Università degli Studi di Ferrara

AN APPLICATION OF MACHINE LEARNING TO FOOTBALL BETTING

BACHELOR THESIS IN INFORMATION ENGINEERING

FRANCESCO MARIA TRASFORINI

SUPERVISOR: PROFESSOR GIACOMO DIMARCO

ACADEMIC YEAR: 2018/2019

Abstract: An Application of Machine Learning to Football Betting

The goal of this thesis is to train a neural network to predict football match results. The network will be trained via supervised learning based on historical odds. The purpose is to verify if the network predictions can beat the bookmakers.

The idea kickstarting this document is that betting markets could be inefficient, from a bettor standpoint. The ground on which I base this thesis is that bookmakers and bettors are playing different games, meaning they are not actually competing with one another. The gambler aims to predict outcomes, while the bookie aims to predict what the public perceives as a reasonable outcome. While these two objectives are related, they are not the same.

The thesis starts with some anecdotes about science in gambling and sports. It follows an introduction on bookmaking. Some machine learning concepts are then described: McCulloch-Pitts Neurons, the Perceptron, multilayered neuronal network, bias/variance tradeoff, cross-validation and supervised learning.

The last part of the thesis displays a classification experiment developed on the Anaconda platform with Python and the Support Vector Machine algorithm[1], which works by mapping the data in a higher space and finding a linear divider there. The goal of the experiment is to predict the results of football matches and make a profit on the betting market. The algorithm correctly categorizes 1314 out of 2893 (45.42%) matches played between August and December 2019. The average odd of those 1314 matches is 2.27. Betting 100 times using a fixed size bet, the return is $(fixed\ bet\ size * 0.4542 * 2.27) - (fixed\ bet\ size * 0.5458 * 0)$. The first term refers to

[1] The SVM algorithm was developed by Vladimir N. Vapnik, Alexey Ya. Chervonenkis and A.Lerner in 1963-1964 in the framework of the "Generalised Portrait Method" for computer learning and pattern recognition (Vapnik, 1963) and "A note on one class of Perceptrons" (A. Y. Chervonenkis, 1964).

the return for wins and the second one to the return for losses. For example, if for any given bet the stake is 1€: $(1€ * 0.4542 * 2.27) - (1€ * 0.5458) \cong 1.03$. Thus, the yield is 3%.

To maximize the return, the Kelly Criterion is applied for bet sizing. The Kelly Criterion is a formula used by gamblers to determine what percentage of capital should be used in each bet to maximize long term growth. A random process of 1000 random variables, where each variable starts from 1000€ and represents a hypothetical fluctuation of a gambler bankroll over 3000 bets, is used to test the profitability of the classification. Each bet is assumed to have a probability of success of 45.42% and a 2.27 average odd as previously calculated. The test shows that applying the Kelly Criterion to the classification leads to gains in 61.1% of the examples. On average, the final bankroll is around 4300€, more than 4 times bigger than the sum invested.

Science in Gambling and Sports

Many intellectuals and scientists developed an interest in games and gambling during the course of history. Poincaré[2], Pearson[3] and many others grew an interest in the roulette. Pearson's goal was to study random data and he thought the roulette could be the perfect random data generator. Poincaré saw the roulette as an example of how simple physics processes could evolve into apparently random ones.

A.Hibbs[4] and R.Walford[5] aimed at finding defective roulettes. They went to casinos to observe the results given by a significant number of throws. They discovered that one out of four roulettes did not give the expected results. One was particularly faulty: gambling at that table, they earned enough money to buy a yacht and travel for one year in the Caribbean Sea, starting with just 100$.

Edward Thorp[6] and Claude Shannon[7] (known as "the father" of Information Theory) were the first to use a wearable computer in a casino to help them calculate how much time the ball took to make one round of the roulette, to estimate when it will hit a deflector. Thorpe is famously responsible for developing the counting cards strategy in blackjack, the "Thorp count" in backgammon and for his studies of stock markets.

---

[2] Jules Henri Poincaré, French mathematician, physicist, engineer, philosopher of science (1854-1912).

[3] Karl Pearson, English mathematician and biostatistician (1857-1936).

[4] Albert Roach Hibbs, American mathematician (1924-2003).

[5] Roy Lee Walford, American professor of pathology (1924-2004).

[6] Edward Oakley Thorp, American professor of mathematics, edge fund manager (born in 1932).

[7] Claude Shannon, American mathematician, electric engineer, cryptographer. Mostly famous for his paper "A Mathematical Theory of Communication" (Shannon, 1948).

Neumann[8] and others looked to find the best strategy in poker. In the last decades, there has been a notable trend towards the development of bots for poker, draughts and chess.[9]

The story of sports betting is rich too. One of the first avenues where science was applied to a betting market is the horse races in Hong Kong, which are almost the perfect science lab for sports betting: the races are an everyday event and the participants are not varying much. Bolton[10] and Chapman[11] built a model on the available information for every horse, such as the average speed, the starting position and the weight of the horse, to estimate the probabilities of any horse to win a race.[12] The results of their research were published in "Management Science".[13] (Kucharski, 2016)

It is important to notice that sports betting and sports analytics are intertwined. The objective of sports analytics is to determine which players are better suited to increase the chance of winning of the team. Betting models built on the analyses of single players estimate the chance of winning of a team based on the sum of the expected performance of any player. Therefore, the relevant data is the same in both fields of study.[14] The most famous example of a former gambler that now is an analyst is Haralabos Voulgaris.[15] He developed a complex model based on the matchups of single players to predict the total points scored in a game. His approach gained him not only a fortune, but also convinced the Dallas Mavericks to offer him a job as Director of Quantitative Research and Development.

---

[8] John Von Neumann, Hungarian-American mathematician, physicist, computer scientist, polymath. He is regarded as one of the most important mathematician of his time. The foundation of the mathematical field of Game Theory is attributed to him. He is also known for the Von Neumann Architecture, a computer architecture where instructions and data share the same memory.

[9] Those bots nowadays are better than the best human player.

[10] Ruth N. Bolton, professor of Marketing at the W.P. Carey School of Business, Arizona State University

[11] Randall G. Chapman, consultant, educator and author.

[12] "The Perfect Bet" Adam Kucharski, Chapter 3: "From Los Alamos to Monte Carlo"

[13] "Searching for Positive Returns at the Track: A Multinomial Logit Model for Handicapping Horse Races", Ruth N. Bolton, Randall G. Chapman – Management Science, August 1986

[14] Sports Analytics is also concerned about player salaries, a crucial data point especially in salary cap leagues

[15] Haralabos Voulgaris twitter account

Two fundamental elements to consider before developing a model for sports betting are data availability and a sense of how much luck is part of the scrutinized sport. Sports where scoring is common, such as basketball, are less affected by the occasional lucky bounce of the ball than sports where scoring is rare, like football.

The availability of data is obviously crucial, because without it there is no way to build a model. Those are the reasons why the first group of team sports in which analytical models were successfully developed were American sports such as baseball and basketball.

Baseball has been substantially changed by the arrival of sabermetrics. "Sabermetrics is the empirical analysis of baseball, especially baseball statistics that measure in game activity" (Sabermetrics, n.d.).[16] Sabermetrics gained popularity after the 2002 season, when the Oakland A's extensively used it during roster construction, under management of Billy Beane[17] and his assistant Paul DePodesta[18]. They went on a 20 game winning streak and qualified for the playoffs, with a roster deemed as poor by media and fans, due to lack of star power.[19] Two seasons later, the Boston Red Sox, with the method pioneered by the Oakland A's and more resources, were able to win the world series after an 86 years drought (1918-2004).

[16] The term is derived from the acronym SABR, which stands for Society for American Baseball Research. The term was coined by Bill James – Wikipedia.

[17] William Lamar Beane III, former American professional baseball player, now an Executive Vice President of Baseball Operations for the Oakland Athletics.

[18] Paul DePodesta, Harvard graduate (Economics). He served as assistant manager for the Cleveland Indians, Oakland Athletics and New York Mets and as general manager for the Los Angeles Dodgers. Now he is chief strategy manager for the Cleveland Browns of the National Football League (NFL).

[19] The story of the 2002 Oakland A's was narrated by Michael Lewis in "Moneyball: the Art of Winning an Unfair Game" (Lewis, 2003). In 2011, a movie based on Lewis's book called Moneyball (Miller, 2011) was released, starring Brad Pitt as Billy Bean.

Football Betting: Market Scenario

Bookmakers consider their lines efficient if they achieve to split gamblers' money equally on any possible outcome of any given event. Their business model relies on the commission fee: the odds offered are never fair, meaning they do not add up to 100%, but more. That extra percentage is the commission fee and is the reason why the betting business is stable and growing.[20] As Matthew Holt, director of Race & Sports Data at Cantor Gaming admitted: "What people need to understand, is that the myth that a sports book is trying to post lines that are there to predict the outcome of a game is wrong. We don't make lines to predict the outcome of a game. We make lines in anticipation of where the action will come on that game. So, it's very possible that very acute players can come up with systems that say "look, my predictive model says the game should land very far apart of it", but due to our ability to do player profiling of some of our substantial players, we anticipate action coming a certain way and thus put out a certain line based on where we anticipate action rather than predictive outcome of a particular event" (Holt, 2013).[21]

Bookmaking works similarly to financial markets. "In financial markets if there is more interest in buying the stock of a company then it will go up, if there is more interest in selling stock then it will go down. The same basic principle basic principle is true in bookmaking" (How Do Bookmakers Set Odds and Make Money, n.d.).[22] The bookie comes out with a line, making a rough prediction of the probabilities and adding in their margin on top. Then, depending on how the public reacts, the bookie will move the lines (which in this similitude are the prices). If many people bet on one outcome, the bookmaker will increase his margin for the popular line and reduce it for the opposite one, creating a common incentive/disincentive system.

"The process isn't quite this risky in reality. The bookmaker doesn't just wait to see which bets will be popular and then move the margins, instead they profile their customer base in attempt to

[20] many people think the reason why this business is growing is because bookies are better than the public at predicting results, ignoring what I stated above

[21] Sloan Sports Analytics Conference 2013 - MIT

[22] www.onlinebetting.org.uk

predict which markets and lines will be more popular than others." (How Do Bookmakers Set Odds and Make Money, n.d.).[22]

The present paper studies the "1X2" market in football. What follows is a quick explanation of the decimal odds system used in Europe.

