AI & Robotics

Boosting

AI & ROBOTICS LAB

Goals

The **junior-colleague**

- can explain boosting in their own words
- can describe the stopping criteria for boosting algorithms
- can explain AdaBoost in their own words
- can explain what a loss function is and why we use them
- can explain the idea behind gradient descent in their own words
- can explain the importance of the learning rate in the context of gradient descent
- can explain gradient boosting in their own words
- can explain the differences between bagging and boosting
- can explain the advantages of XGBoost over other bagging and boosting algorithms

Boosting

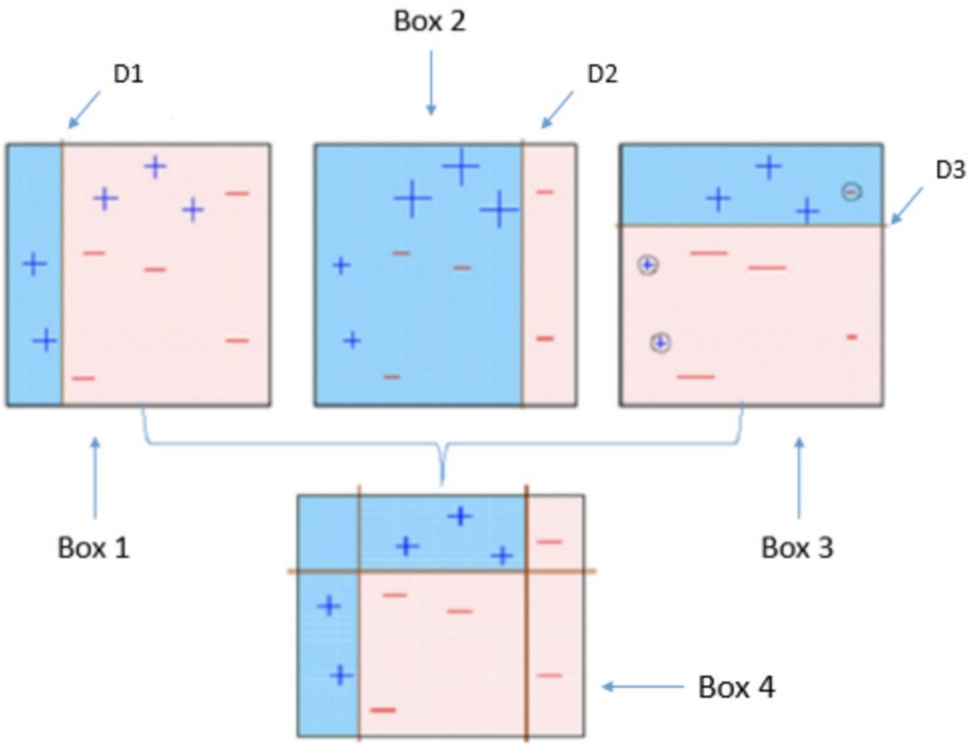
- Ensemble technique

  => Default learner: decision trees

  => Other learners possible
- Reasoning behind ensembles: minimizing bias, variance and the effects of noise
- Boosting is sequential

AdaBoost
(Adaptive Boosting)

