DM & ML
Ensemble learning

Gergely Lukács
Pázmány Péter Catholic University
Faculty of Information Technology
Budapest, Hungary
lukacs@itk.ppke.hu

Combining multiple models

„Crowd can be smarter than the individuals”

- Basic idea of “meta” learning schemes: build different “experts” – or crowd -- and let them vote
  - Sufficiently diverse crowd…
- Advantage:
  - often very good predictive performance
- Disadvantage:
  - interpretation of model ? (black box…)

Stacking / 1

- Train different machine learning models on same dataset
  - *Level-0 Models* (Base-Models)
    - (Different algorithms)
  - Standard ML algorithms
    (e.g., Decision tree, Bayesian network)
  - Special algorithms of the application area
    (e.g. Natural Language Processing, parsing)
  - Models of different teams in competitions („team merging”)
- Combine the results of those models
  - *Level-1 Model* (Meta-Model)
  - Often simple model (in principle: any model)
  - David Wolpert: “relatively global, smooth” model
    - Base learners do most of the work
    - Reduces risk of overfitting

Stacking / 2

- Output of Level-0 models
  - Classification: label
    - Even better: also probabilities!
  - Numeric prediction („Regression”): Value
  - These are used to train Level-1 model
    - Optional: additionally attributes of data as additional context

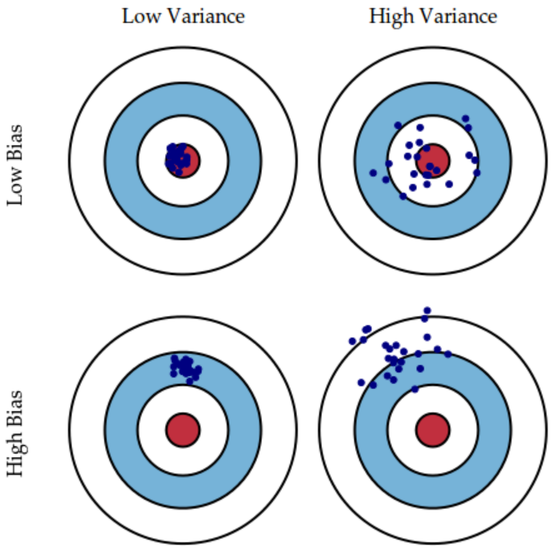
Stacking / 3

- Data used for training level-0 models not to be used to train level-1 model! – overfitting!
  - Cross-validation-like scheme is employed

- Training of Level-0 models
  - Some sort of validation, typical: cross validation – for training Level-1 model
  - Finally: all data used for training
- Training of Level-1 model
  - Data not used to train Level-0 models

Bagging

- **Same algorithm** (learning scheme) for all classifiers
- **Different datasets**
- “Idealized” version:
  - Sample several training sets of size *n* (instead of just having one training set of size *n*)
  - Build a classifier for each training set
- Combining predictions by voting/avera
  - Simplest way (e.g. each model receives equal weight)
    - Reduces variance by voting/averaging, thus reducing the overall expected error



Bagging – creating several datasets

- Problem: we only have one dataset!
- Solution: generate new datasets of size *n* by **sampling with replacement** from original dataset
  - B-Agging = Bootstrap aggregating

