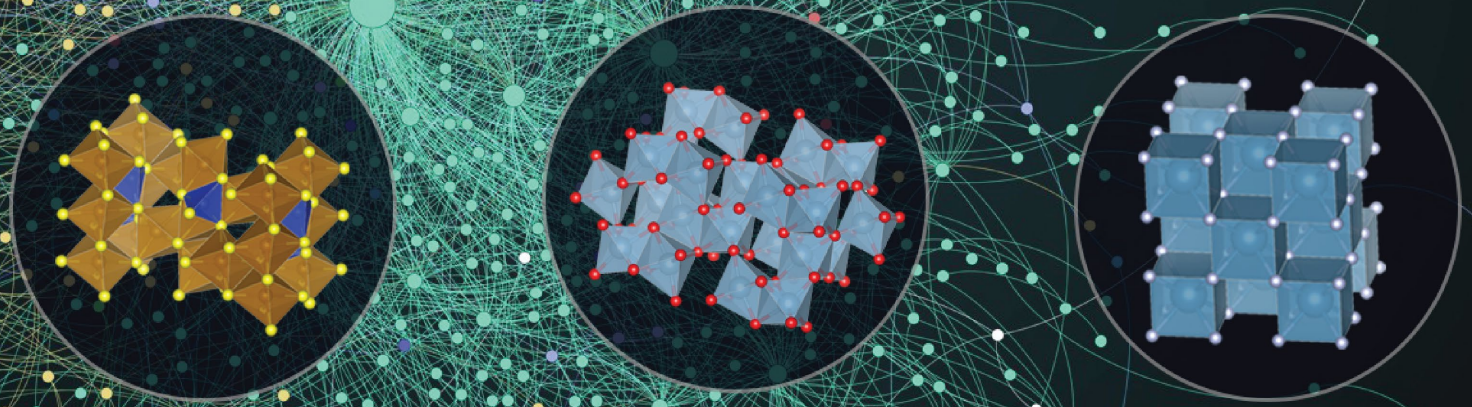
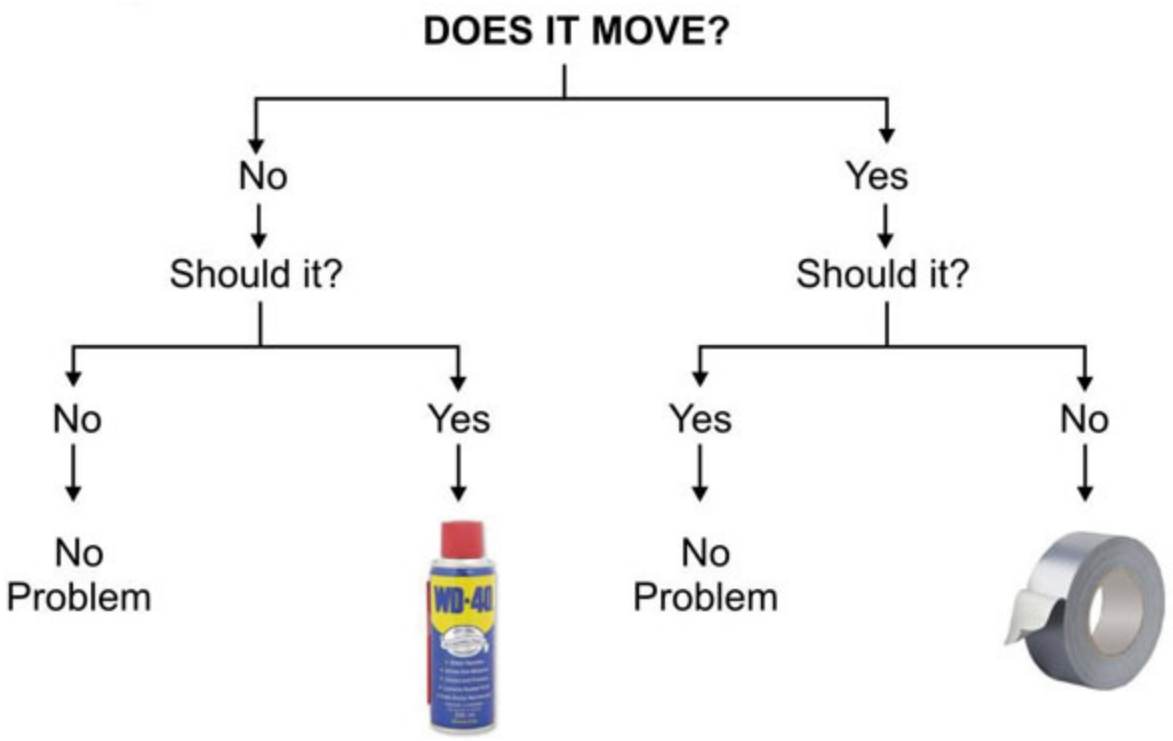
# ensemble techniques



# Ensembling: many weak learners can accomplish what a few strong learners cannot

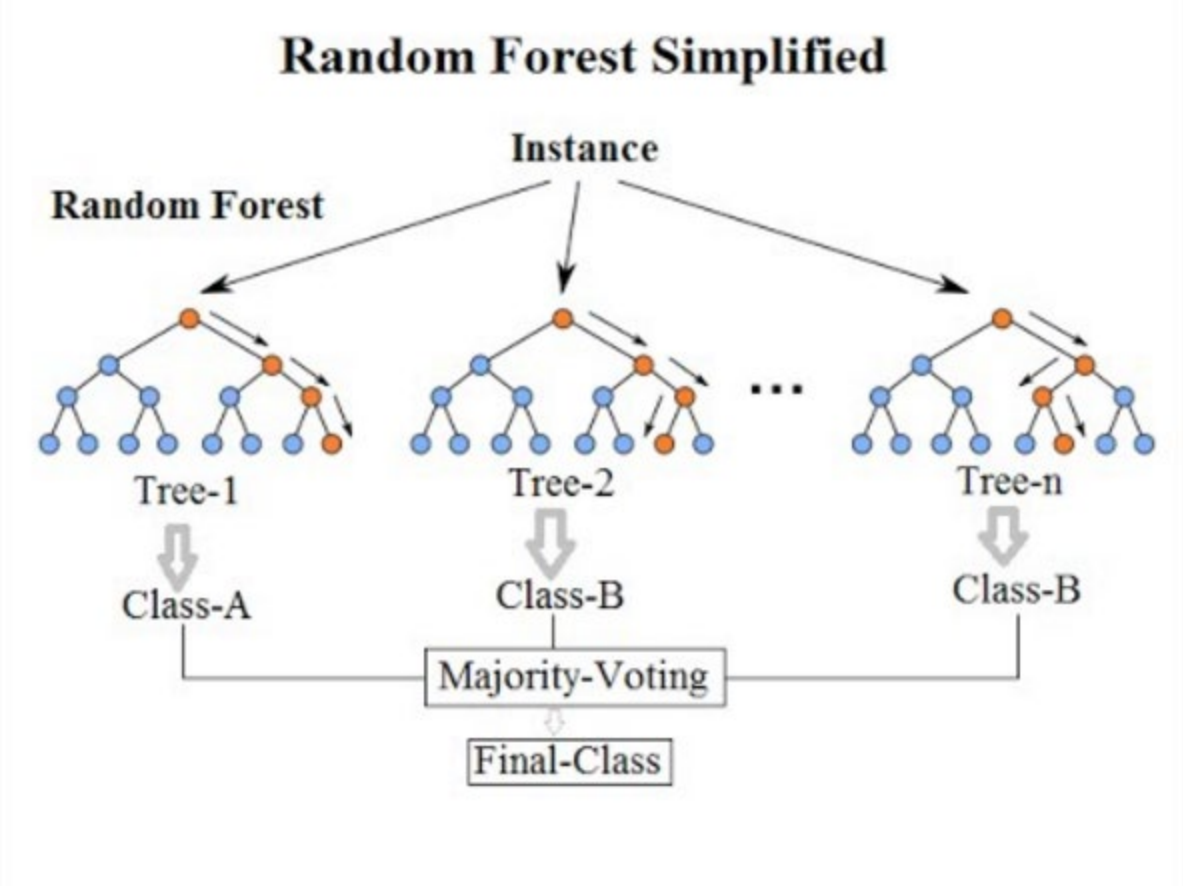
https://knowyourmeme.com/memes/zerg-rush, https://www.youtube.com/watch?v=jNMocWgp9oE&ab_channel=catboi333

# Ensemble techniques rely on a large number of learning events




https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

# A forest of individual trees is powerful for several reasons




Multiple small trees can be calculated in parallel*

Different trees can sample different features

*depending on algorithm!

https://en.wikipedia.org/wiki/Random_forest#/media/File:Random_forest_diagram_complete.png

# Ensemble methods include strong learners made up of weak learners

Weak learners (sometimes called base models):

- Simple
- Building blocks for harder models
- Do poorly alone due to high bias (low complexity) or high variance (high complexity)

Strong learner (ensemble model):

- Combines several weak learners
- Overall improved performance

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

# Which weak learners should be combined and how?

Which?

- Combining weak learners of the same type is most common and most simple
  - “homogeneous” ensemble
- Base models of different types are also possible, but less common
  - “heterogeneous” ensemble

How?

- Guiding principle is “coherent aggregation”
  - If we choose base models with low bias but high variance, it should be with an aggregating method that tends to reduce variance, whereas if we choose base models with low variance but high bias, it should be with an aggregating method that tends to reduce bias.

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

# There are 3 common model combination options

Bagging
- Homogeneous weak learners
- Learning is independent and in parallel
- Combined via deterministic averaging

Boosting
- Homogeneous weak learners
- Learning is sequential and adaptive (depends on previous)
- Combined via deterministic strategy

Stacking
- Heterogeneous weak learners
- Learning is independent and in parallel
- Combines them with a meta-model using weak model outputs as inputs to the meta model

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

# There are 3 common model combination options

Bagging
- Homogeneous weak learners
- Learning is independent and in parallel
- Combined via deterministic averaging

Bagging tends to reduce variance

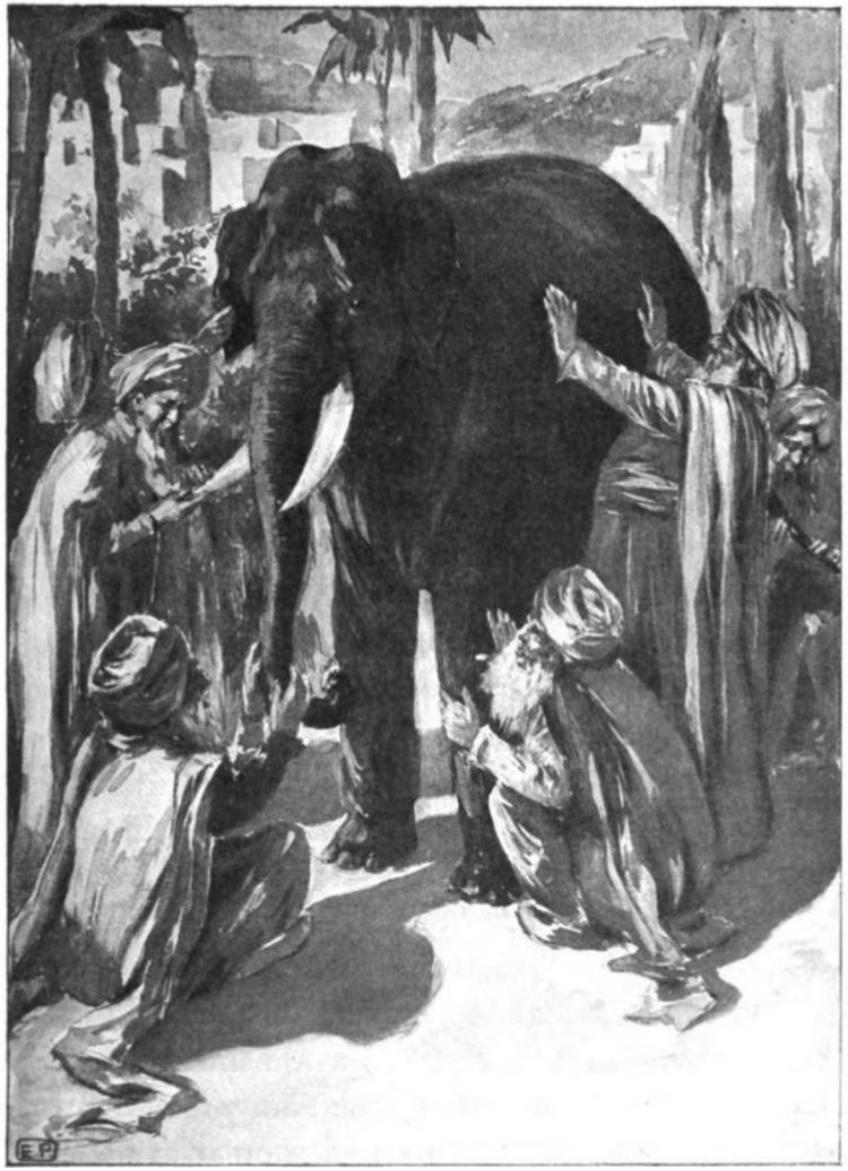
Boosting
- Homogeneous weak learners
- Learning is sequential and adaptive (depends on previous)
- Combined via deterministic strategy

Boosting/Stacking tends to reduce bias but may also reduce variance

Stacking
- Heterogeneous weak learners
- Learning is independent and in parallel
- Combines them with a meta-model using weak model outputs as inputs to the meta model

