

Ensemble learning

DESU IA4Health

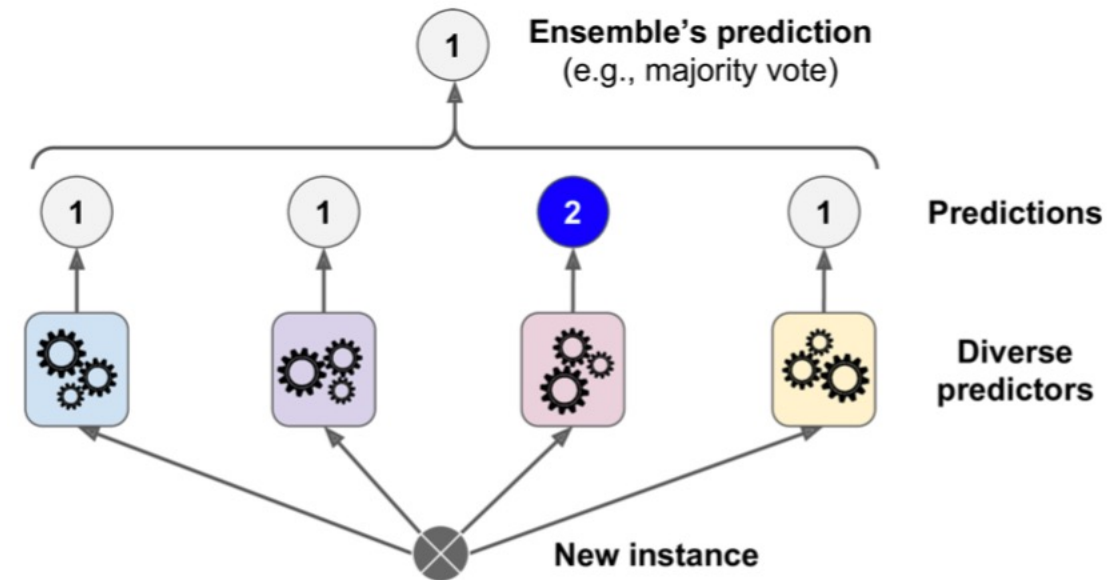
Info-GRAM

Objectives

- Learning the concept of ensemble in machine learning
 - Voting classifier
 - Sampling the dataset :
 - Bagging
 - Pasting
 - Random Forest
 - Boosting
 - Adaboost
 - Gradient boost descending