Bagging classifiers

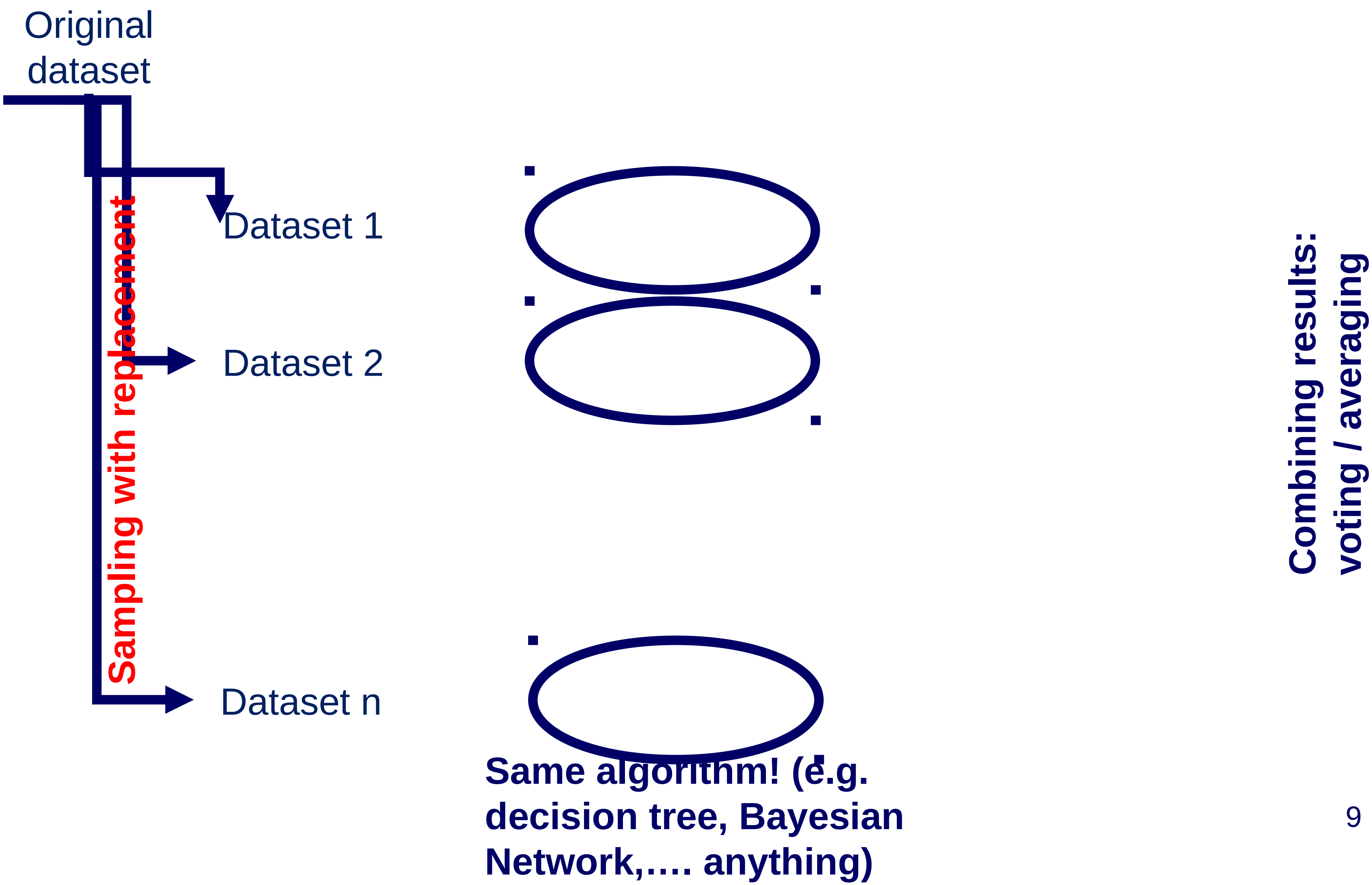
model generation

```
Let n be the number of instances in the training data.
For each of t iterations:
  Sample n instances with replacement from training set.
  Apply the learning algorithm to the sample.
  Store the resulting model.
```

classification

```
For each of the t models:
  Predict class of instance using model.
Return class that has been predicted most often.
```

Bagging



Original dataset
Sampling with replacement
Dataset 1
Dataset 2
Dataset n
Combining results: voting / averaging
Same algorithm! (e.g. decision tree, Bayesian Network,…. anything)

More on bagging

- Learning scheme is unstable
  (Small change in training data can make big change in model;e.g. decision trees)
  -> almost always improves performance
- Theoretically, bagging would reduce variance without changing bias
  - In practice, bagging can reduce both bias and variance
    - For high-bias classifiers, it can reduce bias
    - For high-variance classifiers, it can reduce variance
  - Usually, the more classifiers the better
  - Can help a lot if data is noisy
  - In the case of classification there are pathological situations where the overall error might increase

Bagging with costs

- good probability estimates
  - instead of voting, the individual classifiers' probability estimates are averaged
- also for learning problems with costs

Randomization

- Can randomize learning algorithm instead of data
- Some algorithms already have a random component:
  - e.g. initial weights in neural net
- Most algorithms can be randomized, e.g. greedy algorithms:
  - Pick from the *N* best options at random instead of always picking the best options
  - E.g.:
    - attribute selection in decision trees
    - *random subspaces*: random subsets of attributes in nearest-neighbor scheme
- Can be combined with bagging

Random forest

- „Forest” – multiple decision trees
- Random:
  - Feature randomness:
    Only considers attributes in a random subset of all attributes for a split (instead of all attributes)
  - Data randomness:
    usually bootstrapping data (sampling with replacement)
- Voting

Rotation forest

accurate and diverse ensemble members

(2006, Rodríguez at al.)