As an example, the odds given by Bet365 for the Serie A match Lazio - S.P.A.L. played on February 2nd 2020 are displayed below:

Betting on S.P.A.L. gives a return of 13 times the amount bet if the bet is won (obviously 0 if the bet is lost). Therefore, a 10€ bet returns $10€ * 13 = 130€$.

The betting lines hold further information. They roughly say the chances given by the public for any outcome and the size of the commission fee taken by the book. Using some simple math:

$$Lazio\ win\ probability = \frac{1}{1.22} \cong 0.82$$

$$Draw\ probability = \frac{1}{6.50} \cong 0.15$$

$$Spal\ win\ probability = \frac{1}{13} \cong 0.08$$

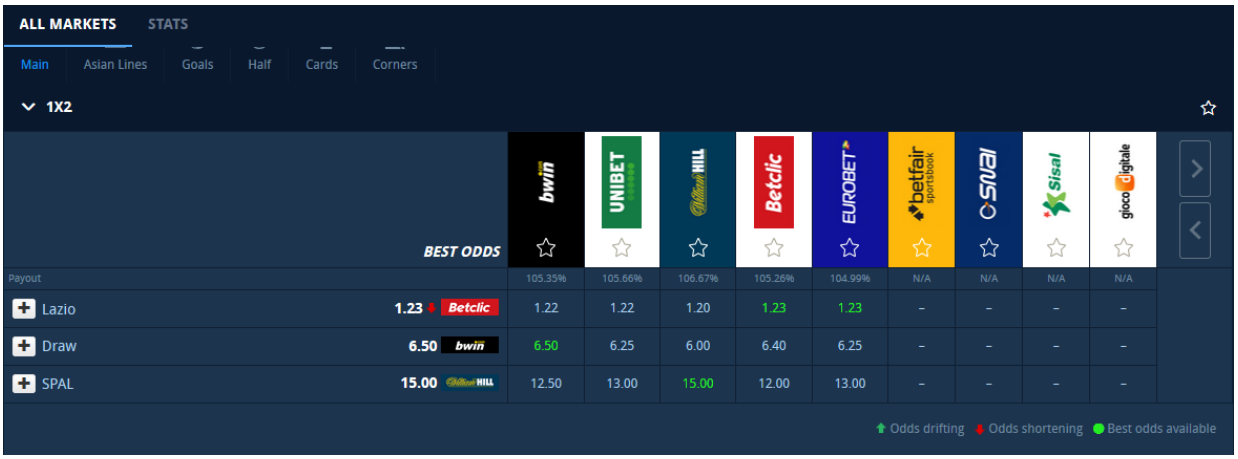
Adding those probabilities and subtracting 1 reveals the commission fee:

$$Commission\ Fee = 0.82 + 0.15 + 0.08 - 1 = 0.05$$

The commission fee on this match on Bet365 is 5%.

To lower the impact of the fee on a bet, the common approach is to take the best odds available on the market, instead of focusing on a single book. There are websites offering useful comparison

tables, such as www.smartbets.com (smartbets, n.d.). The table below displays the "1X2" betting market for Lazio – S.P.A.L., highlights the best odds available and how are they trending:



Taking the odds highlighted in green, the mixed commission fee is:

$$Mixed\ Commission\ Fee = \frac{1}{1.23} + \frac{1}{6.5} + \frac{1}{15} - 1 \cong 0.03$$

Therefore, just by grabbing the best odds, the impact of the commission fee has been reduced by 2%[23].

[23] In the long run, the repercussions are important.

A Basic Introduction to Machine Learning

This short introduction will focus on answering two main questions:

1. what is machine learning?

2. why are we using it?

Let's start from the first question.

With machine learning we indicate the ability of a computer to recognize patterns in a given set of data, called training set. On the basis of that **experience**, the machine comes up with rules to classify or estimate a property of new data. This **inductive process**[24] is the **learning process***:* its brain is the neural network and its methodology is the algorithm. Because any learning method favors some decision rules[25] over others (**Inductive Bias***)*, there is not an optimum algorithm that always outperforms the other. This fact is summarized in several theorems known as "**No Free Lunch Theorems**" (David H. Wolpert, 1997).[26] Due to the No Free Lunch Theorems, "the design of practical learning algorithms is a mix of an art and science" (Sanjeev Kulkarni, An Elementary Introduction to Statistical Learning Theory, 2001, p. 61).[27]

To sum up, "machine learning is about making computers modify or adapt their actions so that these actions become more accurate" (Marsland, Machine Learning - An Algorithmic Perspective,

---

[24] Problem of induction: the conclusions reached from inductive reasoning are not certainly true. Inductive reasoning generalizes a conclusion from gathered observations. It should be evident where the problem lies in this process: just because the observed sample had some characteristics, there is no certainty that the totality of objects of that kind will have them too. This kind of reasoning, while useful and necessary in the evolution process, leads to unsustainable beliefs, reinforced by our innate confirmation bias when the sample used to generalize is too small, which is usually the case in everyday life.

Learning through induction has a statistical value only when the observed sample size is "big enough" and is picked randomly. The galaxy of induction is crammed with logic fallacies: to use it scientifically, we need to ensure our landing on the planet of probability.

[25] Few examples of decision rules are the Bayes decision rule, the nearest neighbor rule and kernel rules.

[26] "No Free Lunch Theorems for Search", David H.Wolpert, William G.Macready (1997)

[27] "An Elementary Introduction to Statistical Learning Theory", Sanjeev Kulkarni, Gilbert Harman (2001), page 61

2009, p. 5).[28] The next required step is generalization, which is the ability to recognize similar features in different data points.[29] As human learning teaches us, generalizing from experiences is a powerful tool, but it also is inherently inaccurate, in some cases to extremes that can be damaging. Again, it is important to be mindful about this. Wrong generalizations can happen due to data bias (arguably the most dangerous and hardest to spot in advance), or the use of an algorithm that does not distinguish data well enough for a given problem.

Why is machine learning useful? One could argue that we should always feed computers not only with data, but also with rules to interpret them. The counter is that letting calculators figure out the rules by themselves is, in many applications, the only possible way to come up with a good rule at all (Sanjeev Kulkarni, An Elementary Introduction to Statistical Learning Theory, 2001, p. 2).

In the case where we know a simple and explicit description of the required features to classify an object in a specific category, we would not need machine learning at all. We would just give our predetermined rules to an algorithm to speed up the desired categorization. This is basically standard programming and it will not lead to any new knowledge about the data: the algorithm would just be used as a calculus tool.[30]

If we would take only the above approach, we would miss opportunities to gain new insights on data. I will use the study case of this thesis as an example. To start off, betting (in legal conditions) means making a prediction on a future event. Due to our lack of superpowers, humans are unable to confidently read the future. The best we can hope for is to make an estimation of what could happen based on past information. Therefore, a bet is naturally defined by its uncertainty. But if we talk about uncertainty, we cannot mechanically apply a general rule to get the desired results. If there were an accurate law to predict football match results, not only there would not be a market for football betting, but arguably interest in football would drastically drop. The fact that I can say

---

[28] "Machine Learning – An Algorithmic Perspective" Stephen Marsland, page 5

[29] Here we can see how machine learning and human learning processes are defined basically in the same way. This is obviously because machine learning is a humanly created concept.

[30] This approach could just enforce the bias of the designer in the definition of the classes.

"every time juventus and S.P.A.L. are going to play, I think juventus is going to win", but I cannot say "every time juventus and S.P.A.L. are going to play, juventus will inevitably win"[31], is a big reason why sports are fun and, as many commentators say "we can talk about numbers all day long, but then we have to play the games".

This obviously does not mean that machine learning will be able to give us the comfort of a sure prediction. Machine learning naturally deals with probabilities and uncertainty, because it is applied to the kind of problems that have an intrinsically uncertain solution. The added value of letting the calculator set its own rules lies mainly in two points: a computer is unbiased[32] and it can evaluate a mass of data much bigger than a single human (or even a group of humans), so it can find patterns we could not spot with our limited vision.

In conclusion: using machine learning reduces the need for explicit programming, increases the adaptability to changing conditions and improves the robustness to errors in assumptions (Sanjeev Kulkarni, An Elementary Introduction to Statistical Learning Theory, 2001, p. 2).[33]

---

[31] It happened: S.P.A.L. - juventus 2-1 (K.Bonifazi, S.Floccari - M.Kean) (Spal - juventus match report, 2019)

[32] While it is true that a computer does not have bias, an important issue regarding machine learning is the bias hidden in data. There are many famous instances where this bias, most of the times unwillingly caused by developers, leads to catastrophic results: software used to predict future criminals is biased against blacks (Julia Angwin, 2016)

[33] "An Elementary Introduction to Statistical Learning Theory", Sanjeev Kulkarni, Gilbert Harman (2001), page 2

McCulloch and Pitts Neurons

The basic component of a neural network is the neuron. An easy but powerful description of a neuron was given in 1943 by McCulloch and Pitts (Warren S. McCulloch, 1943).[34] They modeled a neuron as:

1. **a set of weighted inputs** w_i (acting as the synapses)

2. **an adder** that sums the input signals (equivalent to the membrane of the cell that collects electrical charge)

3. **an activation function** (e.g. threshold function) that decides if the neuron fires for the current inputs (Marsland, Machine Learning - An Algorithmic Perspective, 2009, p. 13).[35]

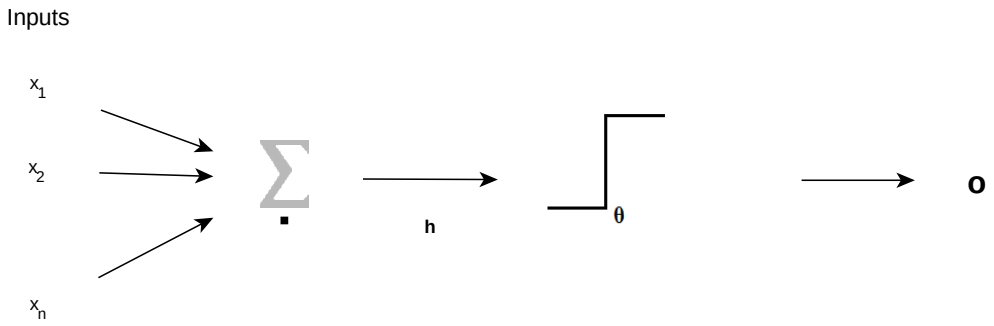


Figure 1: Neuron

Every input x_i is multiplied by the weight w_i assigned to the specific link. When the neuron receives all the signals, it performs the following sum:

---

34 "A logical calculus of the ideas immanent in nervous activity" McCulloch, Pitts (1943)

35 "Machine Learning – An Algorithmic Perspective" Stephen Marsland (2009), page 13

$$h = \sum_{i=1}^{n} w_i x_i$$

The output of the neuron is determined by the decision function. The McCulloch and Pitts neuron works as a binary threshold:

$$o = g(h) = \begin{cases} 1 \; if \; h > \theta \\ 0 \; if \; h \leq \theta \end{cases}$$

where θ is the threshold value, o is the output and $g(h)$ is the output function.

