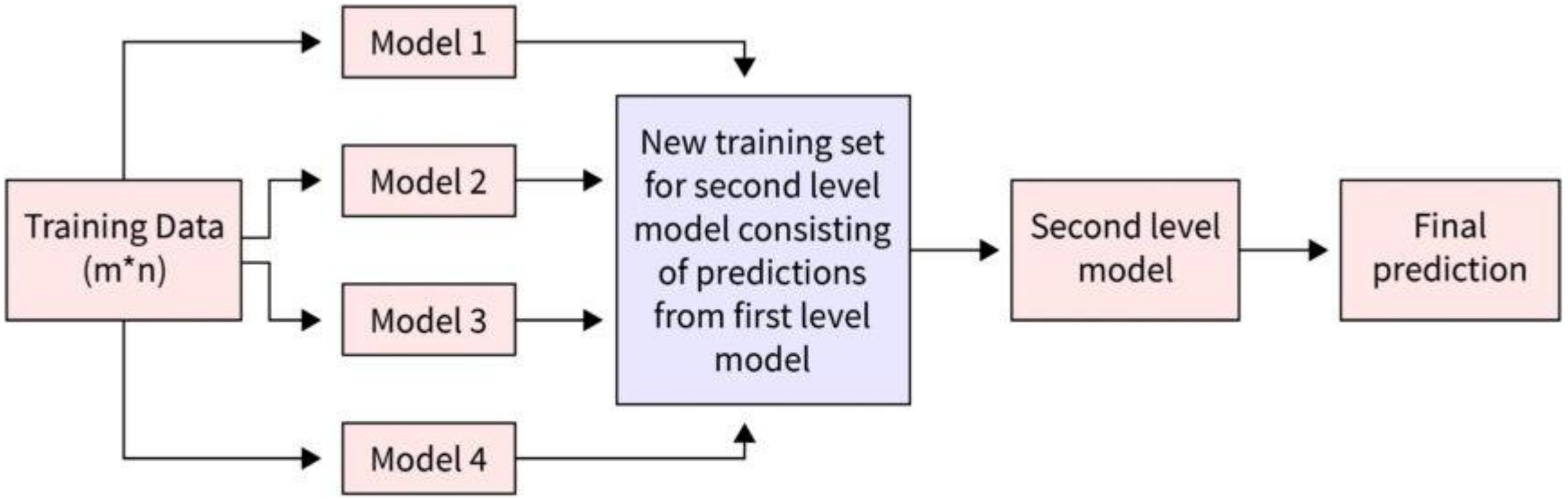
(d) training set for stacking

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

b) Form a feature vector consisting of their predictions

c) Train the meta classifier with these feature vectors

Stacking


Training Data
(m*n)
Model 1
Model 2
Model 3
Model 4
New training set for second level model consisting of predictions from first level model
Second level model
Final prediction

Model centric vs Data centric ML

And now?

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 ?

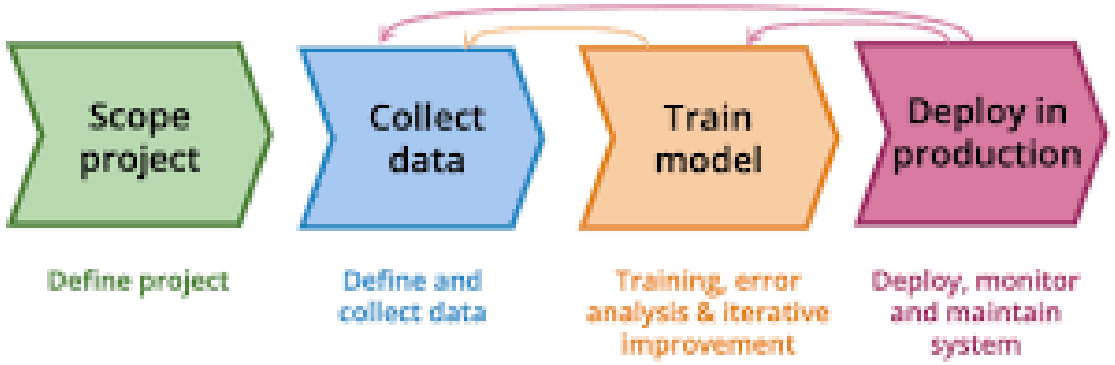
- Get more training examples?
- Try smaller sets of features (feature selection)?
- Try getting additional features (feature engineering)?
- Try using different/nonlinear kernels?
- Try other values of the hyperparameters (e.g. regul. Parameter)?