- Bagging creates ensembles of accurate classifiers with relatively low diversity
  - Bootstrap sampling creates training sets with a distribution that resembles the original data
- Randomness in the learning algorithm increases diversity but sacrifices accuracy of individual ensemble members
- Accuracy-diversity dilemma
- Rotation forests have the goal of creating accurate **and** diverse ensemble members

Rotation forest/2

- An iteration involves
  - Randomly dividing the input attributes into *k* disjoint subsets
  - Applying PCA to each of the *k* subsets in turn
    - (all attributes are kept; **it preserves all information**!)
  - Learning a decision tree from the *k* sets of PCA directions

Rotation forest/3

- „Rotation forest”
  - Base classifier: decision tree → „forest”
  - PCA is a simple rotation of the coordinate axes → „rotation”
- Rotation forest has the potential to improve on diversity significantly without compromising the individual accuracy

Boosting

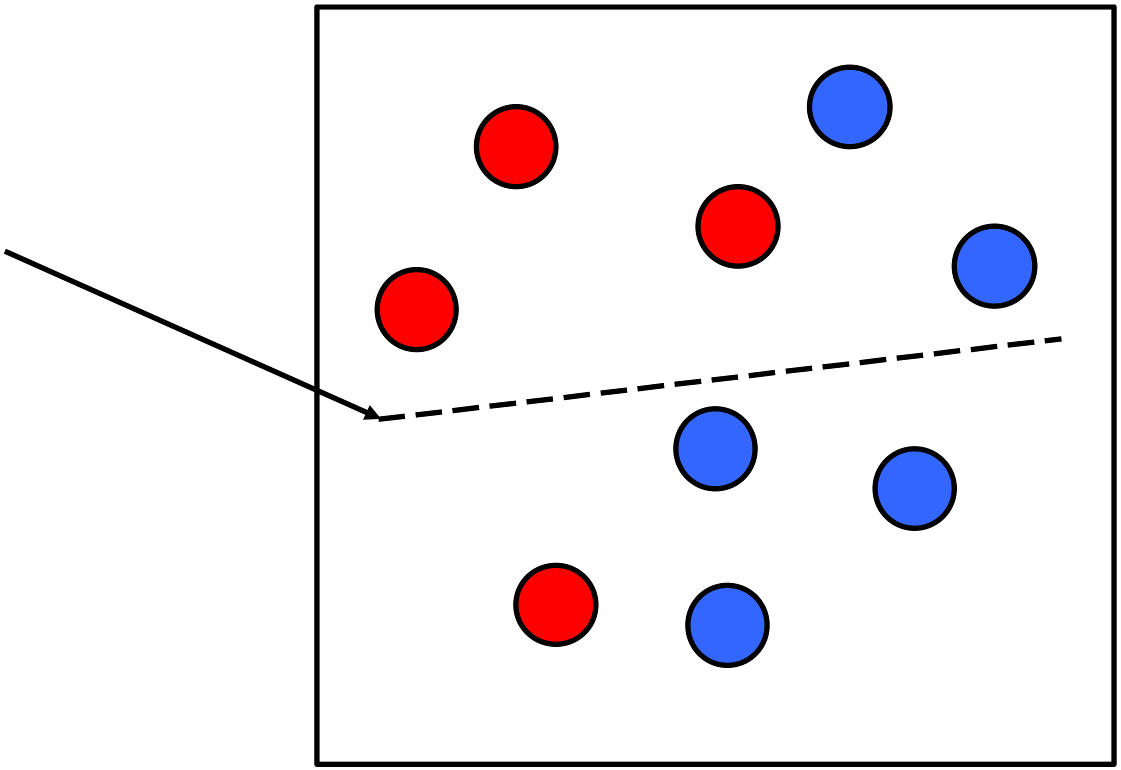
- Boosting: explicitely seeks models that complement one another
- Works well with weak models