The Perceptron

The perceptron is a one-layer collection of independent neurons. Each of them acts as described in the previous chapter.

Presented with a set of inputs x_i and a collection of outputs w_{ij}, every neuron will either fire or not, depending on the assigned weights and activation function. Since the Perceptron is doing supervised learning, it knows the target value. The goal is to make the Perceptron learn how to reproduce the target.

The learning process goes along as follows: every neuron acts differently based on the weights and its activation function, therefore feeding the Perceptron with an input vector **x** of dimension *m* triggers one output per each neuron. Supposing the Perceptron consists of *n* neurons, it provides *n* outputs. The output given by neuron 1 is labelled y_1, the output given by neuron 2 is labelled y_2 and so on until neuron n, which responds with y_n. Comparing the output pattern (which is a vector) to the target identifies which neurons met the requirements and which did not.

Supposing the k neuron gave the wrong answer, it will be necessary to change its weights w_{ik} (i runs from 1 to m). In order to know if the weights need to be increased or decreased, it is necessary to compute the **error function**:

$$\Delta w_{ik} = \eta * (t_k - y_k) * x_i$$

where t_k is the target for the neuron k and η is a parameter called the **learning rate**, which determines how fast the network learns. It typically takes values between 0.1 and 0.4. A small learning rate is more resistant to noise than a big one, which on the contrary makes the network too reactive and unstable. If $(t_k - y_k)$ is positive, the neuron should have fired and did not, so the weights need to be made bigger, vice versa if it is negative. The element of the input could be negative, which would switch the values over. To make the neuron fire it is necessary to render

the value of the weight negative as well. That is the reason why the term x_i is needed (Marsland, Machine Learning - An Algorithmic Perspective, 2009, p. 21).[36]

The new value of the weight will be the old value plus the following value:

$$w_{ij} \leftarrow w_{ij} + \eta * (t_k - y_k) * x_i$$

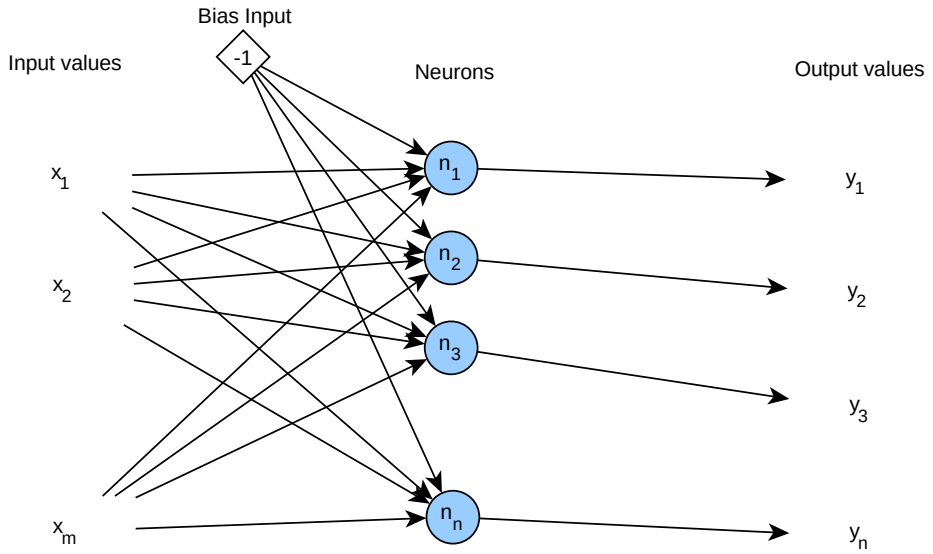


Figure 2: Perceptron Network with Bias

The bias input is a parameter utilized to account for the case where all the inputs to a neuron are zero. In that situation, the size of the weights would not change the behavior of the neuron. Adding an extra input weight to the neuron, with a fixed value (usually -1), allows the neuron to change its behavior by updating this weight, even in presence of all zero inputs (Marsland, Machine Learning - An Algorithmic Perspective, 2009, pp. 22-23).[37]

[36] "Machine Learning – An Algorithmic Perspective" Stephen Marsland (2009), page 21

[37] "Machine Learning – An Algorithmic Perspective" Stephen Marsland (2009), page 22-23

To find the limitations of the Perceptron we need to understand what it computes. Consider one input vector $\mathbf{x}$. A neuron fires if $\mathbf{x} \cdot \boldsymbol{w}^{\boldsymbol{T}} \geq 0$, where $\mathbf{w}$ is the row of the matrix $\mathbf{W}$ that connects the inputs to a particular neuron and $\theta = 0$ for simplicity. By definition of scalar product $\mathbf{x} \cdot \boldsymbol{w}^{\boldsymbol{T}} = \left|\left|\boldsymbol{x}\right|\right| \left|\left|\boldsymbol{w}^{\boldsymbol{T}}\right|\right| \boldsymbol{\cos \gamma}$, so the operation it carries out is a function of the angle γ between the two vectors, scaled by their lengths.

Taking two vectors $\mathbf{x}_1, \mathbf{x}_2$ that satisfy the conditions $\mathbf{x}_1 \cdot \mathbf{w}^T = 0,\ \mathbf{x}_2 \cdot \mathbf{w}^T = 0$, and putting them together holds:

$$\mathbf{x}_1 \cdot \mathbf{w}^T = \mathbf{x}_2 \cdot \mathbf{w}^T \Rightarrow (\mathbf{x}_1 - \mathbf{x}_2) \cdot \mathbf{w}^T = 0$$

Supposing neither $\mathbf{x}_1$ or $\mathbf{x}_2$ is 0, it must be $\cos\gamma = 0 \Rightarrow \gamma = \begin{cases} \pi/2 \\ -\frac{\pi}{2} \end{cases}$ which means that the two vectors are at right angles to each other. Now $\mathbf{x}_1 - \mathbf{x}_2$ is a straight line defined as the **decision boundary** or **discriminant function**, and the weight vector $\mathbf{w}^T$ must be perpendicular to that, as in the figure below.

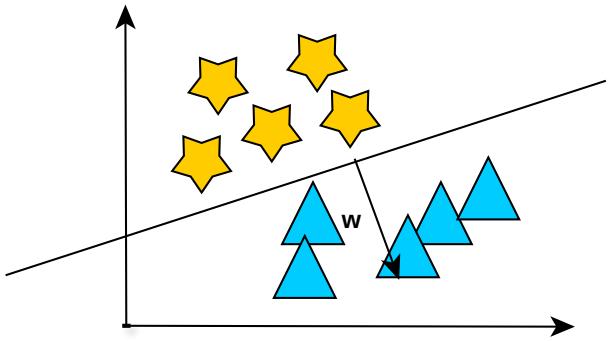


Figure 3: Decision Boundary Separating two Classes of Data

The Perceptron works exactly in this way: it tries to find a straight line that divides the examples where each neuron fires from those where it does not (Marsland, Machine Learning - An Algorithmic Perspective, 2009, p. 33).[38] The problems where it is possible to find such a straight line are called **linearly separable**. The limit of the Perceptron is that it is unable to learn about problems that are not linearly separable. Those problems are not uncommon and a classic example is the XOR function. This inability caused the stalling in machine learning development from the '70s, until the rise in popularity of multilayered neural networks.

Input 1	*Input 2*	*Output*
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*

Table 1: XOR Truth Table

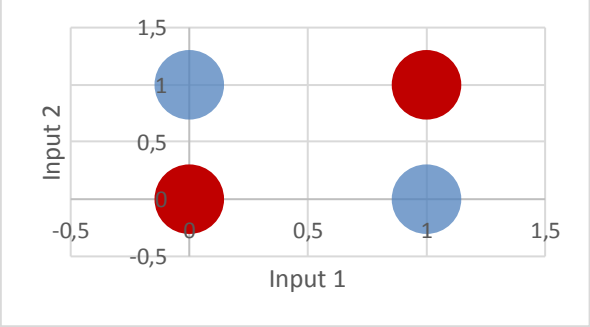


Table 2: Plot of the four data points

[38] "Machine Learning – An Algorithmic Perspective" Stephen Marsland (2009), page 33

Multilayer neural networks

To attack non-linear problems, it is imperative to improve the idea backing up the Perceptron. As previously seen, the learning in the neural network happens in the weights. There are two possible approaches: one is to create backward connections and leads to **recurrent networks**, the other one is to add neurons between the input and the outputs to make a **multilayered network**.

The added layer is called **hidden layer** because it is not possible to know the correct activations for the neurons in the middle of the network. Now, computing the error at the output does not clearly indicate which weights were wrong.

The solution to this problem was found in 1986 by Rumelhart, Hinton and McClelland (Rumelhart, 1986)[39] with a method called **back propagation of error**, which is a form of **gradient descent**.

Before giving the mathematical description of this method, it is necessary to slightly change the form of the threshold function. The step function used previously has a "defect": it is discontinuous (it has a sudden jump in the middle) which makes differentiating it at that point not possible. The solution is to replace the step function with a differentiable function with similar characteristics. A class of S-shaped function with those properties is the **sigmoid functions** (shown in the figure below).

