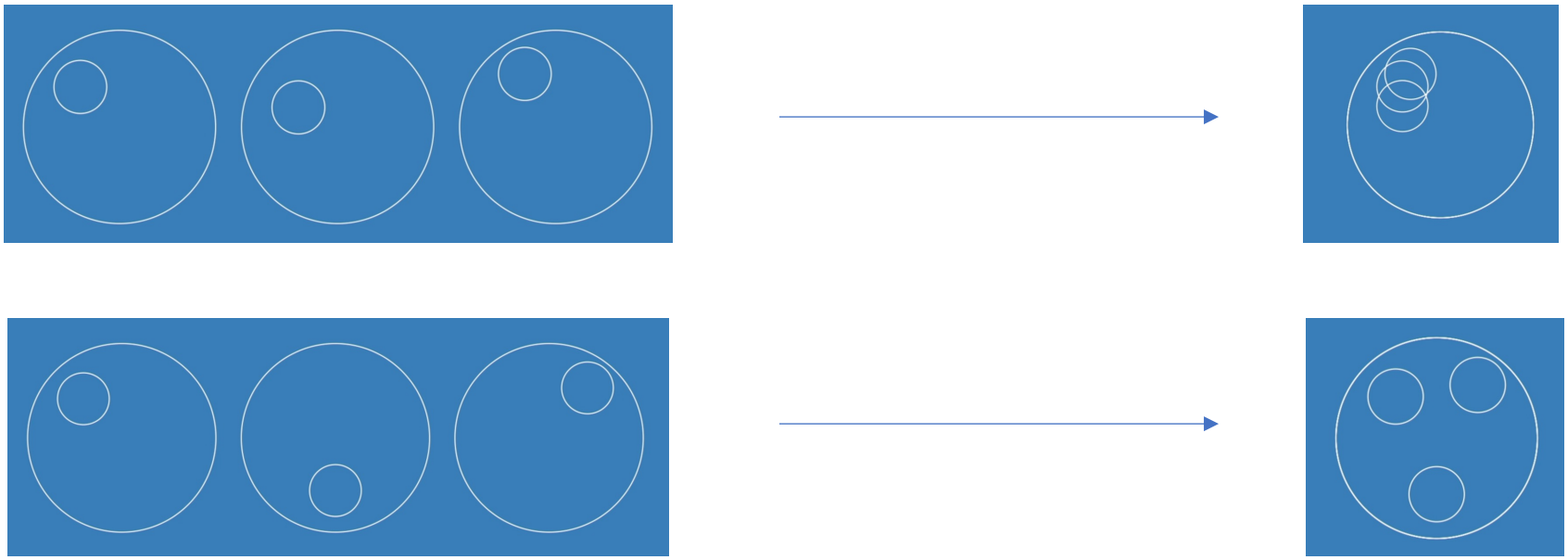
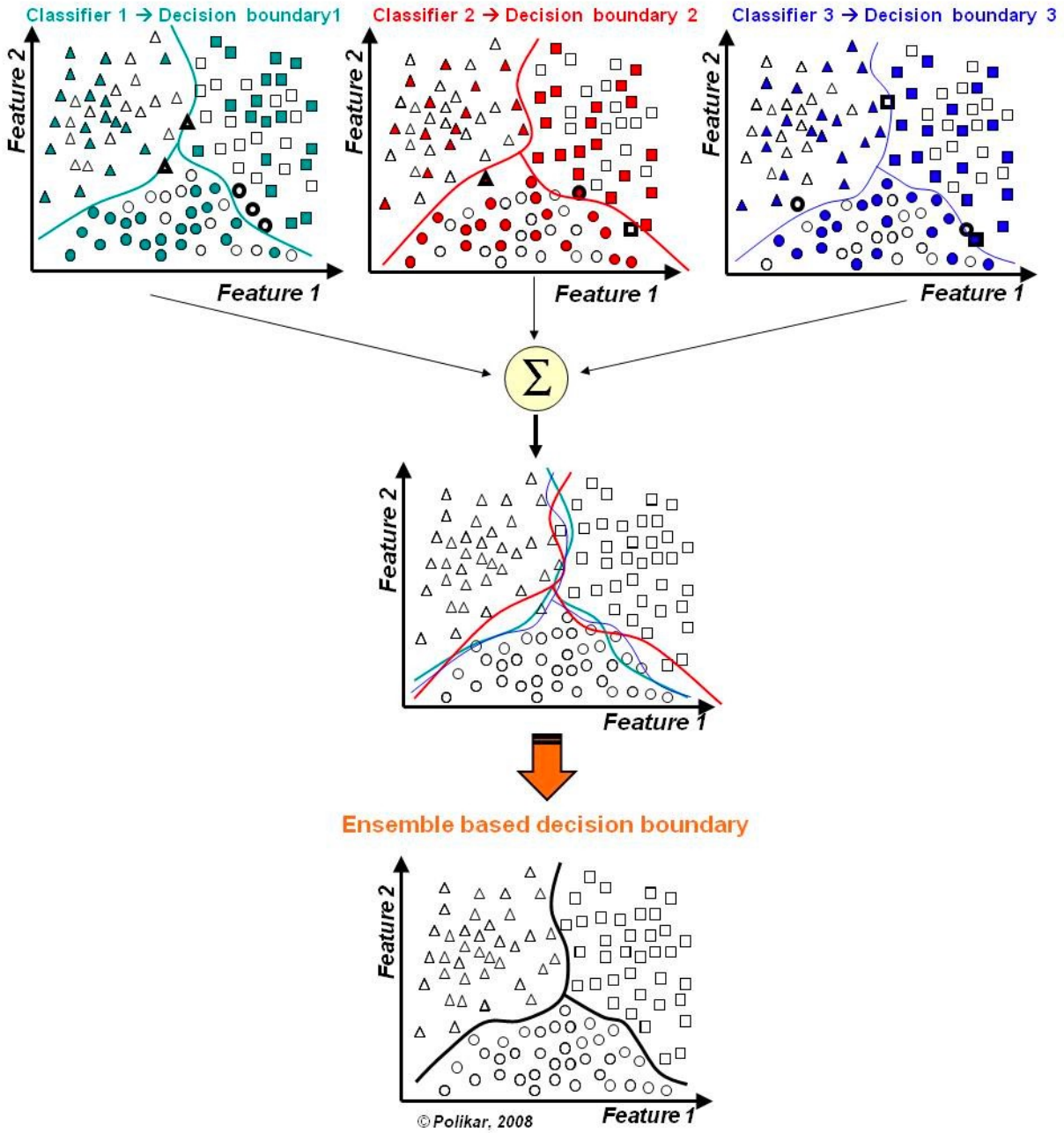
Ensemble

- Minimize the probability of model prediction errors on ***future*** data
- Two Competing Methodologies
  - Build **one** really good model
    - Traditional approach
  - Build **many** models and average the results
    - Ensemble learning (more recent)

- Ensemble methods construct and/or combine collection of predictors with the purpose of improving upon the properties of the individual predictors:
  - stabilize models
  - reduce prediction error
  - aggregate individual predictors that make different errors
  - more resistant to noise

In practice, one reliable approach to improving the performance of ML model by ***a few percent*** is to train multiple independent models, and at test time average their predictions. **As the number of models in the ensemble increases, the performance typically monotonically improves (though with diminishing returns). Moreover, the improvements are more dramatic with higher model variety in the ensemble.**




Classifier 1 → Decision boundary1
Classifier 2 → Decision boundary 2
Classifier 3 → Decision boundary 3
Feature 2
Feature 1
Σ
Ensemble based decision boundary
© Polikar, 2008

The Single Model Philosophy

- Motivation: Occam's Razor
  - "one should not increase, beyond what is necessary, the number of entities required to explain anything"
- Infinitely many models can explain any given dataset
  - Might as well pick the smallest one…

Which Model is Smaller?

$$\hat{y} = f_1(x) = \sin(x)$$

or

$$\hat{y} = f_2(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

- In this case $f_1(x) \equiv f_2(x)$
- It's not always easy to define small!

Exact Occam's Razor Models

- Exact approaches find optimal solutions
- Examples:
  - Support Vector Machines
    - Find a model structure that uses the smallest percentage of training data (to explain the rest of it).
  - Bayesian approaches
    - Minimum description length

Approximate Occam's Razor Models

- Approximate solutions use a greedy search approach which is not optimal
- Examples
  - Kernel Projection Pursuit algorithms
    - Find a minimal set of kernel projections
  - Relevance Vector Machines
    - Approximate Bayesian approach
  - Sparse Minimax Probability Machine Classification
    - Find a minimum set of kernels and features

Other Single Models: Not Necessarily Motivated by Occam's Razor

- Minimax Probability Machine (MPM)
- Trees
  - Greedy approach to sparseness
- Neural Networks
- Nearest Neighbor
- Basis Function Models
  - e.g. Kernel Ridge Regression

Ensemble Philosophy

- Build many models and combine them
- Only through averaging do we get at the truth!
- It's too hard (*impossible*?) to build a single model that works best
- **Two types of approaches**:
  - Models that don't use randomness
  - Models that incorporate randomness

- **Given labeled data, how to construct an ensemble of models?**
- **Given a new unlabeled data instance, how to use the ensemble to predict its label?**

Data:
What (part of the) data to use to train each model in the ensemble?

ML Techniques:
What technique and/or what parameters to use to train each model?

How to combine the individual model predictions into a unified prediction?

