# Introduction

All throughout biology there are differences among organisms. Does the organism have a certain color of hair? Did the organism get a certain type of disease? Did the organism have a specific defect—6 fingers per hand, missing a limb at birth, etc.? These trait differences can be considered the “class” of the organism and whatever attributes of the organism determine it’s class (DNA sequence, phenotypic characteristics, etc.) can be considered the “features” of the organism. This problem formulation naturally lends itself to classification techniques from machine learning. The general structure of a machine learning model is to learn a function that maps an input X to an output y. In this context, we therefore have organism features X and the type of condition present in the organism y.

This project is based on [1]. In this paper, the authors present a method called Deep Feature Selection (DFS), where a neural network is used for a classification but an additional layer is used for feature analysis. After analysis of this paper we believe that these techniques could apply to a broader range of classification and even regression tasks than previously evaluated. We evaluate the methods of the paper, use the same technique on new data sets and evaluate the results obtained.

The remainder of this paper is organized as follows. We introduce the topic of machine learning classification in biological data sets. We introduce the concepts of feature selection and the importance of regularization. We briefly introduce the concepts of neural network analysis. We demonstrate the alterations necessary to transfer a feedforward neural network (FFNN) to a DFS model. We prove an property associated with the regularization parameters in the DFS architecture. Finally, we present the results we obtained from this study’s data set as well as a set of alternative data sets to determine the quality of the feature selection technique.

# Classification in Biological Data Sets

(we should really jazz this up with a lot of resources)

When referring to classification in a biological sense, we mean taking information about an organism and attempting to determine whether an attribute will end up manifesting in the organism. For instance, just looking at the number of chromosomes of a human being you can deterministically say whether or not that human has Down’s Syndrome [2]. In this case, it’s obvious that the classifier would behave as follows:

Figure 1: Pseudo Code for Down’s Syndrome Classification

If number\_of\_chromosomes == 47

Down\_syndrome = true

If number\_of\_chromosomes == 46

Down\_syndrome = false

Else

Raise error

In this case the feature set is just “number\_of\_chromosomes” and the class is the presence or absence of Down’s Syndrome.

The way that classifiers such as these are created is by exposing a general model to a data set and training it to return a class with the greatest accuracy measure. To give an example of how this would work, let’s take a closer look at a common classification technique called logistic regression.

In logistic regression, we are simply trying to determine the probability of two classes based on the inputs (it can be extended to multiple classes but let’s restrict our focus to two for clarity). The model itself is constructed as follows:

1/(1 + e^-(WX + b)) = sigmoid(WX + b)

Where W is a vector of weights, X is the set of input features and b is a bias term. This function is called a sigmoid function. Let’s train a logistic regression model to learn our Down’s Syndrome example. An artificial data set for this example is presented in Figure 2.

Figure 2: Artificial Data Set for Down’s Syndrome



Our model only has two parameters in this case W1 (a multiplier for the number of chromosomes) and b a bias term. X also only has one element X1 which is the number of chromosomes. So the entire model is:

1/(1 + e^-(W1 \* Number\_Of\_Chromosomes + b))

Our task is to dial in W1 and b such that the above function outputs a number close to 1 when the number of chromosomes is 47 and a number close to 0 when the number of chromosomes is 46 for the above data set. To accomplish this we need to define a loss function—that is a measure of how poorly the model is performing. Then we need to manipulate W1 and b in such a way that the loss function is minimized. Typically the loss function in logistic regression is cross entropy which works as follows[3]:

If the actual label is positive (i.e. exhibits Down’s Syndrome) then

-log(sigmoid(WX + b))

Else

-log(1 – sigmoid(WX + b))

Thus the objective function across all data points becomes:

These parameters are tuned using an optimization algorithm such as stochastic gradient descent. We applied this algorithm to the above data and model. After optimization (and constraining all terms to +/-1000), the parameters found for the above problem are W1 = 21.5 and b = -1000. With this model the probability of Down’s Syndrome given 47 chromosomes can be calculated as:

1/(1 + e^-(47 \* -21.5 + 1000) ) = 0.99997

And the probability of having Down’s Syndrome given that the subject has 46 chromosomes is:

1/(1 + e^-(46 \* -21.5 + 1000) ) = 1.7 \* 10 ^ -5

Any time the probability is above 0.5, the True class is outputted and any time it is below 0.5 the False class is outputted. So, this model is exactly equivalent to the pseudo code presented in Figure 1.

To provide a little bit more insight into the behavior of this model, a graph of the probability of Down’s Syndrome in response to chromosome count using our logistic regression model is presented in Figure 3. Even though chromosome count is clearly an integer, this shows the tradeoff as the number of chromosomes transitions from 46 to 47.

Figure 3

# Feature Selection and the Importance of Regularization

In the previous section we developed a logistic regression model that perfectly classified the existence of down’s syndrome based on the number of chromosomes in the subject. This was clearly an engineered example since we had all of the information we needed and nothing else before we even started. Real world data sets are a lot messier and include information that has nothing to do with the problem. But how do you identify this?

To illustrate, let’s make the Down’s Syndrome data set noisier by adding some additional, superfluous features (Figure 4).