AdaBoost

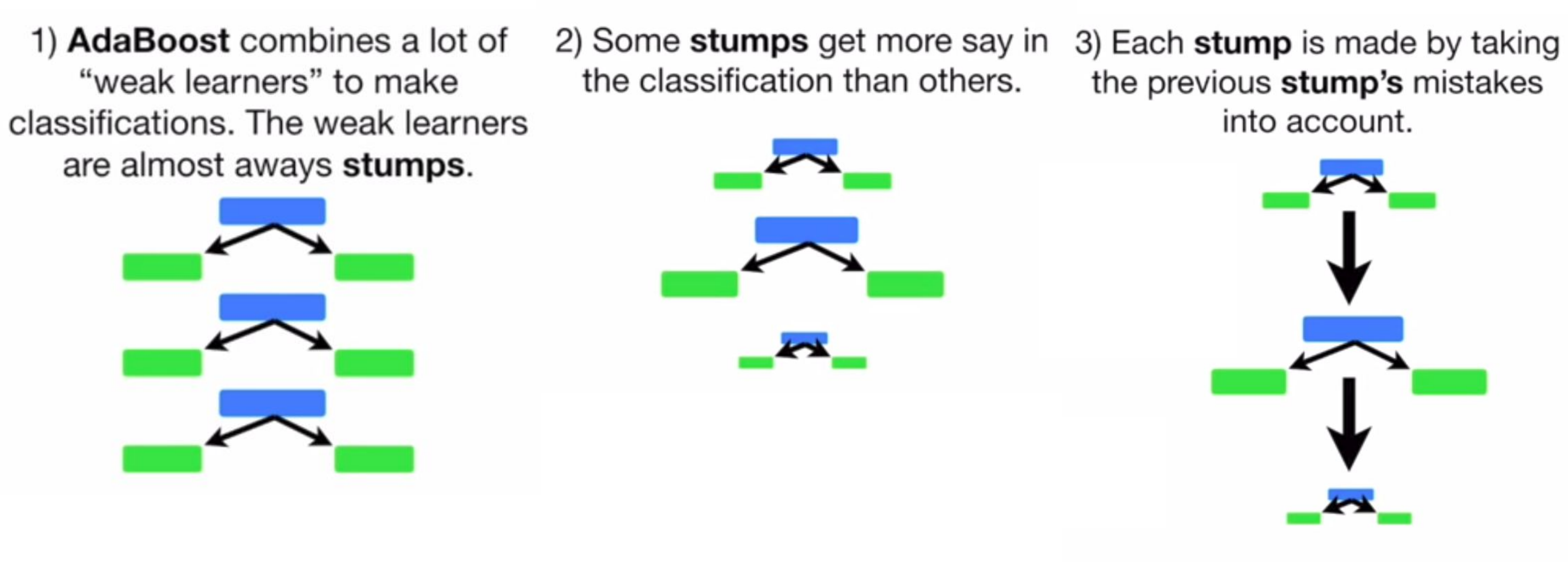


- Start with initial fitting
- Errors:
  - Increase the weights of the incorrectly classified examples
  - Decrease correct classifications
- Fit again
- Continue until stopping criterium
- Average over the different models

(an error less than 50% is required to keep the estimator)

=> Sensitive to noise!

AdaBoost

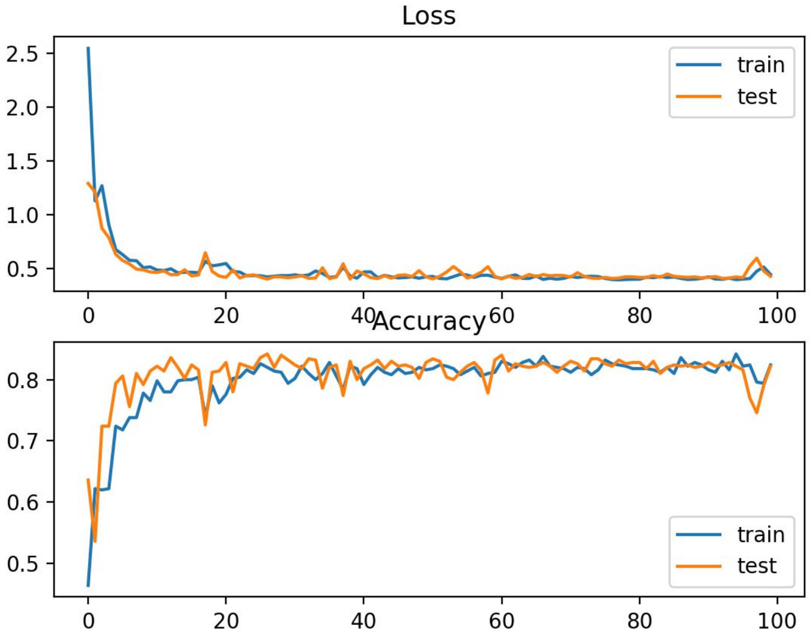

1) **AdaBoost** combines a lot of "weak learners" to make classifications. The weak learners are almost aways **stumps**.
2) Some **stumps** get more say in the classification than others.
3) Each **stump** is made by taking the previous **stump's** mistakes into account.

AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier()
# n_estimators = 50 (default value)
# base_estimator = DecisionTreeClassifier (default value)
clf.fit(x_train,y_train)
clf.predict(x_test)
```

Gradient Boosting

Loss functions



- Optimization objective
  - Minimize Errors
  - == Minimize Loss
- Regression: MSE / MAE

$$\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2$$

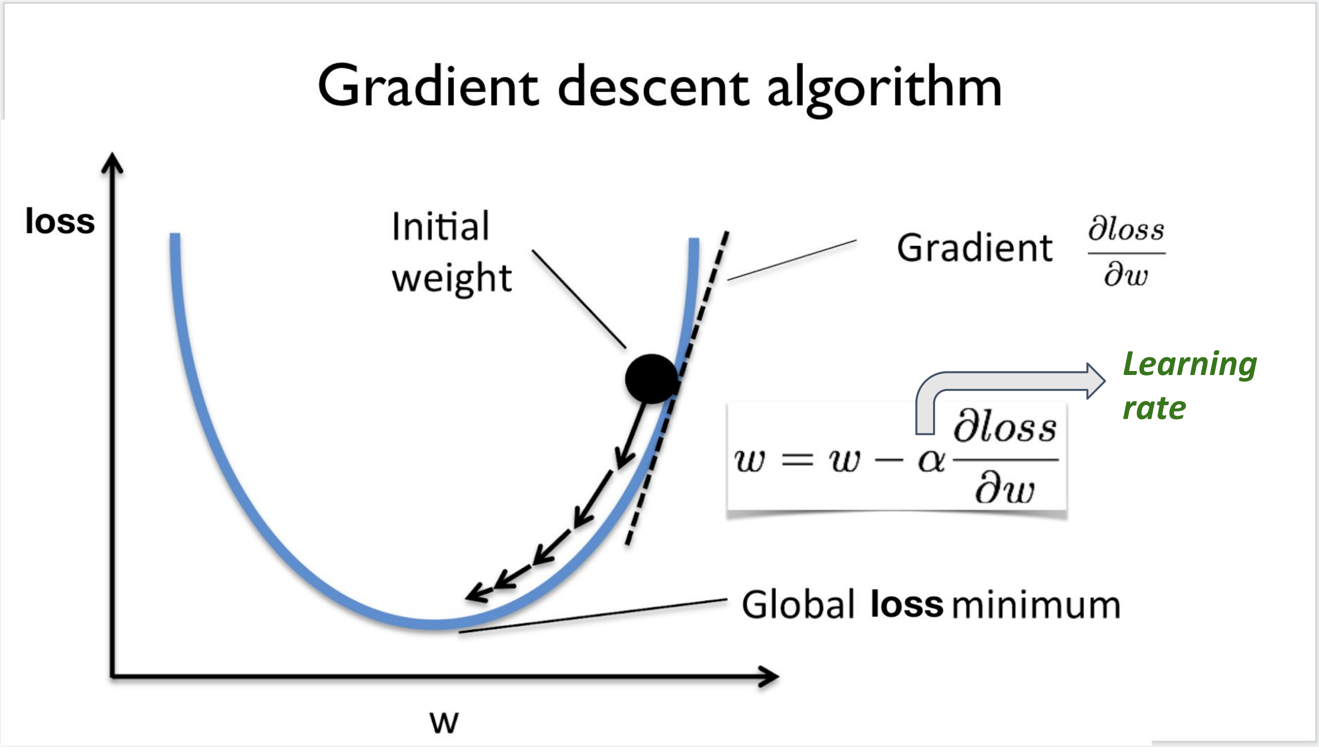
- Classification: Log Loss