Machine learning diagnostics = Model-centric approach

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

Diagnostics is time-consuming, but essential in value creation and quality delivery.

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 than collecting more data
- ✓ Allows more models to do well

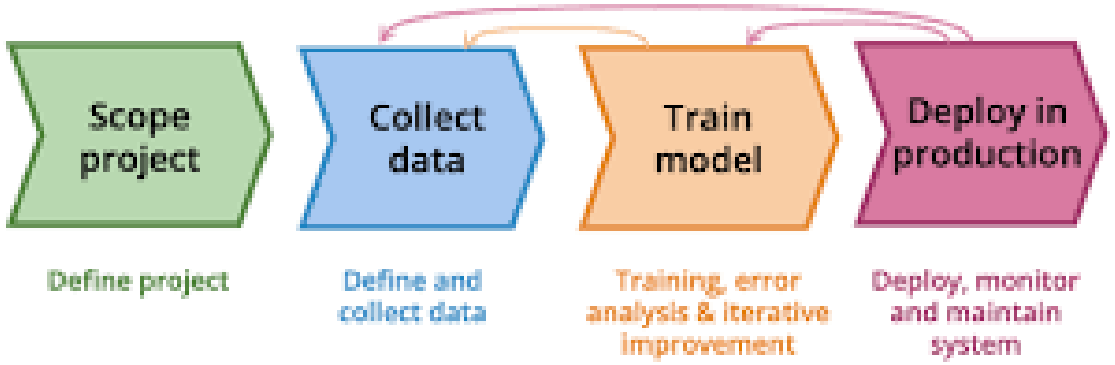
Approach:

- ✓ Hold the code/algorithms fixed
- ✓ Iterate the data quality

Lifecycle of an ML Project

Aspect	Model-centric	Data-centric
Main idea	Collect as much data as possible	Data consistency is key
Strategy	Optimize the model to handle noise in the data	Invest more in data quality tools rather than collecting more data
Impact	Performance mainly improves through better models	Allows more models to perform well
Approach – Data	Data is fixed after standard preprocessing	Data quality is iterated and improved
Approach – Algorithm	The model is improved iteratively	Code/algorithms are kept fixed