Figure 4: Down’s Syndrome Data Set with Additional, Unimportant Features



Now, we already know that the hypothesis sigmoid( -21.5 \* number\_of\_chromosomes + 1000) does a perfect job of classification. What about sigmoid(12,000 \* age – 1000 \* age\_months – 21.5 number\_of\_chromosomes + 1000)? Since, the 12,000 \* age cancels with the -1000 \* age\_months this also does a perfect job! Worse yet, the weight associated with age is 12,000 and the weight associated with age\_months is -1,000. The intuition that you obtain from looking at the weights is that age is around 60 times as important as the number of chromosomes! This is where the concept of feature selection comes in. We need a systematic method to look at the above model and ensure that only the feature number\_of\_chromosomes is selected.

So, let’s perform a technique called regularization. The goal of regularization is to dissuade your model from learning a highly complicated hypothesis. In this case we want to discourage the model from assigning weights of 12,000 and -1,000 to the features age and age\_months. To do this we just need to alter the loss function of the model by adding a term associated with the magnitude of the weights to it. More formally the new loss function can be defined as follows:

Where loss is the original cost function, || is the absolute value operator, is a tradeoff parameter and is the kth term in the weight vector W. This minimizes both the error of the model and the absolute value of the model’s terms. With a value of just 0.01 we obtain the following weights:

= -3.3 \* 10 ^6

= -4.6 \* 10^-6

= 6.6

= -308.6

Now is 2,000,000 times larger than . And thus we can conclude that number of chromosomes is selected by the model and neither age nor age\_months is selected by the model.

It is also worth noting that regularized models tend to be less dramatic. Figure 5 is the same tradeoff example as Figure 3 but for the regularized model. Note that the slope is not as steep as the model transitions from 0 to 1 and only climbs to about 95% confident as the chromosome count tends from 46 to 47.

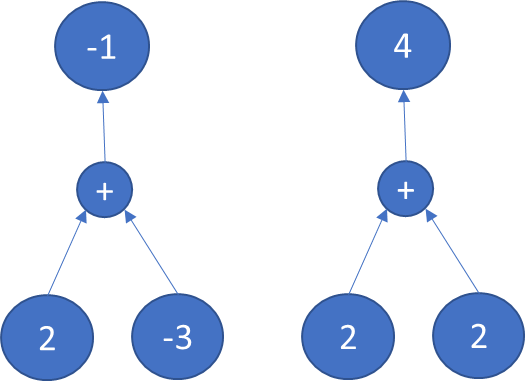
Figure 5

This provides the general framework for a feature selection model. Assign a weight directly to each feature, then minimize both the loss of the model and the weights. What will be returned is a high quality model and a weight vector that reasonably signifies the importance of each feature in the classification task.

# A Brief Introduction to Neural Networks

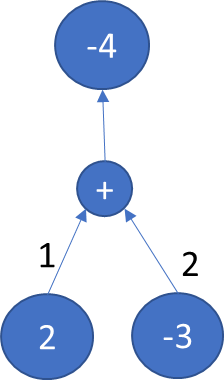
In this section we cover only a simple form of neural networks called feedforward neural networks (FFNN). To introduce the concept let’s look at another concept called a computational graph. Computational graphs work by having numbers and operations. So for instance a “+” node sums all of the nodes feeding into it and outputs that sum. Examples for “2+2 = 4” and “2 + -3 = 1” are provided in Figure 6.

Figure 6: Computational Graph Examples



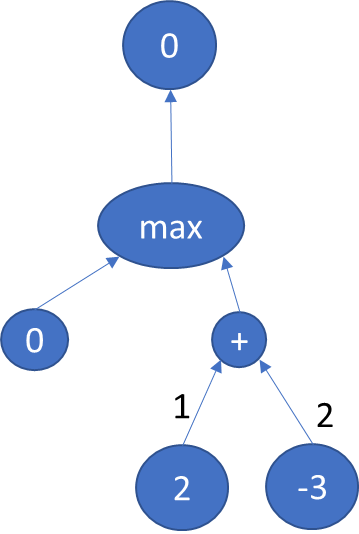
Typically, rather than straight addition there would be a weight term associated with each link in the graph. The weight is multiplied by the result of the previous node and fed into the next node. An example of a weighted computational graph for the equation “2 \* 1 + -3 \* 2 = -4” is provided in Figure 7.

Figure 7



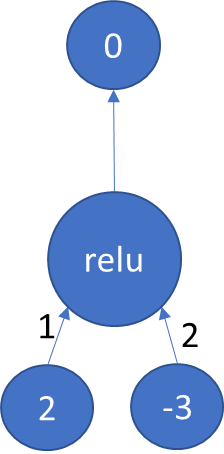
We’re almost there, we just need to make this a non-linear function to transform it to an actually a neural network. The most commonly used non-linear function is to simply check if the output is negative and output 0 if it is, otherwise just pass along the output. Adjusting the figure from Figure 7 to output a 0 signal is shown in figure 8.

Figure 8



This summing and maxing process is condensed into a neural network node called a rectified linear unit or “relu.” So figure 8 can equivalently be expressed as Figure 9.