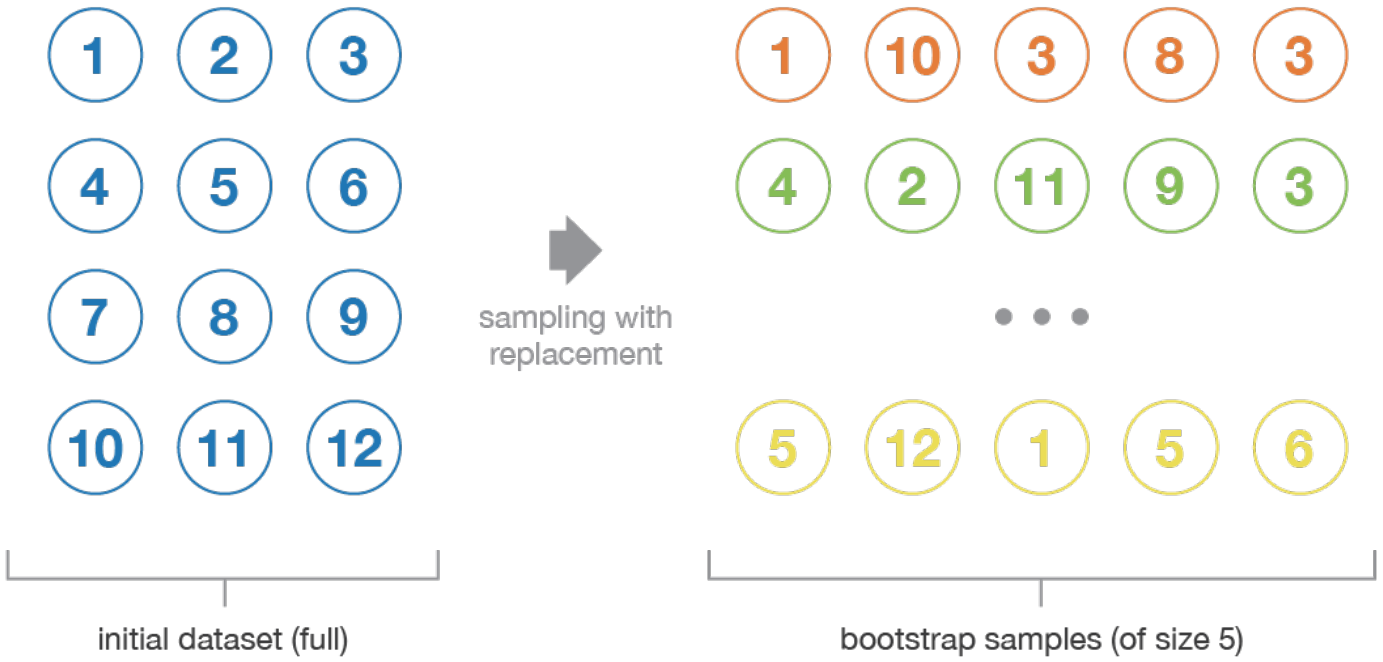
https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

# Each learner has their own perspective and biases



https://en.wikipedia.org/wiki/Blind_men_and_an_elephant#/media/File:Blind_men_and_elephant4.jpg

# Bagging is short for "bootstrap aggregating"




Bootstrapping:
generating samples of size B (called bootstrap samples) from an initial dataset of size N by randomly drawing with replacement B observations.

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

# Is bootstrapping an acceptable way to create a dataset?

In probability theory and statistics, a collection of random variables is independent and identically distributed (I.I.D.) if each random variable has the same probability distribution as the others and all are mutually independent.

| | |
|---|---|
| We are both flipping the same coin (IID) | We are both flipping the same coin, but we scratch it every 1,000 flips (not I ID) |
| We are flipping coins that are imbalanced differently (I not ID) | We are working with differently weighted coins and scratch them every 1,000 flips (not I not ID) |

Example courtesy of Jason Hattrick-Simpers and Brian DeCost

https://en.wikipedia.org/wiki/Independent_and_identically_distributed_random_variables

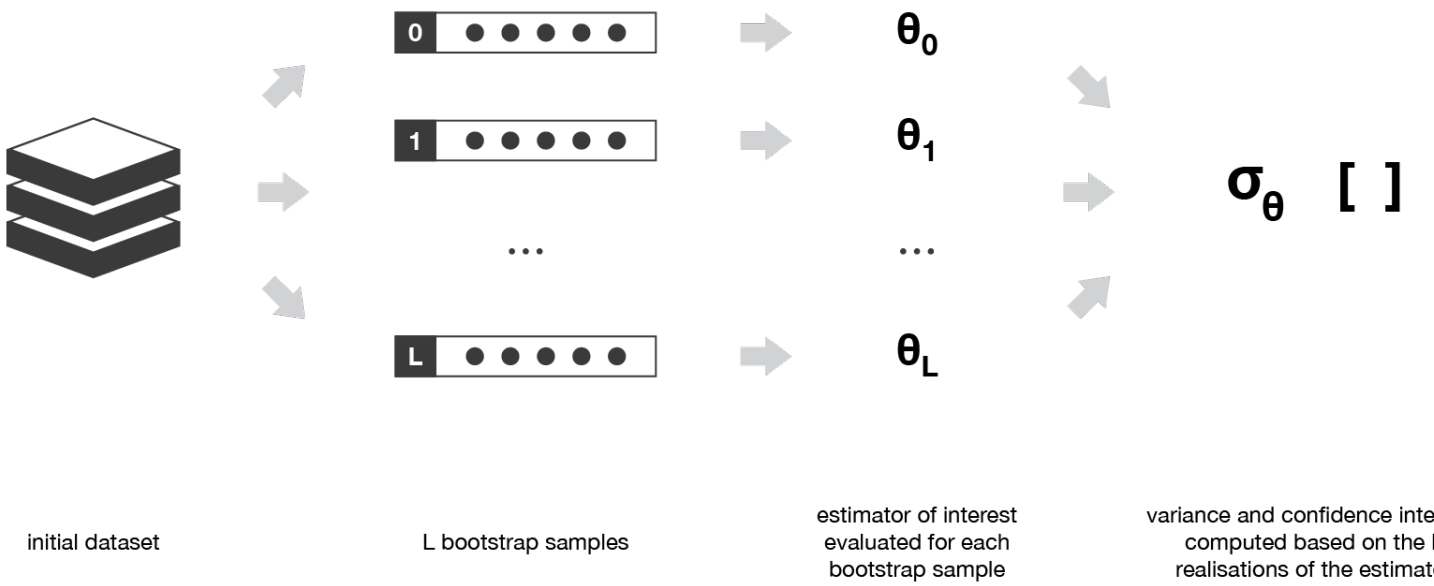
# What does I.I.D. look like in materials data example?

| | |
|---|---|
| We use an oxide database to predict oxide material phases | We use a time-varying oxide database to predict oxide phases (not I ID) |
| We use an oxide database to predict NITRIDE phases (I not ID) | We use a time-varying oxide database to predict NITRIDE phases (not I not ID) |

# For bootstrapping to approximately adhere to I.I.D. we should follow a few guidelines

First, the size N of the initial dataset should be large enough to capture most of the complexity of the underlying distribution so that sampling from the dataset is a good approximation of sampling from the real distribution (**representativity**)

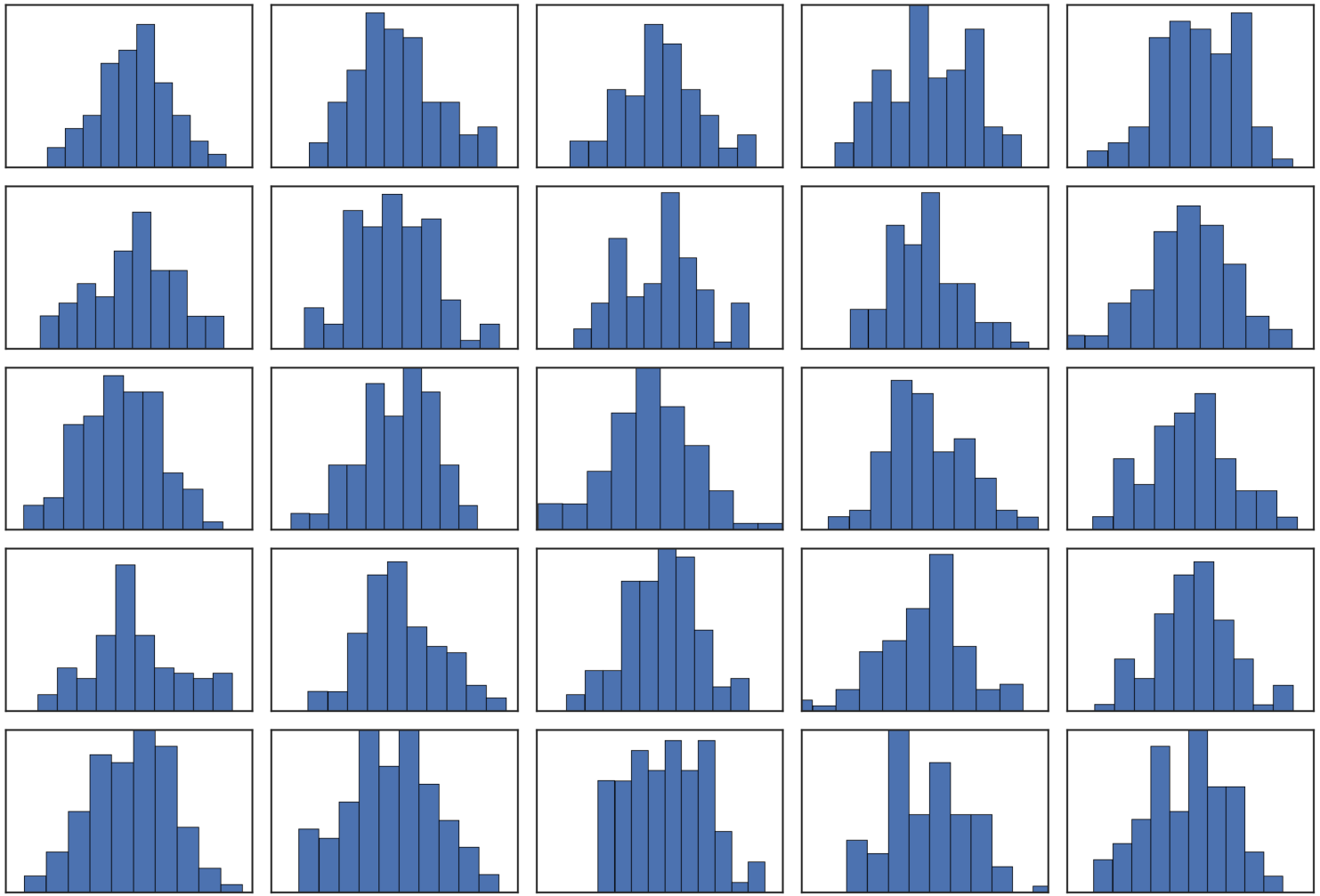
Second, the size N of the dataset should be large enough compared to the size B of the bootstrap samples so that samples are not too much correlated (**independence**).




Bootstrapping allows us to create a large amount of data based on a single distribution where each dataset is approximately i.i.d.

https://en.wikipedia.org/wiki/Independent_and_identically_distributed_random_variables

# Each dataset is a **sample** coming from a true, unknown distribution



Fitted models are trained on datasets, not the underlying true, unknown distribution!

So each model is subject to variability

Bootstrapping gives us many nearly independent model datasets

http://work.thaslwanter.at/Stats/html/statsDistributions.html

# Mathematically, bootstrapping is…

If we have L bootstrap samples of size B

$$\{z_1^1, z_2^1 \dots z_B^1\}, \{z_1^2, z_2^2 \dots z_B^2\}, \dots \{z_1^L, z_2^L \dots z_B^L\}$$

Fit each bootstrap sample to get L weak learners

$$w_1, w_2, \dots w_L$$

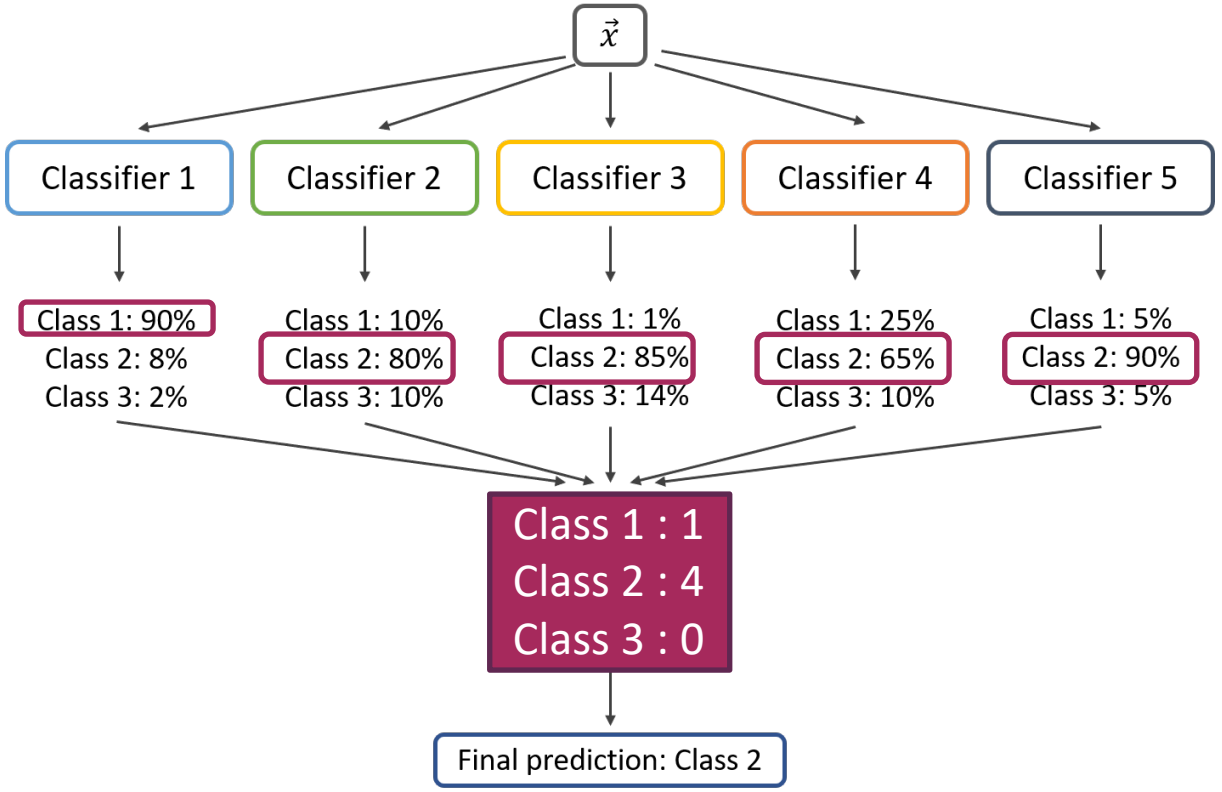
And then aggregate to get an ensemble with lower variance

