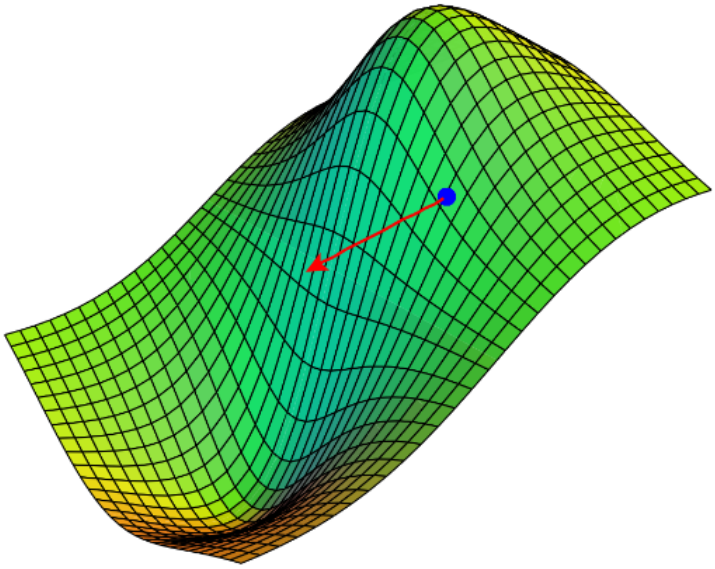
UNIVERSITÉ DE NANTES

IAE NANTES
ÉCONOMIE & MANAGEMENT

Quantitative Research on Market Predictability:
Machine Learning and the Efficient Market Hypothesis



Aurouet Lucas

Master EKAP
IAE Nantes
Under the supervision of Prof. O.DARNE 01/03/2021

Contents

1 Introduction

Trading is futile. This is an oversimplification of the Efficient Market Hypothesis. According to Eugene Fama who formulated this hypothesis in 1970[1], prices at a certain date incorporate all the available and relevant information at that date. There are never overvalued nor undervalued assets, fundamental and/or technical analysis are therefore pointless as all the signals are already reflected by prices on the market.

One cannot beat the market, is another way of expressing the EMH, which brings us to our initial statement; trading is futile. It is quite hard to understand how, if the EMH holds, financial institutions have poured significant amounts of money into trying to outperform the market. Even less understandable is how some have succeeded. We think it is reasonable to define what 'outperforming the market' means. 'Outperforming the market' encompasses the achievement of greater returns than a portfolio containing every assets available on the market, with the same level of risk offered by such portfolio. Alternatively, it could mean achieving the same return, with less risk. Eugene Fama states that the only way to expect returns greater than the market is to take additional risks, or that you cannot achieve that same return on investment with a risk inferior to the market risk.

We are now able to ask the following question: does the EMH holds whnen looking at the data ? Is trading futile ? Can you achieve greater returns than the market for a corresponding market risk ? Or is one constantly better off holding a large index like the S&P 500 for instance ? In our opinion, questioning the EMH is the most riveting topic in quantitative finance as it answers questions from a very theoretical point of view, typically what is observed among academics but also from a very practical point of view. Econometrics is the science of combining these two aspects, using theoretical tools to shine lights on very down to earth problems. Therefore this work will try to reflect both dimensions. There are many ways to prove or disprove the EMH, but consistently beating the market is one of them (consistency is essential here). The objective is to use data science to try and outperform the market. The results should definitely not be taken as an attempt to generate profits but rather as an exercise where the objective is to make use of quantitative techniques to elucidate a question.

Machine Learning is a very powerful tool when a problem comes to identifying patterns and regularities in the data. As we will discuss later on, it is particularly adapted in our case. One fact often played down is that Machine Learning boils down to a list of algorithms applied in a particular order, and it is very easy to make mistakes when using automated techniques. We hope to have sufficient knowledge to question our own results, especially as we are working in quantitative finance. Although we are not, as mentioned above, attempting to concretely apply our findings, We think it is only natural to add some perspective when working with financial data. Finance is connected to the economy by many pipelines and misleading Artificial Intelligence can induce disastrous consequences. Artificial Intelligence is a very broad term and Machine Learning is one of its component, the later relies on the idea of being able to find patterns without explicitly defining these patterns, whereas Artificial Intelligence relies on the idea of writing a program capable of the same thought process as a human being. They can sometimes be interchangeable and the frontier between the two is often blurry, here we are focusing on learning patterns from