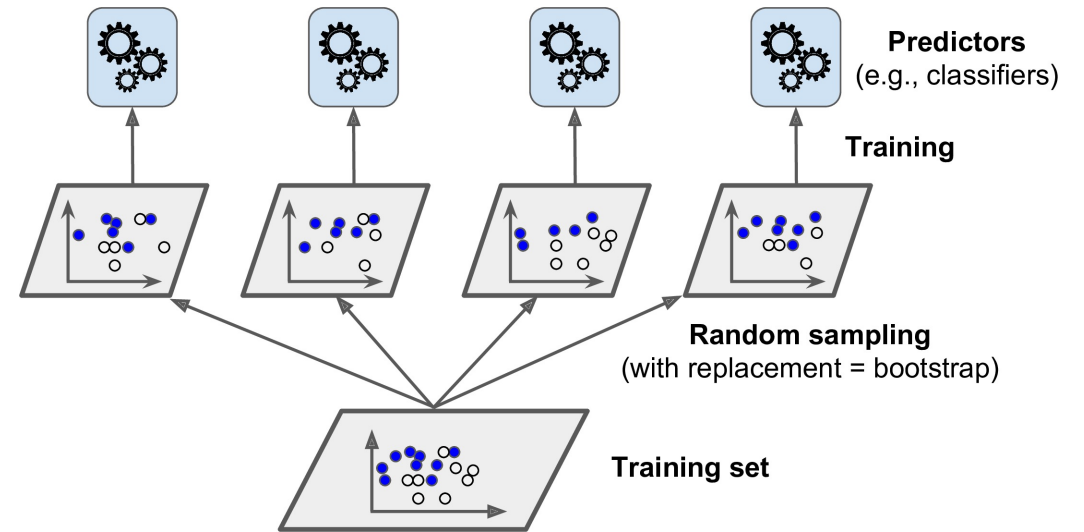
Voting classifier

- voting classifier often achieves a higher accuracy than the best classifier in the ensemble. In fact, even if each classifier is a *weak learner* (meaning it does only slightly better than random guessing), the ensemble can still be a *strong learner* (achieving high accuracy)
- work best when the predictors are as independent from one another as possible. One way to get diverse classifiers is to train them using very different algorithms.



Bagging and pasting

- To use the same training algorithm for every predictor, but to train them on different random subsets of the training set.
 - Bagging : (short for *bootstrap aggregating*) sampling is performed *with* replacement,
 - Pasting : sampling is performed *without* replacement



Why bagging

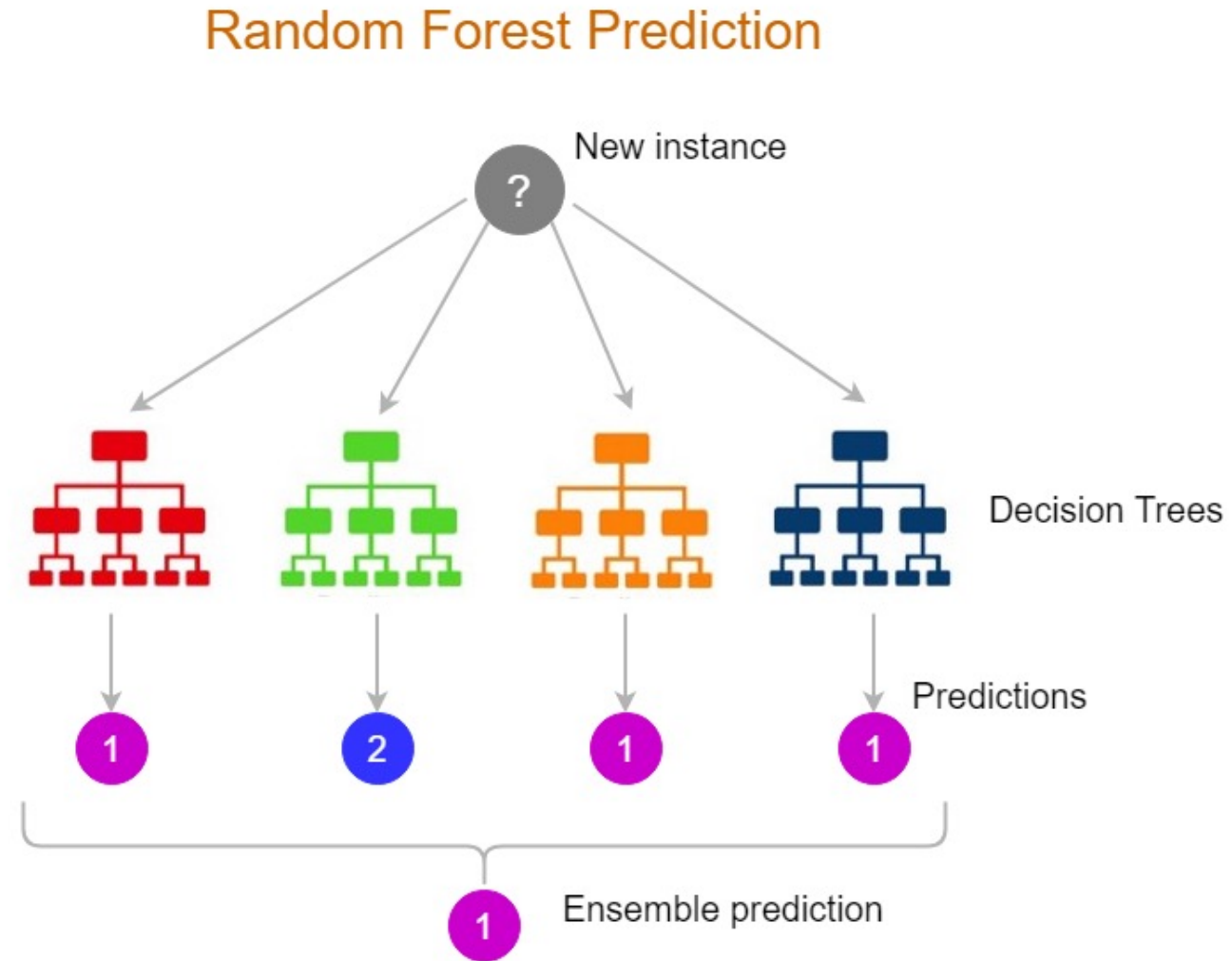
- Designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression.
- It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method.