Varying data used

- subset of the attributes
- subset of the data instances
- …

Varying ML technique/parameters

- different parameters for a technique
- different techniques
- …

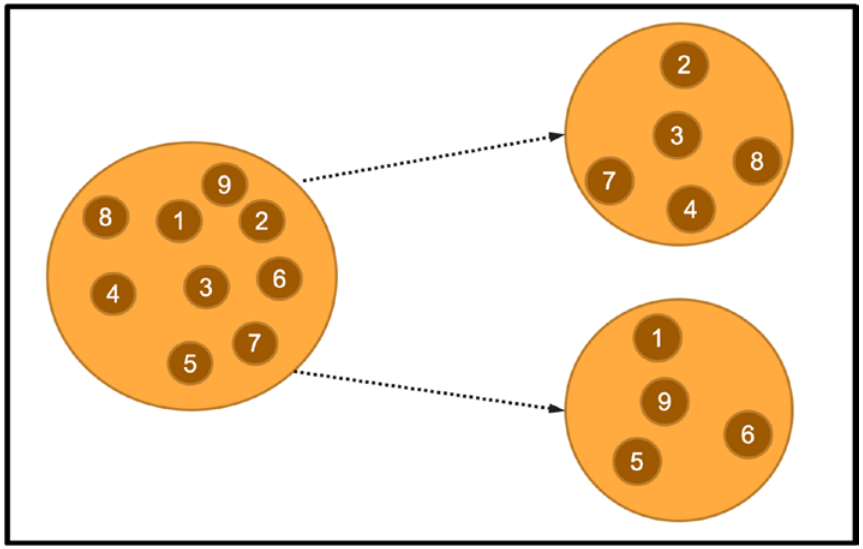
Mixing Training Data

1. k-fold **Cross-Validation**
2. Bootstrap Aggregation (**bagging**)
3. Random Training Subset

Bootstrap

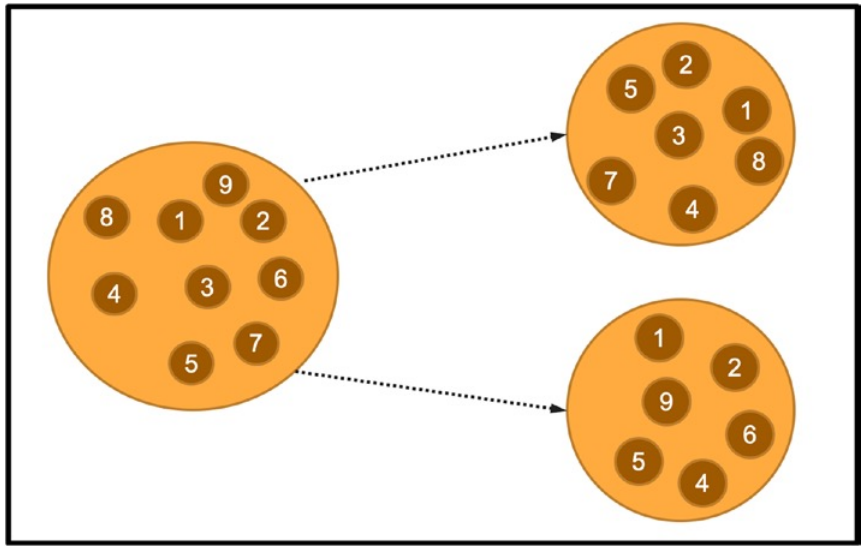
- 약 1900년대 "**to pull (oneself) up by (one's) bootstrap"**은 불가능한 작업을 상징하는 것으로 사용
  - Steels의 "Popular Physics"(1988) 교과서의 1장 마지막에 "실용적 질문들" 중 "30번. 왜 사람은 자기 신발 끈을 잡아당겨 스스로를 들어 올릴 수가 없는가?".
- 1916 년에 의미가 확장되어 "**힘들게 혼자의 노력으로 헤쳐나가다" 라는 뜻**이 포함.
  - "컴퓨터의 운영시스템을 로딩하는 고정된 명령"(1953년)이라는 의미는 처음 로딩되는 프로그램이 자기 자신과 나머지를 구동시킨다는 개념

c.f) from sklearn.utils import resample



Sampling without replacement (WOR)

There are no common samples from the original dataset



Sampling with replacement (WR)

Note some of the samples from the original dataset are common in both samples (item 1,4,5)

Bagging = Bootstrap Aggregation

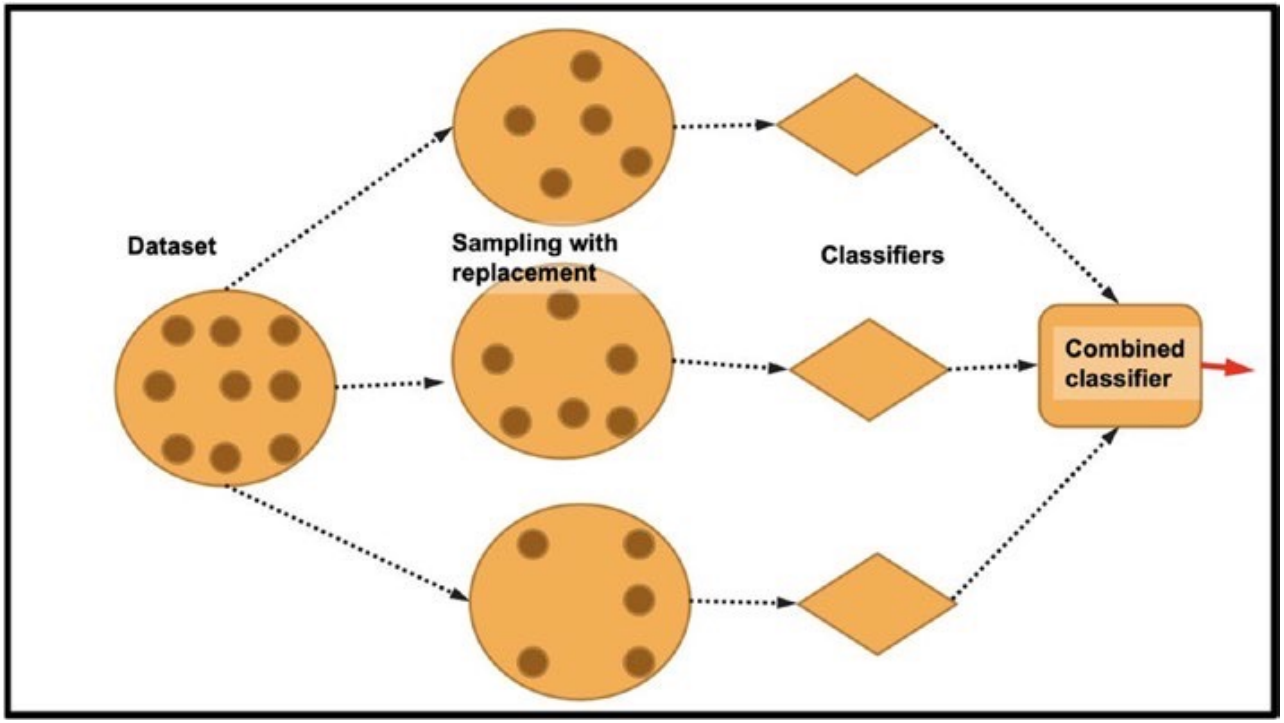
- Main Assumption:
  - Combining many unstable predictors to produce a ensemble (stable) predictor.
  - **Unstable Predictor: small changes in training data produce large changes in the model**.
    - e.g. Neural Nets, trees
    - Stable: SVM (sometimes), Nearest Neighbor.

- Hypothesis Space
  - Variable size (nonparametric):
    - Can model any function if you use an appropriate predictor (e.g. trees)

Given data: $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

For $m = 1 : M$

- Obtain bootstrap sample D_m from the training data D
- Build a model $G_m(\mathbf{x})$ from bootstrap data D_m



Bagging Model

- Regression

$$\hat{y} = \frac{1}{M}\sum_{m=1}^{M} G_m(\mathbf{x})$$

- Classification:
  - Vote over classifier outputs $G_1(\mathbf{x}), \ldots, G_M(\mathbf{x})$

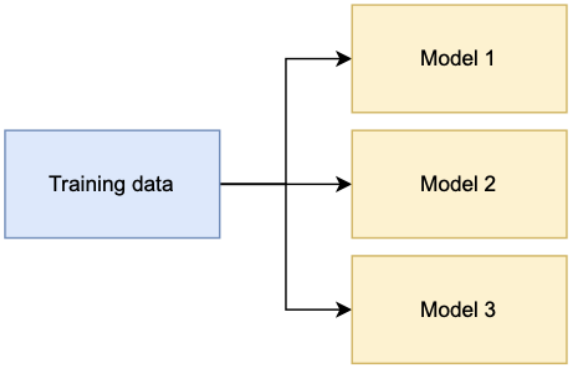
May help stabilize models

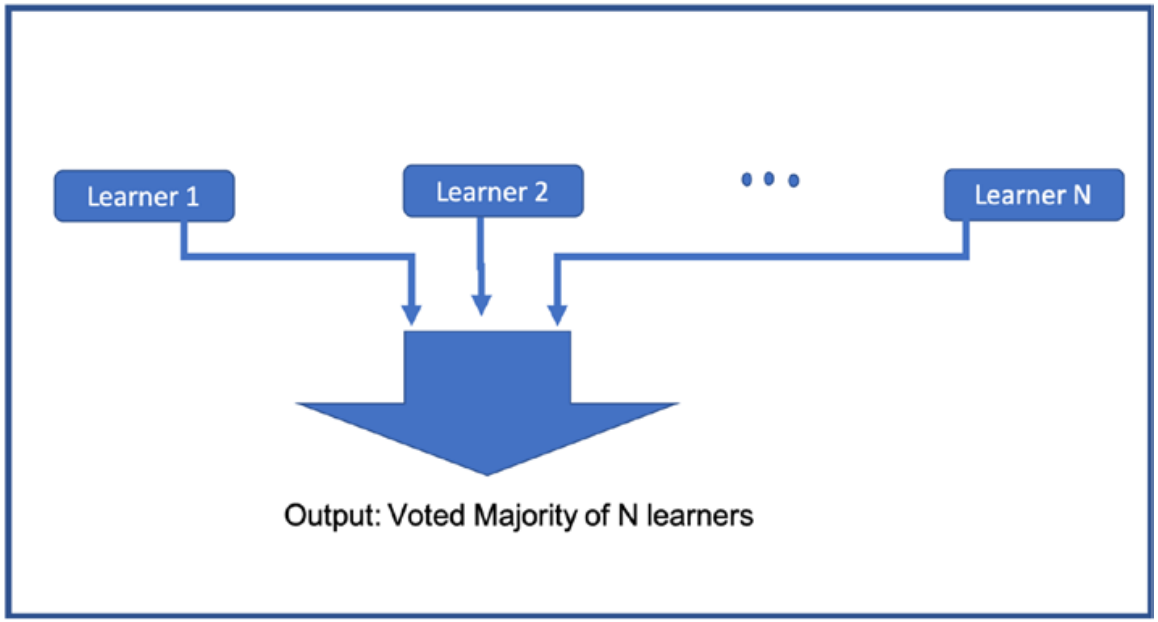
- Bootstrap sample of N instances is obtained by drawing N examples at random, with replacement.
- On average each bootstrap sample has 63% of instances
  - Encourages predictors to have uncorrelated errors
    - This is why it works
- Usually set $M = \sim 30$
  - Or use validation data to pick M
- The models $G_m(\mathbf{x})$ need to be unstable
  - Usually full length (or slightly pruned) decision trees.