[1] Efficient Capital Markets: A Review of Theory and Empirical Work - E.Fama

data, hence we will almost exclusively refer to Machine Learning as opposed to Artificial intelligence. We will also refer to Statistical Learning, which could be seen as the gap between traditional statistical models and Machine Learning models. This gives us a large panel of tools to extract useful information from the data, coerce this information into a functional form that can then be exploited to infer the behavior of previously unseen data. We will first gaze over the Statistical and Machine Learning methods we will be using ranging from logistic regression to Neural Networks and Support Vector Machines. The second part is a step-by-step journal of the progress of the overall project and what modifications have been made to improve the data processing[2].

This project is coded in Python 3.8 and is associated with a Jupyter Notebook, available on GitHub. The integrity of the code is open source.

I often compare open source to science. To where science took this whole notion of developing ideas in the open and improving on other peoples' ideas and making it into what science is today and the incredible advances that we have had. - Linus Torvalds, developer of Linux and Git

[2]Although data processing often refers to the pre-screening of the data before feeding it into a model, here we refer to the entire process from the data gathering to the diagnosis of the results

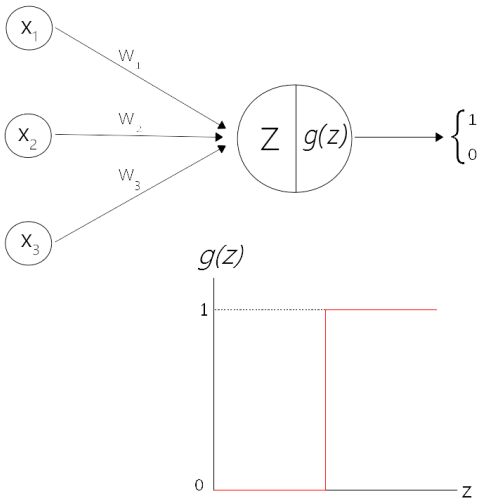
2 Statistics & Machine Learning

2.1 The Perceptron

Artificial Neural Networks (ANN or NN for short) are Machine Learning models designed to replicate the way the human brain processes information and are heavily inspired by biology. In a human brain, the information travels through a network of neurons which can activate to pass the signal to the next neurons. Neurons are connected by branches in which the information transits, starting from a particular point, these branches, along with the neurons that activated create a path to an end point which outputs a decision. The choice for each neuron to carry on the information is what determines the final output. ANN replicate this logic by creating a stream of branches and neurons inside which the information is processed to eventually reach a particular output.

In this section, we will focus on the Perceptron, a particular case of ANN which consists of only one neuron. It is easier to understand the logic behind ANN using a Perceptron, the results can then be generalized to more complex networks with more neurons. We will try to understand how a Perceptron is able to find patterns in the data without being explicitly programmed. This is what we refer to as the learning algorithm, or how the machine learns from the data.

Figure 1: Diagram of a Perceptron



We begin by giving the model inputs (x_p[3]) that account for the exogenous variables. Each input is multiplied by a weight w_j and all weighted inputs are summed. The weighted sum, z goes through an activation function $g(z)$, that outputs 1 if the sum exceeds a certain threshold, 0 otherwise. For instance, we can think of inputs as values for technical indicators such as the RSI, the MACD and the stochastic oscillator and the output as 1 if we should "buy" and 0 if we should "sell". We take any data point and extract the values for the inputs, we compute the weighted sum and pass it through the activation function if the results exceeds the threshold, the Perceptron will output 1, or "buy".

[3] p = 1,..., P where P is the total number of inputs

To learn how to accurately classify data, the Perceptron will update each weight such that we eventually reach a point where every data point is associated to the correct class. If the Perceptron outputs "buy", and the correct position was indeed buying, we can repeat the procedure for another data point. If the correct position was to sell, we update the weights and then we move on to the next data point. Weights are updated whenever the Perceptron outputs the wrong value, and kept the same whenever the Percetron outputs the correct value.

