Implementation Report - NFL Play Prediction Using a Neuro-Genetic Hybrid Model

This document demonstrates the implementation of a Neuro-Genetic hybrid model to predict the success or failure of an NFL Play. The code along with the dataset can be found in our Github Repository (https://github.com/madil27/NFLpredictions).

Dataset

The dataset used contains information on every NFL play that has been run since 2009. It's hosted on Kaggle:

https://www.kaggle.com/maxhorowitz/nflplaybyplay2009to2016

The information on each attribute in the dataset can be located at this link:

https://github.com/maksimhorowitz/nflscrapR/blob/master/R/scrape_play_by_play.R

The step-by-step implementation is explained in subsequent parts of the document.

Data Cleaning

The following steps are implemeneted for cleaning the dataset:

1. Only playtypes "run" and "pass" are considered. Special playtypes such as goals, punts, sacks, fumbles etc are of no interest to us.
2. We only consider those data points in which the value of down is 1,2,3 or 4.
3. We replace NaNs with 'unknown'.

Feature Selection

There is a lot of information in this data set (255 columns).Apart from the actual play the dataset describes injuries, timeouts, quarterback substitutions, penalties, and other information. Since we are only interested in the actual plays we need to filter out all the irrelevant information. The 12 features we ended up using for our classification task are:

Features	Type	Description
posteam	Categorical	Team in possession
defteam	Categorical	Team in defense
game_half	Categorical	Current Half (first, second, overtime)
half_seconds_remaining	Numerical	Seconds remaining in half end
yardline_100	Numerical	Distance to the 100 yardline
down	Categorical	Current down in play (1st,2nd,3rd,4th)
ydstogo	Numerical	Yards to touchdown/firstdown in the play
shotgun	Binary	Binary indicator for whether or not the play was in shotgun
play_type	Categorical	String indicating the type of play
pass_location	Categorical	Location of pass in the ground(left,right)
run_location	Categorical	Location of run in the ground(left,right)
pass_length	Categorical	Length of pass(small,long)

These features best describe the relevant game situation and the play being used by the team in offence.

Targets

Apart from the features we also have to extract the ground truth labels of each play. The possible target variables include:

Label	Type	Description
Success	Binary	Indicates whether a play resulted in a first down or a touchdown.
Yards Gained	Continuous	Indicates how many yards a team has gained through this play.

When we use the binary value success we define successful plays as plays which either obtain a first down or score a touchdown. In all other cases the play will be classified as failure. In this phase of the project, we make use of just the first target variable only i.e “Success”. We plan on predicting the yards gained in further phases of the project.

Data Encoding

Some of our features have categorical values. For example either of the features T eam and Opponent take on a value representing one of the 32 teams. Simply numbering the teams from 1 to 32 would not represent the real situation because team number 1 is not closer to team number 2 than it is to team number 32. Consequently, we need a more sophisticated encoding. For that reason, we have chosen one-hot encoding for our categorical features. That is, each categorical feature is replaced by k binary features where k is the number of possible values.

Encoding all our categorical features of our data set expands the size from 12 to 98 dimensions. Moreover, we standardize features such as yardline100 and half seconds remaining between 0 to 1.

Neuro-Genetic Model

GA creates multiple solutions to a given problem and evolves them through a number of generations. Each solution holds all parameters that might help to enhance the results. For NN, weights in all layers help achieve high accuracy. Thus, a single solution in GA will contain all weights in the NN.

Fitness Function

GA uses a fitness function to returns a fitness value for each solution. The higher the fitness value the better the solution. The best solutions are returned as parents in the parents selection step.

One of the common fitness functions for a classifier such as NN is the accuracy. It is the ratio between the correctly classified samples and the total number of samples.In this implementation, we use NN's accuracy as a measure of the fitness function for the genetic algorithm.

Implementation

Our implementation of the algorithm is as follows:

1. It first reads the features and the target labels as numpy ndarrays.
2. It then creates the NN architecture, using the number of hidden layer and output layer neurons as arguments.
3. It then generates the initial solutions(weights for the NN) randomly.
4. It then loops through a number of generations by calculating the fitness values for all solutions.
5. It then selects the best parents, applies crossover and mutation, and finally creates the new population.
6. The step 4 and 5 are repeated until we either get the desired accuracy or we iterate through all the generation.

Iteration vs Fitness Plot

Based on 1,000 generations, a plot is created at the end of this file using Matplotlib visualization library that shows how the accuracy changes across each generation. It is shown in the next figure. After 1,000 iterations, the accuracy is more than 75%.We can increase this by increasing the number of generations and by using the entire dataset. As of now, we are using the subset of the dataset, as its taking a lot of time on the complete dataset and we plan on running it on a cluster.

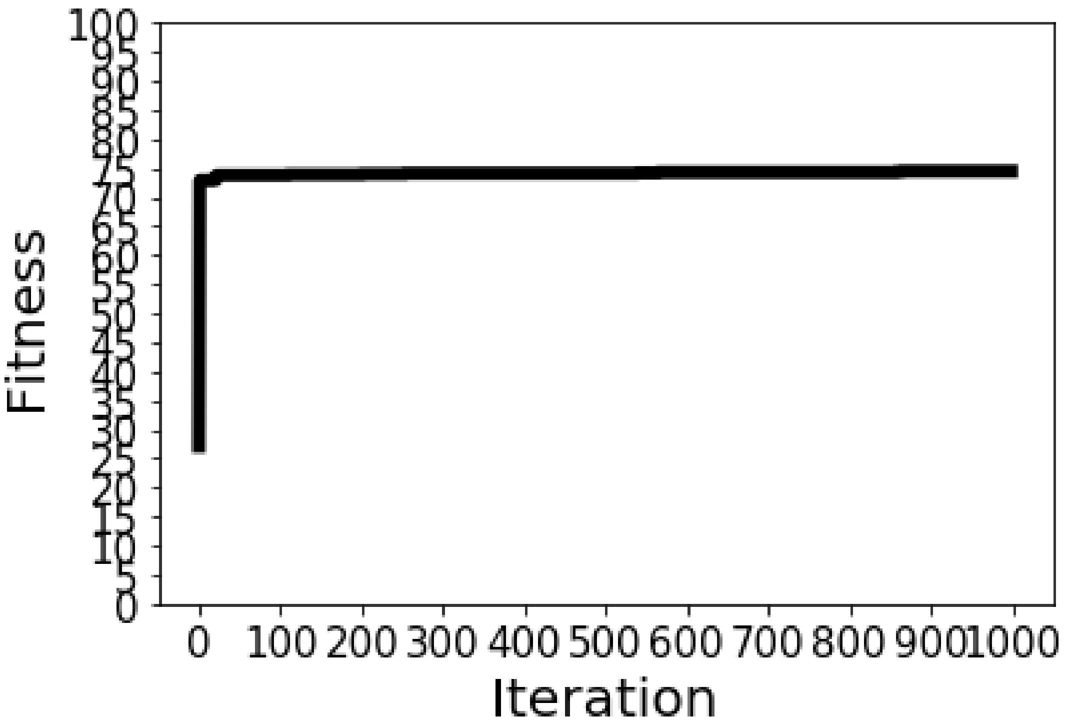

Figure 1: Iterations vs Fitness Plot