Embedded Artificial Neural Network for Data Quality Assessment

Authors

Léo Vieira Peres

Motivation

I'm interested in the scenario where data is travelling around under the risk of being compromised by random noise and the task is to distinguish untrustworthy data (meaning those that might have been affected by noise) from the otherwise trustworthy ones. To tackle this problem I use machine learning. More specifically, I want to train an ANN (Artificial Neural Network) that'll be designed to run on an embedded system capable of aiding this system to decide which data to trust.

More specifically I'll try to answer the following questions:

1. What are some of the machine learning techniques that can be implemented by ANN work that maintain a good performance when noise is an issue?
2. Given that one is unable to obtain a good learner to a specific problem, are there good boosting techniques that can improve the performance of the weak learner?

Goals

The goal is to obtain some good enough insights on the 2 questions listed above before coding the machine learning algorithms I found to be the most interesting. Once I'm done with the coding, I'll train some ANNs and measure their performance before porting them onto EPOS.

One thing to keep in mind is that those ANNs should run on an embedded system.

Methodology

I'll begin by searching in the literature for those algorithms that fit the most the following criteria:

1. They should work well when we're modelling noise in our statistical analysis.
2. They're boosting algorithms.

Once I have a list of algorithms (ideally a number between 2 and 6 of them), I'll survey for each one of them whether their implementation is a feasible task. And then, once the theoretical part of this project is done, I'll focus exclusively in the implementation of the machine learning algorithms before finally using the FANN tool to train the ANNs that'll be ported onto EPOS.

Tasks

1. Write down the project planning
2. Read the FANN documentation, plan the statistical analyses and tests of performance to be made. Also, try to understand how to use the ANNs on EPOS.
3. Research and pick a list of algorithms to be coded.
4. Code each of the algorithms chosen.
5. Use each algorithm to train an ANN to be ported onto EPOS.
6. Run the performance tests.

Deliverables

1. The project planning.
2. Proof of technological viability.
3. A text describing the algorithms I chose.
4. Source code of the algorithms coded.
5. The ANN ported onto EPOS.
6. A report on the tests results and final conclusions.

Schedule

Task	25/09	2/10	9/10	16/10	23/10	30/10	6/11	13/11	20/11	27/11
Task1	D1									
Task2		x	D2							
Task3		x	x	x	D3					
Task4			x	x	x	x	D4			
Task5					x	x	x	x	D5	
Task6					x	x	x	x	x	D6

The PAC learning model

Initially, I'll assume the PAC learning (PAC stands for Probably Approximately Correct) introduced by Leslie Valiant in 1984. The basic idea behind the PAC learning model is as follows: you have an unknown function c mapping a set X to {-1. 1} (which alternatively we can call it a "concept") that belongs to a class of concepts C as a well a set S of examples (x_1, y_1), ..., (x_m, y_m) where each x_i is in X, y_i is in {-1, 1} and $c(x_i) = y_i$. The goal is to obtain a hypothesis h that approximates c. That is, you want to find in polynomial time and with very high probability another function that agrees with c in almost every input from the set X. Now you're probably able to understand why it's called the Probably Approximately Correct learning, since with very high probability (Probably) you get a function that is a very good approximator (Approximately Correct) to your unknown function c.

To model noise in our training data S we could assume the pairs of input and output are taken from a distribution that randomly flip the output bit.

Artificial Neural Networks

Artificial neural networks (ANN) can be seen as DAGs (Directed Acyclic Graphs) where the sources are called the input nodes, the (unique) sink is called the output node and all other nodes are called the hidden nodes. Each node computes a weighted linear threshold function which are functions of the form $sgn(a_1x_1 + ... + a_nx_n - t)$, where a_1, ..., a_n and t are constants. The inputs of an ANN are the values being fed into its input nodes while its output is the value computed by its output node. The topology of an ANN is the arrangement of its nodes and threshold constants that is independent of the weights in each node.

To train an ANN to approximate a function we want to fix the right values of the weights in each node in such a way that makes the output of the ANN and the function to be as correlated as possible.

Boosting

Suppose you're in a situation where you want a good approximation to your unknown function c but all you're able to obtain are extremely weak (i.e. just slightly better than randomly guessing) classifiers. The

goal of boosting is to fine-tune the use of the weak classifiers to obtain a master hypothesis that can correctly compute the values of an example taking from some distribution with probability really close to 1.

As an example here is a rough description of the AdaBoost algorithm:

1. Let (x_1, y_1), ... , (x_m, y_m) be pairs of input/output.
2. Initially, D_1 is the uniform distribution over the m pairs of input/output.
3. For i = 1, ..., T:
4. Train weak learner h_i using distribution D_i.
5. Let e_i be $Pr[h_i(x_k) != y_k]$ with respect to the distribuition D_i.
6. Choose $a_i = 1/2 * \ln(1 - e_i / e_i)$
7. For each pair (x_k, y_k) set $D_{i+1}(j) = D_i(k)*\exp(-a_i y_k h_i(x_k)) / Z_i$ (Z_i is chosen in such way that the probabilities sum to 1).
8. Output the final hypothesis $H(x) = \text{sgn}(a_1 h_i(x) + ... + a_T h_T(x))$.

By looking at the algorithm we see that the output is (the sign of) a weighted majority function. In the inner loop, we see that the weight a_i of the i-th weak hypothesis is the logarithm of a function that goes to infinity as the error e_i gets closer to 0, and goes to 1 as the same error approaches 1/2 (we can assume the error is at most this value). In short, the largest weights are given to the best approximators.

Also, as we update the weights of each pair of input/output we give a higher weight to those pairs for which the i-th weak hypothesis fails to compute the right value (we're considering outputs in {-1, 1}).

Bibliography

1. http://web.mit.edu/~sinhaa/Neural_Net.pdf (covers the basics of PAC learning, artificial neural networks and deep neural networks)