**ENSEMBLE LEARNING**

**Remarks:**

* **Sometimes, adding noise increase performances**
* **Hard to analyze: cannot identify the factors that are contributing to the improved decisions**

Train multiple learners on the same problem and combine them in order to maximize the prediction accuracy.

Basic approaches:

* Bagging
* Boosting
* Stacking

***Bagging*** : combine the decisions of different models by amalgamating the various outputs into a single prediction -> For classification (weighted vote)

* The more the number of models combined, the lower the variance
* Create multiple dataset starting from one, by randomly sampling instances with replacement (replication of some of the data)
* Results can be improved by increasing the diversity in the ensemble of classifiers by making the learning scheme as unstable as possible, while maintaining some level of accuracy.

***Boosting*** : seek for models the complement one another (new models are trained to become experts for instances labeled incorrectly by earlier ones [iterative process])

* ADABOOST.M1 (for classification) -> converts weak learners onto strong ones

***Rotation forests***

***Numeric prediction*** : considering *residuals* (= difference between predicted and observed values on the training dataset)

* Add a second model that tries to predict the observed residuals. To do this, simply replace the original class values by their residuals before learning the second model. Learn the residuals of the residuals and so on.

***Option trees*** : single structure that represent an ensemble of classifiers

* 2 types of node (*decision* nodes, *option* nodes)
* Instances may end up in more than one leaf, and the classification obtained form those leafs must somehow be combined into an overall classification.

***Stacking*** : try to learn which classifiers are the reliable ones, using another learning algorithm [the *meta-learner*] to discover how to best combine the output of the base learners.

**A FRAMEWORK FOR PARALLEL AND DISTRIBUTED TRAINING OF NEURAL NETWORKS**

*Where to apply?*

Distributed environments, where trining data is partitioned over a set of agents that communicate with each other through a sparse, possible time-variant, connectivity pattern.

*Why?*

* Sharing local informations to a central processor might be either unfeasible or not economical/efficient.
* Privacy
* Central processor represents a bottleneck (and in general, an isolated point of failure)

*Possible issue*

Often the implementation of distributed learning schemes requires the training of a *shared* predictive function [a common model accessible independently by each “distributed” learner].

*What is proposed?*

Algorithmic framework for training general NN models is a fully distributed scenario, which encompasses several common loss functions and regularization terms. Built upon the in-network non convex optimization NEXT algorithm.

**IS COMBINING CLASSIFIERS WITH STACKING BETTER THAN SELECTING THE BEST ONE?**

*State-Of-The-Art*

Stacking with :

* Probability Distributions PDs
* Multi response linear regression MLR

*What next? Extensions?*

* Use an extended set of meta-level features
* Multi response model trees to learn at the meta-level ***BEST***

*Proposed solution?*

Stacking with multi-response model trees -> GOOD for combining classifiers that are *heterogeneous* and *strong*.

**FEATURE-WEIGHTED LINEAR STACKING**

*Are non linear combinations of meta-features the only way to improve prediction rate? [note that these procedures require significant tuning and training time]*

*Proposed solution*

FWLS : combine model predictions linearly using coefficients that are themselves linear function of the meta-features.

* Meta-features
* Speed, stability, interpretability

*Important Remark* :

The dataset collected for the stacked regression must consist of out-of-sample model prediction

**ATRIAL FIBRILLATION DETECTION WITH A DEEP PROBABILISTIC MODEL**

* Convolutional NN. Instead of prediction AF probability itself, the model will predict parameters of the beta distribution over this probability.
* Train with fixed-size EC