Lifecycle of an ML Project

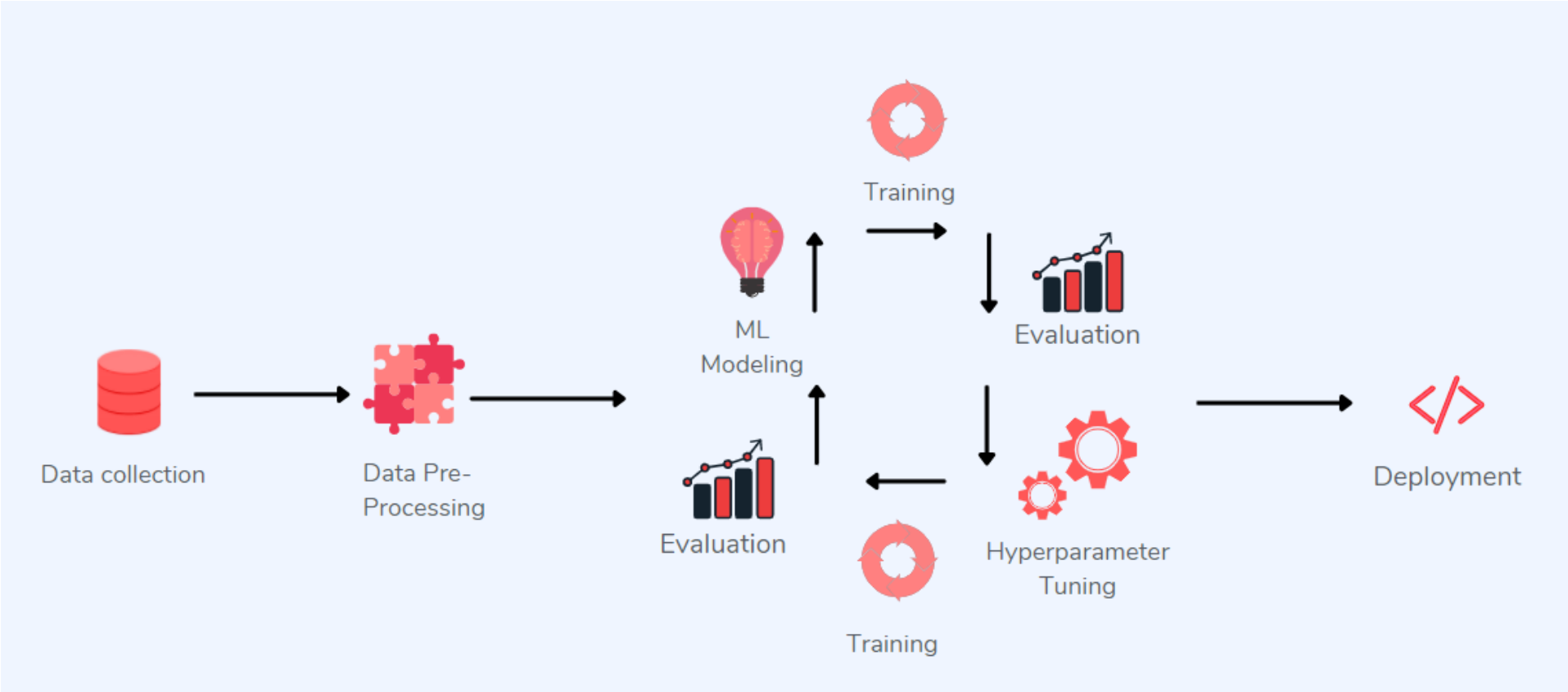


Example:
Error analysis shows your algorithm does poorly in speech with car noise in the background. What to do?