$$-\frac{1}{N}\sum_{i=1}^{N}(y_i\log(p_i)+(1-y_i)\log(1-p_i))$$

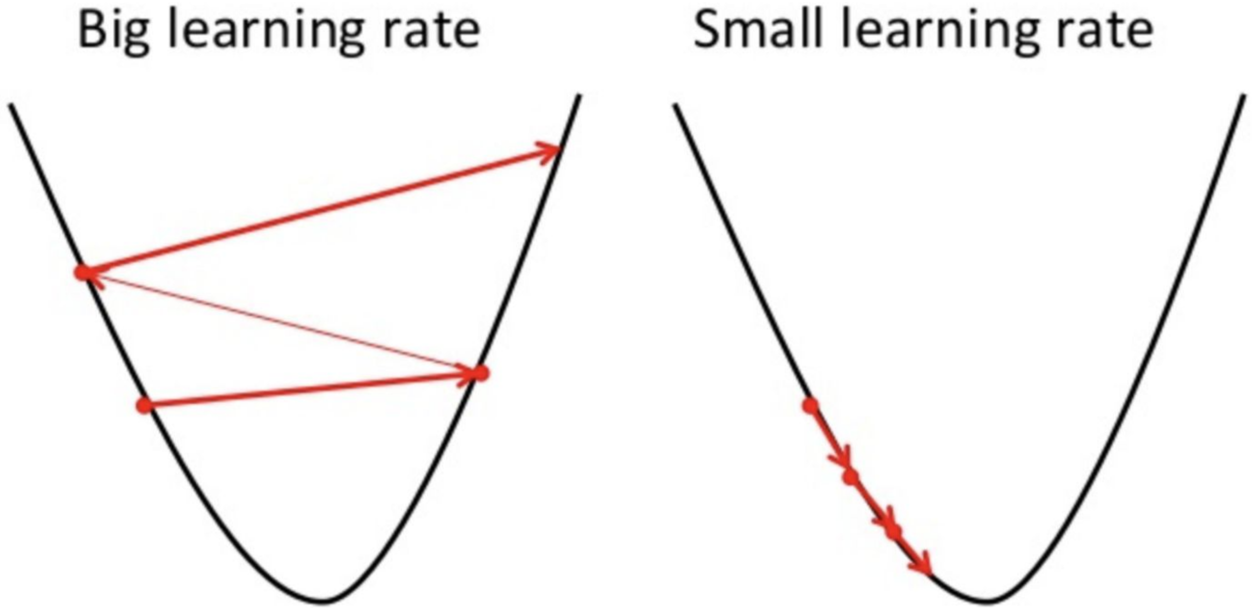
- Multi classification: M Log Loss

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}y_{i,j}\log(p_{i,j})$$

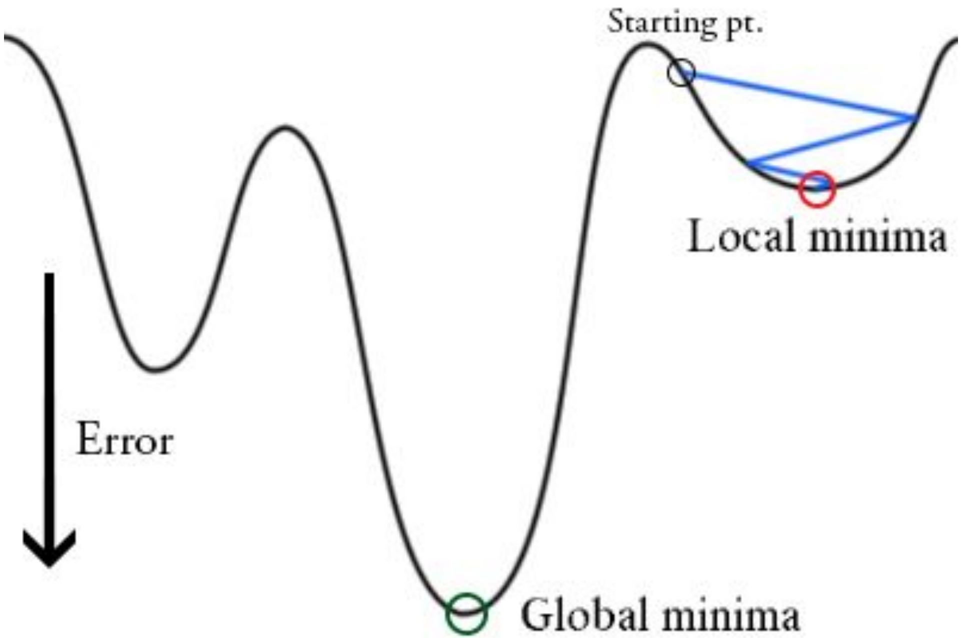
Gradient Descent



Gradient Descent



Gradient Descent



Gradient Boosting

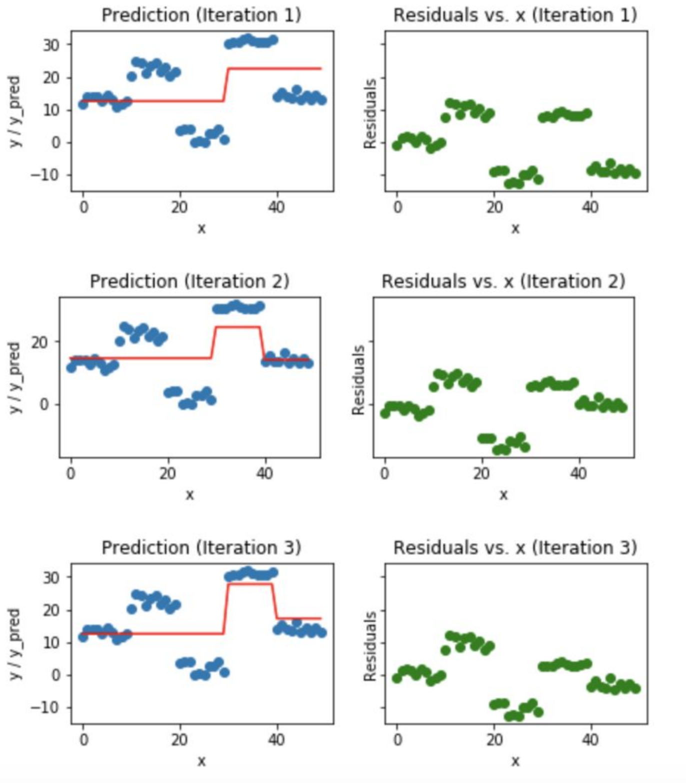
- Every round you're learning errors
- Corresponds to running gradient descent on loss function

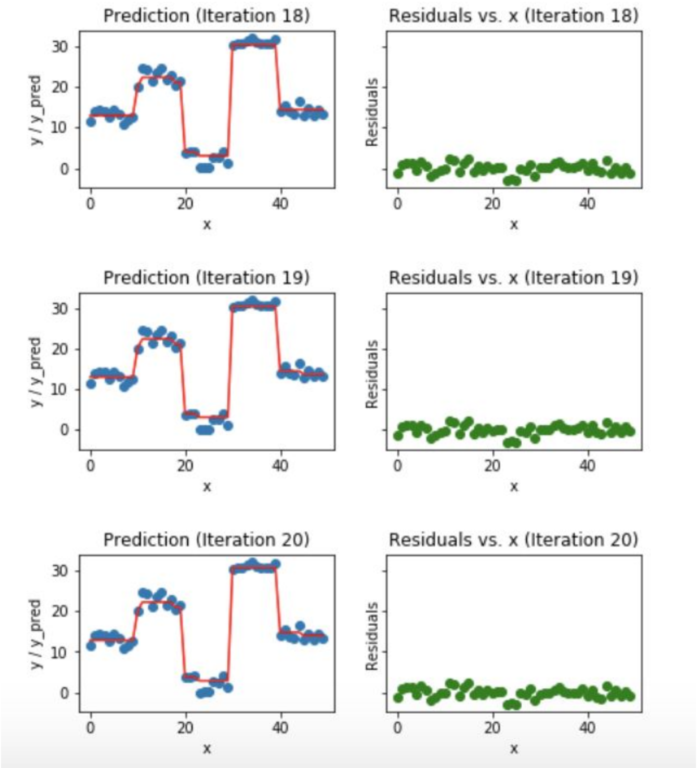