[39] "Parallel Distributed Processing" Rumelhart, McClelland, Hinton, MIT Press, 1986

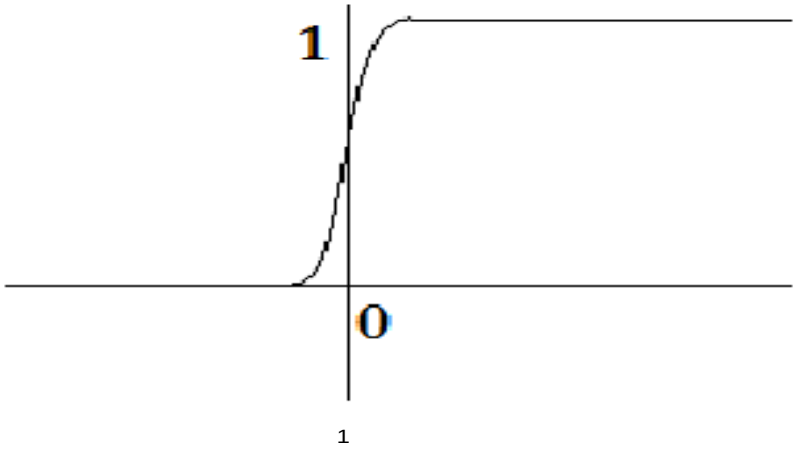


Figure 4: Sigmoid Function

The sigmoid functions form is generally:

$$a = g(h) = \frac{1}{1 + e^{-\beta h}}$$

where β is some positive parameter.

The derivative of this function is:

$$g'(h) = \frac{\beta e^{-\beta h}}{(1 + e^{-\beta h})^2} = \beta g(h)\big(1 - g(h)\big) = \beta a(1 - a)$$

Now it is time to talk about the error of the network. The output of the network is a function of its inputs, the activation function and the weights (**v** for the first layer, **w** for the second). Only the weights can be changed because the inputs are the learning subjects and the activation function is a property of the neurons. The forward part of the algorithm starts from the input and ends with the last layer producing an output. In detail, the hidden neurons receive the inputs **x** and use the

first set of weights **v** to compute their outputs. The outputs of the hidden neurons are then fed as inputs to the last layer and, combined with the second set of weights **w**, determine the outputs **y** of the network. After the outputs are computed, it is time to check their values and compare them with the targets. In the sequent equations, *i* will be an index over the input nodes, *j* will be an index over hidden layer neurons, *k* will be an index over output neurons.

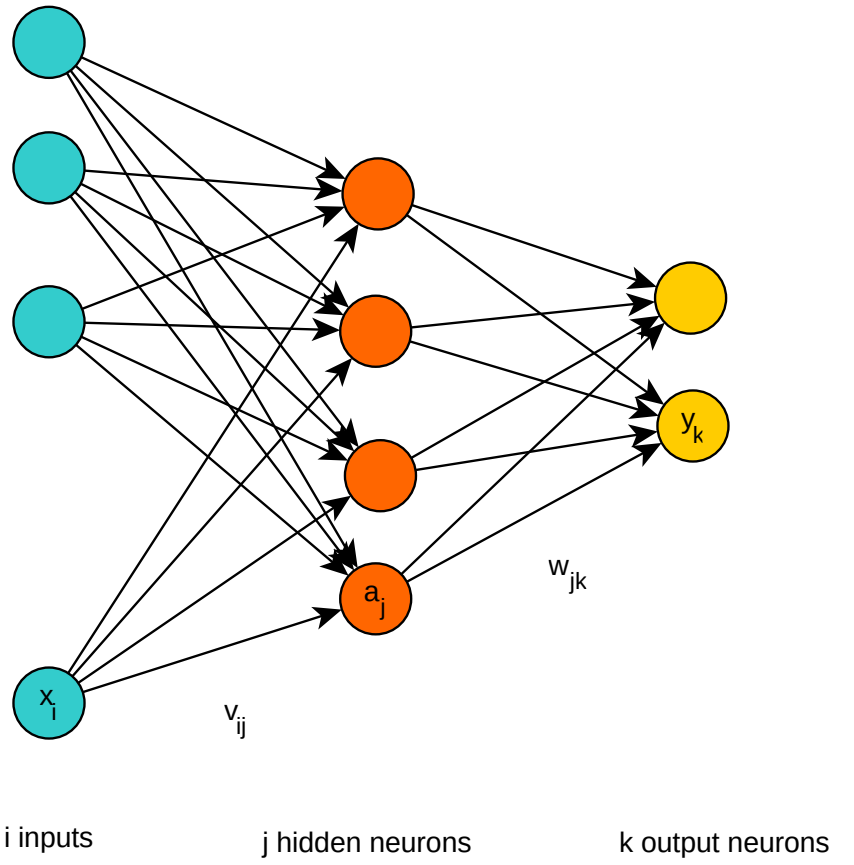


Figure 5: Multilayered Neural Network

The **sum of squares error function** at the output will take the form:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{k}(t_k - y_k)^2 = = \frac{1}{2}\sum_{k}[t_k - g(\sum_{j} w_{jk} a_j)]^2$$

where a_j are the inputs from the hidden layer neurons to the output layer neurons.

The gradient descent algorithm adjusts each weight w_{jk} in the direction of the gradient of $E(\mathbf{w})$. To do so it applies the chain rule:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial h_k}\frac{\partial h_k}{\partial w_{jk}}$$

where $h_k = \sum_l w_{lk}\ a_l$ is the input to output layer neuron k (the sum of the outputs of the hidden layer neurons multiplied by the second layer weights for neuron k). This equation shows that it is possible to study how the error at the output changes following the variations of the second layer weights ($\frac{\partial E}{\partial w_{jk}}$) as the product of two terms:

1. the variation of the output error as the inputs to the output neurons ($\frac{\partial E}{\partial h_k}$) varies;

2. the variation of those inputs as the weights w_{jk} between the hidden layer neurons and the output neurons ($\frac{\partial h_k}{\partial w_{jk}}$) change.

The second term can be rewritten as:

$$\frac{\partial h_k}{\partial w_{jk}} = \frac{\partial \sum_l w_{lk} a_l}{\partial w_{jk}} = \sum_l \frac{\partial w_{lk} a_l}{\partial w_{jk}} = a_j$$

because $\frac{\partial w_{lk}}{\partial w_{jk}} = 0$ for all values of *l* except *l* = *j*.

The first term is called the **error** or **delta term**. To compute it, the chain rule is useful once again:

$$\delta_o = \frac{\partial E}{\partial h_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial h_k}$$

where

$$y_k = g(h_k) = g(\sum_j w_{jk} a_j)$$

is the output of output layer neuron k and g(·) is the activation function.

Therefore, a simpler form for δ_o can be attained via y_k (reminding the expression for the error at the output):

$$\delta_o = \frac{\partial E}{\partial g(h_k)}\frac{\partial g(h_k)}{\partial h_k} = \frac{\partial E}{\partial g(h_k)}\, g'(h_k)$$

$$= \frac{\partial}{\partial g(h_k)}\left[\frac{1}{2}\sum_k (g(h_k) - t_k)^2\right] g'(h_k) = (g(h_k) - t_k) g'(h_k)$$

$$= (y_k - t_k) g'(h_k)$$

Where $g'(h_k)$ denotes the derivative of g with respect to h_k. The activation function g(·) in use is the sigmoid function for which the derivative has already been computed. Now all the ingredients required to update the second layer weights are available:

$$w_{jk} \leftarrow w_{jk} - \eta \frac{\partial E}{\partial w_{jk}} \Rightarrow w_{jk} \leftarrow w_{jk} - \eta \delta_o a_j \Rightarrow$$

$$w_{jk} \leftarrow w_{jk} - \eta (y_k - t_k) y_k (1 - y_k) a_j$$

To get to the update rule for the weights from the inputs to the hidden nodes, we keep following the network backwards, repeating the process above. By employing the chain rule one more time, the expression of the delta term for the hidden layer results:

$$\delta_h = \sum_k \frac{\partial E}{\partial h_k} \frac{\partial h_k}{\partial h_j} = \sum_k \delta_o \frac{\partial h_k}{\partial h_j}$$

where *k* runs over the output nodes and $h_j = \sum_m v_{mj}\, x_m$ is the input to hidden layer neuron j (the sum of the inputs multiplied by the first layer weights v_{mj} for neuron j).

The inputs to the output layer neurons are the outputs of the hidden layer neurons multiplied by the second layer weights:

$$h_k = g\left(\sum_l w_{lk} h_l\right)$$

Which means that:

$$\frac{\partial h_k}{\partial h_j} = \frac{\partial g\left(\sum_l w_{lk} h_l\right)}{\partial h_j}$$

Using again the fact that $\frac{\partial h_l}{\partial h_j} = 0$ unless *l* = *j* which holds:

$$\frac{\partial h_k}{\partial h_j} = w_{jk} g'(a_j) = w_{jk} a_j (1 - a_j)$$

$$\Rightarrow \delta_h = a_j (1 - a_j) \sum_k \delta_o w_{jk}$$

By computing:

$$\frac{\partial E}{\partial v_{ij}} = a_j (1 - a_j) \left(\sum_k \delta_o w_{jk}\right) x_i$$

finally, the update rule for the first layer weights is reached:

$$v_{ij} \leftarrow v_{ij} - \eta \left[a_j(1 - a_j) \left(\sum_k \delta_o w_{jk} \right) x_i \right]$$

Note: if the network has more than one hidden layer, it is necessary to keep cycling this procedure backwards until an update rule for every layer is found (Marsland, Machine Learning - An Algorithmic Perspective, 2009, pp. 50-55, 84-90).[40]