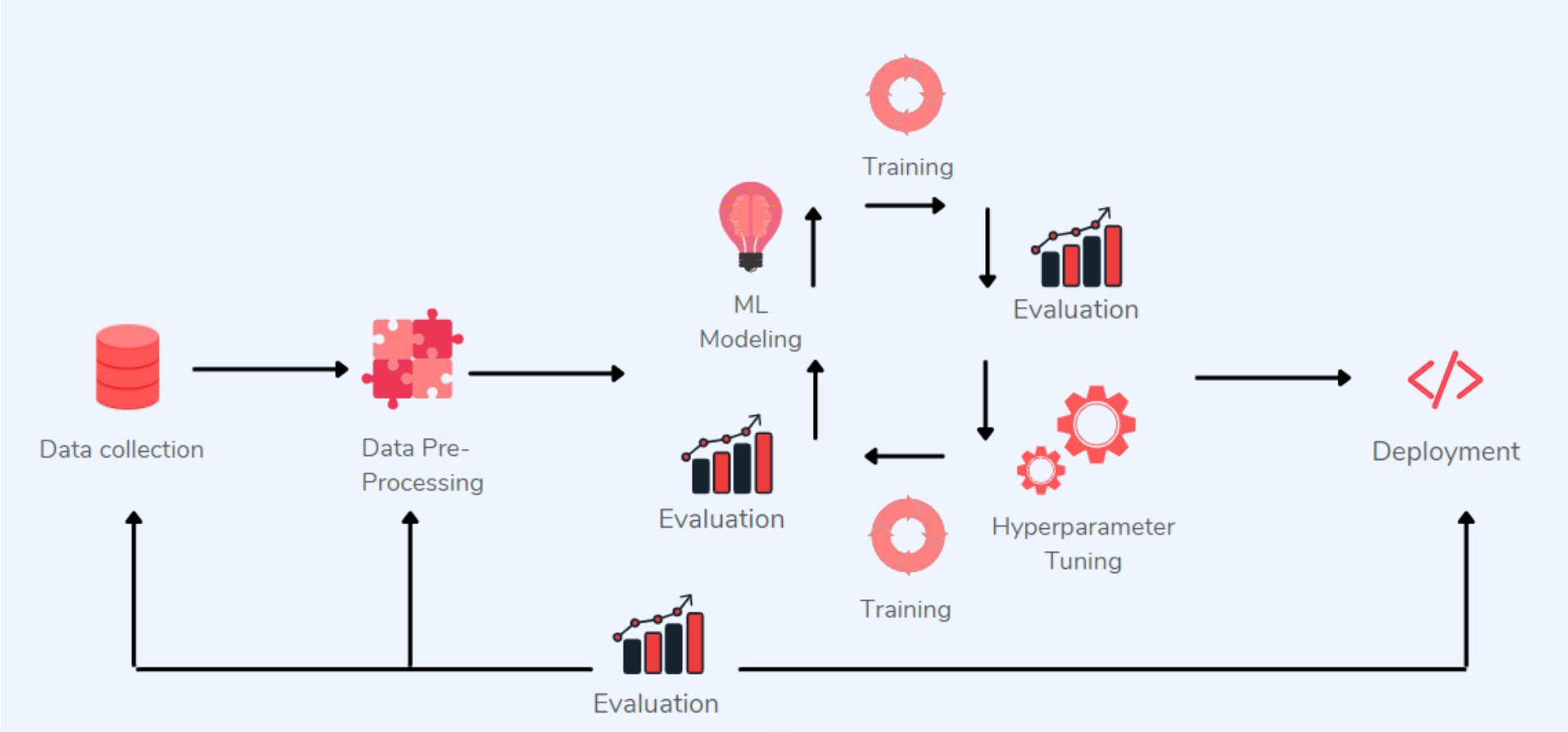
Model-centric view: Try hard to tune the model architecture to improve performance

Data-centric view: Modify the data (get new examples relevant for the car noise scenario) to improve performance.

Model-centric Approach



Data-centric Approach



Data Centric Approach - Label

Labelling instructions:

Use bounding boxes to indicate the positions of the defects.

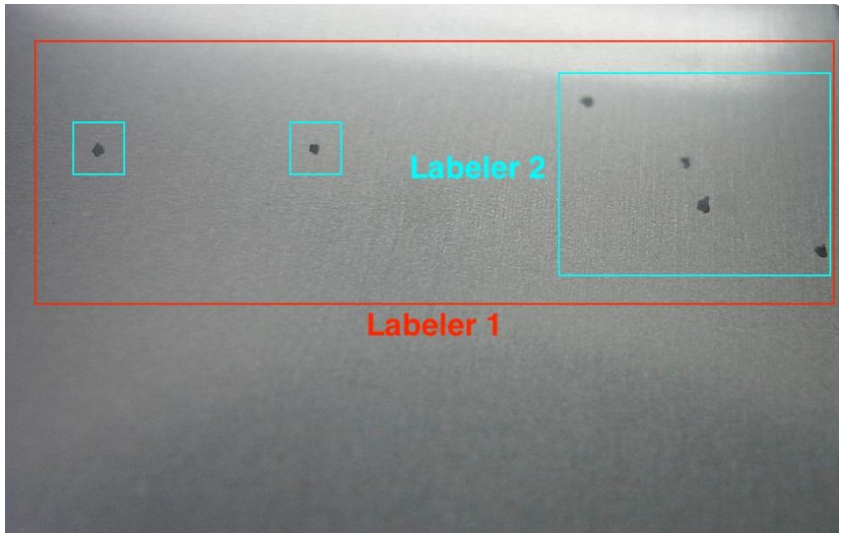
Is the data labelled consistently?

