Using Machine Learning For DOTA2 Match Prediction

Andrius Buinovskij

March 20, 2017

1 Dictionary

There are some more obscure terms littered throughout the text. Here is a short explanation for each one.

- A **feature** is some aspect of something. For instance, a coin may have a mass, an area, an average colour and so on. Each of those are a feature.
- **Classification** is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.[1]

[1]Wikipedia.

2 Introduction

This text describes most of contemporary machine learning techniques, and then shows their application in match outcome prediction in a game of Dota 2, namely, which team will win, based on players' hero selection. In other words, it is classifying team composition as preferring a radiant win, or preferring a dire win.

The game in question, Defence of the Ancients (DOTA) 2, is very nuanced and has a plethora of mechanics. For the purposes of this paper however, it will suffice to say that in a game of DOTA, two teams, "Dire" and "Radiant"[2], five players per team, try to destroy the other team's base. At the beginning of a game, each player chooses a "hero" to play, a character within the game. Different heroes have different abilities, and some heroes synergize, whilst others cancel each other out. As of writing of this text, there are 113 different heroes, resulting in a bewildering number of possible team compositions.

3 Literature Review

The machine learning techniques used are: Nearest Neighbours, Neural Networks, Support Vector Machines and Boosting.

[2]Dire being evil, Radiant being good.

3.1 Nearest Neighbours

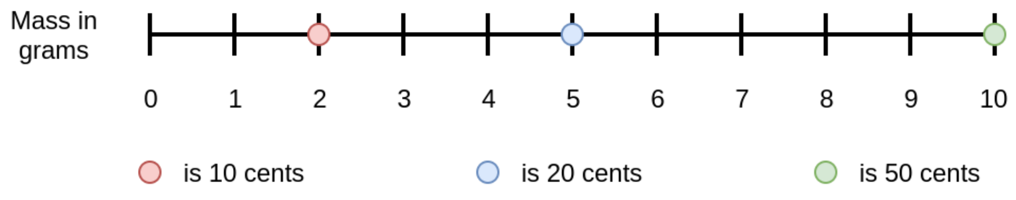
3.1.1 Nearest Neighbours Logic

The simple logic behind the nearest neighbours approach to machine learning is: when asked to do something, one should see if the scenario in question has ocurred before, and what was done then.

Let us say that we are trying to build an algorithm which will be used as a part of vending machines, namely, deciding on the value of coins that are inserted into the machine. Let us also say that the value of the coin will be decided based solely on it's mass (for simplicity), and that we have a database of standard coins already weighed.

A coin is inserted into the vending machine, how can we tell it's precise value? All we have to go by is the coin's mass. Currently, our database of previous cases is illustrated below.

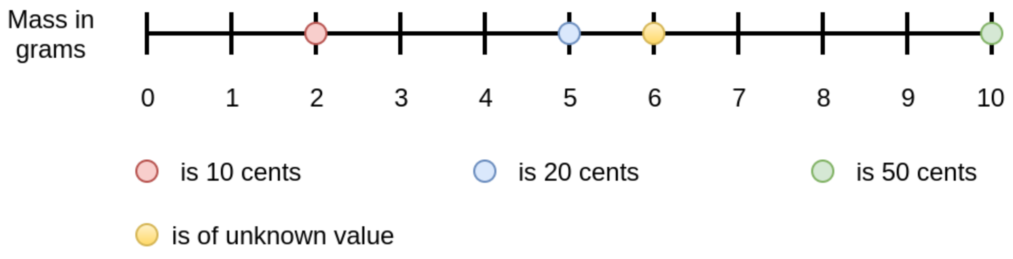
Figure 1: A visualization of coins placed on mass axis



So we have three coins of mass of 2, 5 and 10 grams. Let us say a coin is inserted into the machine, of mass of 6 grams. Here it is on our graph.