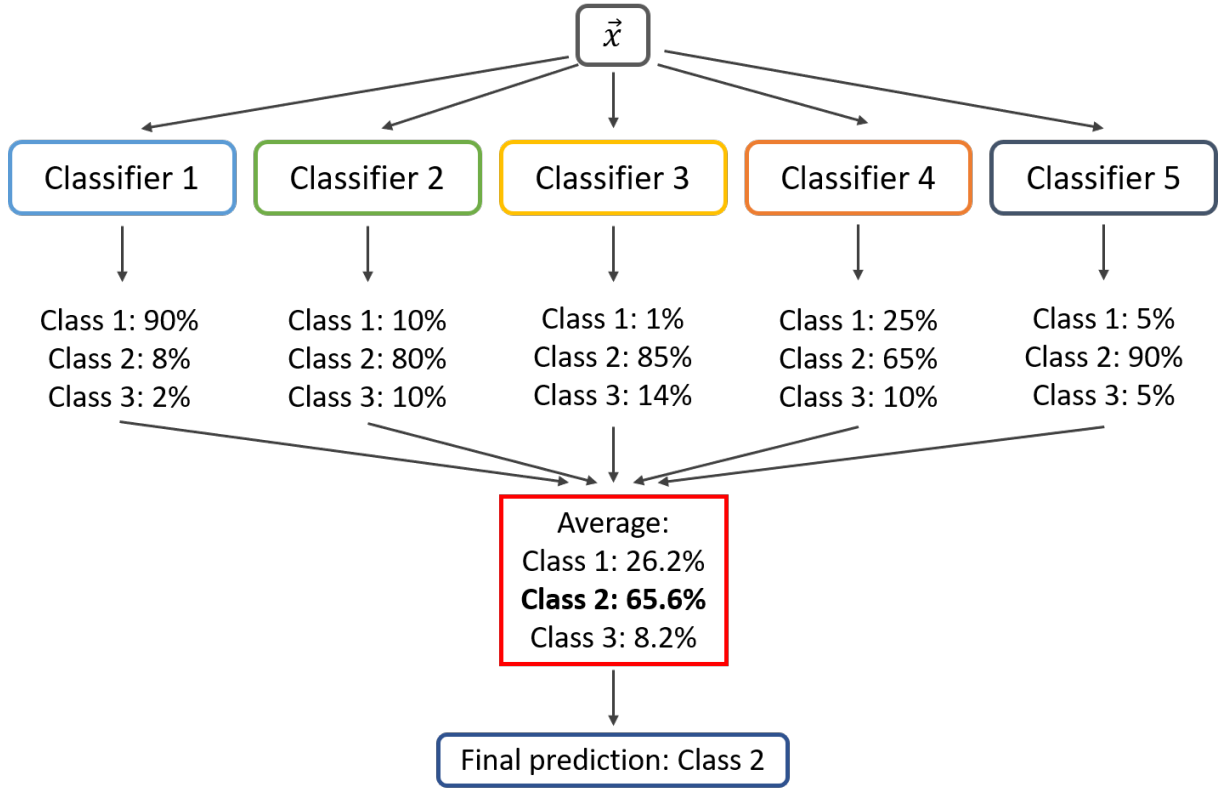
$$regression\ (average):\ \ s_L = \frac{1}{L}\sum_{i=1}^{L} w_i$$

$$classification\ (majority\ vote): \underset{k}{\operatorname{argmax}}[card(i|w_i = k)]$$

http://work.thaslwanter.at/Stats/html/statsDistributions.html

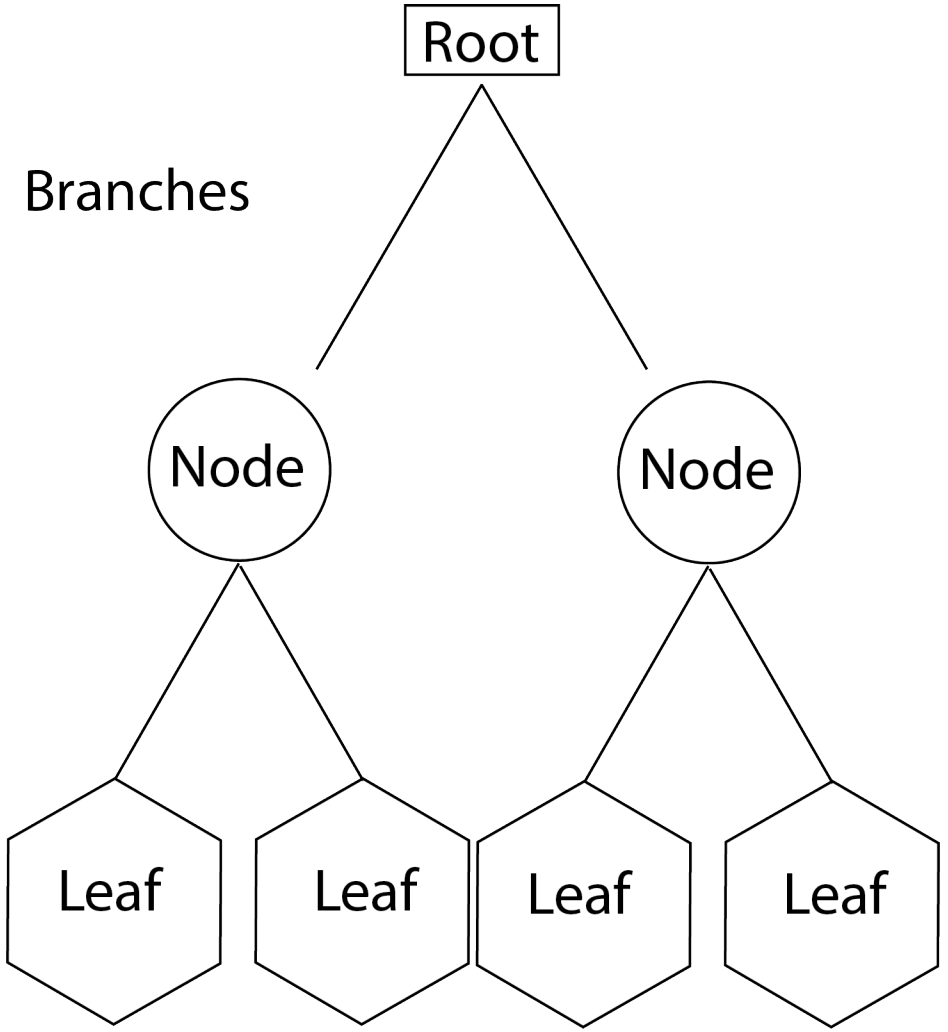
# Classification can occur via hard voting or soft voting




http://work.thaslwanter.at/Stats/html/statsDistributions.html

# There are a number of hyperparameters to select from for decision trees

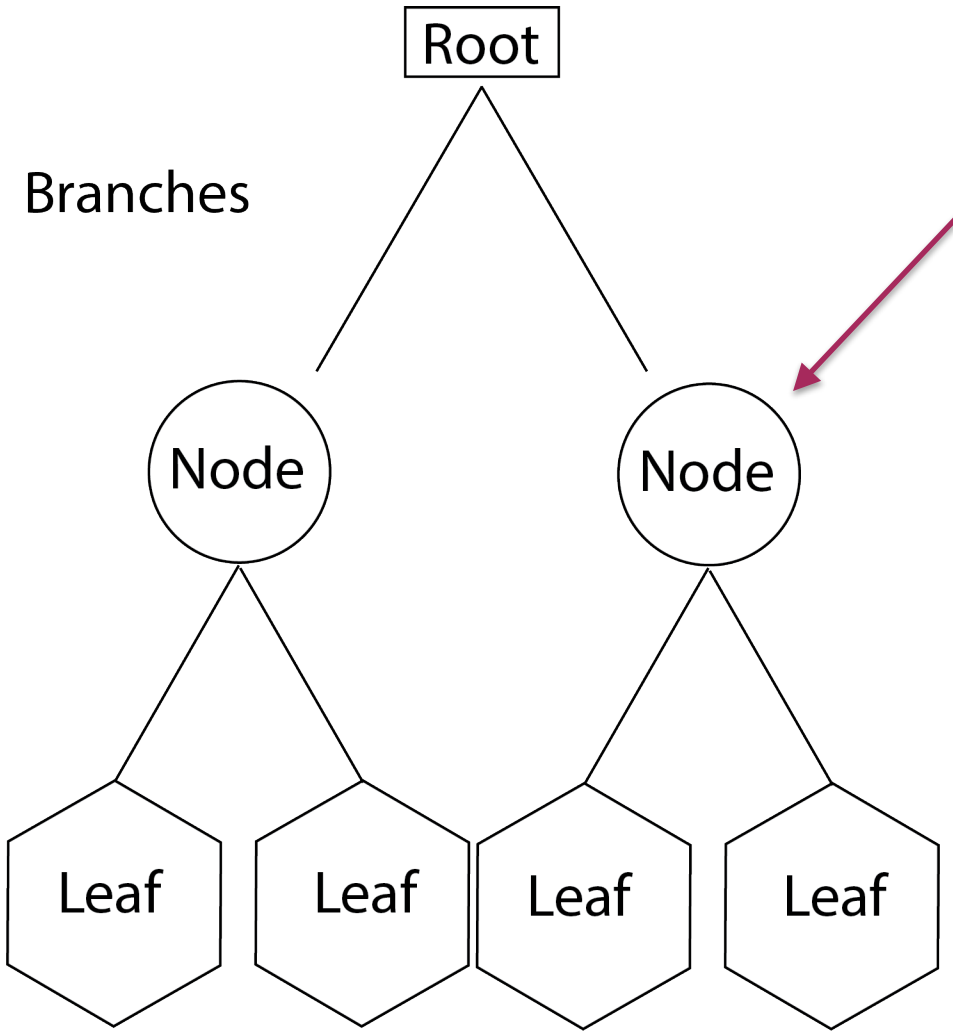



Less variance, more bias
(better for sequential methods)

Increasing tree depth

More variance, less bias
(better for parallel methods)

# There are a number of hyperparameters to select from for decision trees

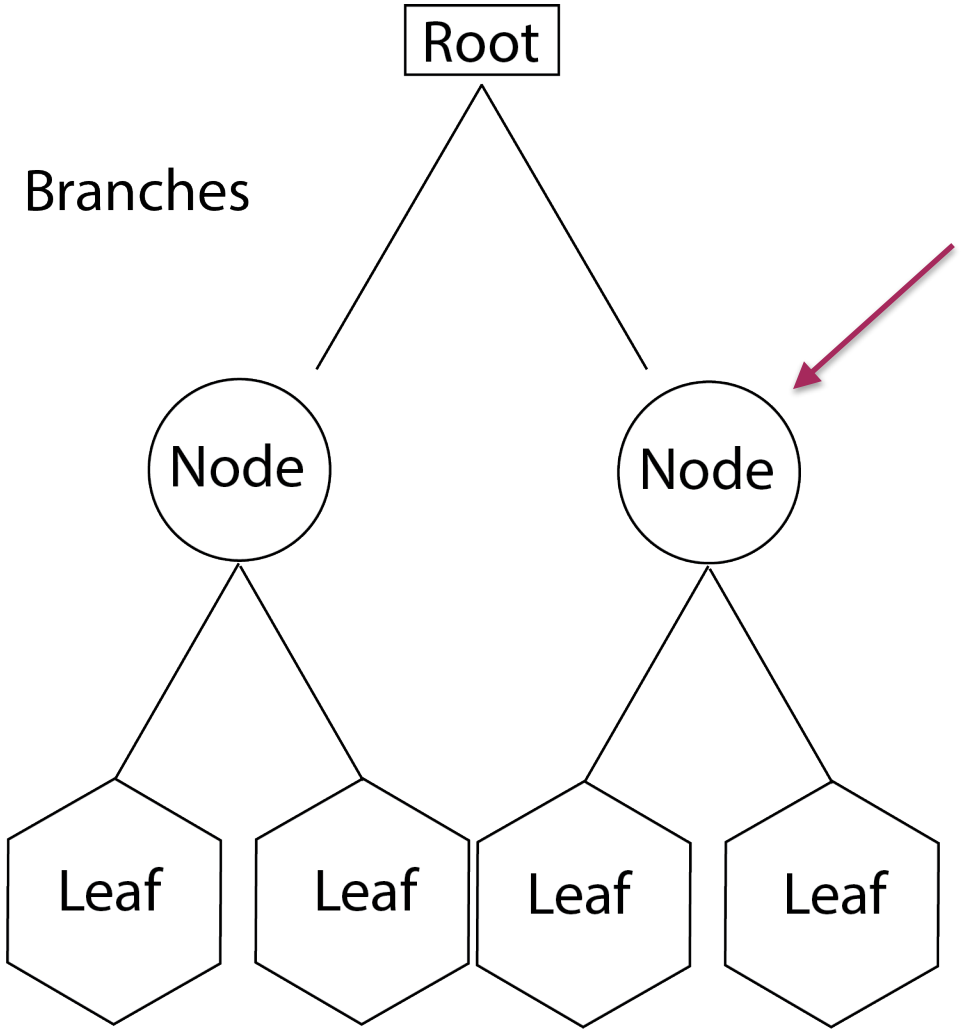



Decision made at each node based on features

Not all features are used
Each tree gets random subset of features
Best if $\sqrt{N}$ where N is total number of features

Makes model more robust to missing data because regression or classification can be done only on trees having required features!