Possible Strategy:

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.

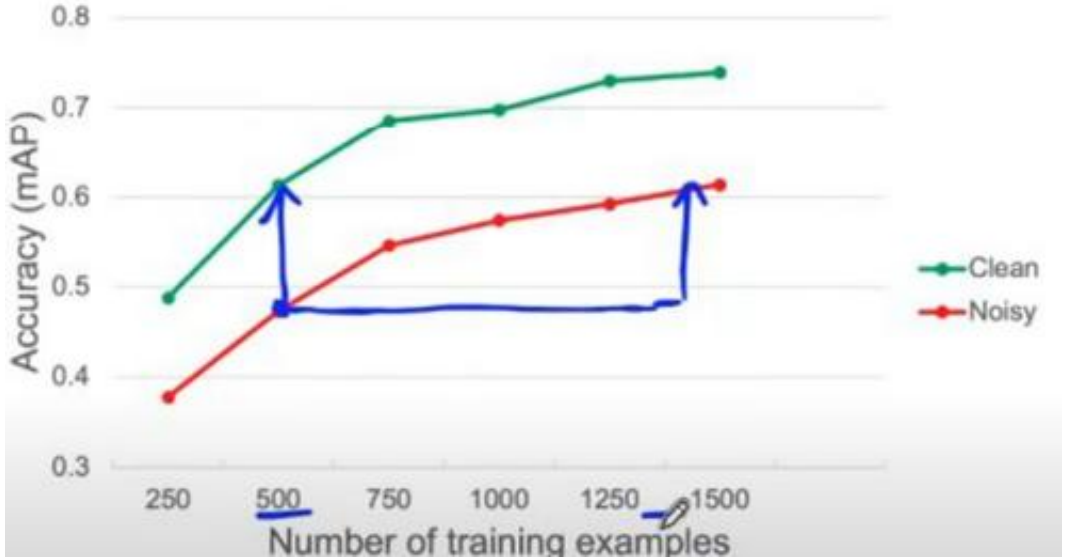


Clean vs Noisy Data

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

Data-centric AI

.

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

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 model's performance on it.

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).

DC-CHECK

A data-centric checklist to guide the development of end-to-end ML pipelines

DATA

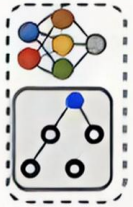
Proactive dataset selection & curation

Data pre-processing & cleaning

Data quality evaluation

Synthetic data improvement

TRAINING



Data informed model design/selection

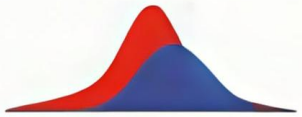
Data infomed training for usage across domains

