Multiple learners

Introduction

The classical approach when working on a machine learning task is to train on a certain dataset and produce a certain output function. Tipically during training the performance in terms of accuracy get better over time, but we will reach a point where accuracy starts to increase at a really lower rate or even start oscillating. Since in practice we might have a limited amount of training time, betting on a single model doesn't feel like a good choice.

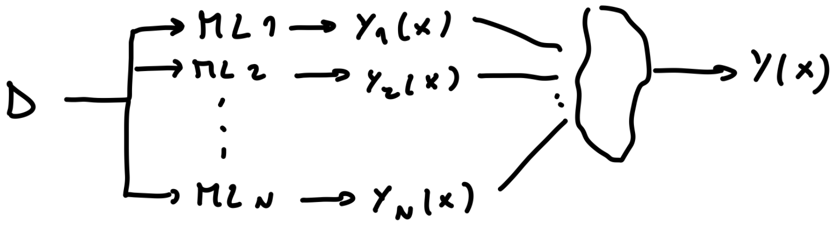
The idea behind multiple learners is to split the computation of training among several modules. This approach almost always leads to better results for the same amount of time with respect to training a single model.

There are three main approaches to multipple learning: voting, bagging and boosting.

Voting

Given a certain dataset D and a certain number of training models $ML_1, ..., ML_N$, we train the every model in parallel (independently) by feeding the same dataset as an input to each of them.

Each of the models will output a certain function $y_i(x)$. The final result will be an ensamble of the results of the different functions.

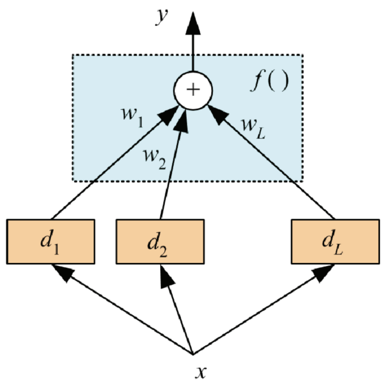


In more detail, when performing voting on **regression**, the result will be the sum of the **weighted average** of the results for each model:

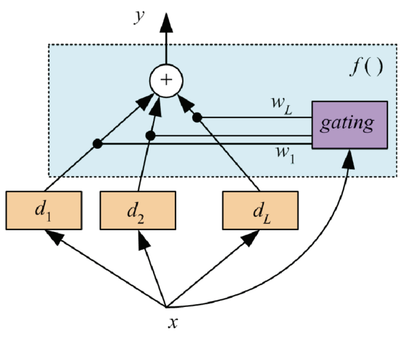
$$y_{voting}(x) = \sum_{m=1}^{M} w_m y_m(x)$$

When dealing with **classification**, the result will be based on the **weighted majority**. This means that we will assign different weights to each prediction, and then computer the argmax of the result:

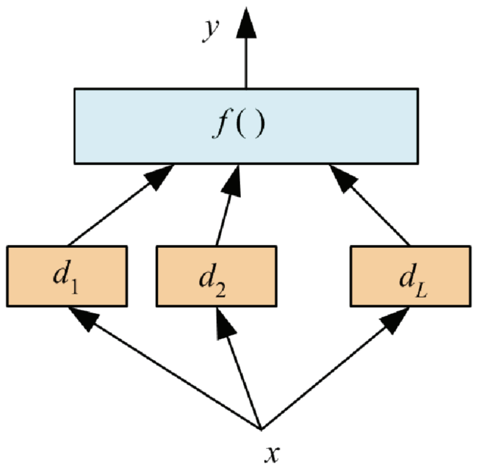
$$y_{voting}(x) = argmax_c \sum_{m=1}^{M} w_m I(y(x_m) = C)$$