Hyperparameters bagging

- **n_estimators** : The number of base estimators in the ensemble. (number of trees)
- **max_samples** : The number of samples to draw from X to train each base estimator (with replacement by default, see bootstrap for more details).
- **max_features** : number of features to draw from X to train each base estimator.
- **Bootstrap** : Whether samples are drawn with replacement. If False, sampling without replacement is performed.

Radom forest

- I an ensemble of Decision Trees, generally trained via the bagging method (or sometimes pasting), typically with `max_samples` set to the size of the training set.
- Instead of building a `BaggingClassifier` and passing it a `DecisionTreeClassifier`, you can instead use the `RandomForestClassifier` class, which is more convenient and optimized for Decision Trees (similarly, there is a `RandomForestRegressor` class for regression tasks).



Radom forest Hypermarameters

- RandomForestClassifier has all the hyperparameters of a DecisionTreeClassifier (to control how trees are grown), plus all the hyperparameters of a BaggingClassifier to control the ensemble itself.

Regularization Hyperparameters Tree

- ***max_depth***
- ***max_leaf_nodes*** (maximum number of leaf nodes),
- ***max_features*** (maximum number of features that are evaluated for splitting at each node).
- ***min_samples_split*** (the minimum number of samples a node must have before it can be split),
- ***min_samples_leaf*** (the minimum number of samples a leaf node must have),
- ***min_weight_fraction_leaf*** (same as `min_samples_leaf` but expressed as a fraction of the total number of weighted instances)

To regularize the model :