Random Forest

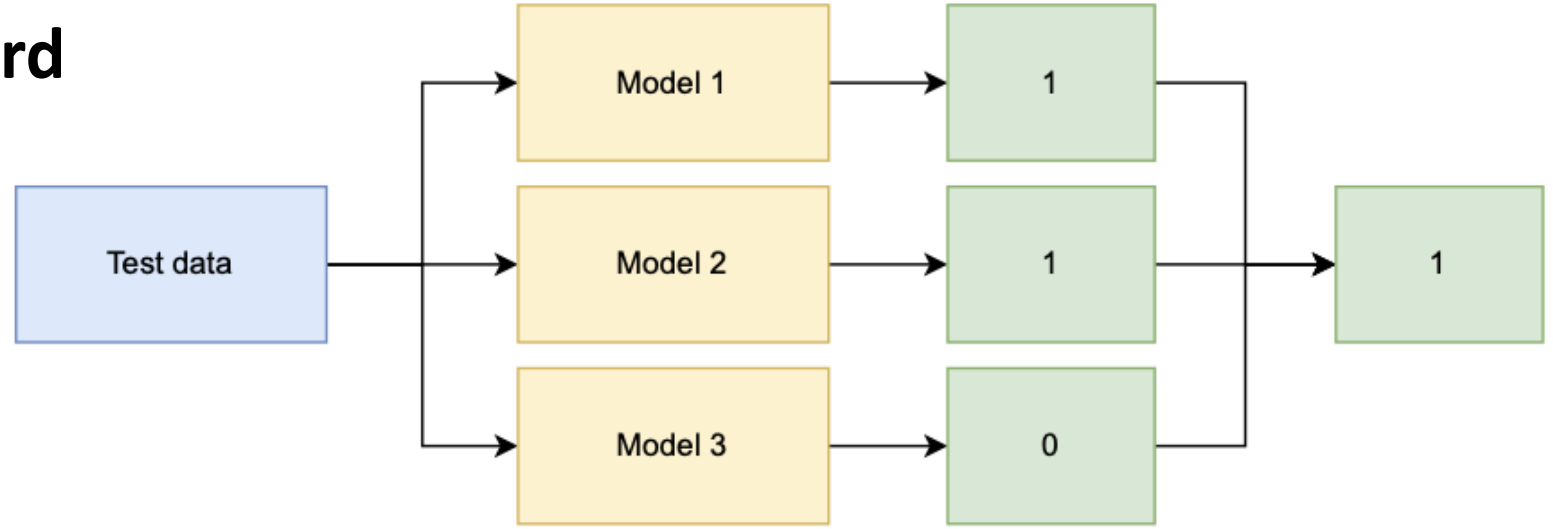
Mixing Models

1. Voting

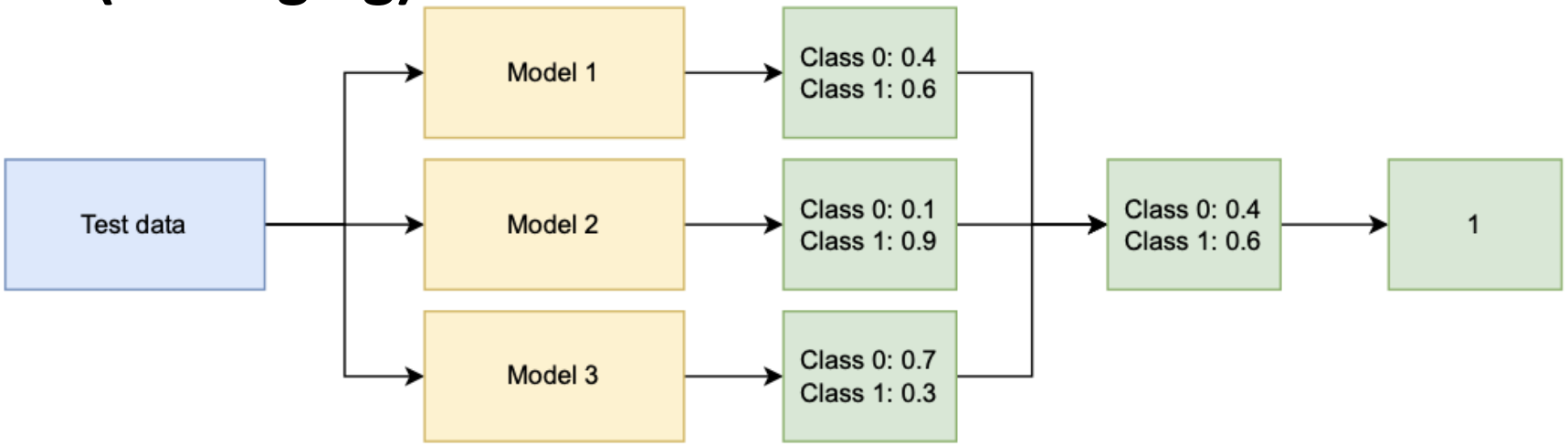




Hard



Soft (Averaging)



Weighted (Average)

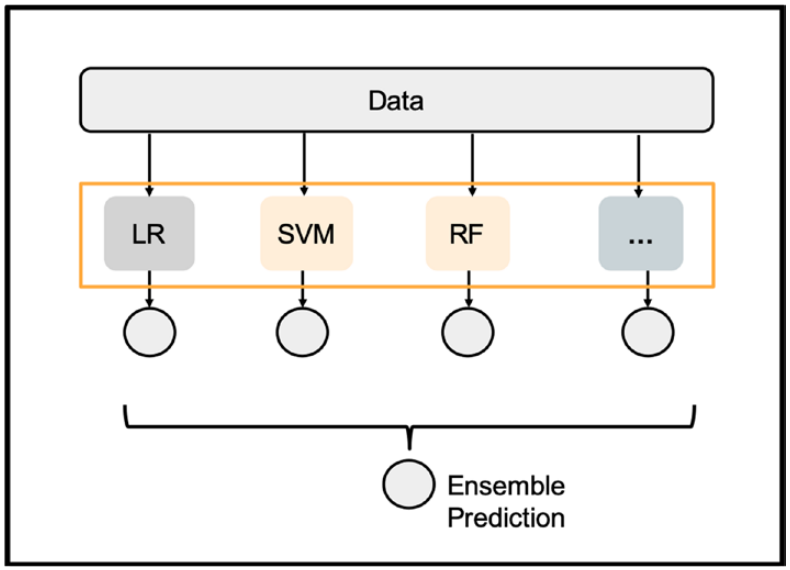
2. Hyperparameter Tuning

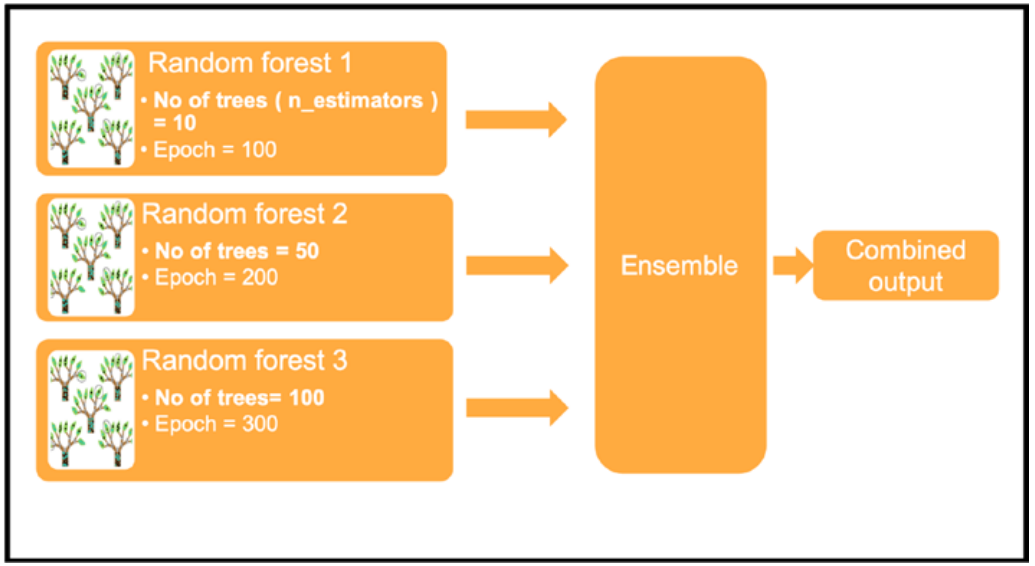
ex) Same model, different initializations

Use cross-validation to determine the best hyperparameters, then train multiple models with the best set of hyperparameters but with different random initialization. The danger with this approach is that the variety is only due to initialization.

ex) Top models discovered during cross-validation

Use cross-validation to determine the best hyperparameters, then pick the top few (e.g. 10) models to form the ensemble. This improves the variety of the ensemble but has the danger of including suboptimal models. In practice, this can be easier to perform since it doesn't require additional retraining of models after cross-validation