As weights are the only elements that can be manipulated, by constantly updating them we will eventually come to the right combination of weights, which outputs the correct buy or sell output for every data point. The weights are then no longer updated and the Perceptron has finished training. For each new set of technical indicators, the weights will be such that the weighted sum of inputs activate the neuron only when necessary, or when buying is the right call, and stay deactivated when the right position is to sell. The process of updating weights is not random, we desire to increment weights in order to increase the accuracy at each update. In order to do so, we can calculate a loss function, the loss will be 0 if the output corresponds to the target value and different from 0 if the Perceptron fails. This loss function represents the error, or how accurate the Perceptron is, with respect to every weights combination. We can minimize this loss function by differentiating it with respect to each weights and set partial derivatives equal to 0, which solves for the weights that outputs the minimal error[4]. This closely relates to Ordinary Least Squares where we minimize the Mean Squared Error in order to find coefficients that minimize the squared error. Although with OLS, one can find the solution to the differential equation analytically, it is often impossible with Perceptrons, and ANN, as they do not have closed-form solutions. To find the correct weights we are required to use numerical methods, and update them gradually towards the minimum instead of directly deriving weights that minimize the loss function.

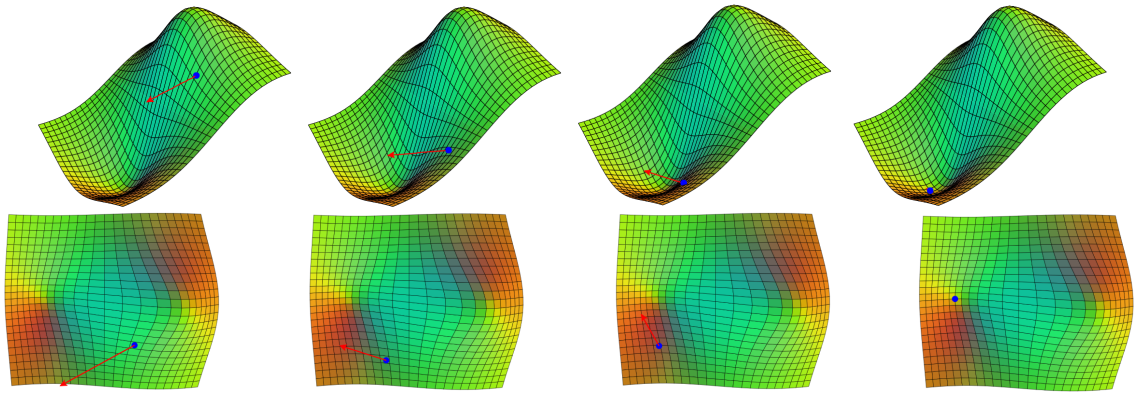
Gradient descent is a numerical algorithm to help us find the local minimum of a loss function (denoted as C), assuming it is differentiable. Starting with random initial values for the weights we compute the vector of partial derivatives of the loss function with respect to each weights, also called the gradient vector of the loss function ($\nabla(C)$). This vector of partial derivatives will point in the direction of steepest ascent where any increment in the weights results in the largest possible increase ib the loss function. If for a particular set of weights we land at a point a on the loss function, the gradient tells us in which direction the loss function increases the most or which new combination of weights will land on a' as far up as possible from a.

As our objective is to find the minimum of the loss function, we take the inverse of the gradient vector, which then leads to the the direction of steepest descent. The inverse gradient tells us how to update weights so that the loss function decreases the most rapidly towards a local minimum. Once we updated the weights, we repeat the procedure and compute the gradient vector of the loss function evaluated at a the new weights combination. We take a step in the direction of steepest descent (inverse of the gradient vector at that point) and this procedure is iterated until we reach a minimum. At a local minimum all the partial derivatives are 0, hence the gradient vector is a null vector. Less formally, once we reach a local minimum, the gradient descent tells us that we should move by 0 in every direction, which is equivalent to not moving at all and weights

[4] 0 ideally, but in reality the minimal error lies somewhere between 0 and 1

are then optimal.