Data subset/subgroup robust training

Data noise robust training

DEPLOYMENT

Model & data monitoring



Understand & address dataset shift and drift

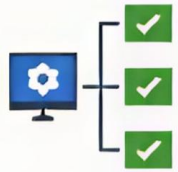
Model retraining & dataset updates

Methods to enable trustworthy predictions

TESTING

Methods to split & assess data

Stress test scenarios



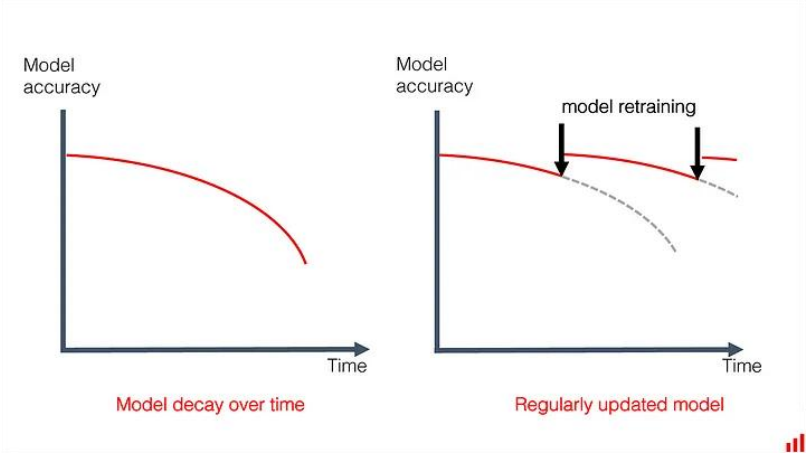
Evaluation beyond average on subgroups

Data and Concept Drift

.

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

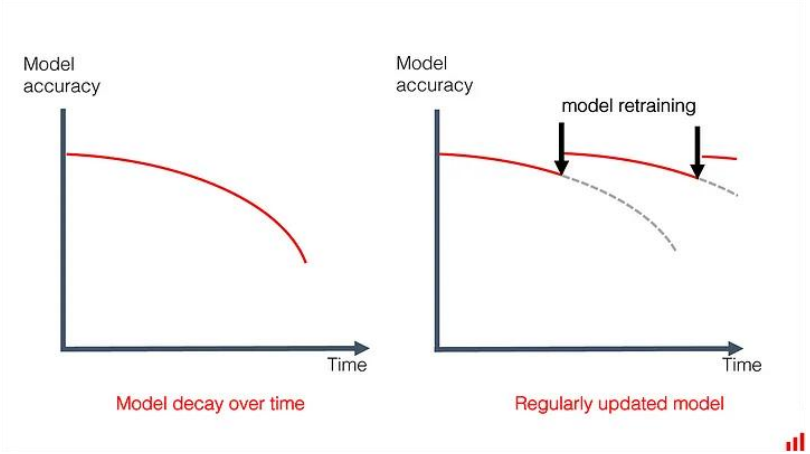
Some models can last years without an update.
For example, certain computer vision or language models.
Or any decision system in an isolated, stable environment.
Others might need daily retraining on the fresh data.



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.

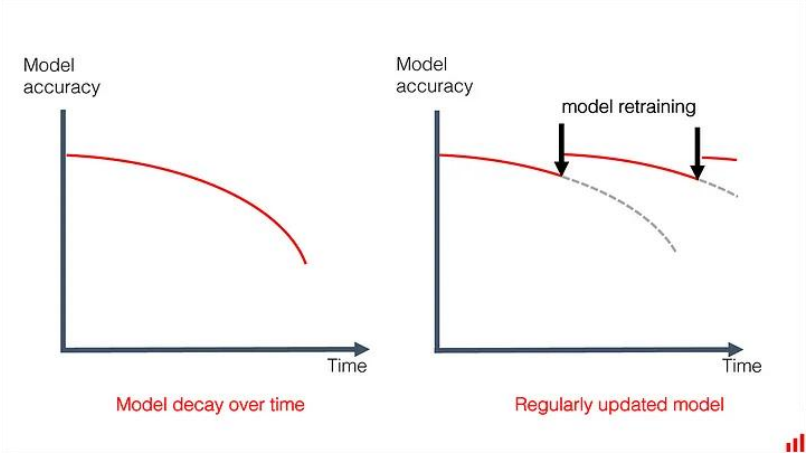
In contrast to the data drift, the distributions (such as user demographics, frequency of words, etc.) might even remain the same. Instead, the relationships between the model inputs and outputs change.

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.