Boosting 2

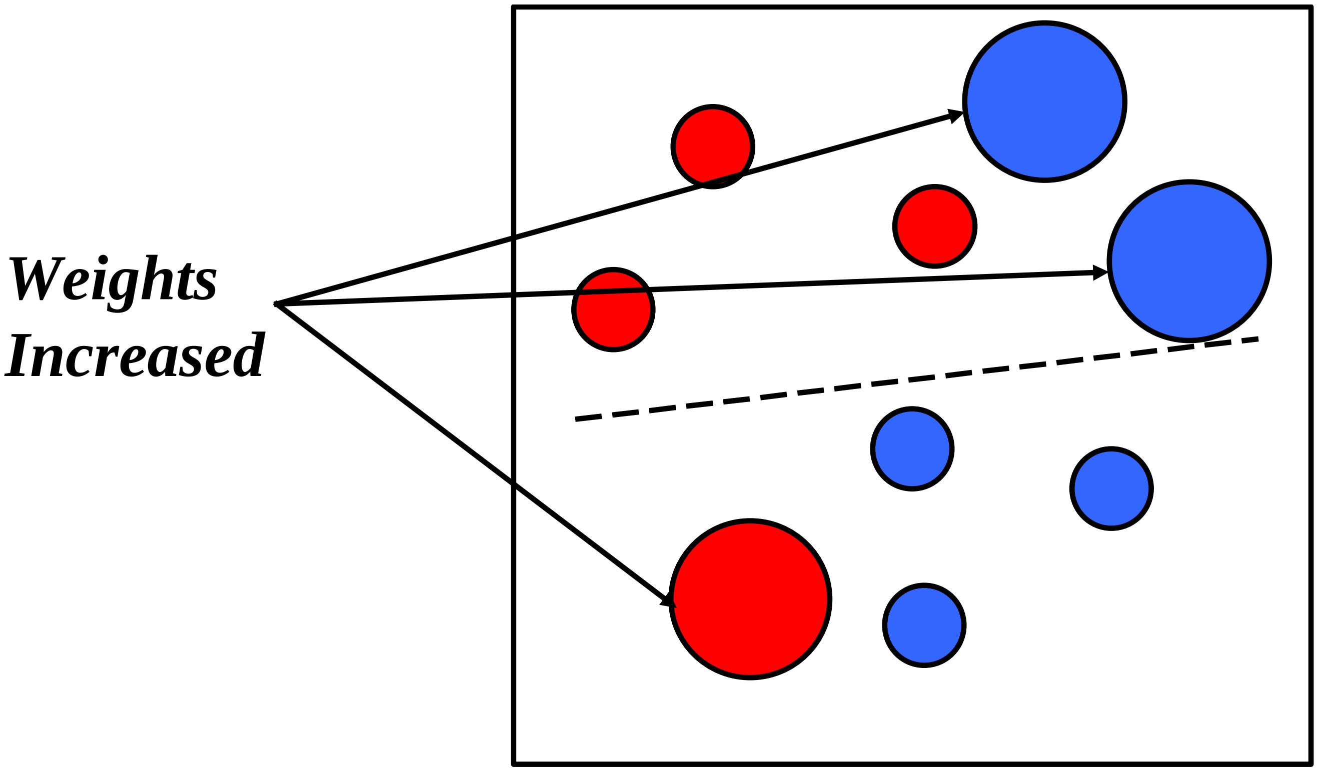
- Similar to Bagging
  - Voting (classification), averaging (numeric prediction)
  - Models of the same type (e.g. decision trees)
- Different from bagging
  - Iterative, models not built separately
  - New model is encouraged to become expert for instances classified incorrectly by earlier models
    - Intuitive justification: models should be experts that complement each other
  - Weighted voting!
- There are several variants of this algorithm