Figure 2: Gradient Descent

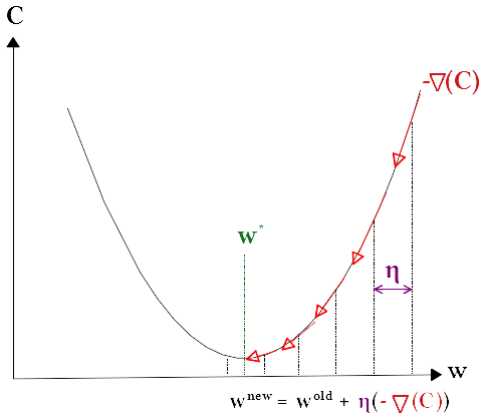


For each data point we compute the error and the gradient at that point, take the inverse and increment the weights in the direction pointed by the inverse gradient. By always choosing the direction of steepest descent we will eventually converge to a local minimum, given that the step size is adapted. The formula used to update weights can be formalized as:

$$\overrightarrow{w^{new}} = \overrightarrow{w^{old}} + \eta(-\nabla(C)) \tag{1}$$

Where $\overrightarrow{w^{new}}$ is the vector of updated weights, $\overrightarrow{w^{old}}$ is the vector of old weights, η is the learning rate which determines the size of the step and $\nabla(C) = \begin{bmatrix} \frac{\partial C}{\partial w_j} \\ ... \\ \frac{\partial C}{\partial w_J} \end{bmatrix}$ is the vector of partial derivatives of the loss function (C) with respect to each weights in $\overrightarrow{w^{old}}$. We often determine stopping rules that stop the gradient descent algorithm instead of reaching a point where all partial derivatives are 0. The gradient vector does not point towards the minimum but towards the direction in which the loss function decreases. If we take a step in that direction once we are really close from the minimum, we might overshoot and land beyond the minimum, therefore if no stopping rules are applied the algorithm keeps iterating indefinitely. Once we reach a certain number of iterations, or that we are satisfied with the model's accuracy, we can voluntarily stop the procedure and get values for the weights that minimize the loss function. It is a possibility that the minimum identified by the gradient descent algorithm is a local minimum, while we ideally want to find the global minimum of the loss function. By construction, gradient descent is not able to tell whether we attained a local or global minimum which can lead to significant inaccuracies in the model. This is one of the crucial drawbacks in gradient descent. But other methods have been developed to minimize the risk of getting stuck at local minimums, such as the mini-batch gradient descent or the adam gradient descent for instance.

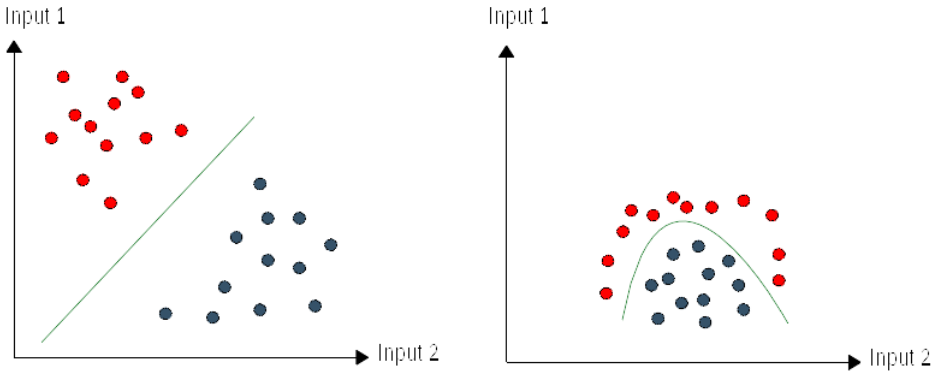
Figure 3: Overshooting the local minimum



In the simple example above, we see how inappropriate step size η can lead the algorithm to oscillate around the local minimum of the loss function which is found at weight $w*$. The algorithm will never converge as weight will "bounce" from right to left, when the weight is to large the gradient points towards the left and because the step size is to large, we land beyond the minimum. The weight is then to small, the gradient points towards the right and we take a step in that direction landing beyond $w*$. The algorithm will iterate indefinitely as we never reach $w*$ where $\nabla(C)$ is null. This shows the importance of determining stopping rules such as number of iterations or minimum acceptable accuracy to prevent infinite iterations.