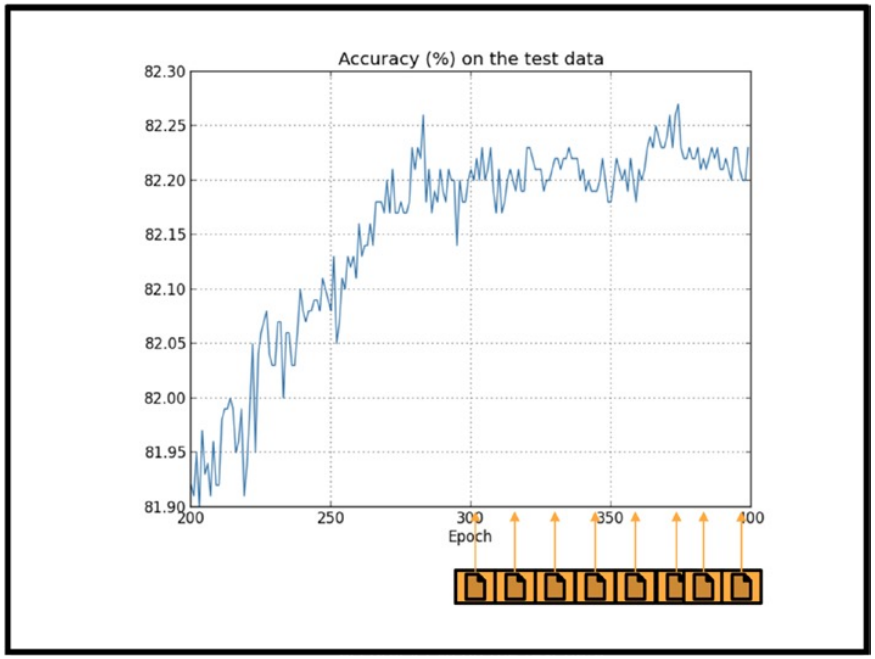


Hyperparameter tuning ensembles

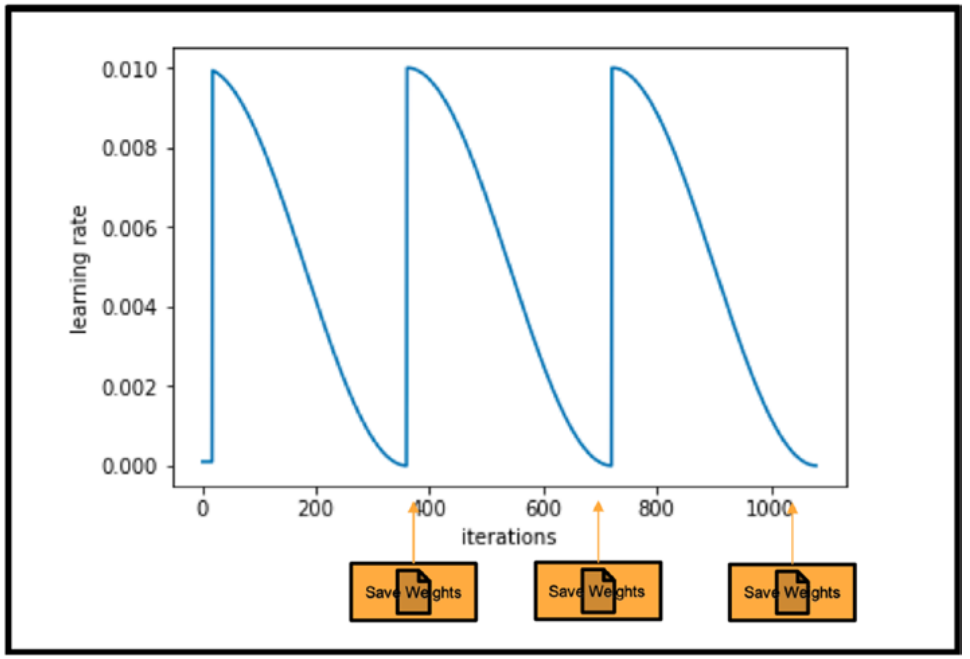
3. Different checkpoints of a single model

If training is very expensive, some people have had limited success in taking different checkpoints of a single network over time (for example after every epoch) and using those to form an ensemble. Clearly, this suffers from some lack of variety, but can still work reasonably well in practice. The advantage of this approach is that is very cheap.

- Snapshot
- Horizontal Epochs
- Vertical Representational



Horizontal voting ensembles



Snapshot ensemble

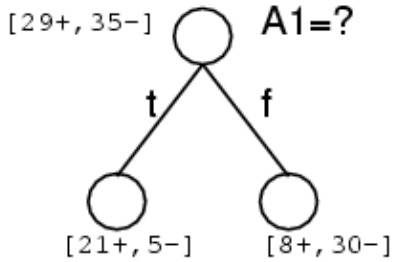
Mixing Combinations

1. **Model (Weighted) Averaging**
2. **Model Weight Averaging (Polyak)**

   ex) Running average of parameters during training.

   a cheap way of almost always getting an extra percent or two of performance is to maintain a second copy of the network's weights in memory that maintains an exponentially decaying sum of previous weights during training. This way you're averaging the state of the network over last several iterations. You will find that this "smoothed" version of the weights over last few steps almost always achieves better validation error. The rough intuition to have in mind is that the objective is bowl-shaped and your network is jumping around the mode, so the average has a higher chance of being somewhere nearer the mode.

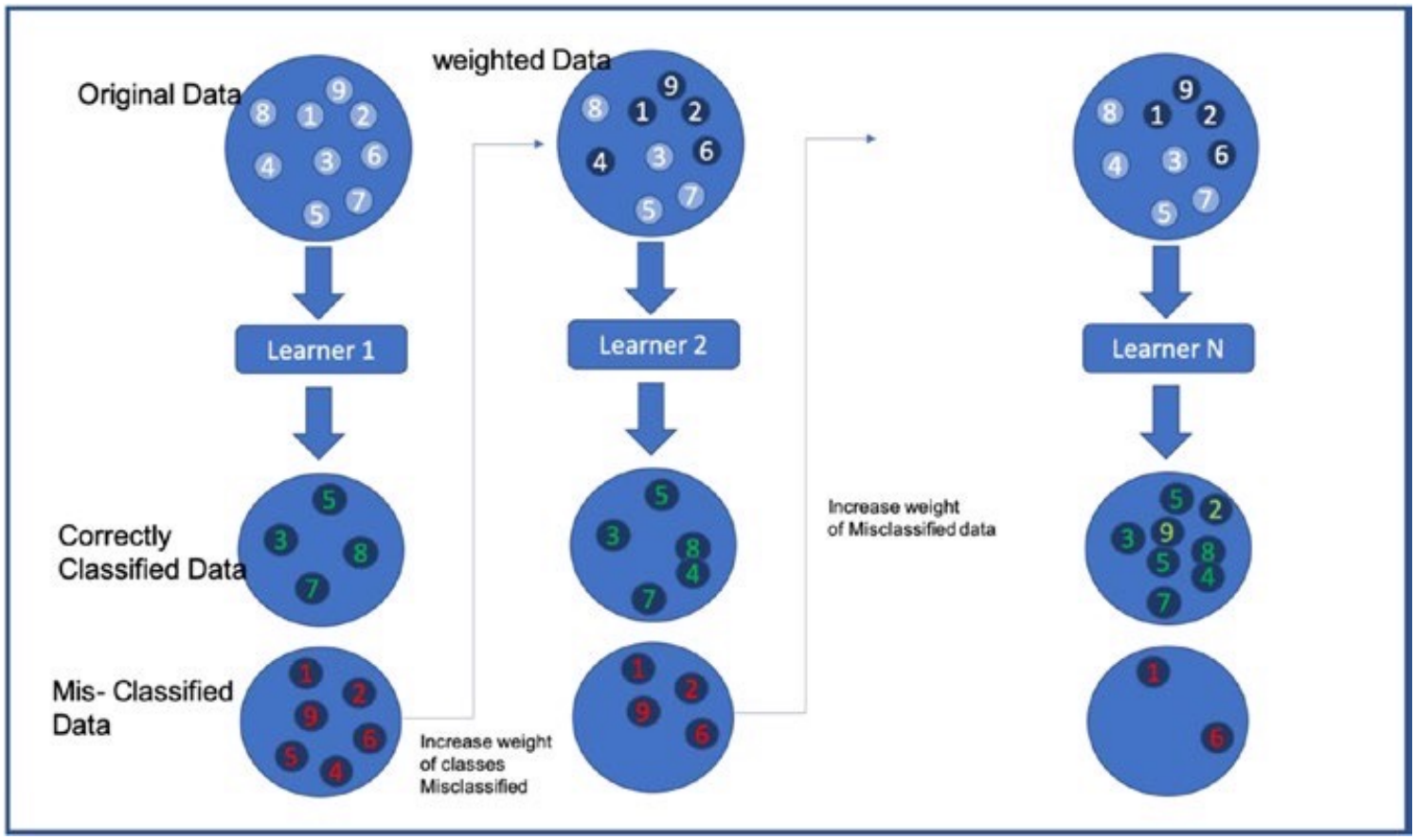
Boosting



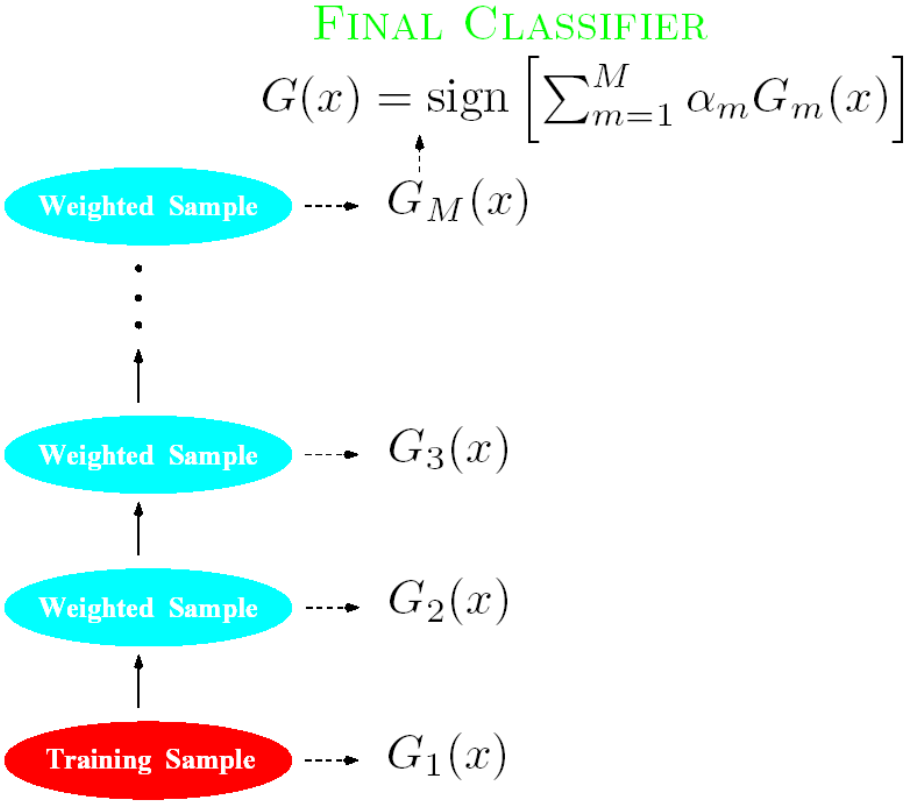
- Main Assumption:
  - Combining many weak predictors (e.g. tree stumps or 1-R predictors) to produce an ensemble predictor
  - The weak predictors or classifiers need to be **stable**

