MACHINE LEARNING IDEAS

ANDREW MAURER

Abstract. This document is just a bunch of my ideas which lie outside the scope of `notes.pdf` but still may be worth writing down.

Contents

1. Social Choice in Random Forest Classifiers

1.1. Overview.

When using a random forest for classification, a number of trees are trained to predict the response variable and the class with the most trees is declared the winner. At its essence, trees are trained to be voters and plurality voting is applied. My question is: what happens if another voting system is applied? Two rank-choice voting systems of interest are:

(1) *Hare Voting:* In which the candidate(s) with the fewest first-place votes are eliminated sequentially.
(2) *Coomb's Rule:* In which the candidate(s) with the most last-place votes are eliminated sequentially.

By *Arrow's Impossibility Theorem* and *Gibbard-Satterthwaite Theorem*, there is no perfect voting system. Because of this, I am interested in whether using an alternative voting system that *really* cares about the rankings of voters will yield better results.

The only thing that makes me nervous is that the time taken for prediction will increase by up to a factor of the number of categorical classes to be predicted.

2. Sparse Bipartide Expander Graphs

This is something I chatted about with Daniel McKenzie at University of Georgia.

2.1. Overview.

When training a neural network, the dropout technique demonstrates that not every connection is valuable. Expander graphs remain extremely well-connected while containing far fewer edges than their fully connected counterparts. Of interest to us is the bipartide expander case, as an alternative to fully connected layers in neural networks. Neural network layers with m input variables (including bias) and n output variables will have $m \cdot n$ edges. Replacing this number of connections with something $o(m \cdot n)$ would yield much faster training while perhaps maintaining performance.

Date: March 3, 2019.

In implementation, I would build a `SBELayer` with n input neurons and m output neurons by randomly selecting $\lfloor\sqrt{n}\rfloor$ input neurons $\texttt{in}(1), \ldots, \texttt{in}(\lfloor\sqrt{n}\rfloor)$ for each output neuron `out`.

$$\texttt{out} = \sum_{i=1}^{\lfloor\sqrt{n}\rfloor} w_i \cdot \texttt{in}(i) \tag{2.1.1}$$

The alternative would be selecting $\lfloor\sqrt{m}\rfloor$ output neurons for each input neuron. This will ensure all inputs are equally represented in the output layer, but I believe it is more important to make sure each output neuron has an equal amount of information.

E-mail address: `andrew.b.maurer@gmail.com`

URL: `andrewmaurer.github.io`