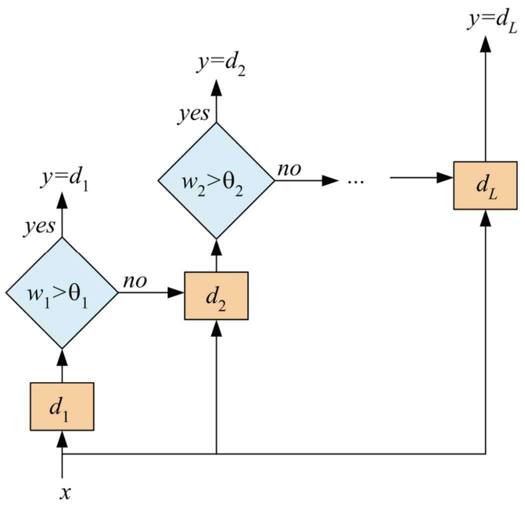
When implementing voting, a finer approach is to introduce **gating**. Basically, we adjust the weights of the voting function based on the kind of input by using a non-linear gating function:



The **scaling** approach is to treat the function that combinates the different outputs as a trainable set of parameters. The function will learn how to assign the proper weights to our outputs:



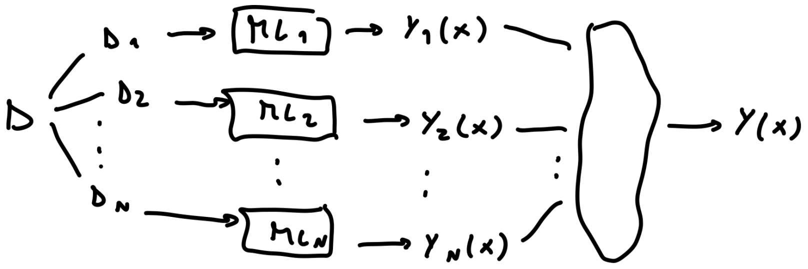
Finally, another approach is **cascading**. We set a certain threshold for confidence and subsequently evaluate the instance on a certain model, until we get an acceptable confidence value:



A general problem of the voting approach is that we are using the same dataset to train every model. This does not produce a good variability on the models obtained, meaning that me might risk of having bad performances for every mode.

Bagging

When implementing bagging, the idea is to pre-process the dataset in order to divide it in different subsets $D_1, ..., D_N$ and feed a different subset to each model:



This method introduces a better approach with respect to voting in terms of variance, but still doesn't implement any interaction between models during training.

Boosting

The main concept of boosting is to train each model on a subset of the entire training dataset, but make the training sequential such that each step will be influenced by the previous one.

Starting from a dataset $D = \{(x_n, y_n)_{n=1}^{N}\}$, the idea is to assign a weight $w_n^{(m)}$ to each of its samples (where m is the m-th step). At the beginning every weight will be initialized with the same value $w_n^{(1)} = \frac{1}{N}$.

The weights will regulate the importance of each sample during training. In particular the strategy is to evaluate the results and increse the weights when the error between prediction of the sample and the actual value is high, and lower their value when it's low. This will ensure that in the next step training will have produce better results.

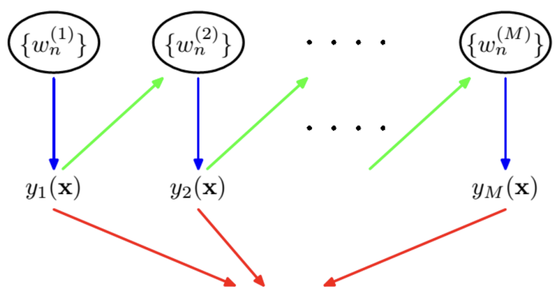
We implement boosting by adding the weights to the computation of the error. The formula for the error will be:

$$E(\theta) = w_n(y(x_n) - t_n)$$

This means that samples with higher weights will produce bigger errors.

A popular implementation of boosting is AdaBoost. Basically we compute the weighted error function, evaluate the results and update the weights accordingly to our strategy, before passing to the next step.

The final classifier will be an ensemble model weighted on the performances of each trained model:



$$Y_M(\mathbf{x}) = \operatorname{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right)$$