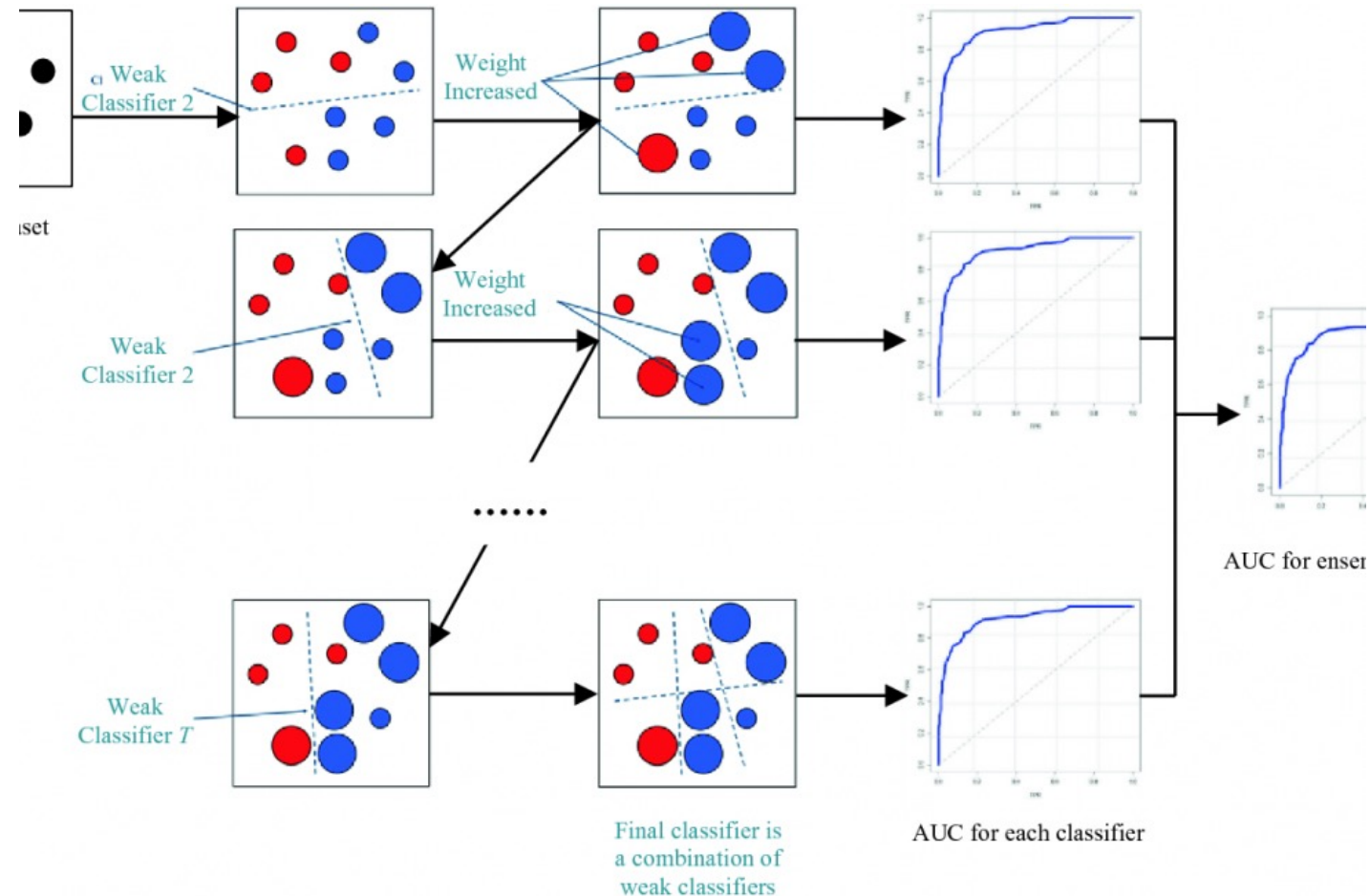
- Increase `min_*` hyperparameters
- Reduce `max_*` hyperparameters

Random Forest, feature importance

- RF makes easy to measure the relative importance of each feature.
- Scikit-Learn measures a feature's importance by looking at how much the tree nodes that use that feature reduce impurity on average (across all trees in the forest). It is a weighted average, where each node's weight is equal to the number of training samples that are associated with it.
- Scikit-Learn computes this score automatically for each feature after training. You can access the result using the `feature_importances_` variable.

