**Casting (boosted parallel ensemble, ant colony ensemble)**

Here we define a novel meta-algorithm for reconstructing serial and parallel ensembles. This meta algorithm - called casting - can be used to create a continuum of base learners which are either loosely connected (i.e. a parallel ensemble) or communicate tightly via a boosting algorithm (i.e. a serial ensemble). While casting can be defined generally to create a network of any set of machine learning algorithms, here we develop the case where each base estimator is a weak learner (a decision stump). This spectrum of learners is anchored at the parallel end by a random forest while its serial counterpart is Adaboost. Casting works by bagging subgroups of stumps (castes) which use Adaboost to communicate internally but operate independently. Casting introduces a new hyperparameter which controls the number of groups. If this hyperparameter is equal to the number of groups, casting functions as a random forest. If this hyperparameter is 1, then it’s equivalent to Adaboost. If this parameter is set some value in between these two extremes, then we have a novel ensemble which mimics natural systems.

This design could allow us to optimally trade-off bias and variance. Highly communicative ensembles might be more prone to bias than models with low social adhesions, as the decision of each base estimator depends on the decisions of previous estimators in the chain. Conversely, the decisions of independent networks are likely to be more variable. Intermediate networks may be able to get the best of both worlds, at least for some datasets.

This algorithm was inspired by the observation that groups of organisms (in particular ant colonies) tend not to be perfectly connected nor disconnected. They tend to form subgroups, or castes. For example, ants tend to communicate more with individuals that perform the same tasks (brood care workers tend to communicate more often with other brood care workers than foragers, for instance). This could be due to spatial limitations imposed on the colony. Brood tends to be located in the same part of the nest, so ants are more likely to encounter other brood care workers than they are other types of workers. However, subgroup formation could be adaptive as well, as overcommunication with groups that do not need to know the status of your work could be energetically taxing. Moreover, including irrelevant information could muck up optimal decision making (including many shopping items for humans, for example, make it more difficult for us to choose the best deals) while not including relevant information could also lead to bad decision making. In a sense, then, they might also be trying to trade off bias (doing the wrong thing because other ants are doing the wrong thing) and variance (doing the wrong thing because you don’t know anything about the environment beyond your local bubble). This model could yield more insights into the adaptive functionality of partially-connected networks. As individual ants do not have the cognitive capacity to evaluate global colony health, they can be considered weak learners. Therefore, another name for the particular algorithm we develop here is an ant colony ensemble.

It should be noted that a bag of decision stumps is not equivalent to a random forest. This is primarily due to the fact that in a random forest each tree only receives a subset of features. If there are F features, typically each tree in the random forest is given features. Conversely, trees in Adaboost will use all F features. We therefore introduce another parameter ⍴ to control the proportion of features a tree in the ant ensemble will be trained on. In the case of a random forest, ⍴ = /F

Here is description of how the algorithm works:

1. Set hyperpameters W, N, and ⍴
   1. N = the number of weak learners. These weak learners are analogous to ants, which might be able to differentiate between different classes but not very well. N could be equal to colony size, then.
   2. W = window size, or group size. This determines the number of ants that are connected via a boosting algorithm.
      1. These groups are then bagged together for final classification. There are N/W groups. If N/W is not a whole number, then the remainder will be grouped together to form the final group, which will have fewer ants than the other evenly-sized groups.
      2. When W = 1, then the ant ensemble is the equivalent of a random forest (a type of parallel ensemble which uses decision stumps). When W = N, then the ant ensemble is the equivalent of AdaBoost
   3. ⍴ (proportion of columns a base learner is trained on) is defined in terms of W. See equations below
   4. Set typical hyperparameters as well, such as learning rate.
2. Define the type of weak learner
   1. Here we use decision stumps, which are one-level decision trees
3. Divide data into training/test sets
4. Determine if this is a regression or classification problem
5. Clean data if necessary
   1. Remove outliers
   2. Group sparse classes
   3. Transform data to a symmetric distribution (log transform, box-cox, etc.)
   4. Remove NAs
6. Bag AdaBoost groups defined by N and W
7. Train the model
8. Perform k-fold cross validation
9. Do a grid search to find optimal hyperparameter combinations
10. Measure score of the final model on the test dataset
    1. Accuracy, balanced accuracy, f2, etc.

The proportion of randomly selected columns for each base estimator (or string of base estimators) can also depend on W. When W is low, this proportion is relatively small, resembling a random forest. When W is high, this proportion will approach 1. Let J be the number of base estimator groups. When W = 1, then J = N. If N = 10 and W = 2, then there are 5 groups of Adaboost estimators with two decision stumps each, and thus J = 5. If N = 10 and W = 3, then there would be 3 groups of adaboost estimators with 3 decisions stumps each and a remainder adaboost estimator with only 1 decision stump, so J = 4. Finally, when W = N, then J = 1.

Let sj  be a random sample (with replacement) of columns from the dataset S for group j. For a random forest (W = 1), the length of this vector (the number of columns selection, or the magnitude of the vector, ||sj ||) should be a fixed proportion (⍴) of the number of columns in the dataset, ||S|| and thus can be written in the form ||sj || = ⍴||S||. ||sj || should then be rounded to the nearest positive integer. For an adaboost algorithm (W = N), ⍴ = 1 as it uses the entire dataset, so ||sj || = ||S||. We can now find a function in which ⍴ steadily increases with W so that it starts off at the standard random forest value when W = 1 but then goes to 1 when W = N. This function can be introduced as k, which can be introduced to the exponent of ⍴, so:

||s_j|| = \rho^k||S|| (1)

When k = 0, then the term in front of ||S|| is 1, so the equation reverts to its adaboost state. When k = 1, the equation is in its random forest state.

The equation that governs k can be arbitrarily defined given some constraints. The domain of k is W ∈ {1, 2, …, N}, the output must vary between 0 and 1, and at W = 1 k must equal 1, and at W = N k must equal 0. The simplest possible continuous function that meets these requirements is given by the following line:

k = -\frac{1}{N-1}W + \frac{1}{N-1} (2)

With equations 1 and 2, we have now use W to set the proportion of features each group of base estimators is trained on. This proportion varies monotonically with W, and is anchored at both ends by a true random forest at one end and a true adaboost algorithm at the other.