Boosting illustration

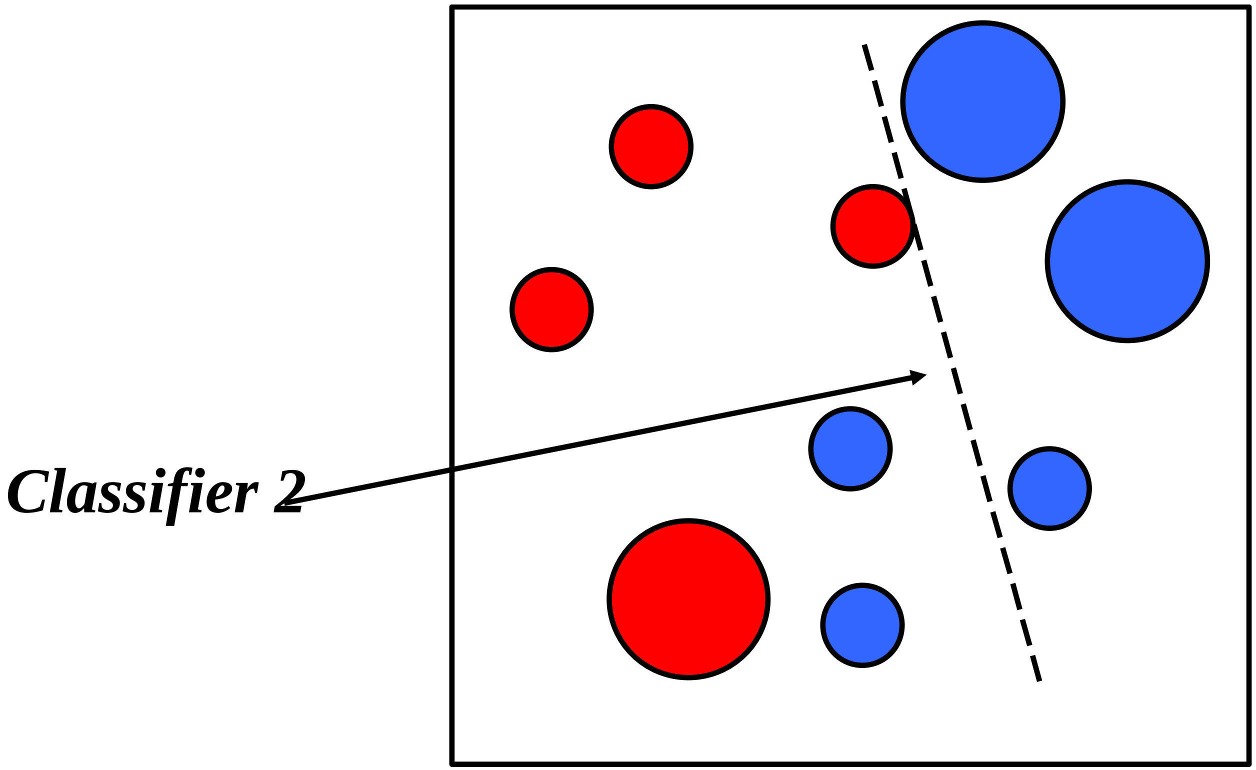
Classifier 1



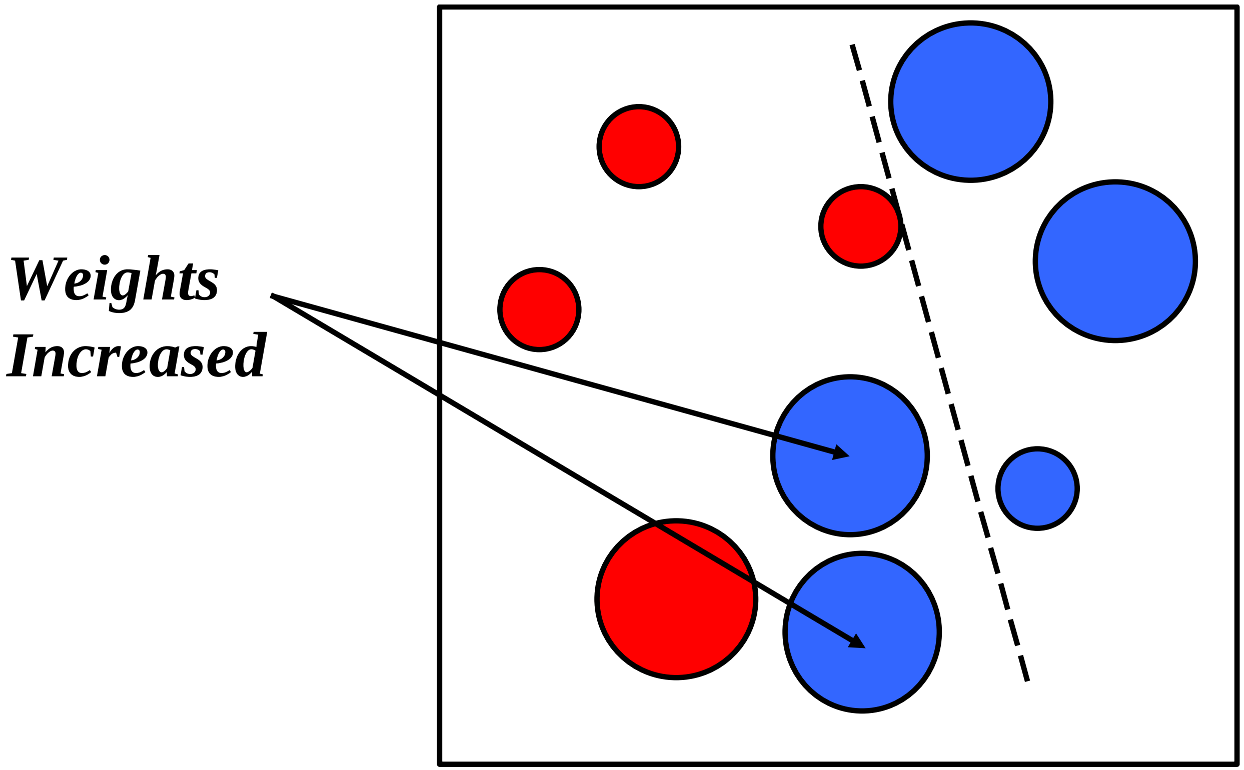
Boosting illustration



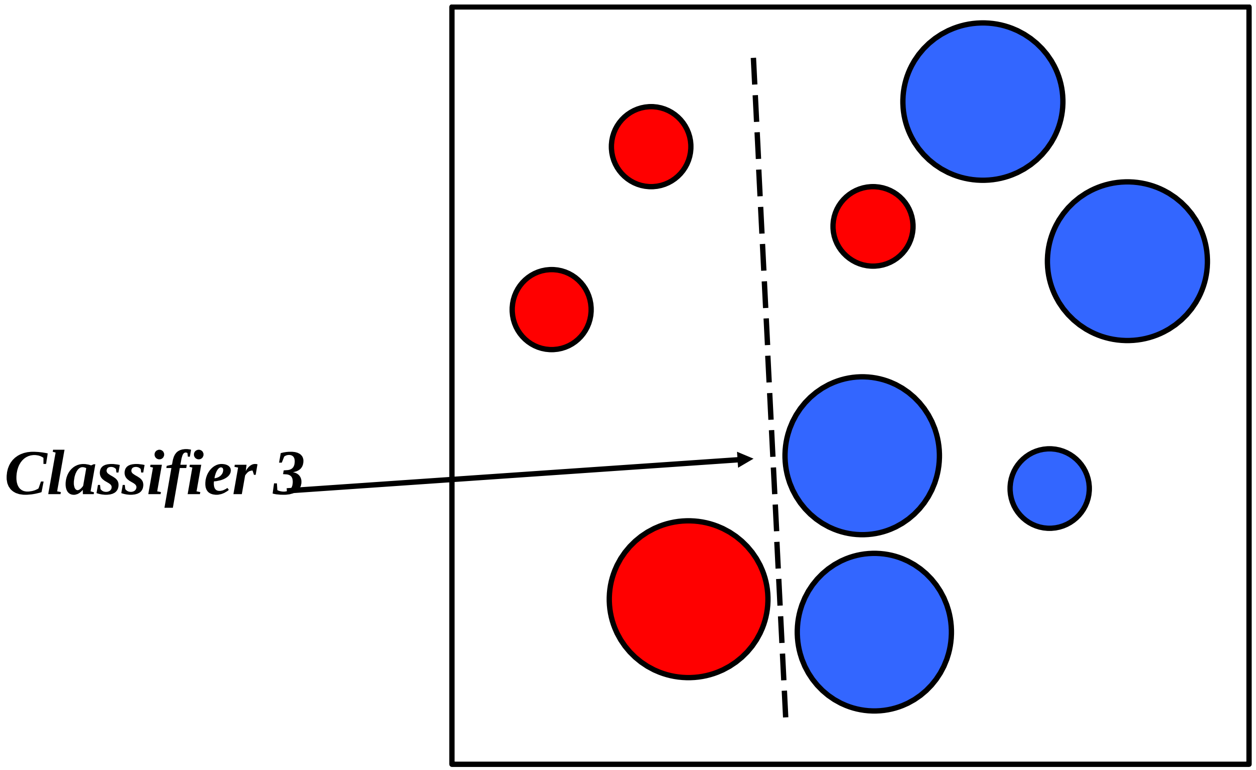
Boosting illustration



Boosting illustration

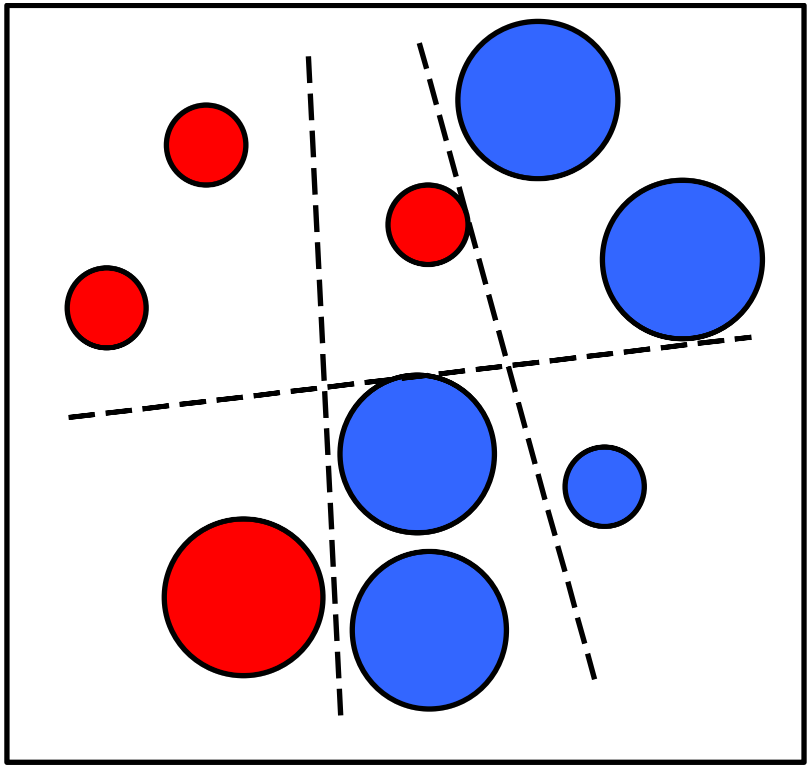


Boosting illustration



Boosting illustration

Final classifier is a combination of single classifiers



AdaBoost.M1

Model Generation

```
Assign equal weight to each training instance.
For each of t iterations:
  Apply learning algorithm to weighted dataset and
   store resulting model.
  Compute error e of model on dataset and store error.
  If e equal to zero, or e greater or equal to 0.5
    Terminate model generation.
  For each instance in dataset:
    If instance classified correctly by model