# The value of the feature used for splitting data must be learned




Based on minimized residual

$$RSS = \sum_{left} (y_i - \bar{y}_{left})^2 + \sum_{right} (y_i - \bar{y}_{right})^2$$

Based on maximizing split proportions

$$GINI = n_{left} \sum_{k=1..K} p_{kleft}(1 - p_{kleft}) + n_{right} \sum_{k=1..K} p_{kright}(1 - p_{kright})$$

# There are a number of hyperparameters to select from for decision trees

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(random_state = 42)

from pprint import pprint

# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rf.get_params())
```

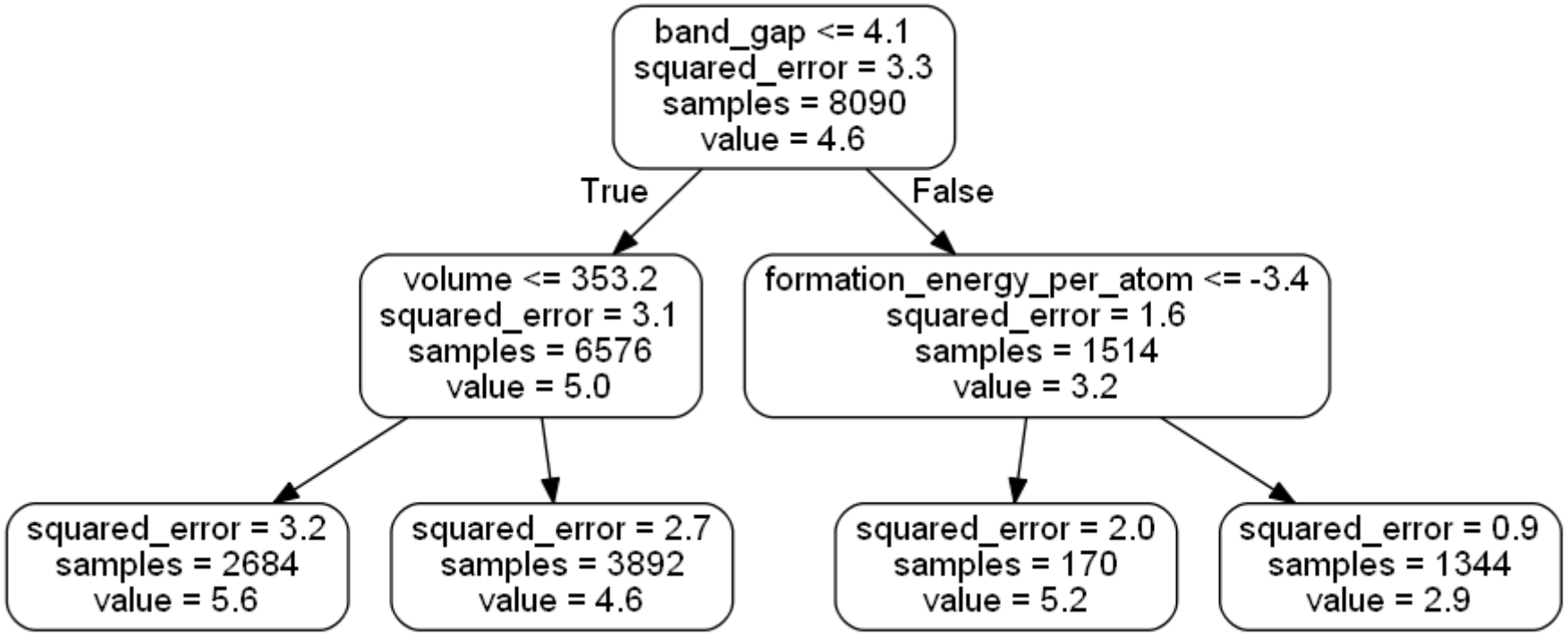
```
Parameters currently in use:

{'bootstrap': True,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 10,
 'n_jobs': 1,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

We can read all about the different model parameters (hyperparameters) through the documentation

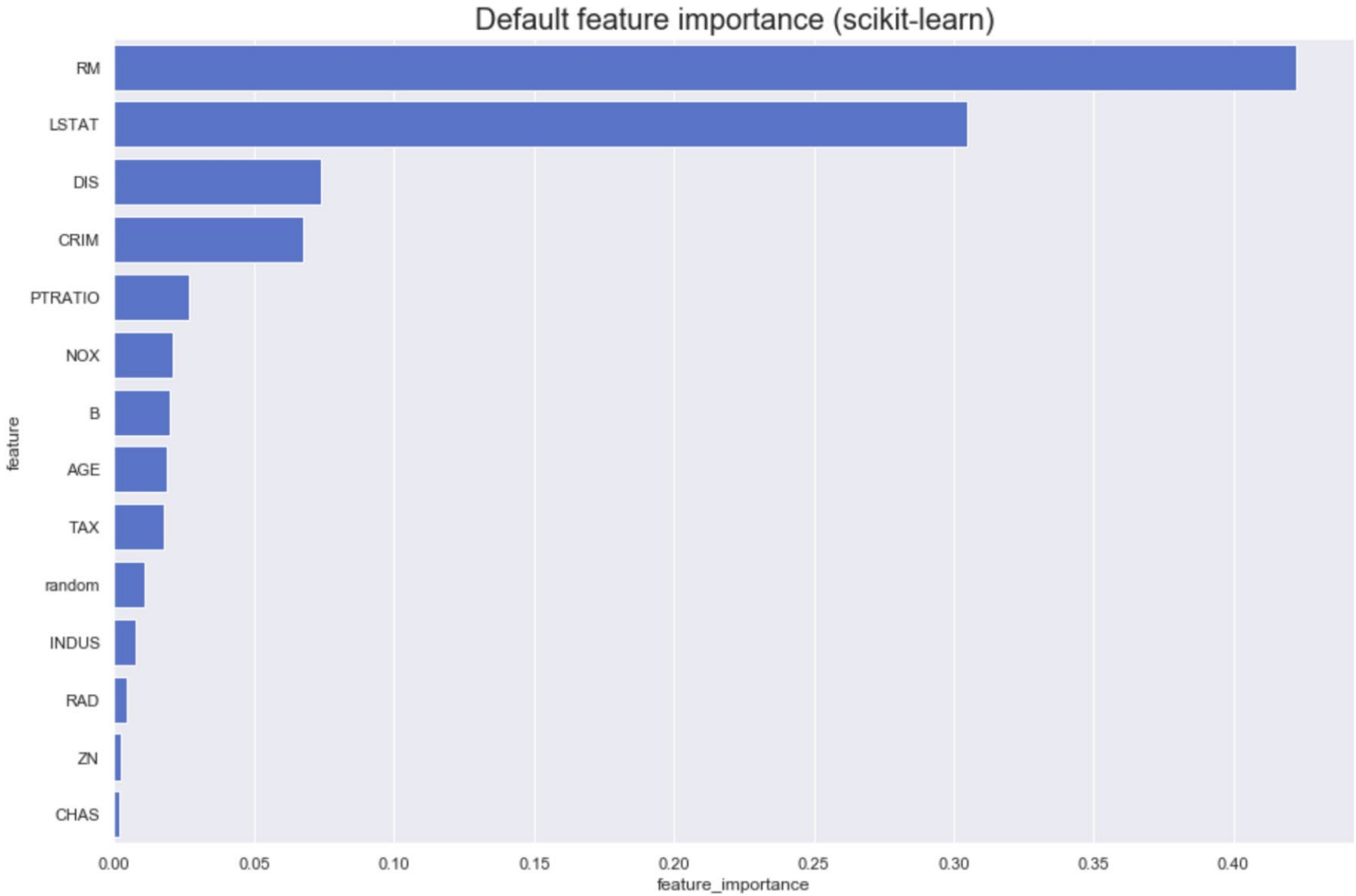
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

# Awesome libraries like graphviz and pydot allow us to actually visualize a specific tree!




https://towardsdatascience.com/random-forest-in-python-24d0893d51c0

# Random forests can easily give us information on feature importance!



# Feature weight or importance can be calculated in several different ways

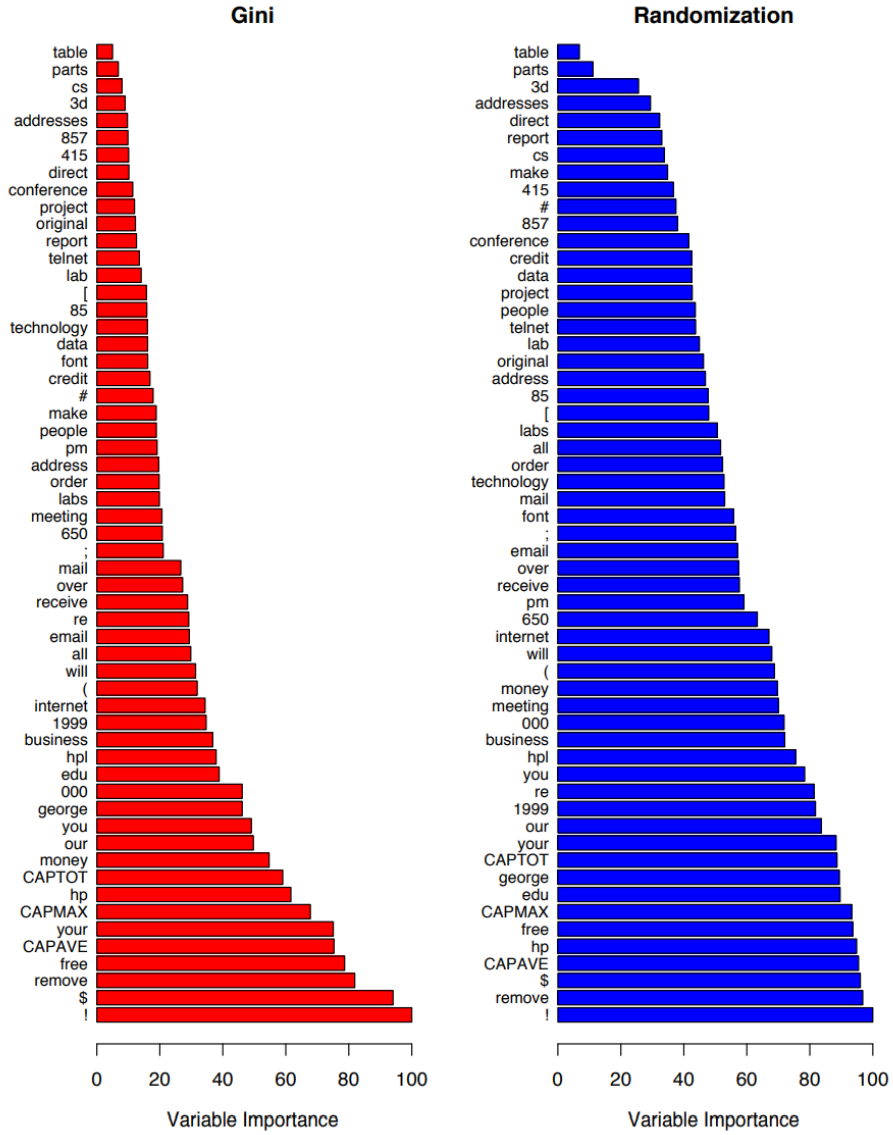
## GINI importance (mean decrease in impurity)

- "Defined as the total decrease in node impurity (weighted by the probability of reaching that node (which is approximated by the proportion of samples reaching that node)) averaged over all trees of the ensemble."

## Permutation importance (mean decrease in accuracy)

- Random forests also use the OOB samples to construct a different variable-importance measure, apparently to measure the prediction strength of each variable. When the b-th tree is grown, the OOB samples are passed down the tree, and the prediction accuracy is recorded. Then the values for the j-th variable are randomly permuted in the OOB samples, and the accuracy is again computed. The decrease in accuracy as a result of this permuting is averaged over all trees.

# Comparison of two feature weighting techniques for spam features




https://www.math.mcgill.ca/yyang/resources/doc/randomforest.pdf

# Scikit-learn calculates feature weight using GINI importance

For each decision tree, we calculate all node importance values, $ni_j$

$$ni_j = w_j C_j + w_{left(j)} C_{left(j)} + w_{right(j)} C_{right(j)}$$

Where $C_j$ is the impurity of node j

$$C_j = \sum_{i=1}^{C} f_i(1 - f_i)$$

Importance of feature on a specific tree becomes

$$f_{i_i} = \frac{\sum_{j,node\ j\ splits\ on\ feature\ i} ni_j}{\sum_{k\ all\ nodes} ni_k}$$

Normalize by dividing by sum of all features

$$norm\ f_{i_i} = \frac{f_{i_i}}{\sum_{j\ all\ features} fi_j}$$

Then average feature weight across all trees

$$RF f_{i_i} = \frac{\sum_{j\ all\ trees} norm\ fi_{ij}}{T}$$

# Boosting for sequential ensemble methods

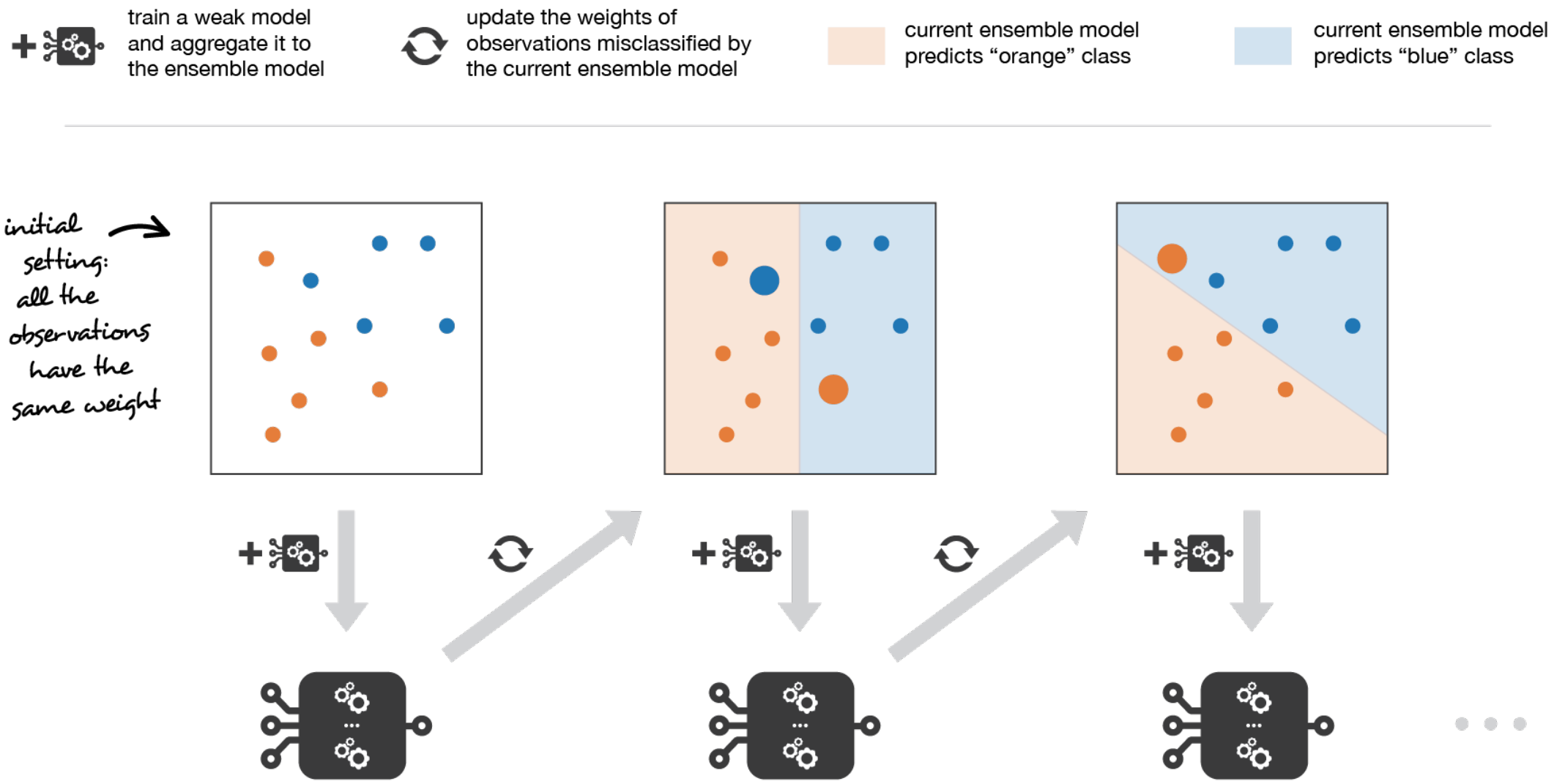
Iterative model creation where model *i* is based on previous models.