  => We will see this in depth for Neural Networks

Stopping criteria

- After a certain number of trees
- When the error goes up
- When there’s no more improvement on the validation set

Gradient Boosting

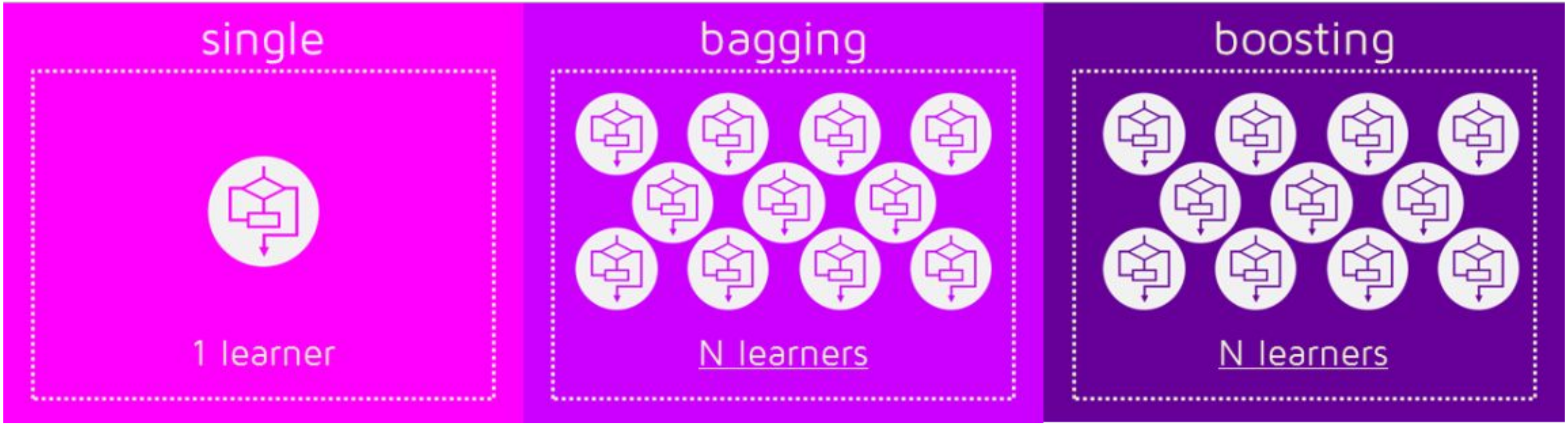




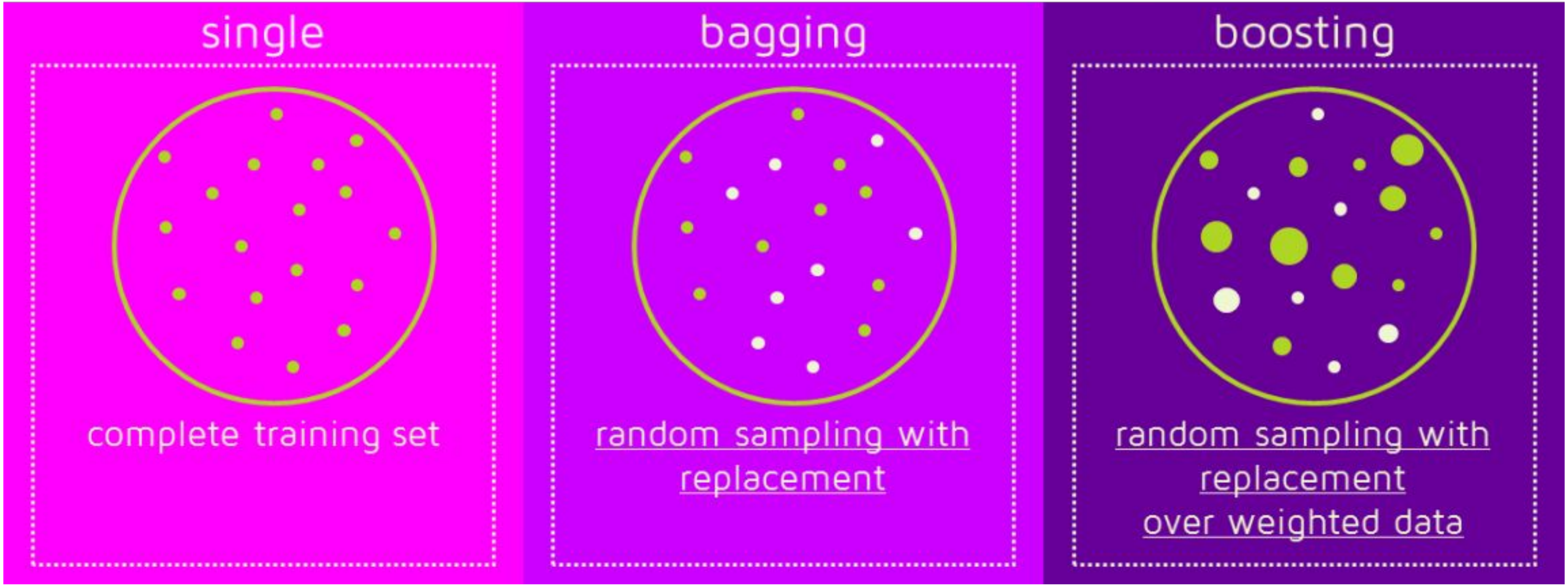
Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier()
# n_estimators = 100 (default)
# loss function = deviance(default) used in Logistic Regression
clf.fit(x_train,y_train)
clf.predict(x_test)
```