- Hypothesis Space
  - Variable size (nonparametric):
    - Can model any function if you use an appropriate predictor (e.g. trees)

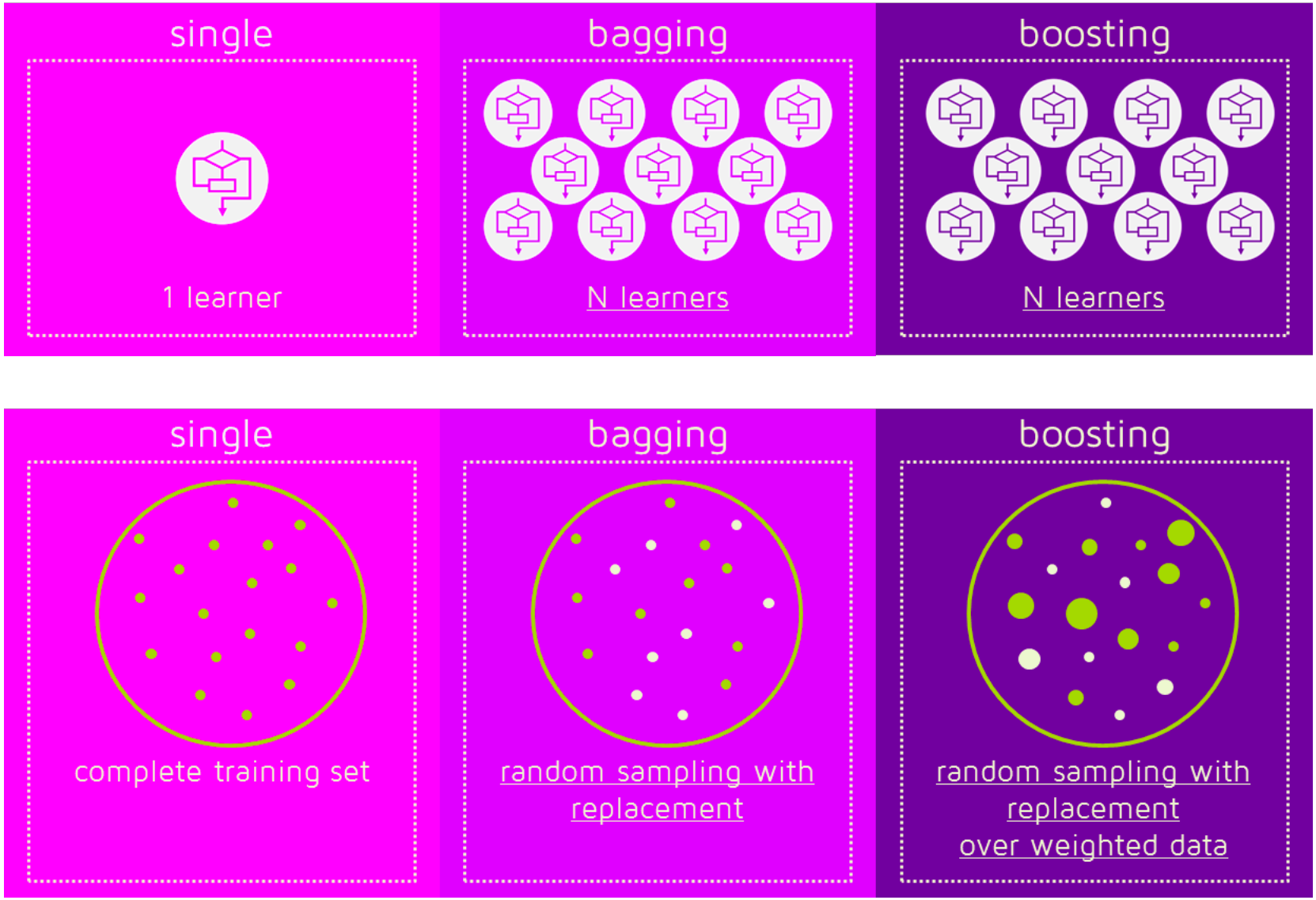
May help decrease prediction error


Original Data
weighted Data
Learner 1
Learner 2
Learner N
Correctly Classified Data
Mis- Classified Data
Increase weight of classes Misclassified
Increase weight of Misclassified data

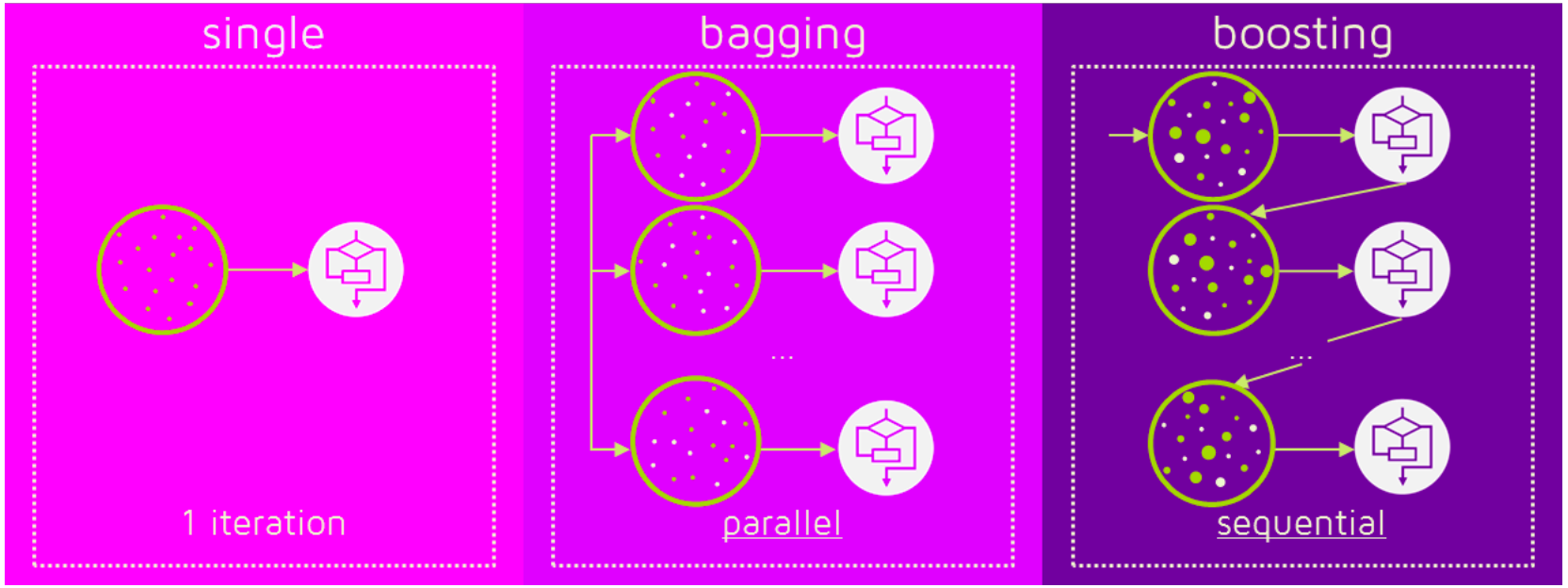
- Each predictor is created by using a biased sample of the training data
  - Instances (training examples) with high error are weighted higher than those with lower error
- Difficult instances get more attention
  - This is the motivation behind boosting



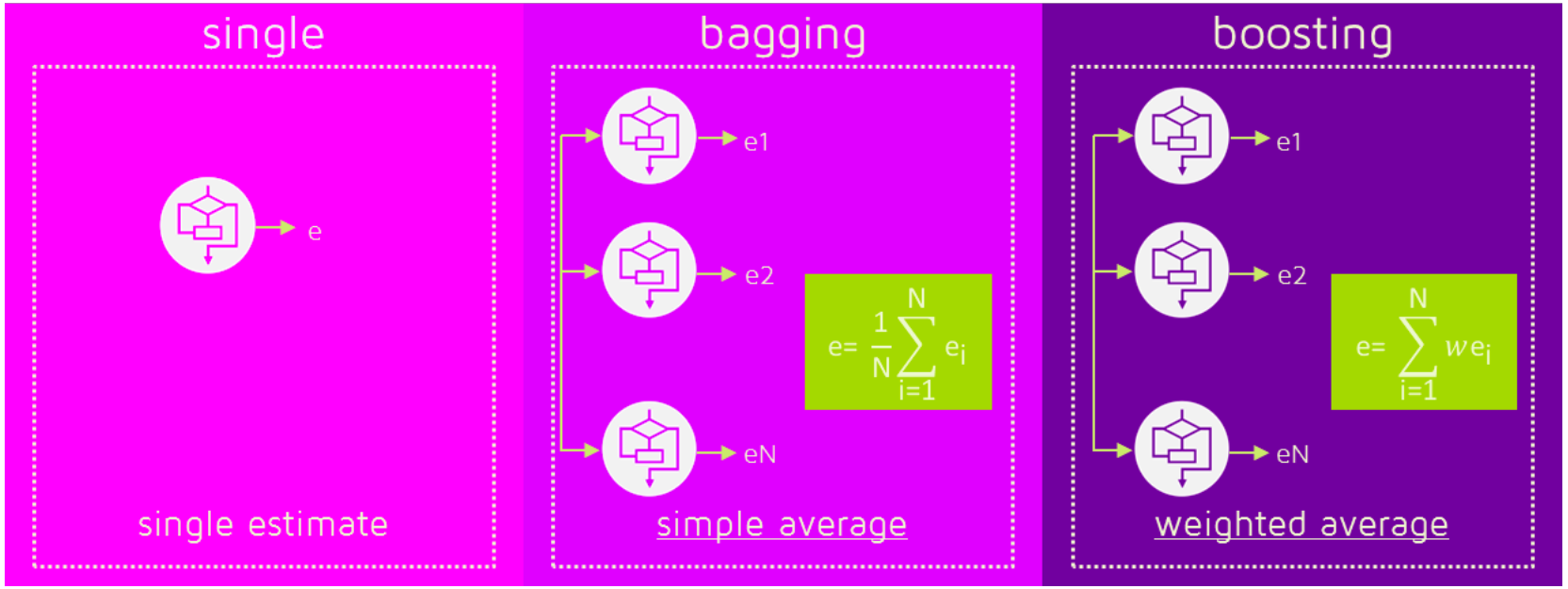
Each classifier $G_m(\mathbf{x})$ is trained from a weighted Sample of the training Data