- More attention given to observations badly handled in previous iteration
- Focus on hardest observations to fit
- Implement with shallow trees (low variance, high bias)

How will sequential models be fit?

- What info from previous model is taken into account?
- How is current and previous model aggregated?

# Adaptive boosting AKA adaboost is one way to select next model



# Adaptive boosting AKA adaboost is one way to select next model

Ensemble model as weighted sum of weak learners

$$s_L = \sum_{l=1}^{L} c_l w_l$$

Where $c_l$ is coefficient, $w_l$ is a weak learner. Super hard to solve this equation….

So break it up and add new learners iteratively

$$s_l = s_{l-1} + c_l w_l$$

Select $c_l$ and $w_l$ so that $s_l$ fits training data best. (max improvement over $s_{l-1}$)

$$c_l, w_l = argmin\ E\ (s_{l-1} + cw) = argmin \sum_{n=1}^{N} e\big(y_n, s_{l-1}(x_n) + cw(x_n)\big)$$

E is model error, e is the loss function so optimization takes place on most recent model, not all models.

# Adaboost in plain English

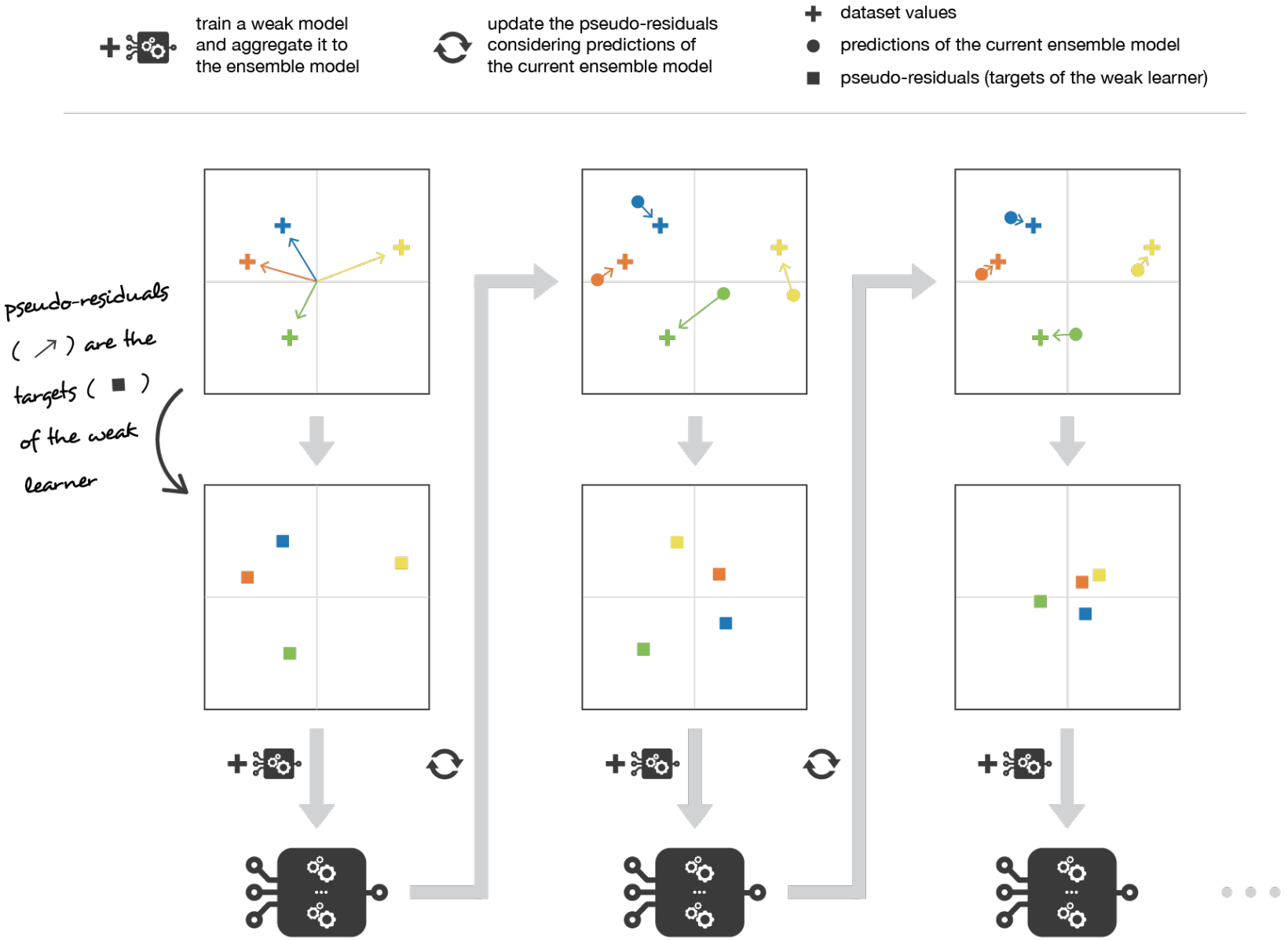
First, it updates the observations weights in the dataset and train a new weak learner with a special focus given to the observations misclassified by the current ensemble model. Second, it adds the weak learner to the weighted sum according to an update coefficient that express the performances of this weak model: the better a weak learner performs, the more it contributes to the strong learner.

Consider N observations

- Fit the best possible weak model with the current observations weights (1/N)
- Compute the value of the update coefficient
- Update the strong learner (add new weak learner times its coefficient)
- Compute new observations  and repeat

Variations like LogitBoost, L2Boost available with slightly different loss functions

# Gradient boosting is another way to select next model



# Gradient boosting is another way to select next model

Ensemble model as weighted sum of weak learners

$$s_L = \sum_{l=1}^{L} c_l w_l$$

At each iteration we fit a weak learner to the opposite of the gradient of the fitting error.

$$s_l = s_{l-1} - c_l * \nabla_{s_{l-1}} E(s_{l-1})$$

Where *E* is the fitting error of the given model, $c_l$ is coefficient corresponding to step size, the final term is the gradient of the fitting error with respect to the previous model $s_{l-1}$.

In essence, each observation has a pseudo-residual describing how well the model fits it. So we fit a weak learner to the pseudo-residuals themselves!

# Gradient boosting in plain English

Each observation has pseudo-residual equal to observation and then…

- Fit weak learner to residuals
- Compute the value of the optimal step size that defines by how much we update the ensemble model in the direction of the new weak learner
- Update the ensemble model by adding the new weak learner multiplied by the step size (make a step of gradient descent)
- Compute new pseudo-residuals that indicate, for each observation, in which direction we would like to update next the ensemble model predictions

Thus, gradient boosting can be considered as a generalization of adaboost to arbitrary differentiable loss functions.

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

# Stacking allows us to combine heterogeneous weak learners

"A model made up of models"

Basic steps

- We split data into two folds
- Choose weak learners, fit to one fold of data
- Use weak learners to make predictions for second fold observations
- Fit the meta-model on the second fold, using predictions made by weak learners as inputs

Data used to train weak learners is not used again for training meta-model so we lose ½ of our data unless we do k-fold splitting and training

Multi level stacking is possible with more than one meta-model.

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

# Let’s consider an example of stacking / transfer learning for band gap prediction

Integrating Materials and Manufacturing Innovation
https://doi.org/10.1007/s40192-020-00178-0

TECHNICAL ARTICLE

## Extracting Knowledge from DFT: Experimental Band Gap Predictions Through Ensemble Learning

Steven K. Kauwe[1] · Taylor Welker[2] · Taylor D. Sparks[1]

Received: 4 May 2020 / Accepted: 18 June 2020


**Abstract**
Many of the machine learning-based approaches for materials property predictions use low-cost computational data. The motivation for machine learning models is based on the orders of magnitude speedup compared to DFT calculations or experimental characterization. High-quality experimental materials data would be ideal for training these models; unfortunately, experimental data are typically costly to obtain. As a result, experimental databases are often smaller and less cohesive. Using band gap, we demonstrate how an ensemble learning approach allows us to efficiently model experimental data by combining models trained on otherwise disparate computational and experimental data. This approach demonstrates how disparate data sources can be incorporated into the modeling of sparsely represented experimental data. In the case of band gap prediction, we reduce the root mean squared error by over 9%.

**Keywords** Machine learning · Band gap · Transfer learning · Ensemble learning

### Introduction

The discovery of new materials is critical for achieving the grand engineering goals of the future [1–3]. Despite the importance of materials discovery, revolutionary discoveries are a rarity and often serendipitous. The resources needed for the synthesis and characterization of new materials frequently limit the materials discovery process [2]. Because of the often prohibitive costs of synthesis, researchers in the field of materials science and engineering routinely use first principles calculations such as density functional theory (DFT) to aid in the materials discovery process [1, 2, 4, 5]. These computational techniques are used to estimate materials properties and effectively screen candidates [6–9], providing insight into promising synthesis actions. Despite the success of first principles approaches, these physics-based calculations can take days or weeks and consequently are not well suited for exhaustively screening chemical composition space [10, 11]. Moreover, these approaches require crystal structure to be known prior to calculations which precludes unknown structures from most screening methods.

The Materials Genome Initiative in 2011 is evidence that the current rate of discovery will not address the technological and scientific needs we face. The opportunity for machine learning (ML) to help accelerate materials discovery has already been made evident by companies and research programs including Citrine Informatics [12], the Materials Project (MP) [1], the AFLOW distributed materials property repository (AFLOW) [13], as well as many individual research groups. These organizations have made considerable progress toward providing simple access to computational data, machine learning techniques, and experimental materials information.

A simple approach for using such data relies on creating a composition-based feature vector (CBFV) [14–16]. The CBFV is a permutation-invariant representation of the composition; this is often achieved using the sum, mean, variance, and range of element properties in the composition. Models using this type of approach have successfully predicted materials properties on both experimental and first principles data [15, 17–21]. That said, researchers are not

**Electronic supplementary material** The online version of this article (https://doi.org/10.1007/s40192-020-00178-0) contains supplementary material, which is available to authorized users.

✉ Taylor D. Sparks
sparks@eng.utah.edu

[1] Materials Science and Engineering Department, University of Utah, Salt Lake City, UT 84112, USA

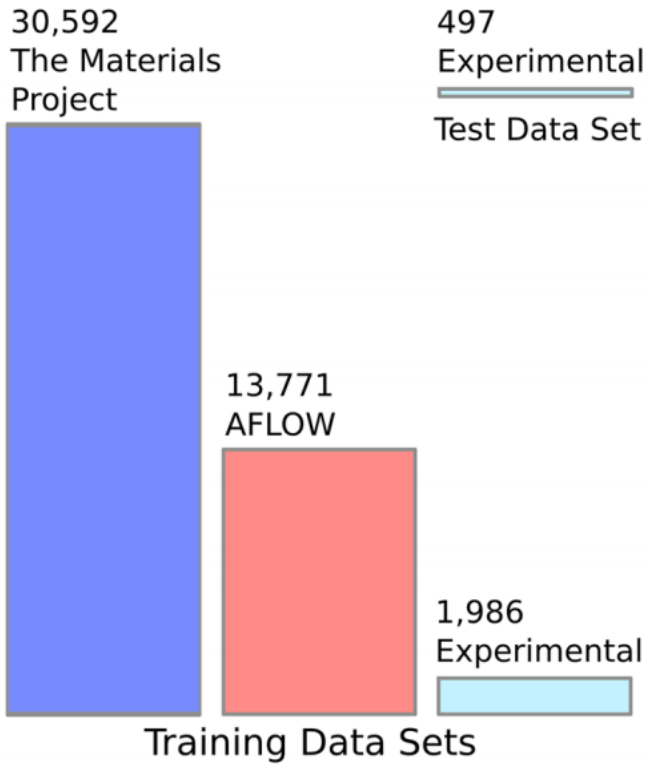
[2] School of Computing, University of Utah, Salt Lake City, UT 84112, USA