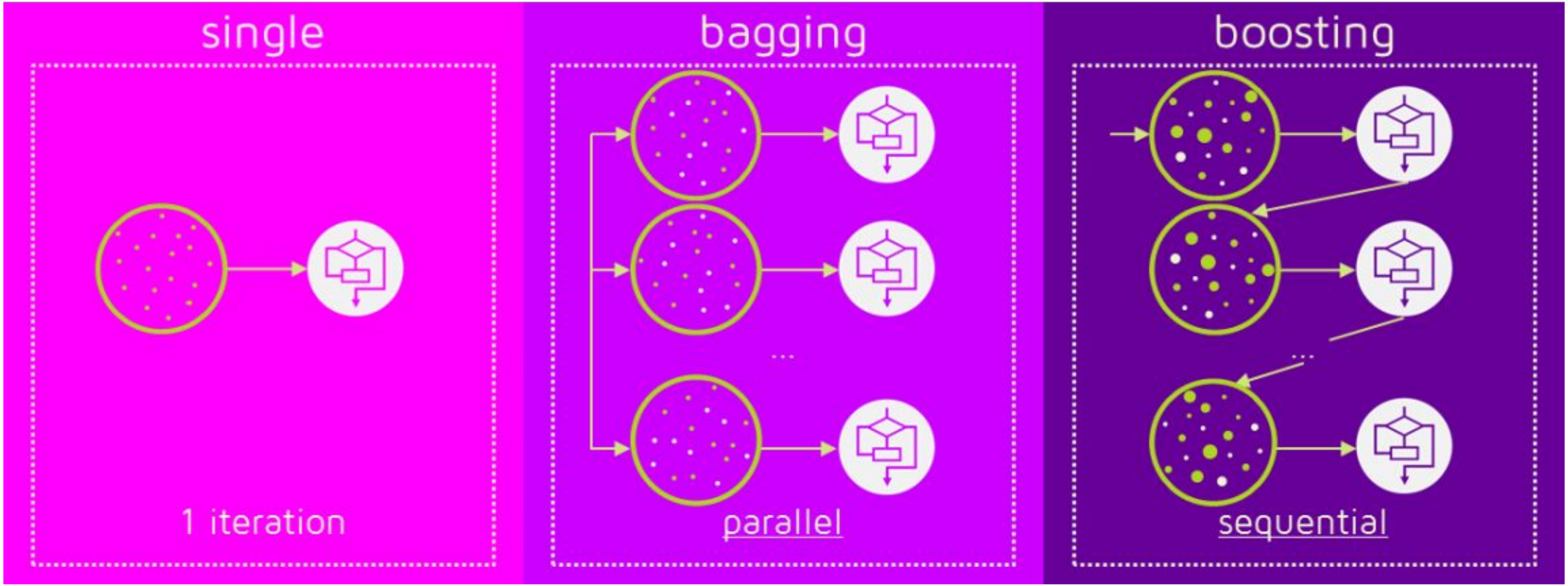
Bagging vs. boosting



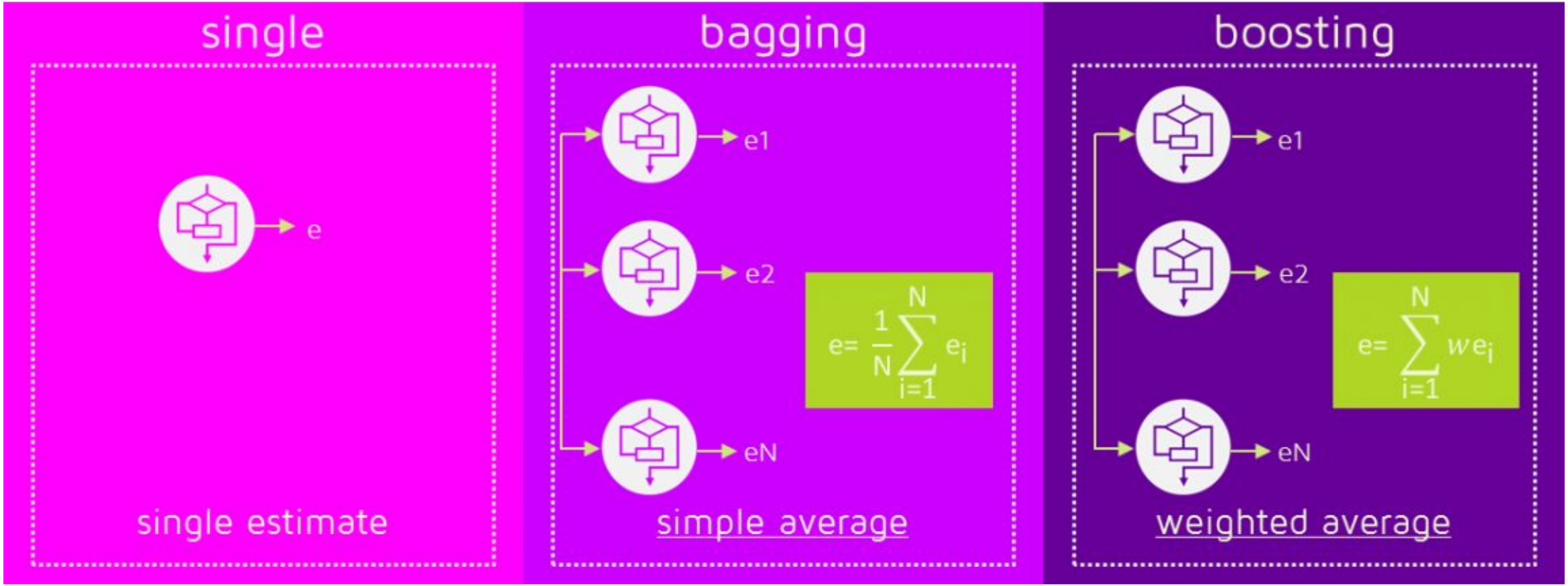
Bagging vs. boosting



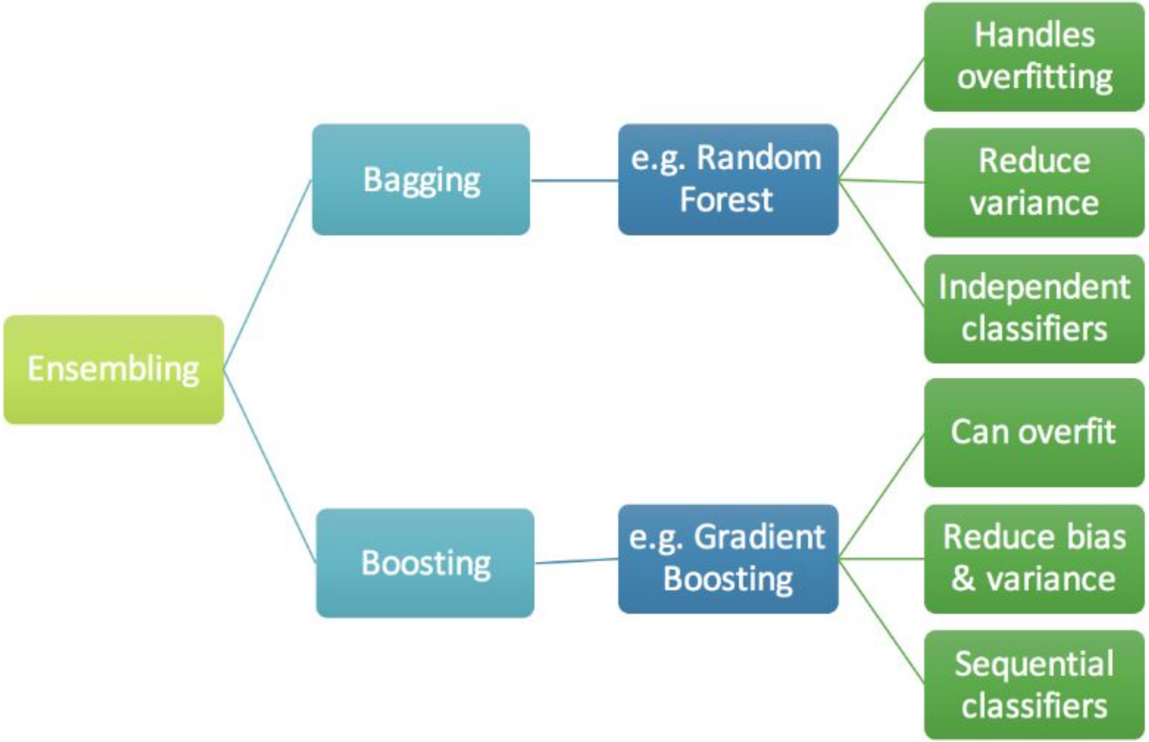
Bagging vs. boosting



Bagging vs. boosting



Bagging vs. boosting



Bagging vs. boosting

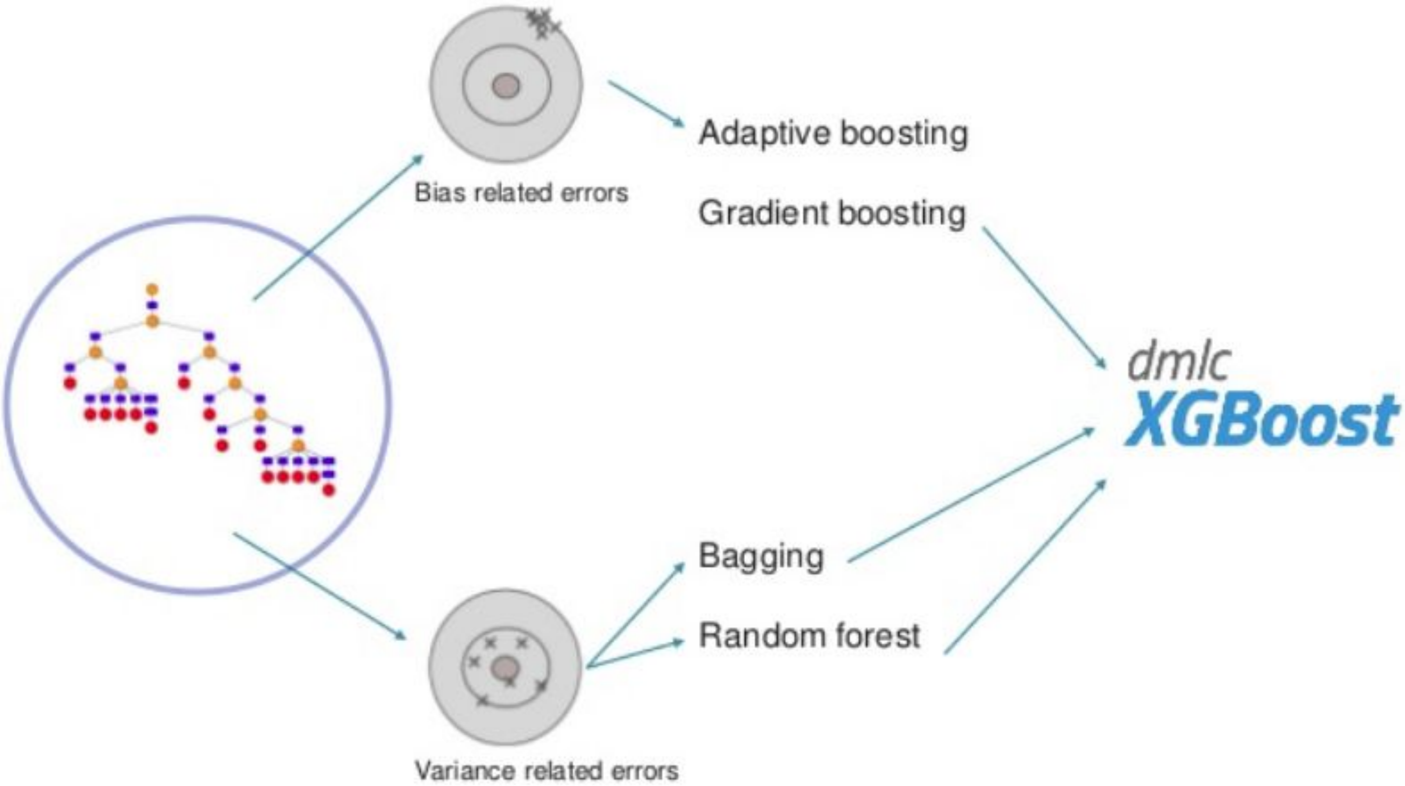
Bagging	Boosting
Bootstrapping (resample data points)	Bootstrapping (reweight data points)
Better suited for reducing variance and overfitting	Better suited for reducing bias and underfitting
Models are built independently (parallel)	A model is built, based on the previous model (sequential)

EXTREME

XGBoost