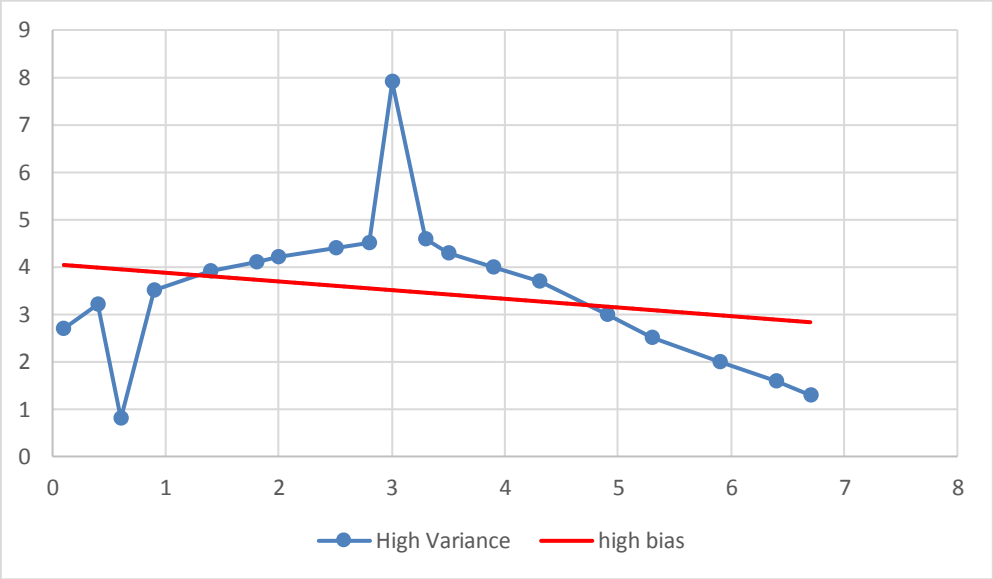
[40] This section relied heavily on "Machine Learning: An Algorithmic Perspective" by Stephen Marsland, see chapter 3.2 and 3.6

The Bias/Variance Tradeoff

The goal of machine learning is to generalize from the training examples to new inputs. In order to achieve this goal, it is necessary to train the network for enough time. What may be counter intuitive is that, after some time, more training worsens the results. This problem is known as **overfitting**.

When a neural network trains for "too long", it starts picking up the noise in the data. This means that instead of finding a function that approximates the data generating function, the network looks for a function which matches the training data perfectly. While it may seem desirable, it reduces the generalization ability of the network. Since the function learned by the network goes through every training inputs, it will be too sensitive to variations in new inputs. To explain overfitting, one can think about how the curve would look if it perfectly matched outliers. It would have sudden spikes and steep holes: in other words, it would change fast. If I test the network with inputs holding similar values, the network will predict the output based on this function. Due to its variability, the output values are probably going to be far from each other. A network with these characteristics has **low bias** (0 at the extreme) and **high variance**.

Bias is defined as the inability of a machine learning method to capture the true relationships between the training set inputs and the target outputs. The term variance refers instead to the difference in fits between the training set and the test set.



An ideal predictive model will have low bias and low variance. In order to attain those goals, it is necessary to find the sweet spot between a simple model and a complex one, by figuring out when to stop learning (Marsland, Machine Learning - An Algorithmic Perspective, 2009, p. 66) (Starmer, 2018).[41]

[41] Sources for this chapter were "Machine Learning: An Algorithmic Perspective" chapter 3.3.4 and Machine Learning Fundamentals: Bias and Variance StatQuest with Josh Starmer

Cross Validation

A fundamental part of machine learning is **testing**.

To start, the training set is divided into subsets. Those subsets are alternatively used as training, testing and validation sets, in a procedure called **cross validation**. The split between sets is arbitrary and is usually guided by the mass of training data available (typical splits are 50:25:25 for large datasets).

Cross validation is necessary to be able to check how well the network learned after the training process is complete. It is also crucial to check how the learning is going during training, because this is a critical information in order to decide when to stop and prevent overfitting. Since a test made on the same set used for training would not be informative about how well the network generalizes and would not spot overfitting, it is not wise to train the network with the entire dataset available. This explains why a testing set is kept. The validation set is used to monitor the generalization ability of the network at its current stage of learning (Marsland, Machine Learning - An Algorithmic Perspective, 2009, p. 68).[42] At first, the network trains for a predetermined amount of time. Then the validation set is used to check the ability to generalize acquired by the network. This process of learning and validating is cycled until the sum of squares error on the validation set will start increasing again. When this happens, the network is starting to overfit, which means it is learning about the noise in the data. At this stage, the training should be stopped, because carrying it out longer would worsen the predictive performance on new data.

The test set is used to analyze the results by comparing the predictions with the target outputs.

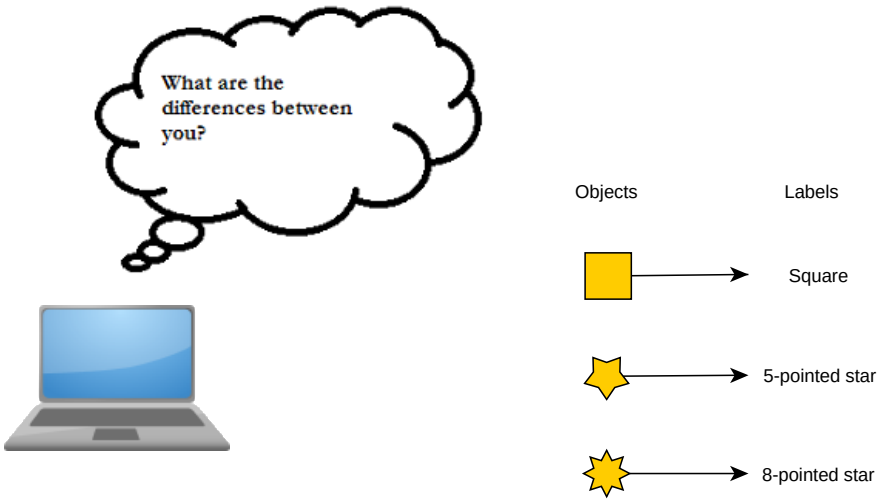
[42] "Machine Learning – An Algorithmic Perspective" Stephen Marsland, page 68

Supervised Learning

In the above discussion, the implicit assumption was that I was applying a specific kind of machine learning, called **supervised learning**.

Supervised Learning consists in providing the computer with a **training set** and the correct associated **target labels**, so that the algorithm can understand the underlying patterns linking them, generalize and respond correctly to new inputs.

The process can be split into two parts: the first one is the one where the computer is actually learning. In this section, the machine tries to understand the rules separating the training set. In other words, the algorithm asks the data why a target label is assigned to a feature vector and looks for similarities between vectors related to the same label. It does so by trial and error.



In the second part, the rule is employed to make predictions on the labels associated with new data.

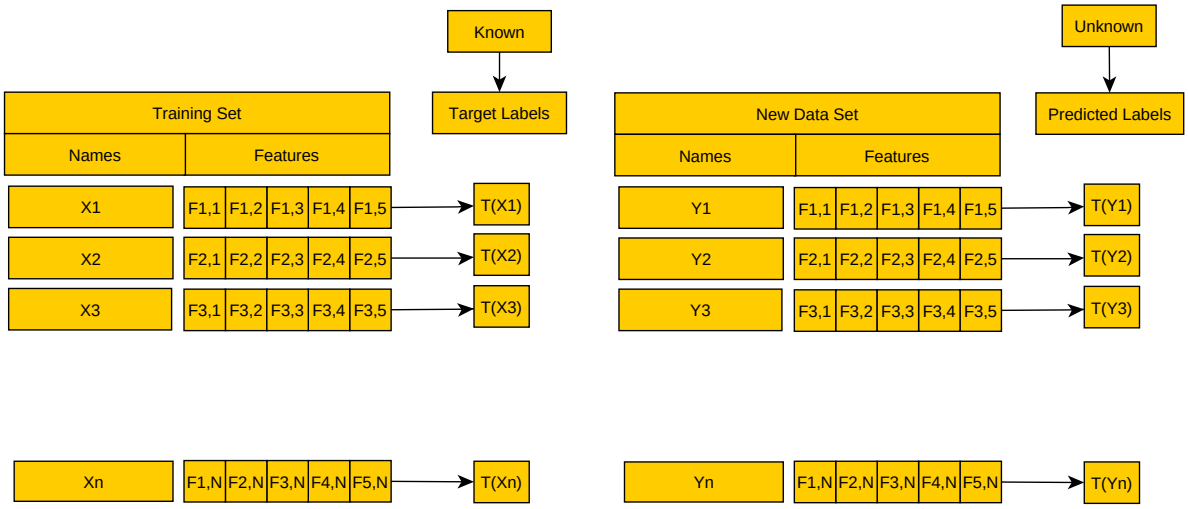


Figure 6 - Supervised Learning

Supervised learning can perform classification and regression:

1. the output of a classification algorithm is a class or category

2. the output of a regression algorithm is a function *f* such that $f(\boldsymbol{x})$ provides a good estimate of the value of *y* given a feature vector **x** (Sanjeev Kulkarni, An Elementary Introduction to Statistical Learning Theory, 2001, p. 150).[43]

[43] "An Elementary Introduction to Statistical Learning Theory" Sanjeev Kulkarni, Gilbert Harman, Chapter 15

The Experiment Ratio

The intent is to find a good rule that predicts the final results of football matches. I define a good rule as one that allows a bettor to consistently beat the odds offered on the market by bookmakers. In other terms, creating a good rule implies finding a way to make a profit on the betting market.[44]

The reasons why I think good rules may exist are two.

1. The first argument is deductive. The job of bookmakers is (surprise!) to make money. This may seem a trivial consideration, but it has a crucial consequence: bookmakers are not strictly trying to guess the results of an event. What they intend to do is understanding what the public *feels*[45,46] the result of an event will be[47]. When they open the market, they offer some odds. Then, depending on how the gamblers place their bets, they move the lines to induce people to split money evenly on any outcome. So, the odds are decided by the bettors as a collective[48]. The goal of gamblers, instead, is to make money by the wonderful thing called being right. This group is made by wannabe Nostradamus[49] who try to predict the outcome of an event by applying some rules. Some use intuition successfully, many use it unsuccessfully. Others follow gurus and, sometimes, even octopuses[50]. Few run statistically sound analyses, which leads me to the next point.