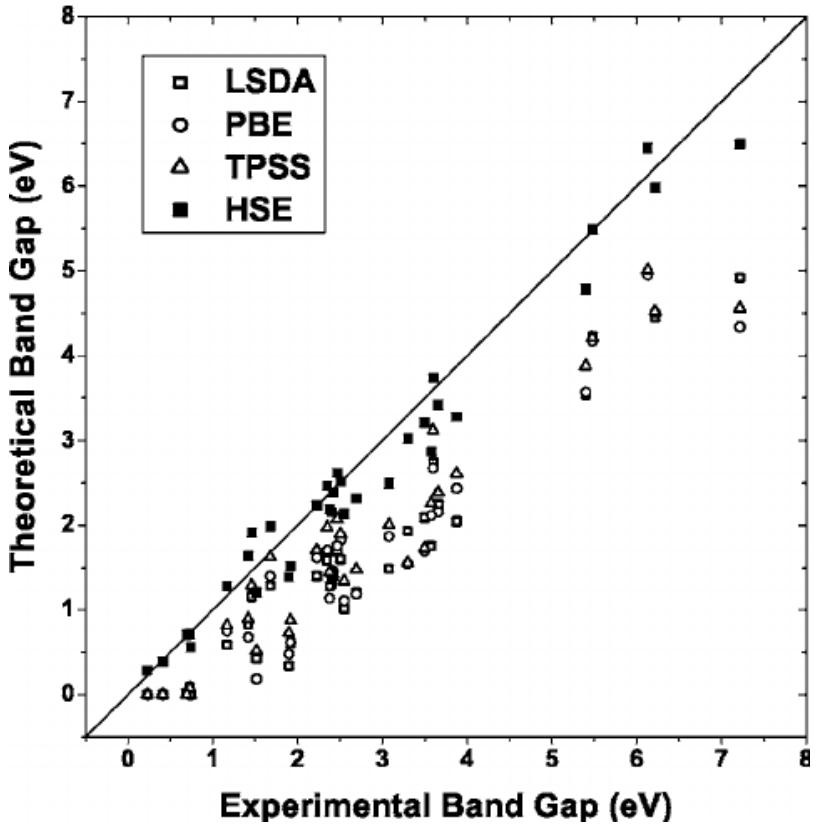
Published online: 24 July 2020



Kauwe, Welker, Sparks IMMI (2020)

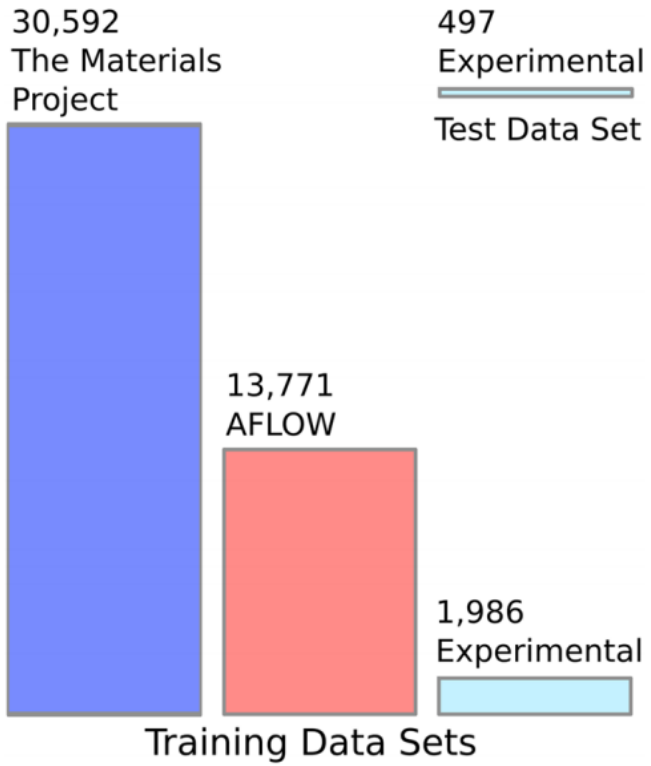
# Transfer learning is when a trained model can be applied to another related task

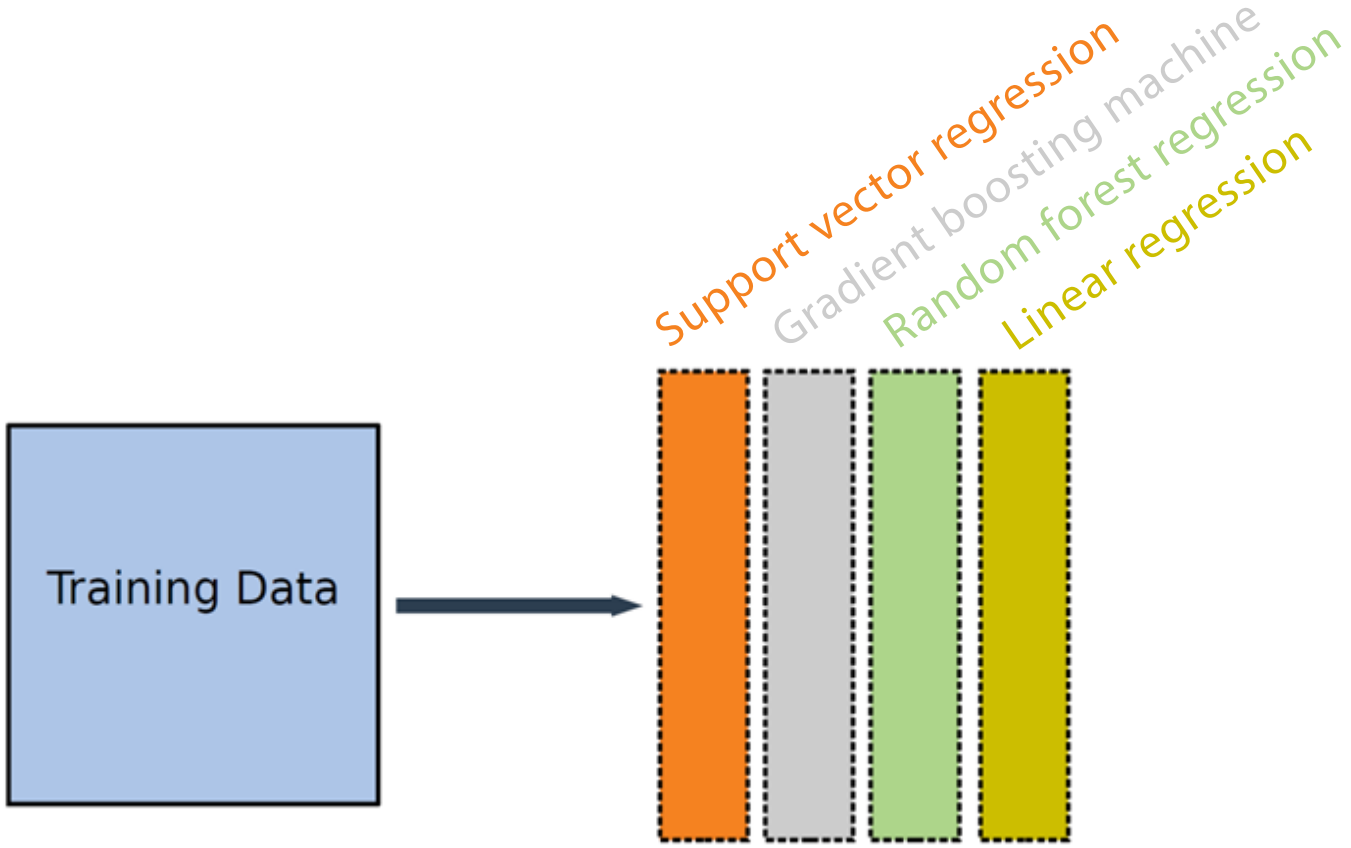



Kauwe, Welker, Sparks IMMI (2020)

# Stacking allows us to employ various types of weak learners




Kauwe, Welker, Sparks IMMI (2020)

# What do we do with diverse, heterogenous data sets?

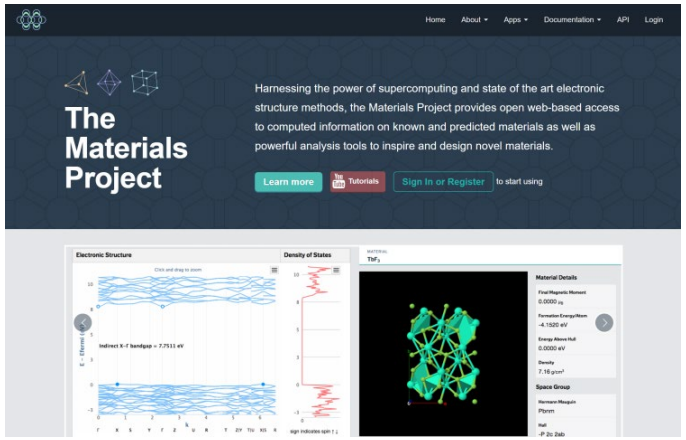
DFT data

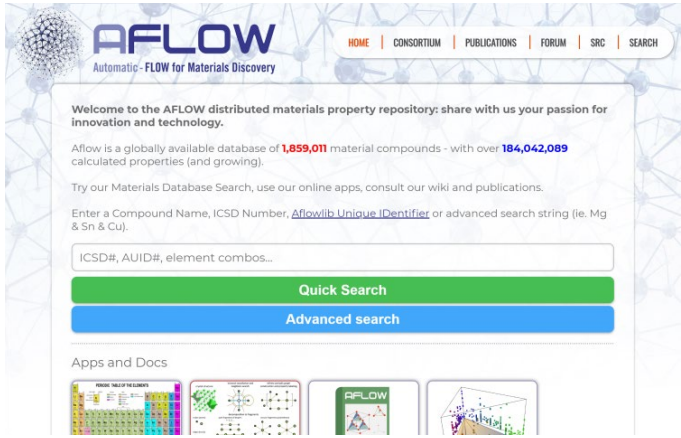
30,592
Materials Project

Experimental data

2,483

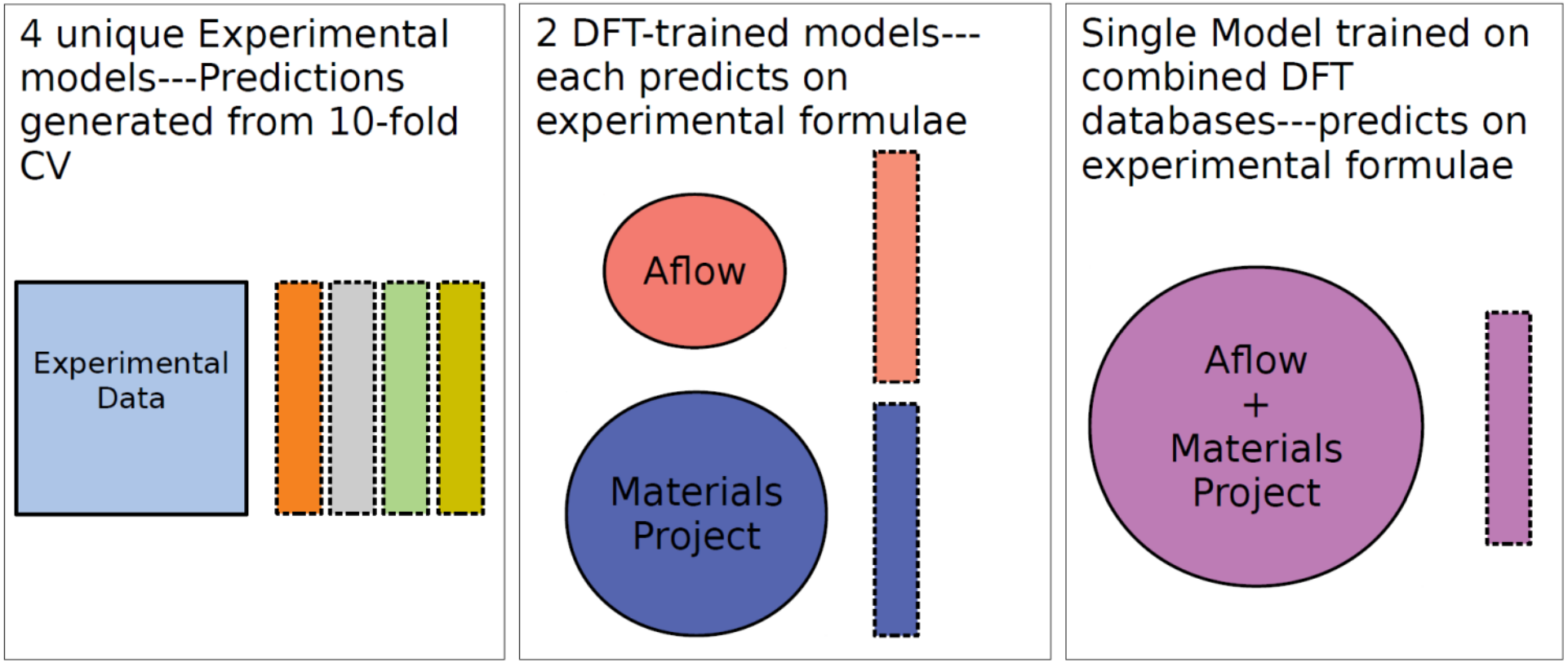
13,771
AFLOW



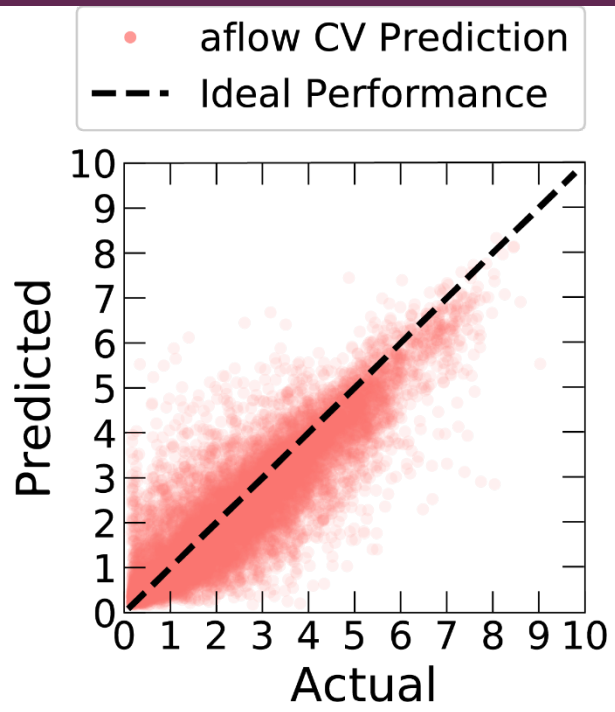

Kauwe, Welker, Sparks IMMI (2020)