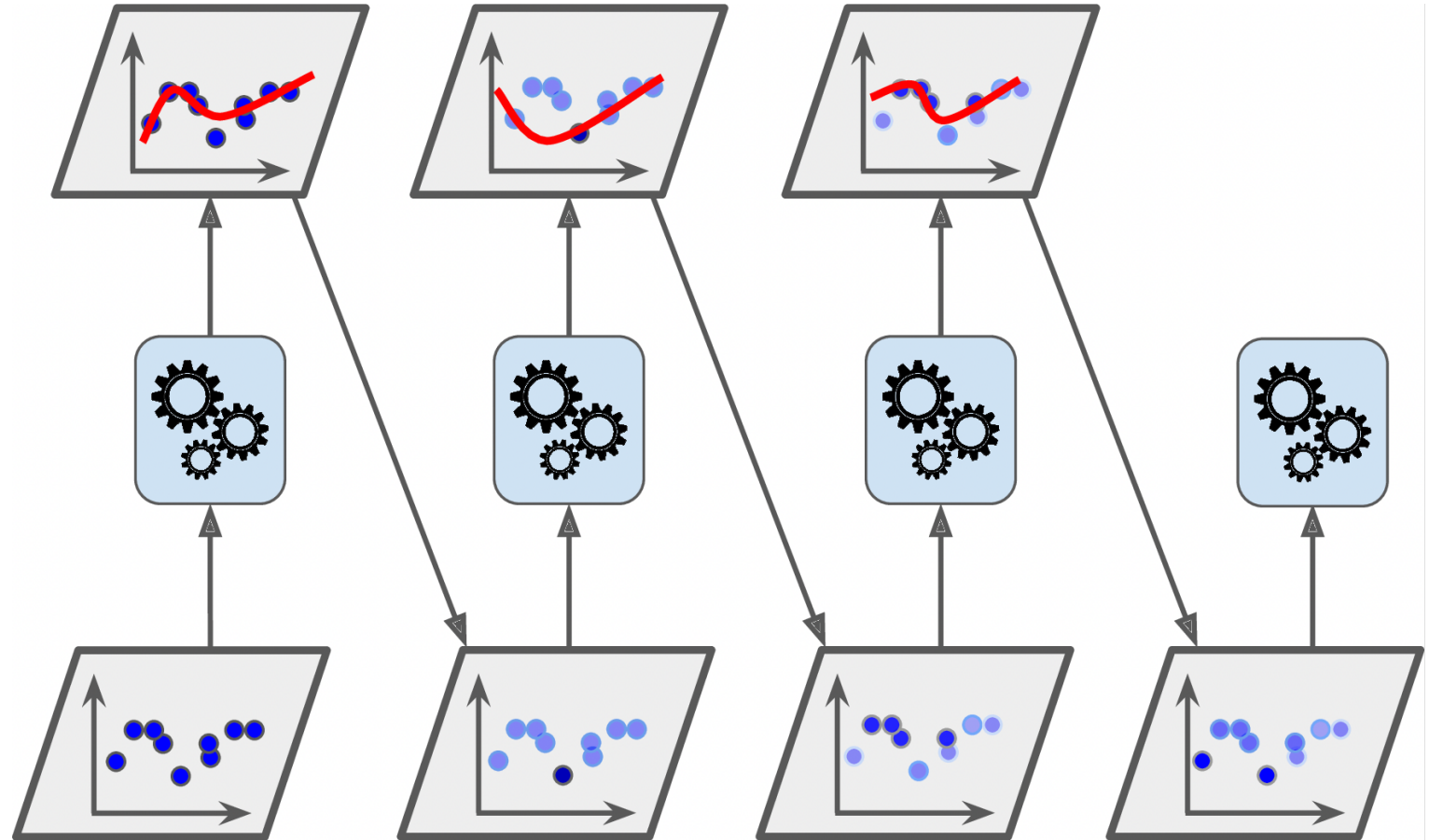
Boosting

- *Boosting* (originally called *hypothesis boosting*) refers to any Ensemble method that can combine several weak learners into a strong learner. The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor.



Adaboost

- One way for a new predictor to correct its predecessor is to **pay a bit more attention to the training instances that the predecessor underfitted**. This results in new predictors focusing more and more on the hard cases
- Example : A first base classifier (such as a Decision Tree) is trained and used to make predictions on the training set. The relative weight of misclassified training instances is then increased. A second classifier is trained using the updated weights and again it makes predictions on the training set, weights are updated, and so on.



Adaboost Hyperparameter tuning

If your AdaBoost ensemble is overfitting the training set, you can try reducing the number of estimators or more strongly regularizing the base estimator.

- **base_estimator** *object, default=None* The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper `classes_` and `n_classes_` attributes. If `None`, then the base estimator is [DecisionTreeClassifier](#) initialized with `max_depth=1`.
- **n_estimators** *int, default=50* The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.
- **learning_rate** *float, default=1.0* Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the `learning_rate` and `n_estimators`

Gradient boost descending

- Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor.
- Instead of tweaking the instance weights at every iteration like AdaBoost does, this method tries to fit the new predictor to the *residual errors* made by the previous predictor.
- The `learning_rate` hyperparameter scales the contribution of each tree. If you set it to a low value, such as 0.1, you will need more trees in the ensemble to fit the training set, but the predictions will usually generalize better. This is a regularization technique called *shrinkage*.