eXtreme Gradient Boosting
custom tree building algorithm
XGBoost
by Tianqi Chen
Used for:
• classification
• regression
• ranking
with custom loss functions
Interfaces for Python and R, can be executed on YARN

XGBoost


Bias related errors
Adaptive boosting
Gradient boosting
dmlc
XGBoost
Bagging
Random forest
Variance related errors

XGBoost

- Very fast Gradient Boosting
- Reduce correlation by sampling
  - Examples
  - Columns for each tree / split
- Tree Constraints
  - Number of trees
  - Tree depth

    => shorter trees are preferred (4-8 levels)
  - Number of nodes
  - Minimum improvement to loss

    => constraint on the improvement of any split added to a tree

XGBoost: Data

- Categories to numeric
- One-hot encoding
- Missing values:
    - Separately handled
    - Always added to a branch in which it would minimize the loss

      => treated as either very large / very small value

XGBoost

- Robust
- Insensitive to noise
- Underfitting vs overfitting
    - Underfitting => Not enough rounds: Start where you finished and continue
    - Overfitting => Limit number of rounds

XGBoost

```
from xgboost import XGBClassifier
clf = XGBClassifier()
# n_estimators = 100 (default)
# max_depth = 3 (default)
clf.fit(x_train,y_train)
clf.predict(x_test)
```