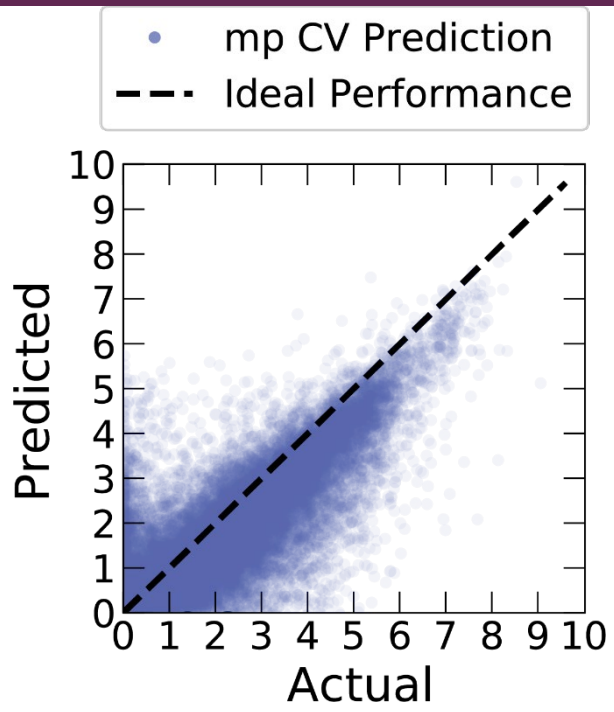
# We can learn on different data with different models in a way that makes the most sense

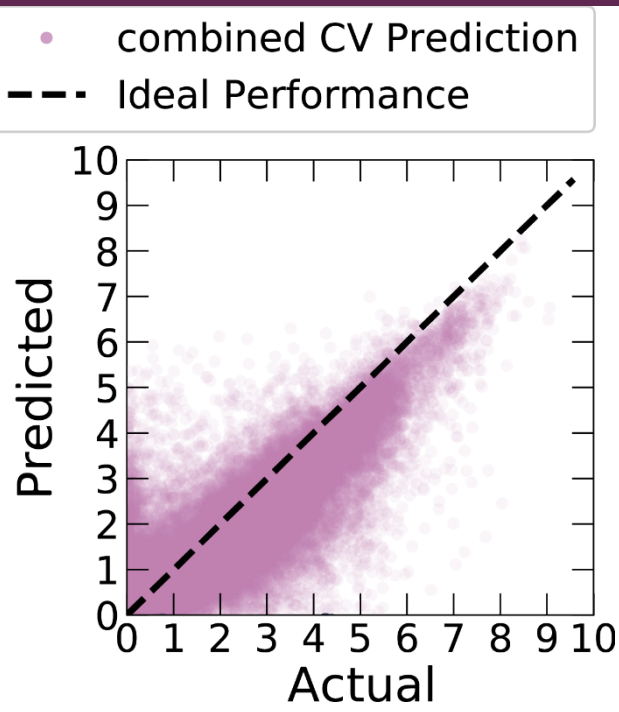


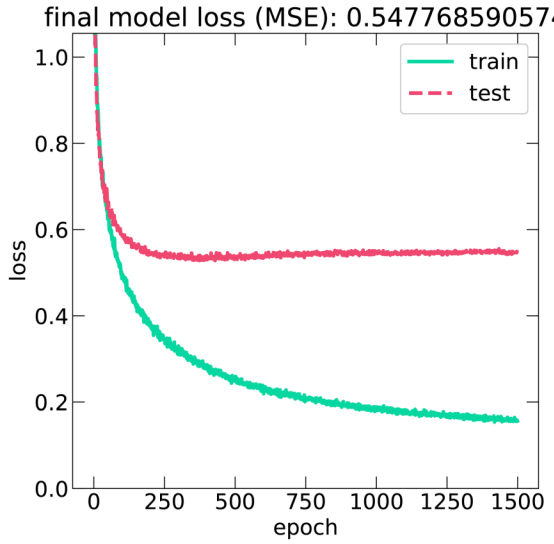

Kauwe, Welker, Sparks IMMI (2020)

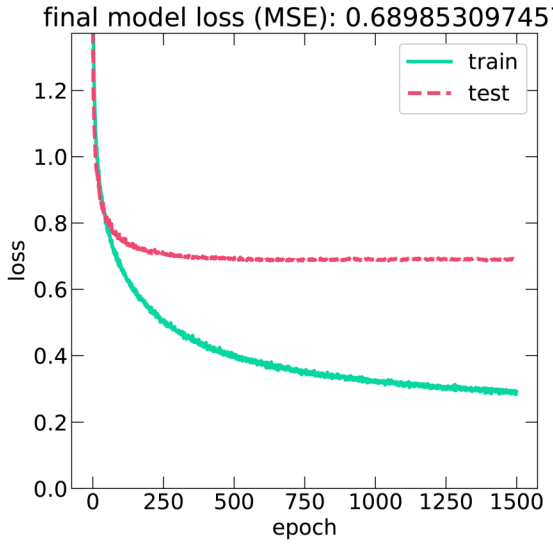
# Neural nets do well with large amounts of DFT data

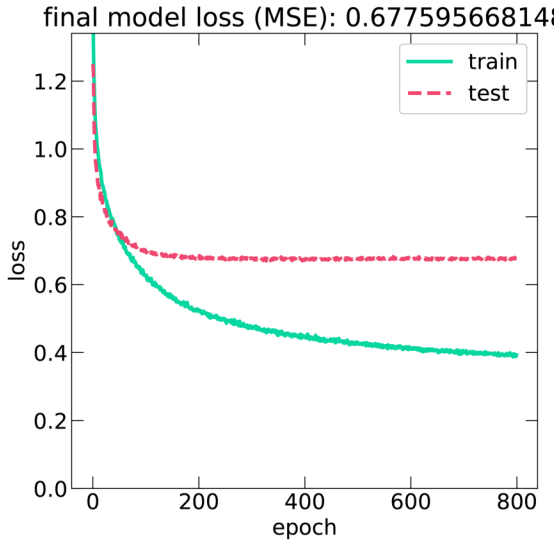


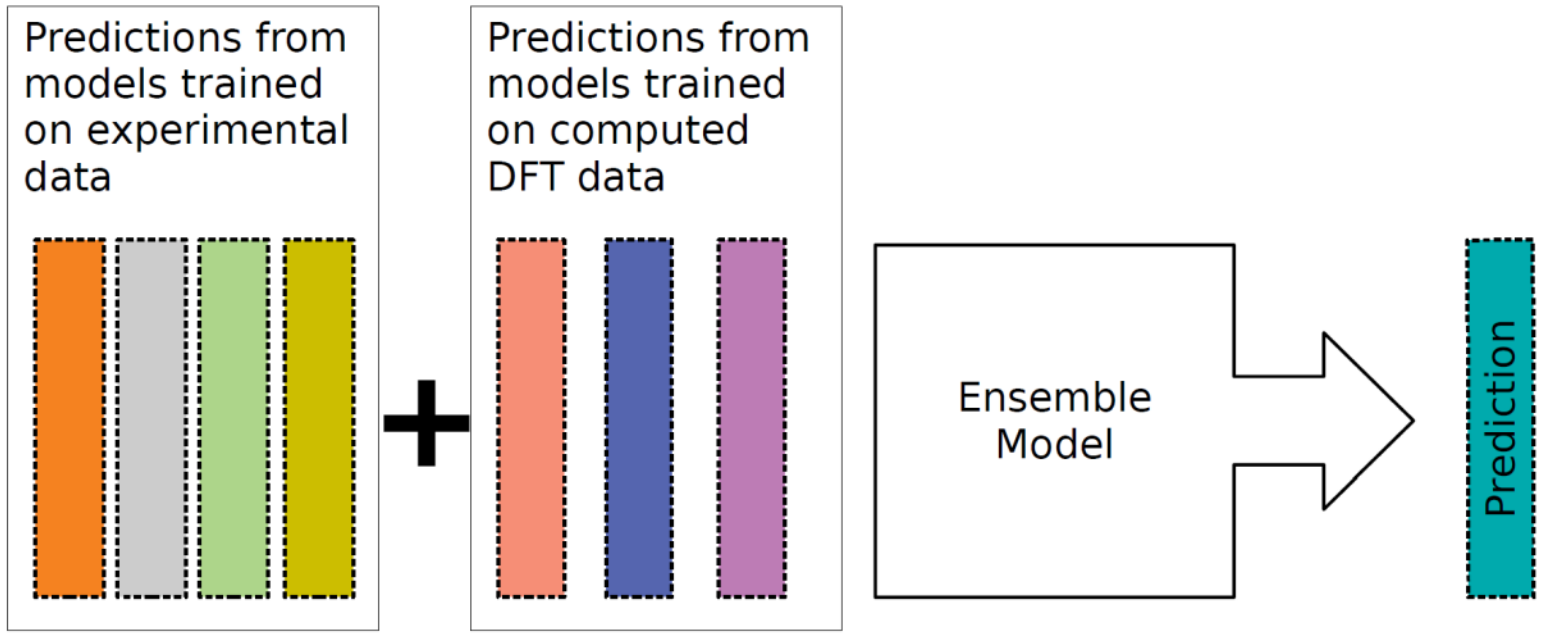



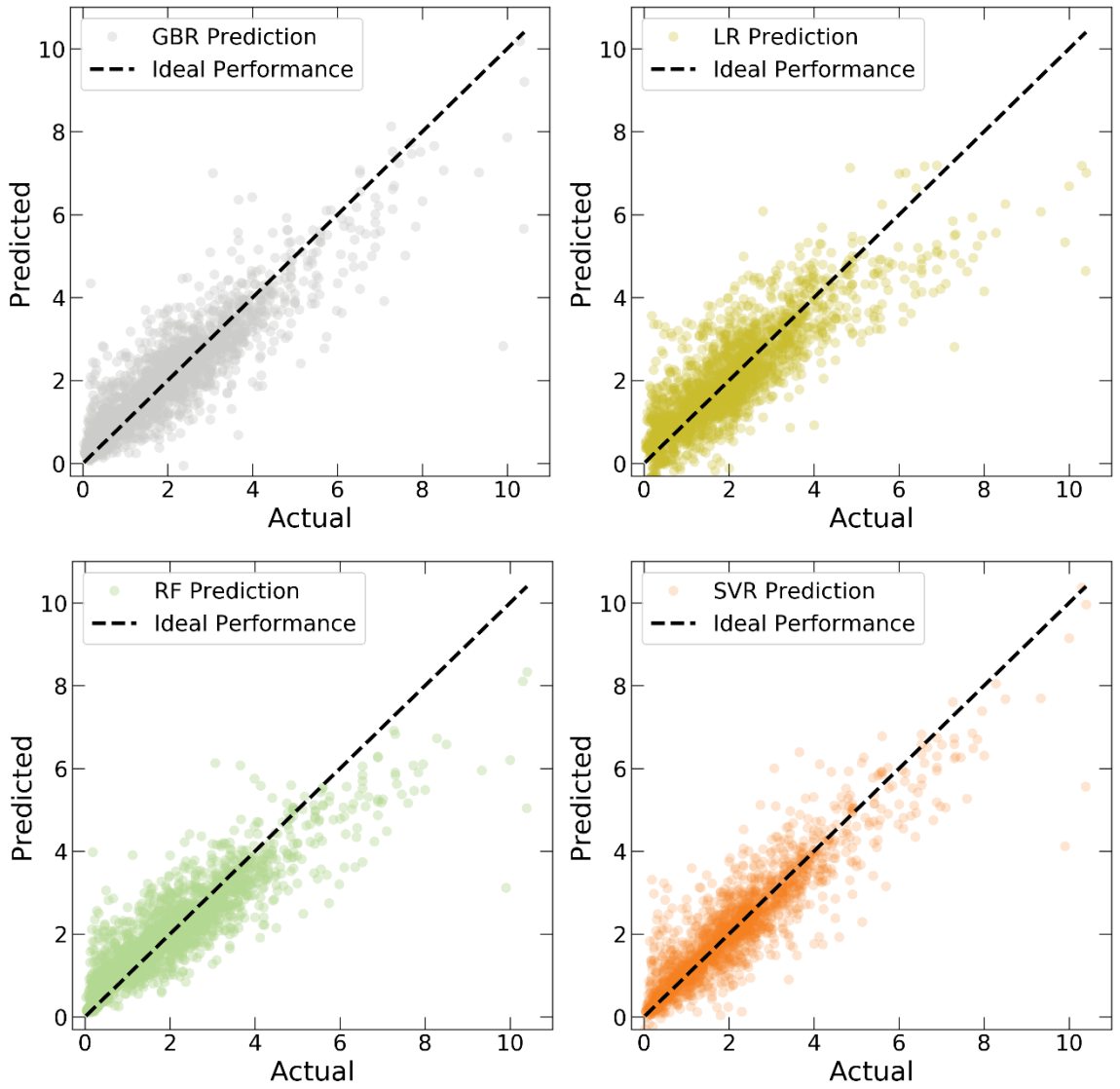



# The outputs from individual models become inputs for ensemble, meta-model




Kauwe, Welker, Sparks IMMI (2020)