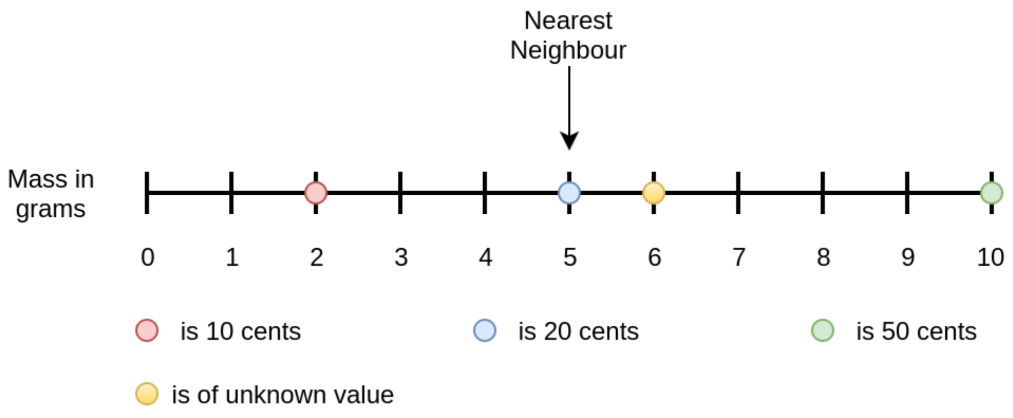
We would like to decipher the value of the newly inserted coin. An immediatelly obvious approach is to query our databse of already known mass-value pairs of coins to find a coin that most closely resembles the new coin, and say that the two are the same. Within the graph, this is simply looking

Figure 2: Coin graph including mismatching coin



for the nearest neighbouring coin[3] Here it is again on our graph.

Figure 3: The nearest neighbour to the unknown coin



Having found the nearest neighbour, the 20 cent coin, we conclude that the new coin must also be a 20 cent coin, perhaps with some imperfections.

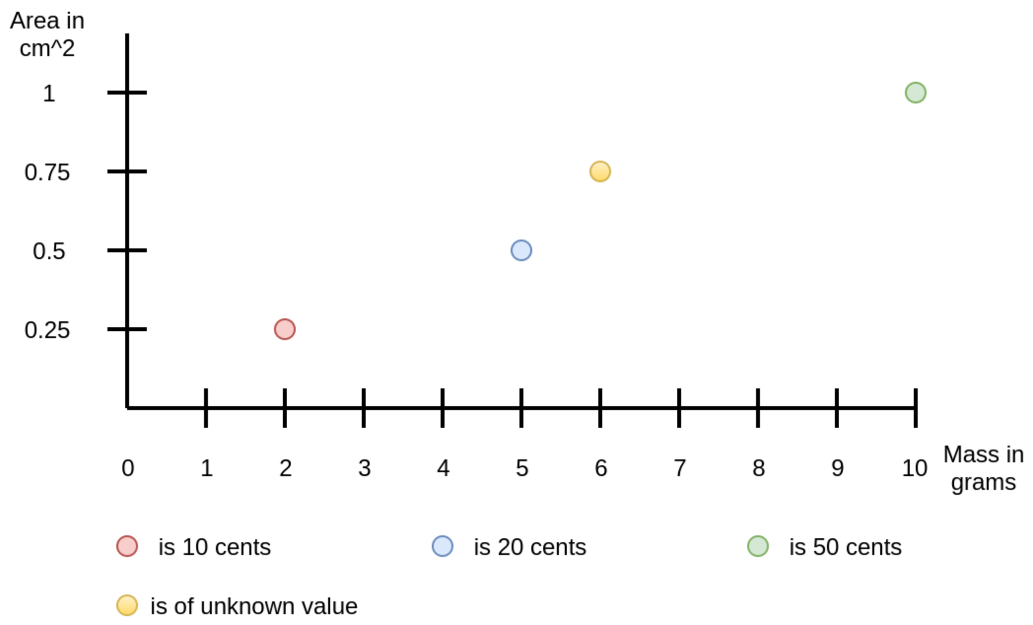
This short example illustrates the benefits and shortcomings of Nearest Neighbours. The method is simple and logical, and directly uses experience of the past. However, it is not extracting underlying patterns out from underneath the data, but is instead simply[4] looking to the past for similar cases.

[3]Hence the name of the method, Nearest Neighbours.
[4]Not for long.

3.2 Adding complexity and kD Trees

Our coin example was quite simple, using a single feature. A single feature gives rise to a single axis, however, we could have used more. For instance, we could have used the coin's area, which would have looked like the following:

Figure 4: A visualization of coins places on area and mass axes



The search for a nearest neighbour is now a two-dimensional one, which we can still handle intuitively. However, as the number of features, and consequently axes, grows, the question of how does one find nearest neighbours arises.

There are a number of algorithms