---

[44] This means realizing the ambition of every gambler: to turn a bet into an investment

[45] There is no more volatile thing than feelings, especially in sports, politics, TV competitions, where a relevant degree of fanaticism comes into play. Notice that those are all fields where the betting market operates.

[46] Sometimes they may try to lead the public a certain way by coming out with misleading odds.

[47] What will happen if everyone starts betting following the prediction of an algorithm trained on a good rule? My hypothesis is that then there would be no more value in the market, because the crater between the feeling of the public and the "true" odds of an event would close. That means bookmakers will start offering "true" odds instead of "feel" odds.

[48] Conceptually, bookmakers work in the same way as market makers operate in financial markets, such as the stock market.

[49] Latinization of Nostredame, a French astrologer, physician and reputed seer: Nostradamus on Wikipedia (Nostradamus, n.d.)

[50] 2010 Octopus Paul was made an absolute legend: a crowd of media follows Octopus Paul's World Cup final prediction (Press, 2010)

2. The second argument is inductive. From my experience as an avid consumer of sports media, I came to know about people consistently winning their bets. I found out that what they have in common is some scientific background, or at least the use of a scientific method. This is true also in fantasy sports, which is debatably another form of gambling (Oliver, 2015).[51]

There are many ways to approach the challenge of finding value bets. A successful (and complex) one is to analyze how individual players impact the outcome of a match. Another one is to examine previous performances of teams and look for trends.

What I am aiming to see is if there is value in the odds themselves.

Is it possible to find a pattern to exploit by looking at the odds? How do they relate with actual final results?

[51] See Last Week Tonight with John Oliver: Daily Fantasy Sports (HBO)

Tools

The code I developed uses the Anaconda Distribution of the Python language. It is a free open source distribution of the Python and R languages for scientific computing (data science, machine learning applications, large scale data-processing, predictive analytics, etc.) (Anaconda (Pythondistribution), n.d.).[52]

In particular, I used the Jupyter Notebook graphical user interface, which is a web based interactive computational environment.

The main libraries I worked with were:

- scikit-learn, a library featuring classification, regression and clustering algorithms;
- pandas, a library providing data structures and data analysis tools;
- NumPy, a package for scientific computing;
- Matplotlib, a Python 2D plotting library to generate plots, histograms, bar charts, scatterplots and many other useful visualization charts.

[52] From Wikipedia: Anaconda (Python distribution)

The Dataset

The dataset I will use in the following experiment is taken from the website football-data.co.uk (football-data download, n.d.), which has a rich collection of data about every match played in 22 European Football Leagues from the 1993/1994 season.

The data is stored in Excel files, each containing one season worth of data. Every league has its own worksheet. Football-data keeps track of the following leagues:

- the top 5 leagues in England
- the top 4 leagues in Scotland
- the top 2 leagues in Germany
- the top 2 leagues in Spain
- the top 2 leagues in Italy
- the top 2 leagues in France
- the top leagues in Belgium, Netherlands, Portugal, Turkey and Greece.

For every match a lot of information is recorded, including the statistics on the match, the result, the odds offered by bookmakers for a number of events (final result, half time result, various kinds of over/under…).

Since I intend to predict future results considering purely the odds on the market, I have to drop all the data columns containing information about how the match played out. The dataset is shown in the figure below:

In [13]: ndf

Out[13]:

	B365H	B365D	B365A	BWH	BWD	BWA	IWH	IWD	IWA	PSH	...	PSA	WHH	WHD	WHA	VCH	VCD	VCA	PSCH	PSCD	PSCA
0	2.40	3.3	3.25	2.45	3.10	2.95	2.50	3.30	2.65	2.47	...	3.19	2.50	3.20	2.90	2.50	3.2	3.25	2.79	3.16	2.89
1	2.00	3.3	4.50	2.00	3.20	3.90	2.10	3.30	3.30	2.06	...	4.32	2.05	3.10	4.00	2.00	3.3	4.40	2.25	3.15	3.86
2	3.20	3.4	2.40	2.95	3.20	2.40	2.65	3.30	2.50	3.25	...	2.37	3.10	3.10	2.40	3.25	3.4	2.38	3.64	3.54	2.16
3	4.50	3.6	1.91	4.33	3.40	1.90	3.30	3.30	2.10	4.43	...	1.95	4.20	3.25	1.95	4.40	3.5	1.95	4.68	3.50	1.92
4	1.25	6.5	15.00	1.22	6.00	11.50	1.25	5.50	10.30	1.27	...	13.15	1.25	5.50	13.00	1.25	6.5	15.00	1.25	6.50	14.50
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
22710	1.75	3.1	5.75	1.75	3.10	5.25	1.75	3.30	4.85	1.76	...	4.94	1.78	3.30	4.80	1.75	3.3	4.80	1.85	3.35	4.95
22711	3.60	3.2	2.10	3.90	3.10	2.00	3.75	3.15	2.05	3.95	...	2.04	3.80	3.10	2.05	3.60	3.2	2.05	4.10	3.57	1.94
22712	6.50	3.3	1.65	5.50	3.40	1.65	5.50	3.50	1.65	5.17	...	1.64	5.25	3.60	1.65	5.40	3.4	1.67	4.11	3.45	1.97
22713	1.07	10.0	29.00	1.07	9.50	26.00	1.09	8.60	22.00	1.07	...	24.54	1.07	9.50	29.00	1.07	9.5	23.00	1.06	13.65	37.74
22714	1.16	6.5	17.00	1.18	5.75	16.50	1.20	5.90	13.50	1.16	...	15.76	1.17	6.50	17.00	1.18	6.0	15.00	1.20	6.99	12.79

22715 rows × 21 columns

Figure 7: Training Set containing data of 3 seasons and odds given by 7 bookmakers

Each row contains the odds given for a home team win, a draw and an away team win by Bet365, Bet&Win, Interwetten, Pinnacle, William Hill, VC Bet, Pinnacle Sports. The row number relates the odds to the appropriate match.

To improve the learning performance, the data have been normalized before feeding them into the network. To normalize data means making every column have zero mean and unit variance. This data processing "helps to stop the weights from getting too large unnecessarily […] it does not allow outliers to dominate as much" (Marsland, Machine Learning - An Algorithmic Perspective, 2009, p. 63).[53] To do it, I used the preprocessing function from the scikit-learn library:

```
In [7]: # training set normalization
        from sklearn import preprocessing
        norm_data = preprocessing.normalize(ndf)
        norm_data = norm_data.astype('float')

In [8]: norm_data_df = pd.DataFrame(norm_data, columns=ndf.columns)
```

[53] "Machine Learning: An Algorithmic Perspective", Stephen Marsland, chapter 3.3.1

Classification

The experiment carried out regards a categorization problem. The network has been trained to predict the final result for each match. The experiment has been considered potentially successful if:

$$\frac{returned\ capital}{invested\ capital} = \frac{(bet\ size * number\ of\ bets * MP * AO)}{(bet\ size * number\ of\ bets)} > 100$$

where MP is the percentage of rightfully classified matches and AO is their average odds. To check those values, it was required to have a set of data of which the final score was known and that was not used to train the network. The games played in the 2019-2020 season from August to December were used for this task. In the above expression, the product $number\ of\ bets * MP = winning\ bets$. Cancelling out the common terms, holds:

$$\frac{returned\ capital}{invested\ capital} = MP * AO > 100$$

The reason why the term on the right is 100 is that MP is a percentage. It would have been 1 if a probability had been used.

After I prepared the data, I ran a cross validation test on the network, training it with the SVM[54] algorithm:

[54] Support Vector Machines: it is an algorithm that works by mapping the data in a higher space and finding a linear divider in that higher space.

```
#evaluation svm multiclass, decision function shape one vs one

from sklearn import model_selection
from sklearn.svm import SVC
name = 'svm'
results = []
scoring = 'accuracy'
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = SVC(kernel = 'rbf', cache_size = 1000, gamma = 'scale', shrinking = True, class_weight = 'balanced',
            decision_function_shape = 'ovo')
cv_results = model_selection.cross_val_score(model, norm_data_df, target, cv=kfold, scoring=scoring)
results.append(cv_results)
msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print(msg)
```

```
svm: 0.464276 (0.020076)
```

```
model.fit(norm_data_df,target)
```

The cross-validation test resulted in around 46% good predictions, with a 2% standard deviation, as shown in the figure above.

Next, I prepared the dataframe with the new data. The program read it from the excel file of the 2019-2020 season, dropped the entries with null values and normalized it:

```
# prediction dataframe

df_p = pd.DataFrame(index=[],columns=[])

file_p = []
file_p.append("D:/Bets/eurodata/all-euro-data-2019-2020.xlsx")

xls_p = pd.ExcelFile(file_p[0])
for sheet_count in range(0,22):
    df_p = df_p.append(pd.read_excel(xls_p, sheet_name = sheet_count, usecols = ['FTR', 'B365H', 'B365D', 'B365A',
                                                                                 'BWH', 'BWD', 'BWA', 'IWH', 'IWD', 'IWA',
                                                                                 'WHH', 'WHD', 'WHA', 'VCH', 'VCD', 'VCA',
                                                                                'PSH', 'PSD', 'PSA', 'PSCH', 'PSCD', 'PSCA']))
```

```
# drop rows with null values
ndf_p = df_p.dropna()
```

```
# one column dataframe with target values (results) to test the prediction

target_p = pd.DataFrame(index=[],columns=[])
target_p = target_p.append(ndf_p[['FTR']])
```

```
ndf_p = ndf_p.drop(['FTR'],axis=1)
```

```
# data normalization
npdata = preprocessing.normalize(ndf_p)
npdata = npdata.astype('float')
npdata_df = pd.DataFrame(npdata, columns = ndf_p.columns)
```

I saved the column 'FTR' containing the final results to test the prediction down the road.