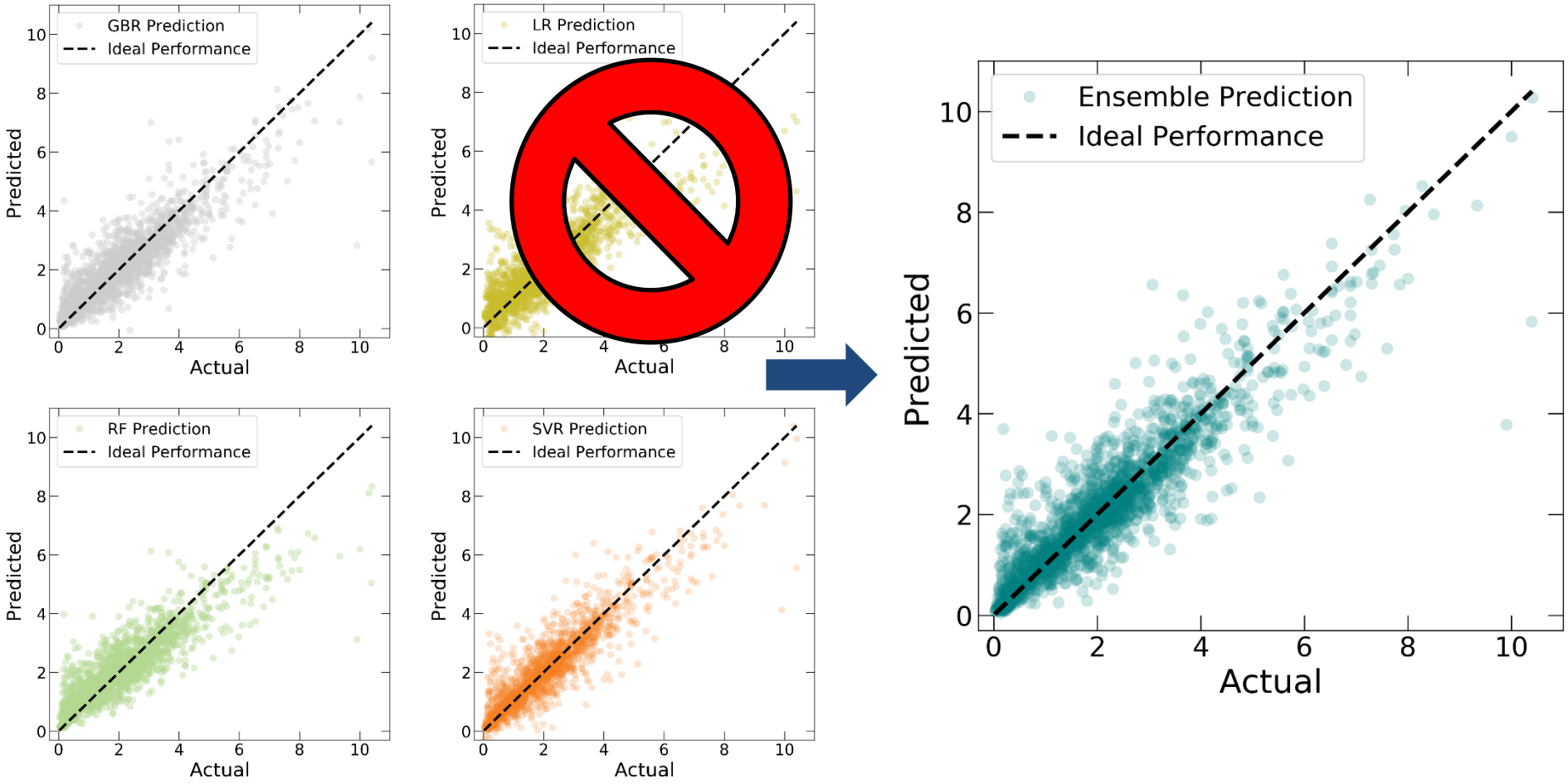
# We can examine the error from each weak learner




| MODEL | R^2 | RMSE |
|---|---|---|
| SVR | 0.808 | 0.627 |
| BGM | 0.779 | 0.672 |
| RFR | 0.773 | 0.687 |
| LR | 0.676 | 0.822 |

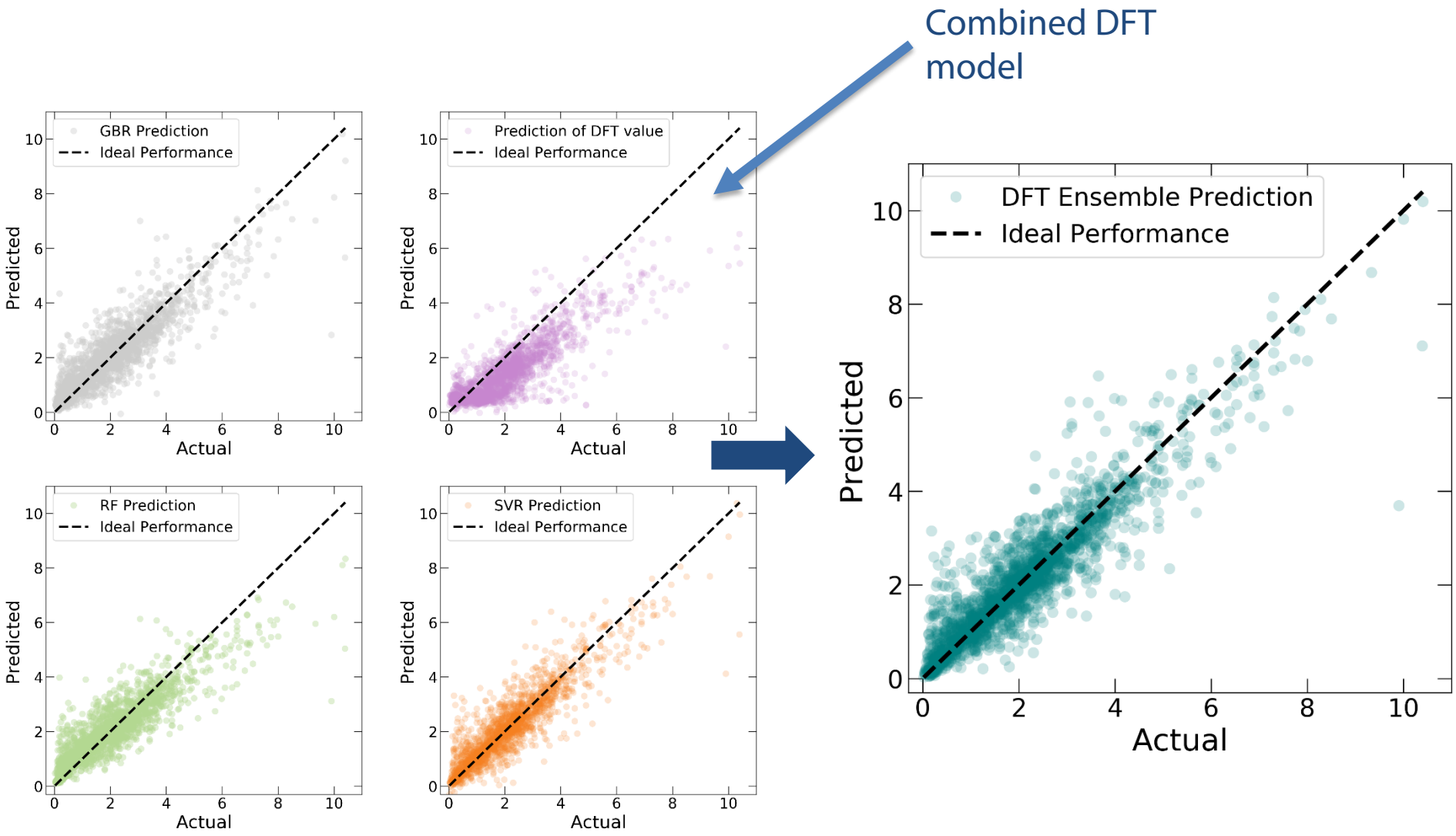
Kauwe, Welker, Sparks IMMI (2020)

# Meta-model might eliminate contributions outright from a weak learner



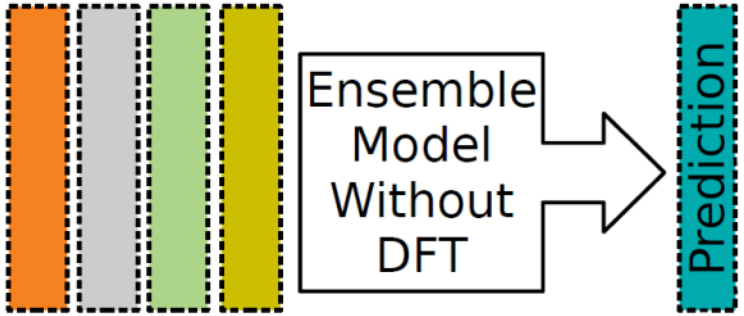

Kauwe, Welker, Sparks IMMI (2020)

# Instead of the linear model, we can incorporate diverse data from the neural network model

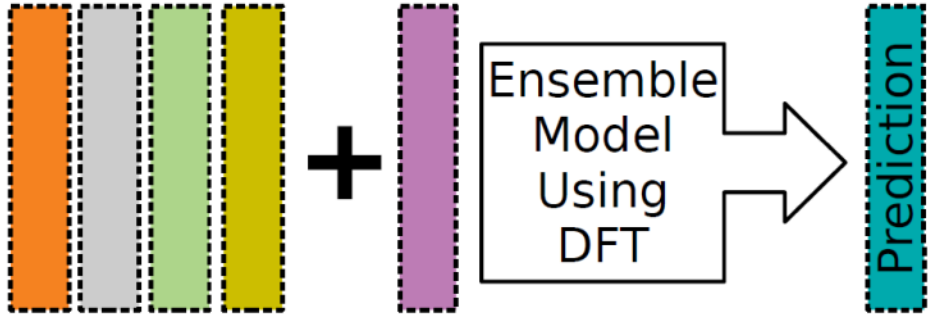



Kauwe, Welker, Sparks IMMI (2020)

# Stacking ensemble does significantly better overall



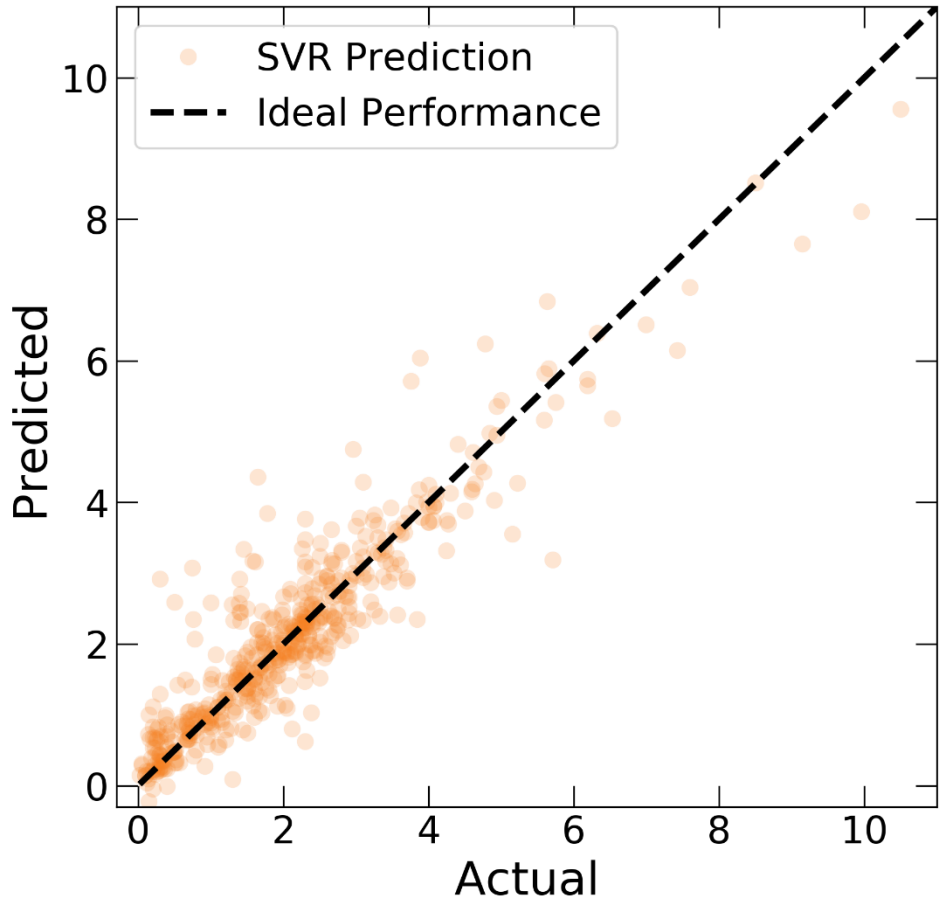

1.65 % improvement to R2
5.09% improvement to RMSE
0.35 % improvement to MAPE

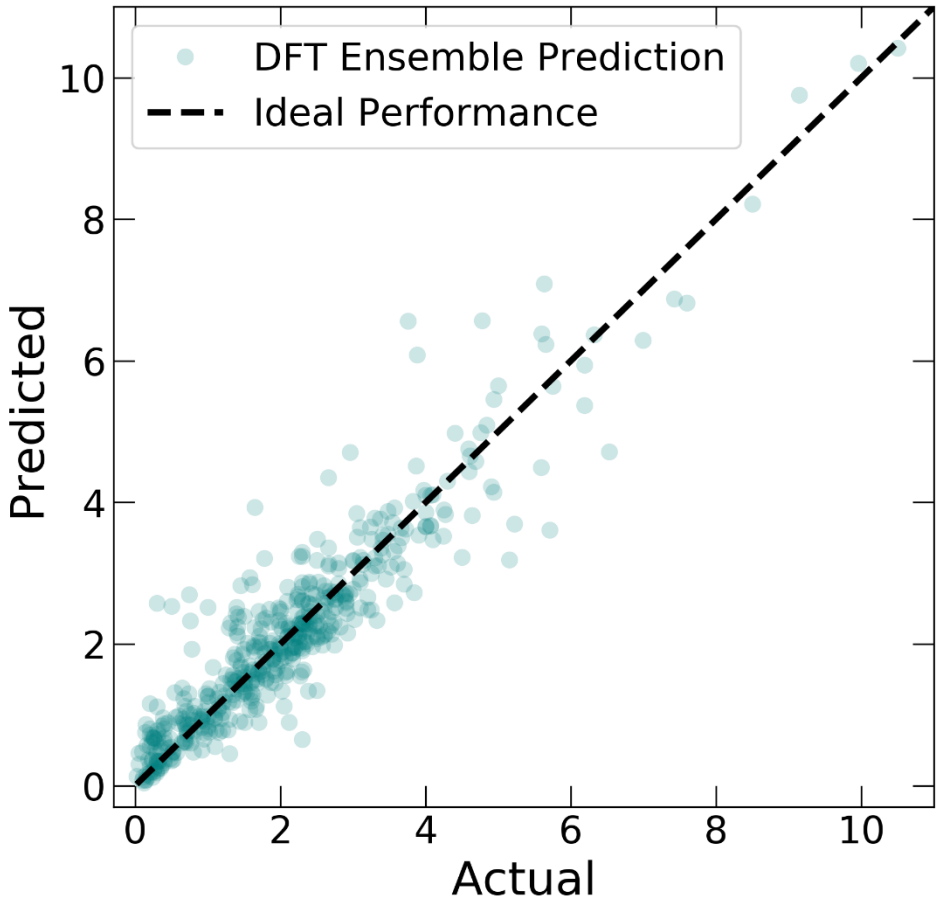



3.01 % improvement to R2
9.51% improvement to RMSE
2.46 % improvement to MAPE

Kauwe, Welker, Sparks IMMI (2020)

# Side by side comparison of best individual vs ensemble predictions




Kauwe, Welker, Sparks IMMI (2020)

# Support Vector Machines (SVMs)