       Multiply weight of instance by e / (1 - e).
  Normalize weight for all instances.
```

Classification

```
Assign weight of zero to all classes.
For each of the t (or less) models:
  Add -log(e/(1-e)) to weight of class model predicts.
Return class with highest weight.
```

More on boosting

- Theoretical result:
  - Upper limit for training error decreases exponentially
  - Boosting works with weak learners only condition: error doesn’t exceed 0.5
    - Week learner: only slightly better, than random predictor
- Practical issue: Boosting needs weights
  - adapting learning algorithm to use weights
  - resample with probability determined by weights

- AdaBoost in fact
  - Additive model
  - Particular loss function (exponential loss)

MIT OpenCourseWare:

(17. Learning: Boosting)

- https://www.youtube.com/watch?v=UHBmv7qCey4&t=2752s

Gradiant Boosting

- AdaBoost: adjusting weights of data points…
- Difference between the prediction and the ground truth.
  - Optimization problem
    - on suitable cost function
    - over function space

  - Iterative, greedy
  - by choosing function pointing in negative gradient direction

Elements of Learning

Example:
Linear regression
$\hat{y}_i = \sum_j \theta_j x_{ij}$

- Model
  - Parameters
- Objective Function
  - Training loss

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

  - + regularisation!!!

$$\text{obj}(\theta) = L(\theta) + \Omega(\theta)$$

XGBoost – theoretical advantages

- Decision tree ensembles
- Regularisation!
  - Formalizes complexity of tree classifiers overfitting reduced!!