The next line of code runs the predictive model over the new data:

```
prediction = model.predict(npdata_df)
```

To test the results, I wanted to know how many matches were predicted right and their odds. The odds I picked were the best available in order to get the best value on the market.

```
correct = 0
oddsH = pd.DataFrame(index=[],columns=[])
HW = 0
oddsD = pd.DataFrame(index=[],columns=[])
D = 0
oddsA = pd.DataFrame(index=[],columns=[])
AW = 0

for i in range (len(prediction)):

    if (prediction[i] == ntarget_p.at[i,'FTR']):
        correct += 1

        if(prediction[i] == 0):
            HW += 1
            oddsH = oddsH.append([[i,NEWdf_p.loc[[i],['B365H', 'BWH', 'IWH', 'WHH', 'VCH', 'PSH', 'PSCH']].values.max(),
                                  NEWdf_p.loc[i,['B365H', 'BWH', 'IWH', 'WHH', 'VCH', 'PSH', 'PSCH']].idxmax(axis=1)]])

        if(prediction[i] == 1):
            D += 1
            oddsD = oddsD.append([[i,NEWdf_p.loc[[i],['B365D', 'BWD', 'IWD', 'WHD', 'VCD', 'PSD', 'PSCD']].values.max(),
                                  NEWdf_p.loc[i,['B365D', 'BWD', 'IWD', 'WHD', 'VCD', 'PSD', 'PSCD']].idxmax(axis=1)]])

        if(prediction[i] == 2):
            AW += 1
            oddsA = oddsA.append([[i,NEWdf_p.loc[[i],['B365A', 'BWA', 'IWA', 'WHA', 'VCA', 'PSA', 'PSCA']].values.max(),
                                  NEWdf_p.loc[i,['B365A', 'BWA', 'IWA', 'WHA', 'VCA', 'PSA', 'PSCA']].idxmax(axis=1)]])
```

HW, D, AW contain the total amount of home team wins, draws and away team wins respectively.

The dataframes oddsH, oddsD and oddsA store the code of the matches rightfully predicted, the best odds available for those matches and the name of the bookmakers who offered those odds, respectively for matches won by the home team, for drawn matches and for matches won by away teams.

The next screen shows the results of the test.

```
# home team wins, draws, away team wins (right prediction)

print(HW,D,AW)
```

```
583 302 429
```

```
# odds total of rightfully classified matches
total_odds = oddsH[1].sum() + oddsD[1].sum() + oddsA[1].sum()
total_odds
```

```
2980.87
```

```
# average odds
average = total_odds / correct
average
```

```
2.268546423135464
```

```
# percentage right class
perc = (correct / len(ntarget_p)) * 100
perc
```

```
45.41997926028344
```

```
# expected value
perc * average
```

```
103.03733148980297
```

The network predicted correctly 1314 outcomes out of 2893 events (around 45,42%) with average odds of almost 2,27. As shown above, the yield is 3%.

The Kelly Criterion

"The Kelly Criterion is a formula for bet sizing relating to the long-term growth of capital developed by John L. Kelly, Jr[55]. The formula is currently used by gamblers and investors for risk and money management purposes, to determine what percentage of their bankroll/capital should be used in each bet/trade to maximize long-term growth" (Kenton, 2019).[56]

For bets with two outcomes (win/lose) like in our case, the Kelly bet is:

$$f^* = \frac{bp - q}{b} = \frac{bp - (1 - p)}{b} = \frac{p(b + 1) - 1}{b}$$

In the above formula:

- f^* is the fraction of the current bankroll to wager;
- b is the net odds received on the wager;
- p is the probability of winning;
- $q = 1 - p$ is the probability of losing (Kelly Criterion, n.d.).[57]

[55] "A New Interpretation of Information Rate" J.L.Kelly Jr. (J.L.Kelly, 1956)

[56] Investopedia - Kelly Criterion

[57] Wikipedia - Kelly Criterion

Application of the Kelly Criterion to the Classification Experiment

To give further data on the results of the experiment, I wanted to check how well I could do by betting using the Kelly Criterion. In this context, I assumed not to know which matches were predicted right. The only relevant information is:

- the probability of winning;
- the average net odds received on the wager.

I created a random process of 1000 random variables, where each variable tracks the movements of the bankroll for 3000 bets. I set the starting bankroll at 1000€. What distinguishes any variable is the flow of wins and losses, which relates to the bet sizing. For example, let's assume that the first variable tracks a win for the first bet, while the second variable computes the first bet as a loss. That would result in a bigger bankroll for the first variable, which means the Kelly Criterion will suggest a bigger bet size for the second bet. The opposite would happen for the second variable.

Applying the Kelly formula with $b = 1.2686, p = 0.4542, q = 0.5458$ (which are the values given by the experiment) holds:

$$f^* = \frac{bp - q}{b} \cong 0.024$$

Therefore, for every bet, the investment would be 2.4% of the capital.

The following code shows the creation of the process.

```
import numpy as np

bankroll = np.empty([1000,3001],dtype = float)
```

```
#set starting point at 1000€ for all random variables

bankroll[:,0] = 1000
```

```
kelly = 0.024
average = 1.268
```

```
betresult = np.empty([1000,3000],dtype = float)
betsize = np.empty([1000,3000],dtype = float)

for i in range (1000):

    for j in range (3000):

        betsize[i,j] = bankroll[i,j] * kelly
        betresult[i,j] = np.random.choice([0,1],p = [0.55,0.45])
        if(betresult[i,j] == 0):
            bankroll[i,j+1] = bankroll[i,j] - betsize[i,j]
        if(betresult[i,j] == 1):
            bankroll[i,j+1] = bankroll[i,j] + betsize[i,j]*average
```

I computed the average return and plotted it:

```
# expected value function

Eval = bankroll.mean(0)
```

```
Variance = bankroll.var(0)
```

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
fig = plt.figure(figsize=[17,9])
ax = plt.axes()
x = np.linspace(0,3000,num = 3001)
ax.plot(x,Eval)
```

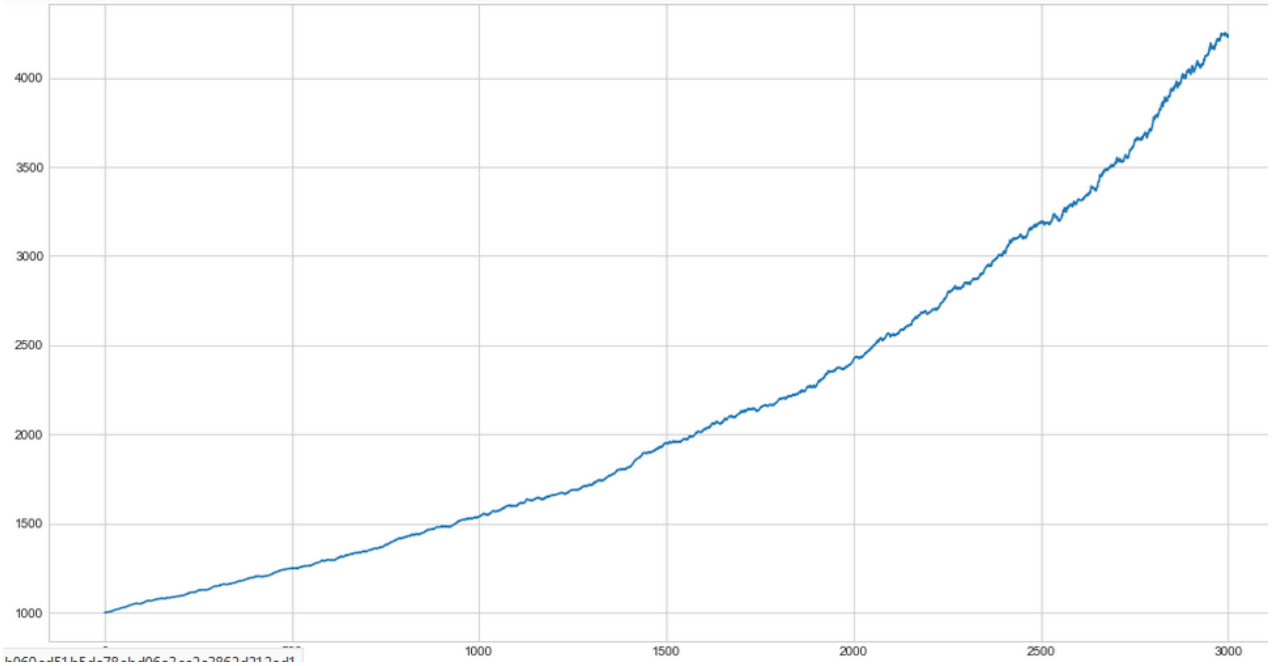


Chart 1: Average Bankroll

While the average is a meaningful information, it may be skewed by extremely big outliers. A simple way to see the probabilities of gain, is to create ranges of values and to count how many of the random variables fall in any of them.

I elected to create 8 ranges, named as follows:

1. "under" contains the number of variables which ended up with a bankroll<1k
2. "range1" for random variables with final bankroll from 1k to 2k
3. "range2" for random variables with final bankroll from 2k to 4k
4. "range3" for random variables with final bankroll from 4k to 7k
5. "range4" for random variables with final bankroll from 7k to 12k

6. “range5” for random variables with final bankroll from 12k to 20k

7. “range6” for random variables with final bankroll from 20k to 40k

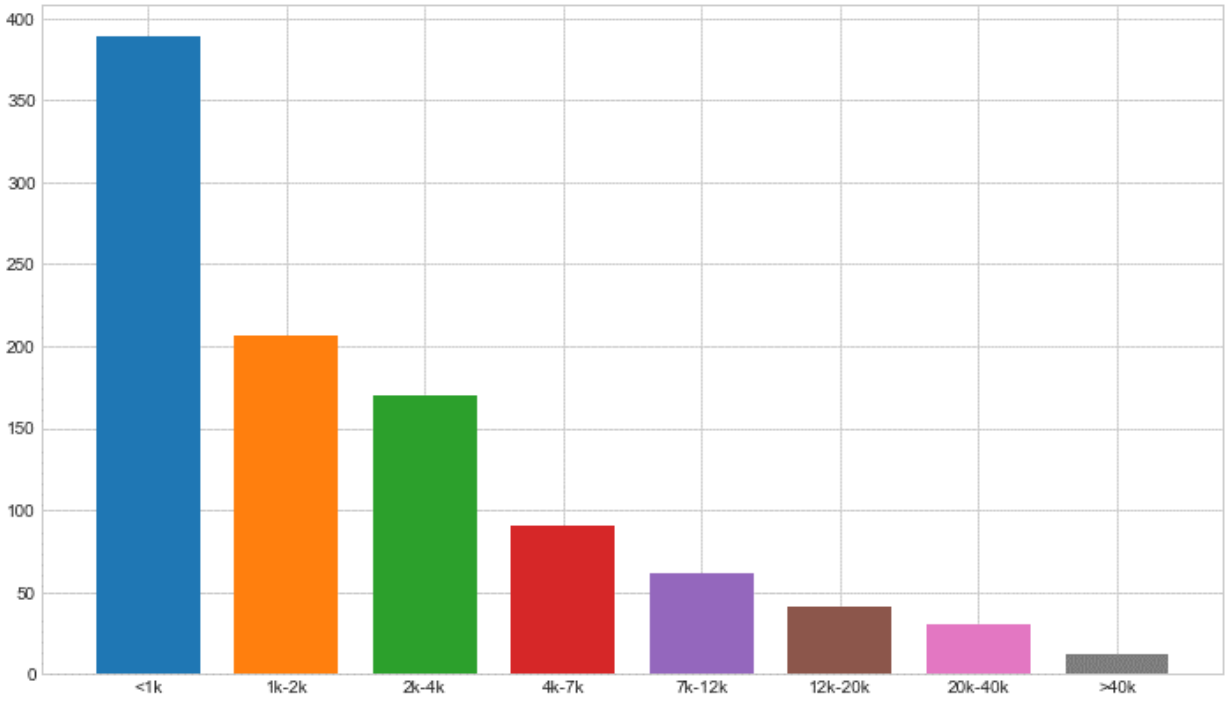
8. “range7” for random variables with final bankroll>40k