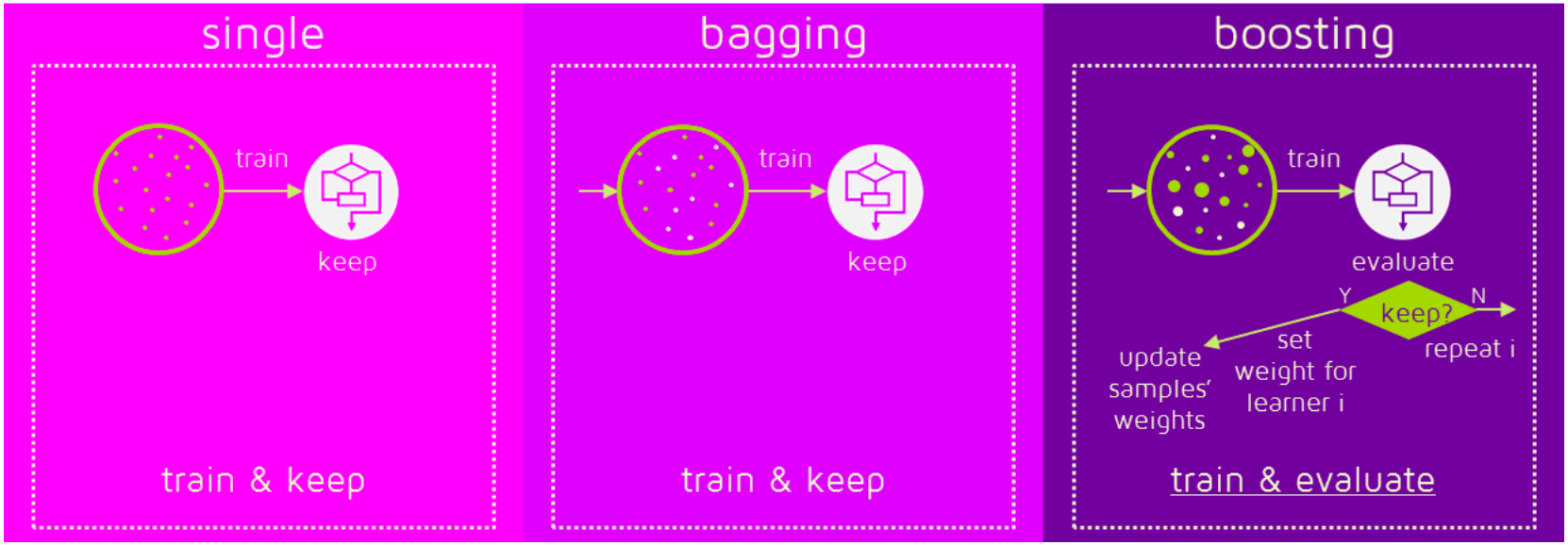


https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/



Usually same ML technique





Similarities	Differences
Both are ensemble methods to get N learners from 1 learner…	while they are built independently for Bagging, Boosting tries to add new models that do well where previous models fail.
Both generate several training data sets by random sampling…	only Boosting determines weights for the data to tip the scales in favor of the most difficult cases.
Both make the final decision by averaging the N learners (or taking the majority of them)…	it is an equally weighted average for Bagging and a weighted average for Boosting, more weight to those with better performance on training data.
Both are good at reducing variance and provide higher stability…	only Boosting tries to reduce bias. On the other hand, Bagging may solve the over-fitting problem, while Boosting can increase it.

AdaBoost

Given data: $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

1. Initialize weights $w_i = 1/N, i = 1, \ldots, N$
2. For $m = 1 : M$
   a) Fit classifier $G_m(\mathbf{x}) \in \{-1, 1\}$ to data using weights w_i
   b) Compute $$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(\mathbf{x}_i))}{\sum_{i=1}^{N} w_i}$$
   c) Compute $\alpha_m = \log((1 - err_m)/err_m)$
   d) Set $w_i \leftarrow w_i \exp[\alpha_m I(y_i \neq G_m(\mathbf{x}_i))], \quad i = 1, \ldots, N$

- The $I(s)$ function is defined as:

$$I(s) = \begin{cases} 1 \text{ if } s \text{ is true} \\ 0 \quad \text{otherwise} \end{cases}$$

Gradient (tree) boosting

Combines a set of weak learners to form a strong learner. There are three main items to note, as far as gradient boosting trees are concerned:

- a differential loss function (Can use any cost function) has to be used,
- decision trees are used as weak learners,
- it's an additive model, so trees are added one after the other. Gradient descent is used to minimize the loss when adding subsequent trees.

- Stochastic (Gradient) Boosting
  - Bootstrap Sample: Uniform random sampling (with replacement)
  - Often outperforms the non-random version

eXtreme Gradient Boosting

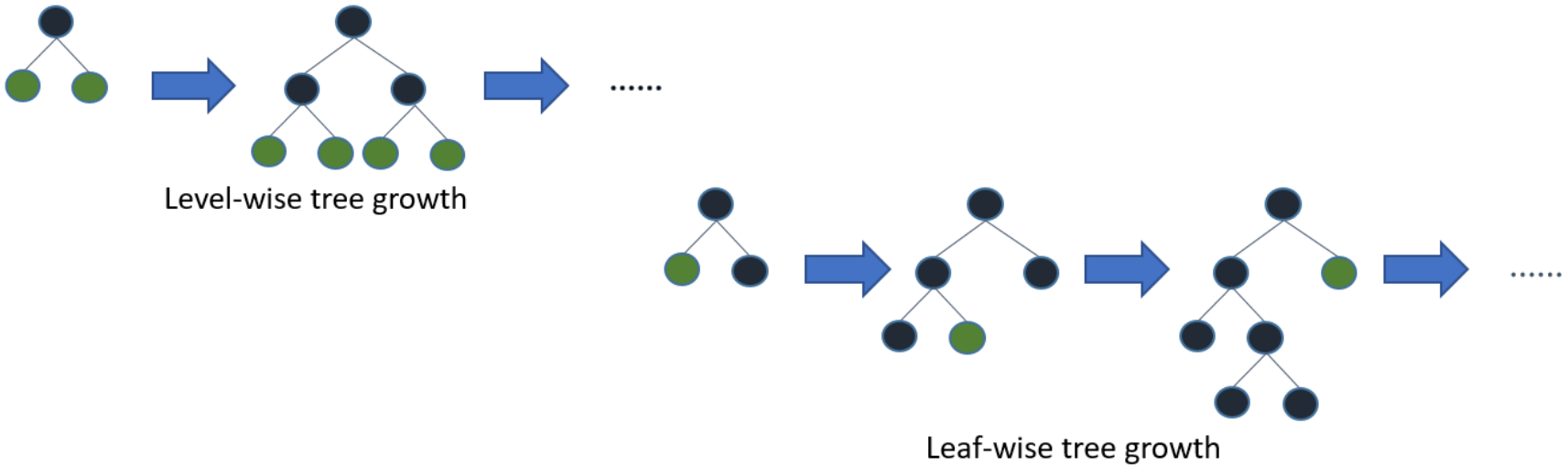
XGBoost Documentation — xgboost 2.0.0-dev documentation

- eXtreme Gradient Boosting, popularly known as XGBoost, is a top gradient boosting framework.
- It's based on an ensemble of weak decision trees. It can do parallel computations on a single computer.
- The algorithm uses regression trees for the base learner. It also has cross-validation built-in.
- ML love it for its accuracy, efficiency, and feasibility.

LightGBM

[Welcome to LightGBM's documentation! — LightGBM 3.3.5.99 documentation]

a gradient boosting algorithm based on tree learning. Unlike other tree-based algorithms that use depth-wise growth, LightGBM uses leaf-wise tree growth. Leaf-wise growth algorithms tend to converge faster than dep-wise-based algorithms.



LightGBM can be used for both regression and classification problems by setting the appropriate objective.

CatBoost

catboost/catboost: A fast, scalable, high performance Gradient Boosting on Decision Trees library, used for ranking, classification, regression and other machine learning tasks

a depth-wise gradient boosting library developed by Yandex. It grows a balanced tree using oblivion decision trees. The same features are used when making left and right splits at each level.

