# Neural Net Classification Algorithm

Alexander Heckett

September 7, 2015

## Abstract

This paper provides a relatively in-depth look at the algorithm used behind my neural net classifier. It does not cover the background corresponding to, the progression of ideas that led to, or any results coming from the program implemented.

## Contents

## 1 The pre-computation

### 1.1 The Neural Net

#### 1.1.1 Simulated Data

#### 1.1.2 Training Scheme

### 1.2 Boosting

#### 1.2.1 BrownBoost

#### 1.2.2 LogitBoost

## 2 Per-dataset computation

### 2.1 Evaluation and Error

Having constructed our monster classifier, we now brace to use it. We have already laid out the general principle behind how it is to be used. For some starting position, look forward some fixed number of points into the future, where each point is separated by a constant number of frames, find the error bar at these points, and pass them into the neural net. We will thus generate a sequence of classifications, starting at every frame in the dataset except for the small slice of the set which would require looking at frames past the end.

It is now our task to turn these individual classifications into one final conclusion.

However, this is not trivial. In an ideal world in which the master classifier is perfect, we would simply look at every classification, and if a star comes up with an exoplanet, record our identification. However, due to the many times we call our classifier, many misclassifications will occur. With this method, practically every star will come through as having an exoplanet, since at least one frame will be incorrectly identified as a false positive. Thus, we must contrive some algorithm to weed out the error.

Before proceeding to eliminate the error, we must define some expectation of how our error is going to behave. This model follows from two very basic propositions which I am now going to take as axioms:

1. The likelihood of one star to be misclassified is not dependent on another star's likihood.

2. The stars are all treated the same way by the master classifier.

These two propositions form the basis of my theory. Due to these two axioms, the probability that a truly exoplanet free star comes out incorrectly positive is some constant number - say $\alpha$. The probability that a star with an exoplanet comes out incorrectly negative is some other constant number - say $\beta$. We will use these two quantities to "decode" our noisy output into one final result.

## 2.2 The Decoder

We can finally start tackling the decoding process. Throughout the course of the recording, the most common classification will change. For instance, a star with an exoplanet experiencing a transit will come out positive for a region around the transit, but in the parts of the dataset where there are no transits, the most common classification will show nothing. Thus, we must find some way of partitioning up the dataset into similarly classified chunks. I'm again going to call these chunks "bins" or "blocks" just like in Dynamic Blocking. I assume that the master classifier has been given data similar enough that classifications within a bin will be *homogeneous*, i.e. the only variation in classification is due to random fluctuation in the input, and we can assume that the actually correct classification will be the most common one amid all the identifications in the bin. We thus wish to break our data up so as to minimize the lack of homogeneity.

Just as we did with Dynamic Blocking, I'm going to go hunting for a solution involving Dynamic Programming. The general trick with Dynamic Programming is to find some metric of the "badness" of a partitioning as a function of the "badness" of the left most block plus, times, or any other function of the "badnesses" of the other blocks put together. In this situation, a fitting definition is that one partitioning is worse if we are less certain that it comes from a series of homogeneous blocks. How can we find the probability that a partitioning comes from a series of homogeneous blocks?

Let's focus on one block. I define:

$$\mathfrak{F} = \{(f_0, f_1, \ldots, f_N) \,|\, f_i \in \{0, 1\}\} \qquad (1)$$

to be the set of all possible identifications when tracking $N$ stars. I am examining some block of classifications, $(A_1, A_2, A_3, ..., A_n)\,, A_i \in \mathfrak{F}$, and I believe for some reason that the true classification the master classifier was trying to output is $X$. What is the probability that, given $X$ is the true classification, the program would have generated the output $(A_1, A_2, ..., A_n)$? Let's say there are $a$ stars both classified as positive in $X$ and $A_i$, $b$ stars newly classified as positive in $A_i$, $c$ stars newly classified as negative in $A_i$, and $d = N - a - b - c$ stars both classified as negative in $X$ and $A_i$. The probability that $A_i$ would be generated by a homogeneous block with true value $X$ is then:

$$P_{X \to A_i} = (1 - \beta)^a \alpha^b \beta^c (1 - \alpha)^d \qquad (2)$$

If we take the product of these probabilities for each $A_i$, we get the probability that such a string would originate from homogeneous $X$.

An algorithm is finally beginning to appear. For a given bin, we look at every possible classification $X \in \mathfrak{F}$ and compute $\prod_{i=0}^{n} P_{X \to A_i}$. Our "badness" metric is then:

$$b_j = -\max_{X \in \mathfrak{F}} \left[ \prod_{i=0}^{n} P_{X \to A_i} \right] \qquad (3)$$

The "max" finds the classification $X$ which maximizes the probability that $X$ created the homogeneous block $(A_1, ..., A_n)$. The $j$ subscript on the $b$ signifies that we are working with the $j$th block. The minus sign is present because the quantity following it is something we want to be as *large* as possible. Thus, we can switch all our logic with badness metrics to be the same logic, just with "goodness metrics":

$$g_j = \max_{X \in \mathfrak{F}} \left[ \prod_{i=0}^{n} P_{X \to A_i} \right] \qquad (4)$$

## 2.3 Finding Alpha and Beta

Up until now, I have assumed we somehow knew $\epsilon$. However, there is no obvious way to predict this from theory. We thus must resort to using the dataset to find the most probable value of it. This is similar in concept to finding $\sigma$ in my Dynamic Binning algorithm, just harder to implement.

The program has been tasked with the problem of, given the classifications in an interval, find the best $\epsilon$ to work with. Let's start by working this problem in reverse: if we knew $\epsilon$, what would be the distribution of classifications? At this point, we must define some more terminology before we can proceed.

## 2.4 The Decoder Revisited