```
# final range check

under = 0 # 0<under<1000
range1 = 0 # 1000<=range1<2000
range2 = 0 # 2000<=range2<4000
range3 = 0 # 4000<=range3<7000
range4 = 0 # 7000<=range4<12000
range5 = 0 # 12000<=range5<20000
range6 = 0 # 20000<=range6<40000
range7 = 0 # range7>=40000

for i in range(1000):
    if(bankroll[i,3000] < 1000):
        under += 1
    if(1000 <= bankroll[i,3000] < 2000):
        range1 += 1
    if(2000 <= bankroll[i,3000] < 4000):
        range2 += 1
    if(4000 <= bankroll[i,3000] < 7000):
        range3 += 1
    if(7000 <= bankroll[i,3000] < 12000):
        range4 += 1
    if(12000 <= bankroll[i,3000] < 20000):
        range5 += 1
    if(20000 <= bankroll[i,3000] < 40000):
        range6 += 1
    if(bankroll[i,3000] >= 40000):
        range7 += 1
```

The results are below, shown explicitly as pure numbers and visually as a histogram.

```
print("<1k: ",under)
print("1k-2k: ",range1)
print("2k-4k: ",range2)
print("4k-7k: ",range3)
print("7k-12k: ",range4)
print("12k-20k: ",range5)
print("20k-40k: ",range6)
print(">40k: ",range7)
```

```
<1k:  389
1k-2k:  206
2k-4k:  170
4k-7k:  91
7k-12k:  61
12k-20k:  41
20k-40k:  30
>40k:  12
```



Conclusions

The experiment consisted in training a neural network for classification via supervised learning. The training data counted 22715 matches played over 3 seasons: 2016/2017, 2017/2018 and 2018/2019. The relevant features for each game were the odds given by 7 bookmakers for any of the 3 possible outcomes of a football match (home team win, draw, away team win), adding up for a total of 21 features for each match. A test on the network has been run, splitting the training data in 10 subsets for cross-validation: on average, the network classified correctly 46.43% of the data, with a standard deviation of 2%.

The network has then been used to predict the results of 2893 matches played between August and December 2019. The network gave the right prediction for 1314 matches. Therefore, the network has been accurate 45.42% of the time. The average of the best odds available for the correctly classified matches was 2.27. Hence, betting 100 times using a fixed size bet, the return would be:

$$(fixed\ bet\ size * 0.4542 * 2.27) - (fixed\ bet\ size * 0.5458 * 0)$$

The first term accounts for wins and the second one for losses. For example, if for any given bet the stake is 1€: $(1€ * 0.4542 * 2.27) - (1€ * 0.5458) \cong 1.03$. Thus, the yield is 3%.

To maximize the return on investment I utilized the Kelly Criterion, which is a formula to determine the optimal bet size. Holding the above values, the Kelly bet has been estimated to be a 2.4% fraction of the available capital per any given bet.

To estimate the possible profits of staking with the Kelly strategy, I created a random process made of 1000 random variables, each starting with a bankroll of 1000€. Using the Kelly bet size and the average of the best odds available for each game, I observed the behavior of the process, cycling each variable for 3000 bets. The mean of the process ended up being around 4300€ after 3000 bets. The mean curve shows a progressive and steady increase on the expected return.

I then checked how many of the 1000 variables had a negative return and, if positive, what the value range was. The return has been negative 38.9% of the times and positive, with varying degrees, 61.1% of the times. In 23.5% of the cases, the bankroll ended up being at least 4 times bigger than the initial one (40 times bigger in 1.2% of the cases).

Overall the experiment seems to suggest that it is possible to make a profit using the predictions given by the neural network.

Future Developments

A possible advancement could be to focus on one league at a time and see if the behavior of bookmakers is particularly skewed by the public in any of them. This could be true especially in popular leagues, where fandom has a more dominant role. The amount of training data should not be a problem because football-data has a database that goes back to 1993 and each league has between 240-552[58] games per season (depending on its size). However, the value of old data is debatable (and what is "old"?), assuming there have been some adjustments in the way the odds are set over time.

Another plausible way would be to train the network with more or less data to check if the amount used in the above experiment was suboptimal due to underfit or overfit.

A different strategy could be to use svm for regression to identify the matches where the estimation of the network is the most deviant from the odds and exploit them.

Eventually, it would be interesting to see how different algorithms would perform on this problem against svm, both in classification and regression.

[58] The smallest league has 16 teams, facing each other 2 times in a season: $number\ of\ matches = 15 * 2 * 8 = 240$. The biggest league has a total of 24 teams, playing each other 2 times per year: $number\ of\ matches = 23 * 2 * 12 = 552$

Bibliography

(n.d.). Retrieved from smartbets: https://www.smartbets.com/

Anaconda (Pythondistribution). (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)

David H. Wolpert, W. G. (1997, April). No Free Lunch Theorems for Search. IEEE Transactions on Evolutionary Computation.

football-data download. (n.d.). Retrieved from football-data.co.uk: http://football-data.co.uk/downloadm.php

Holt, M. (2013). Predictive Sports Betting. (J. Ma, Interviewer)

How Do Bookmakers Set Odds and Make Money. (n.d.). Retrieved from onlinebetting: https://www.onlinebetting.org.uk/betting-guides/how-do-bookmakers-set-odds-and-make-money.html

J.L.Kelly. (1956). A New Interpretation of Information Rate.

Julia Angwin, J. L. (2016, May 23). Machine Bias Risk Assessments in Criminal Sentencing. Retrieved from propubblica: https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

Kelly Criterion. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Kelly_criterion

Kenton, W. (2019, July 21). Tools for Fundamental Analysis. Retrieved from Investopedia: https://www.investopedia.com/terms/k/kellycriterion.asp

Kucharski, A. (2016). The Perfect Bet - How Science and Math Are Taking the Luck Out of Gambling. Basic Books.

Lewis, M. (2003). Moneyball: the Art of Winning an Unfair Game. W. W. Norton & Company.

Marsland, S. (2009). Machine Learning - An Algorithmic Perspective. Chapman & Hall/CRC Press. (p.5)

Marsland, S. (2009). Machine Learning - An Algorithmic Perspective. Chapman & Hall/CRC Press. (p.13)

Marsland, S. (2009). Machine Learning - An Algorithmic Perspective. Chapman & Hall/CRC Press. (p.21)

Marsland, S. (2009). Machine Learning - An Algorithmic Perspective. Chapman & Hall/CRC Press. (pp.22-23)

Marsland, S. (2009). Machine Learning - An Algorithmic Perspective. Chapman & Hall/CRC Press. (p33)

Marsland, S. (2009). Machine Learning - An Algorithmic Perspective. Chapman & Hall/CRC Press. (pp.50-55, pp.84-90)

Marsland, S. (2009). Machine Learning - An Algorithmic Perspective. Chapman & Hall/CRC Press.(p.63)

Marsland, S. (2009). Machine Learning - An Algorithmic Perspective. Chapman & Hall/CRC Press.(p.66)

Marsland, S. (2009). Machine Learning - An Algorithmic Perspective. Chapman & Hall/CRC Press.(p.68)

Nostradamus. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Nostradamus

Oliver, J. (2015, November 15). Daily Fantasy Sports: Last Week Tonight with John Oliver. Last Week Tonight with John Oliver. HBO.

Press, A. (2010, July 6). Raw Video: Oracle Octopus Picks Spain.

Rumelhart, M. H. (1986). Parallel Distributed Processing. MIT Press.

Sabermetrics. (n.d.). Retrieved from Wikipedia.

Sanjeev Kulkarni, G. H. (2001). An Elementary Introduction to Statistical Learning Theory. Wiley. (p.2)

Sanjeev Kulkarni, G. H. (2001). An Elementary Introduction to Statistical Learning Theory. Wiley. (p.61)

Sanjeev Kulkarni, G. H. (2001). An Elementary Introduction to Statistical Learning Theory. Wiley. (p.150)

Sanjeev Kulkarni, G. H. (2001). An Elementary Introduction to Statistical Learning Theory. Wiley.

Spal - juventus match report. (2019, April 13). Retrieved from legaseriea: http://www.legaseriea.it/en/serie-a/match-report/2018-19/UNICO/UNI/32/SPAJUV

Starmer, J. (2018, September 17). Machine Learning Fundamentals: Bias and Variance. StatQuest.

Vapnik, Lerner (1963). Generalised Portrait Method

Vapnik, A. Y. Chervonenkis (1964). A note on one class of Perceptrons

Warren S. McCulloch, W. P. (1943). A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics.

Table of Contents