Researchers need Catboost for the following reasons:

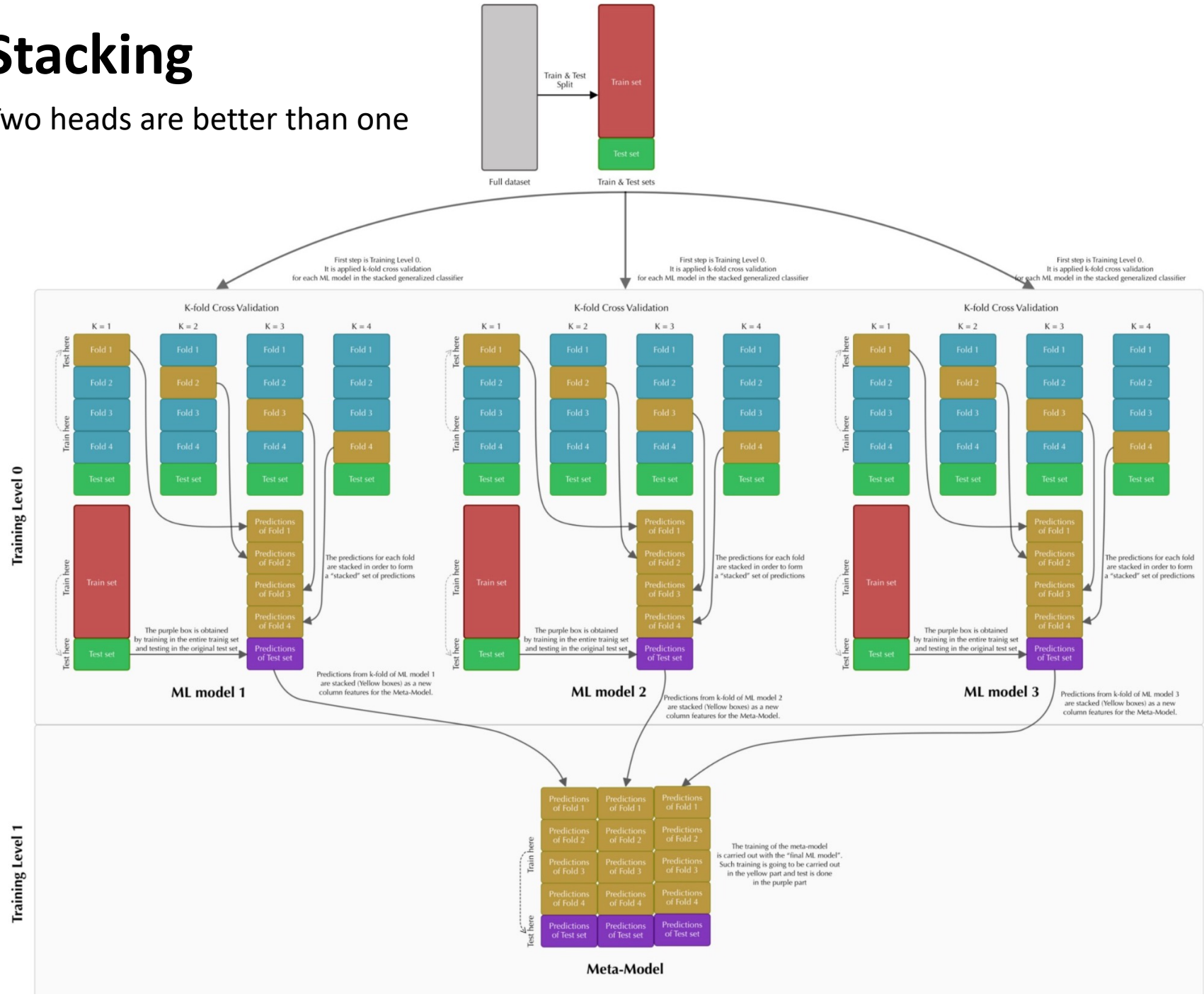
- The ability to handle categorical features natively,
- Models can be trained on several GPUs,
- It reduces parameter tuning time by providing great results with default parameters,
- Models can be exported to Core ML for on-device inference (iOS),
- It handles missing values internally,
- It can be used for both regression and classification problems.

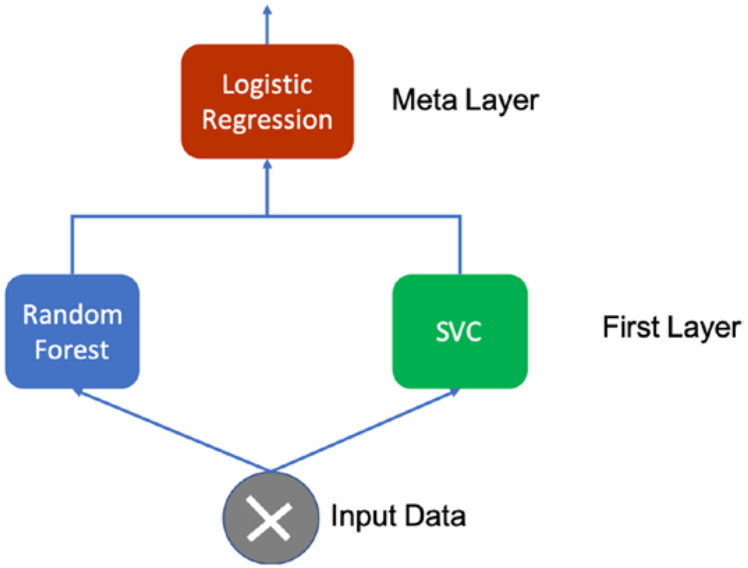
Boosting Summary

- Good points
  - Fast learning
  - Capable of learning any function (given appropriate weak learner)
  - Feature weighting
  - Very little parameter tuning
- Bad points
  - Can overfit data
  - Only for binary classification
- Learning parameters (picked via cross validation)
  - Size of tree
  - When to stop