Figure 9



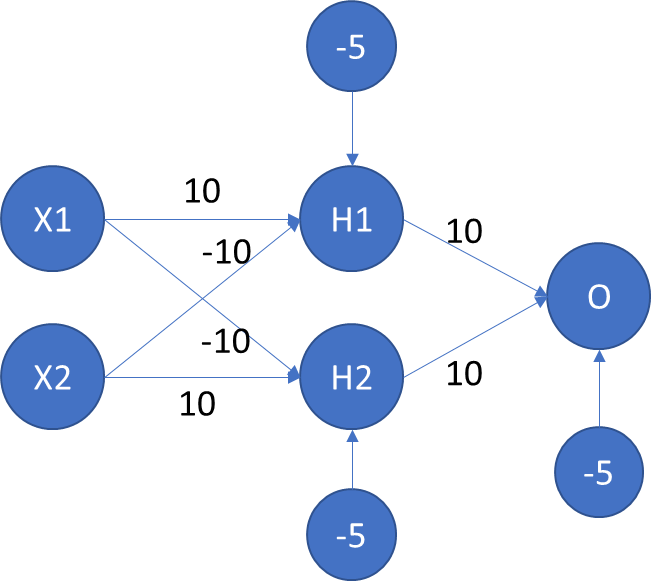
There are many other activation functions and experimentation or research is required to determine which fits your problem appropriately. Other examples of activation functions include sigmoid (the same function used in logistic regression), hyperbolic tangent, exponential linear unit and many others.

An example of a more complex neural network with **sigmoid** activation functions is provided in Figure 11 [5]. This neural network is an example of implementing the XOR function. The truth table for XOR is provided in Figure 10.

Figure 10

|  |  |  |
| --- | --- | --- |
| X1 | X2 | Truth Value |
| 0 | 0 | False |
| 0 | 1 | True |
| 1 | 0 | True |
| 1 | 1 | False |

Figure 11



Walking through the computation in figure 11, if x1 and x2 are both zero or both one, then the hidden nodes, H0 and H1 receive a signal of 0 from the input layer. Then the bias term of -5 is added to each. So the output of both H0 and H1 are very small since sigmoid(-5) is near 0. The small output from each is multiplied by 10 then summed then 5 is subtracted. Again this results in a very small signal. Thus the output would be less than 0.5 and the label would be 0 (or False).

Conversely, suppose X1 is 1 and X2 is 1. Then H1 will receive a signal of 10, H2 will receive a signal of -10. This will cause the output of H1 to be near 1 and the output of H2 to be near 0 (again since sigmoid(5) ~= 1 and sigmoid(-5) ~= 0). The high output from H1 and the near zero output from H2 will be multiplied by 10 and summed and 5 will be subtracted. Thus the output from O will be roughly sigmoid(5) which is near 1 and the label would be 1 (or True) if only one of X1 and X2 is 1.

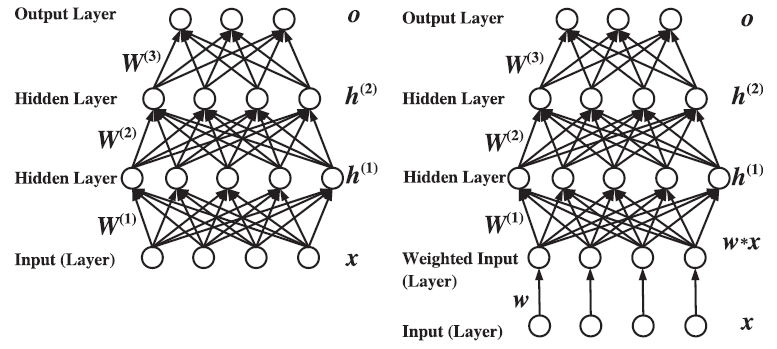
Training for neural networks works similarly to logistic regression and involves the use of a solver such as stochastic gradient descent or adagrad.

This section only described the simple case of feedforward neural networks. There are other types that are used for other applications. Many of them have features in common but this version is probably the easiest to conceptualize and the one used for the this paper.

# Transforming Feedforward Neural Networks to Deep Feature Selection

Neural networks are notorious for being a “black box.” Once the weights are learned it is often quite difficult to determine why it is outputting the correct label even though it is. From the example provided in Figure 11, I’m sure it wasn’t immediately obvious at first glance what the network was computing for instance. Realistically neural net works can have several layers of nodes each with 1000+ nodes. They are fully connected to one another so there end up being literally millions of weights and bias terms in some cases. So we need a method to extract the features that the neural network used. This model is presented in [1]. The concept is actually pleasantly simple—take a neural network and simply interrupt the input layer by multiplying by a scalar. This allows the training to learn a weight that only applies to one feature. A pictorial representation is provided in Figures 12 A and B. Figure 12A is a standard feed forward neural net, figure b has the input multiplied by a scalar to produce a DFS model.

A B



Resources

1. (main paper)
2. <https://www.ndss.org/about-down-syndrome/down-syndrome/>
3. <https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html>
4. ML Book
5. <https://gist.github.com/stewartpark/187895beb89f0a1b3a54>