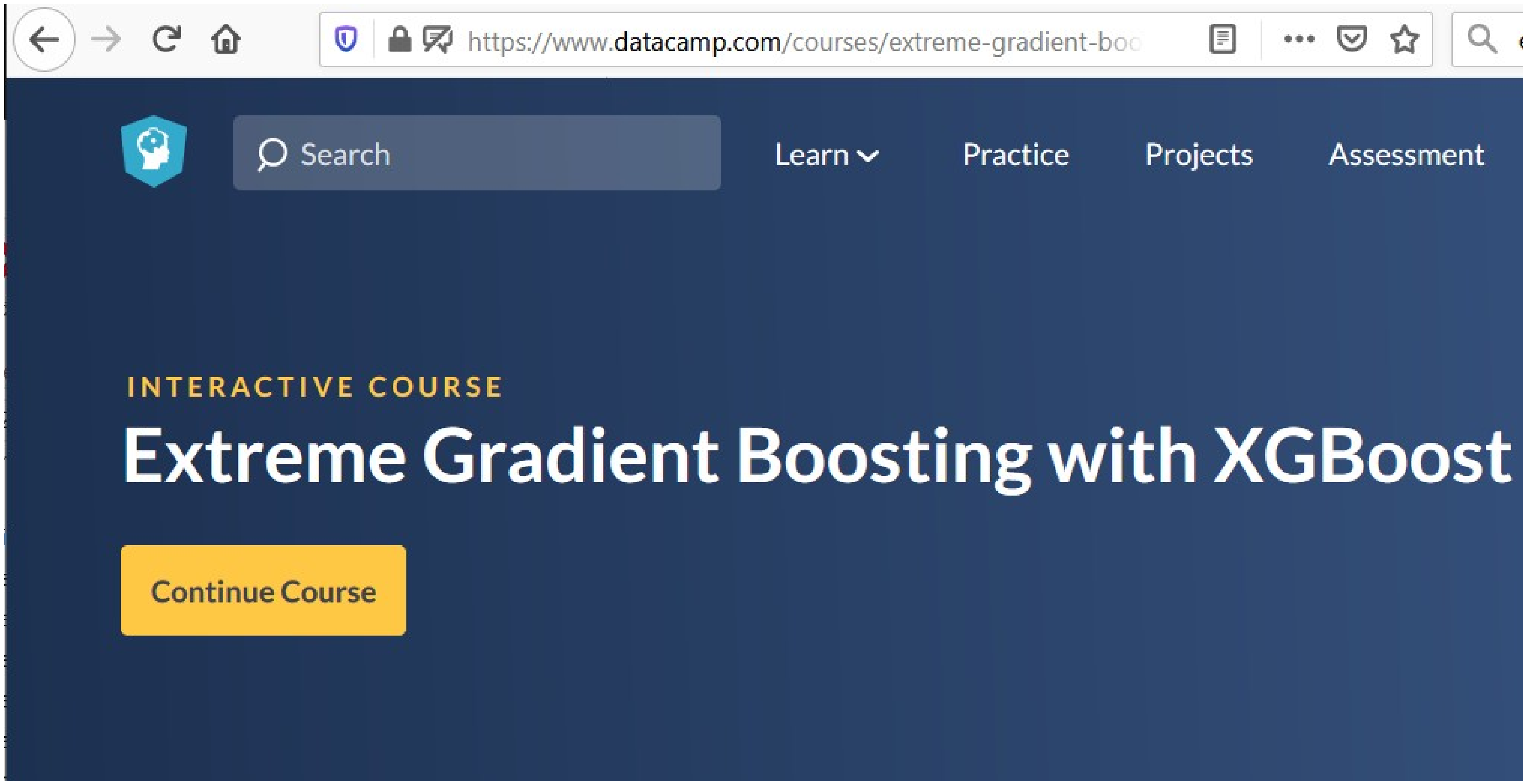
- Custom optimization objectives and evaluation criteria
- +… (handling missing values, built-in cross validation, continue existing model…)

XGBoost – computing advantages

- Use of **sparse matrices** with sparsity aware algorithms
- Improved **data structures** for better **processor cache utilization** which makes it faster.
- Better support for **multicore processing** which reduces overall training time.

XGBoost – Parameter Tuning!

- https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/


https://www.datacamp.com/courses/extreme-gradient-boo
Search
Learn
Practice
Projects
Assessment
INTERACTIVE COURSE
Extreme Gradient Boosting with XGBoost
Continue Course

XGBoost vs Deep learning

- Start with simple models!!

- XGBoost
  - Easier to train
  - Less computational resources
  - Better if categorical + numeric features
- Deep learning
  - Image Recognition, Computer Vision, Natural Language Processing (some sort of structure, space)

Comparison

Stacking	Bagging	Boosting
Data1 = Data2 = … = Data m	Data1 ≠ Data2 ≠ … ≠ Data m	
	Resample training data	Reweight training data
Learner1 ≠ Learner2 ≠ … ≠ Learner m	Learner1 = Learner2 = … = Learner m	
Level-1 model	Vote	Weighted vote

Summary/concepts/questions

- Bias-variance decomposition
- Stacking
- Bagging
- Randomization
  - Random Forest
  - Rotation forest
- Boosting
  - AdaBoost
  - Gradient Boosting
    - XGBoost