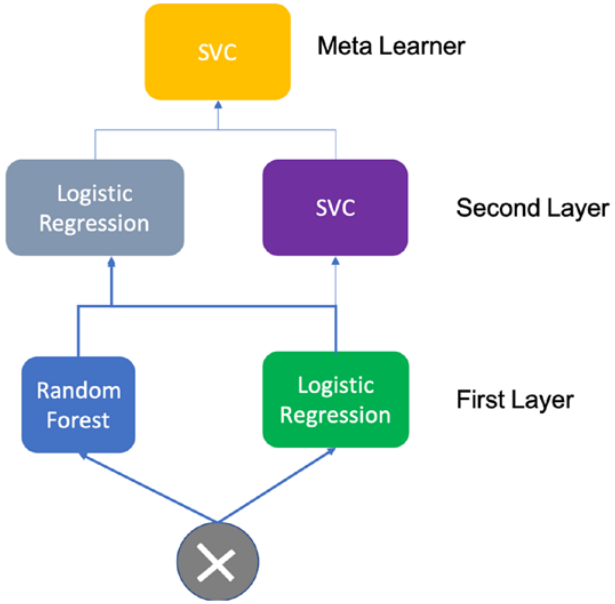
Stacking

Two heads are better than one



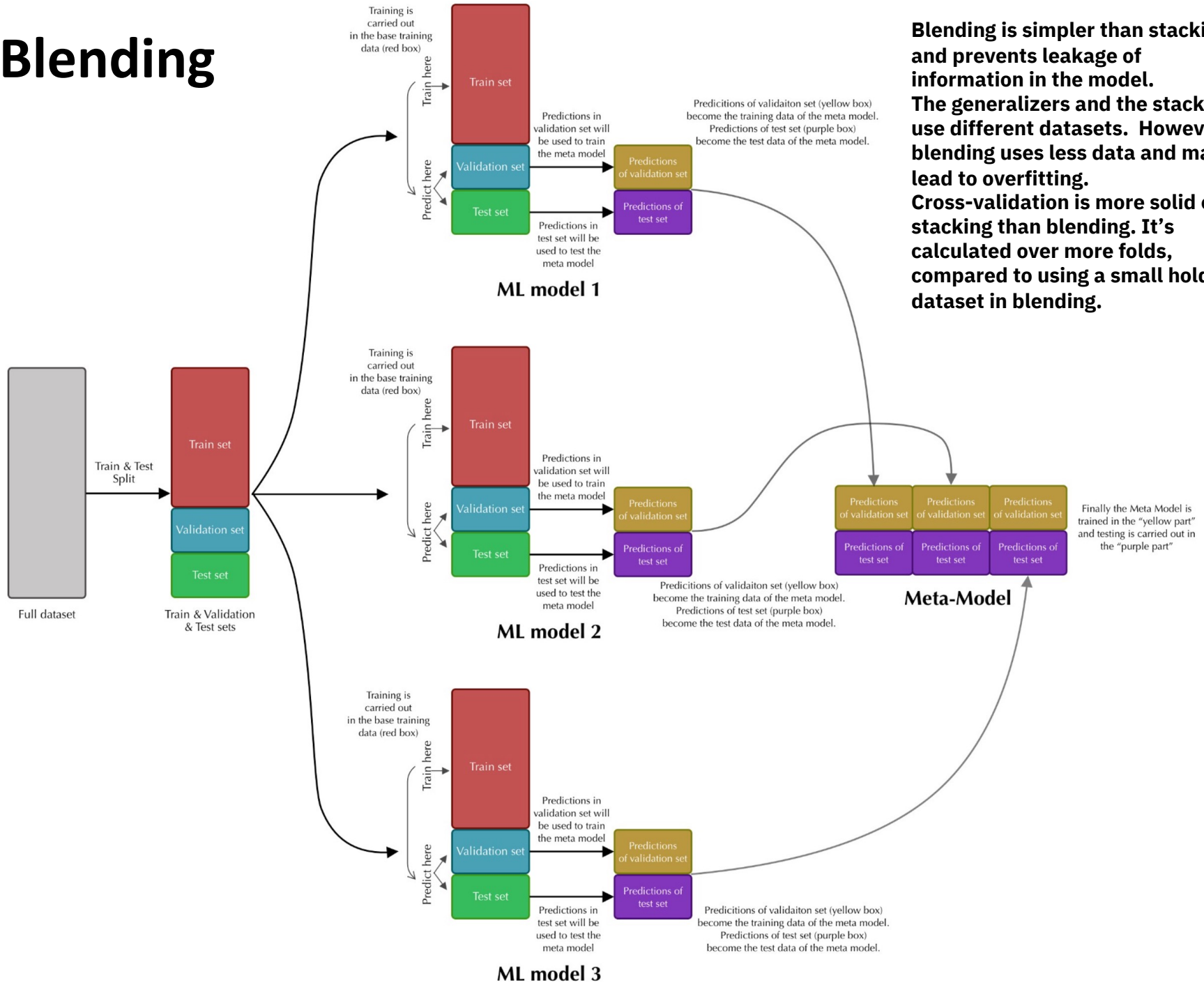


Single layer stacked ensemble



Multilayer stacked ensemble

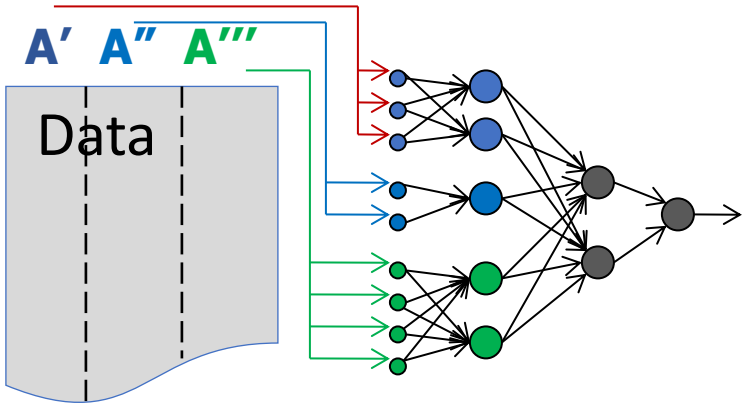
Blending

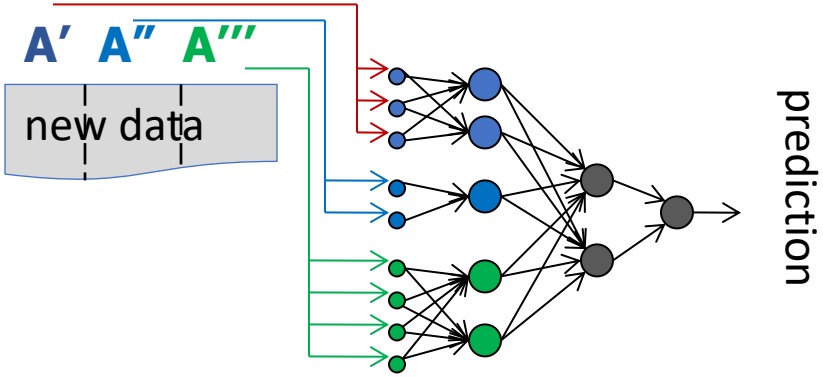


Blending is simpler than stacking and prevents leakage of information in the model.
The generalizers and the stackers use different datasets. However, blending uses less data and may lead to overfitting.
Cross-validation is more solid on stacking than blending. It's calculated over more folds, compared to using a small hold-out dataset in blending.

Mixture of Experts

1. **Split data attributes into domain meaningful subgroups: A', A", …**
2. **Create and train a Mixture of Experts Feed-Forward Neural Net:**
3. **Given a new unlabeled data instance, feed it forward through the mixture of experts**





the mixture of experts prediction is the output produced by the network

May help speed-up ANN training without increasing prediction error

Difficulties in ensemble learning

Weak or noisy data

The most important ingredient of a successful model is the dataset. If the data contains noise or incomplete information, there is not a single machine learning technique that will generate a highly performant model.

Let's illustrate this with a simple example. Suppose we study populations (in the statistical sense) of cars and we gather data about the color, shape, and manufacturer. It is difficult to generate a very accurate model for either variable, as a lot of cars are the same color and shape but are made by a different manufacturer. The following table depicts this sample dataset.

The best any model can do is achieve 33% classification accuracy, as there are three viable choices for any given feature combination. Adding more features to the dataset can greatly improve the model's performance. Adding more models to an ensemble cannot improve performance:

Noise, bias, and Variance

The combination of decisions from multiple models can help improve the overall performance. Hence, one of the key factors to use ensemble models is overcoming these issues: noise, bias, and variance. If the ensemble model does not give the collective experience to improve upon the accuracy in such a situation, then a careful rethinking of such employment is necessary.

Generalizations

There are many claims that ensemble models have more ability to generalize, but other reported use cases have shown more generalization errors. Therefore, it is very likely that ensemble models with no careful training process can quickly produce high overfitting models.

Understanding interpretability (Simplicity and Explainability)

Machine learning models, especially those put into production environments, are preferred to be simpler than complicated. The ability to explain the final model decision is empirically reduced with an ensemble.

By employing a large number of models, interpretability is greatly reduced. For example, a single decision tree can easily explain how it produced a prediction, by simply following the decisions made at each node. On the other hand, it is difficult to interpret why an ensemble of 1,000 trees predicted a single value. Moreover, depending on the ensemble method, there may be more to explain than the prediction process itself. How and why did the ensemble choose to train these specific models. Why did it not choose to train other models? Why did it not choose to train more models?

When the model's results are to be presented to an audience, especially a not-so-highly-technical audience, simpler but more easily explainable models may be a better solution. Furthermore, when the prediction must also include a probability (or confidence level), some ensemble methods (such as boosting) tend to deliver poor probability estimates:

Computational cost (especially, Inference time)

Another drawback of ensembles is the computational cost they impose. Training a single neural network is computationally expensive. Training a 1000 of them requires a 1000 times more computational resources. Furthermore, some methods are sequential by nature. This means that it is not possible to harness the power of distributed computing. Instead, each new model must be trained when the previous model is completed. This imposes time penalties on the model's development process, on top of the increased computational cost. Computational costs do not only hinder the development process; when the ensemble is put into production, the inference time will suffer as well. If the ensemble consists of 1,000 models, then all of those models must be fed with new data, produce predictions, and then those predictions must be combined in order to produce the ensemble output. In latency-sensitive settings (financial exchanges, real-time systems, and so on), sub-millisecond execution times are expected, thus a few microseconds of added latency can make a huge difference.

Although we might be relaxed to accept a longer time for model training, inference time is still critical. When deploying ensemble models into production, the amount of time needed to pass multiple models increases and could slow down the prediction tasks' throughput.

Google AI Blog: Model Ensembles Are Faster Than You Think (googleblog.com)

c.f) Knowledge Distill

Choosing the right models

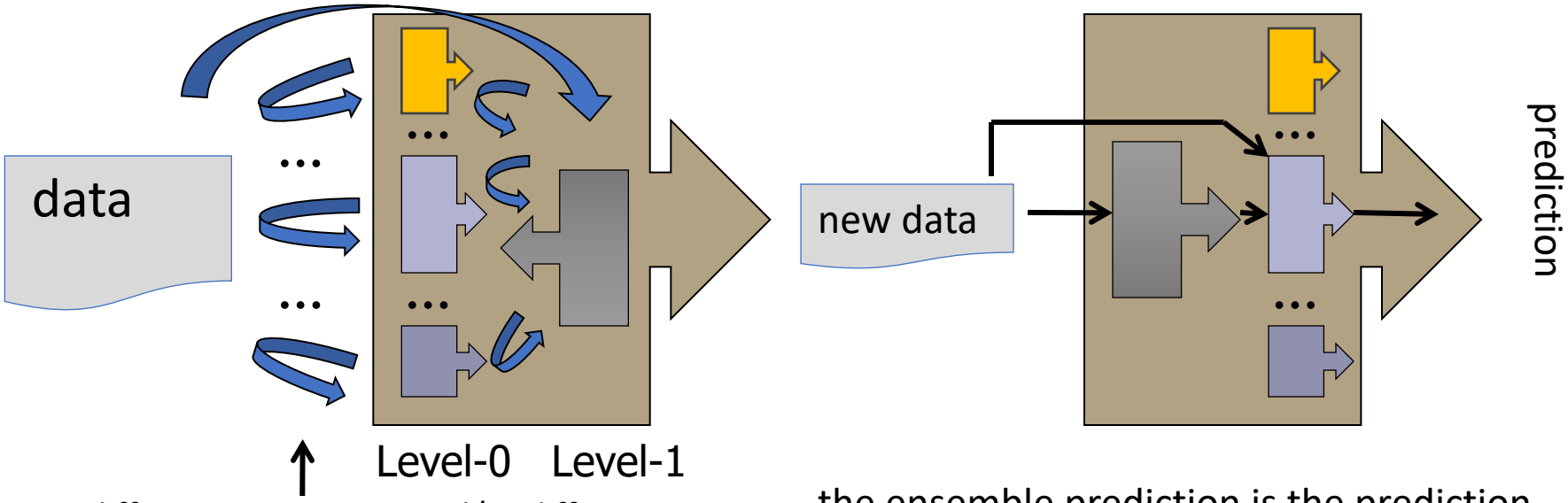
the models that comprise the ensemble must possess certain characteristics. There is no point in creating any ensemble from a number of identical models. Generative methods may produce their own models, but the algorithm used as well as its initial hyperparameters are usually selected by the analyst. Furthermore, the model's achievable diversity depends on a number of factors, such as the size and quality of the dataset, and the learning algorithm itself.

A single model that is similar in behavior to the data-generating process will usually outperform any ensemble, both in terms of accuracy as well as latency. In our bias-variance example, the simple sine function will always outperform any ensemble, as the data is generated from the same function with some added noise. An ensemble of many linear regressions may be able to approximate the sine function, but it will always require more time to train and execute. Furthermore, it will not be able to generalize (predict out-of- sample) as well as the sine function.

Meta-learning

1. **Train different Level-0 models**
2. **Train a Level-1 model to predict which is the best Level-0 model for a given data instance**

2. Given a new unlabeled data instance, input it to the Level-1 model:



Using different parameters and/or different ML techniques

the ensemble prediction is the prediction of the Level-0 model selected by the Level-1 model for the input data instance

May help determine what technique/model works best on given data

Reference

- **Bagging (Bootstrap Aggregating)**
  - Breiman, UC Berkeley
    - "Bagging Predictors". *Machine Learning,* 24(2), 123-140. 1996.
- **Boosting**
  - Schapire, ATT Research (now at Princeton U). Friedman, Stanford U.
    - "The strength of weak learnability." *Machine Learning.* 5(2), 197-227. 1990
    - "Experiments with a new boosting algorithm." *Proc. of the 13th Intl. Conf. on Machine Learning.* 148-156. 1996.
    - "Additive Logistic Regression: a statistical view of boosting". *Annals of Statistics.* 1998.
- **Stacking**
  - Wolpert, NASA Ames Research Center
    - "Stacked Generalization." *Neural Networks*. 5(2), 241-259. 1992.
- **Mixture of Experts in Neural Nets**
  - Alvarez, Ruiz, Kawato, Kogel, Boston College and WPI
    - Faster neural networks for combined collaborative and content based recommendation. *Journal of Computational Methods in Sciences and Engineering (JCMSE).* IOS Press. Vol. 11, N. 4, pp. 161-172. 2011.
- **Model Selection Meta-learning**
  - Floyd, Ruiz, Alvarez, WPI and Boston College
    - "Model Selection Meta-Learning for the Prognosis of Pancreatic Cancer", full paper, *Proc. 3rd Intl. Conf. on Health Informatics (HEALTHINF 2010)*, pp. 29-37. 2010.