The Perceptron is limited to linearly separable problems, where it is possible to find a linear decision frontier in the input space that separates classes. In two dimensions, the frontier is a straight line, in three dimensions it would be a plane and for all further dimensions the frontier is an hyperplane. Once the neuron has been trained and we the finals weights are obtained we can calculate $b(x) = W^T x + w_0$ to trace a decision boundary. Every point falling on a particular side of the boundary will be associated with the corresponding class and any point falling on the other side of the frontier will be associated to the other class. The diagram below illustrates two different problems, the first one which is linearly separable and the second one which is not. Because Peceptron is a linear classifier, we need to upgrade it in order to solve non linear problems such as the one presented on the diagram above.

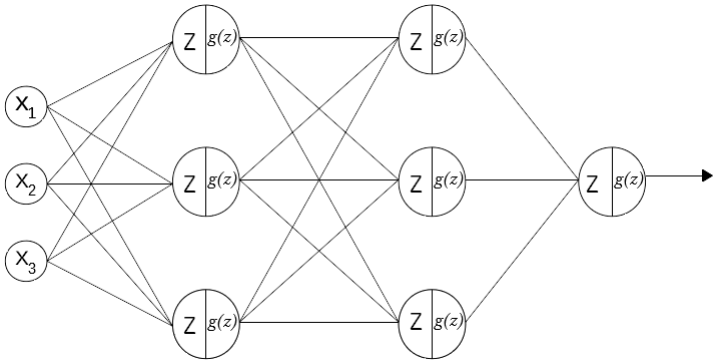
Figure 4: Linear separability



2.2 Networks & Backpropagation

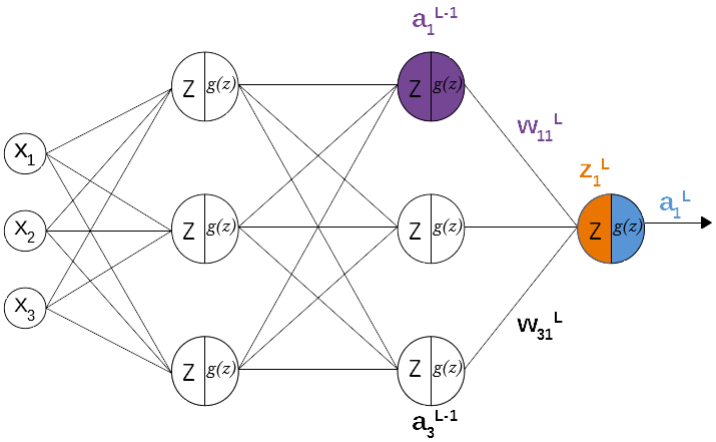
The use of Neural Networks allows us to find non-linear decision boundaries and separate classes that could not be indentified by a Perceptron. Neural Networks are a series of Perceptrons interconnected with each other. The Neural Network consist in one entry layer, one output layer and intermediate "hidden layers" which size may vary. The size of a layer is determined by the number of neurons and the ultimate size of the network is determined by the size of each layer multiplied by the number of layers. The following example is a Neural Network with one entry layer of 3 neurons, two hidden layers of 3 neurons each and one output layer of 1 neuron.

Figure 5: Neural Network with two hidden layer



The core principles are identical to the Perceptron, although one major modification has to be made. The gradient descent algorithm requires the ability to compute all partial derivatives with respect to each weights. Because the first weights, on the left side of the network, are passed through an activation function before reaching the next layers, we need this activation function to be differentiable[5]. In that case, we can apply the chain rule to compute partial derivatives from one end of the network to the other. Let us first define the notation.

Figure 6: simple network



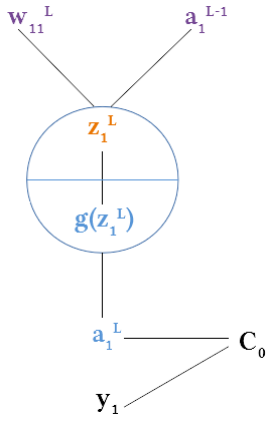
[5]We give further details on why that is when we compute partial derivatives

We associate each weight w to the two nodes it is connecting such that w_{jk}^L connects node j in the previous layer to node k in the current layer. The current layer is denoted as L and previous layers are denoted as $L-1$, $L-2$, ...,$L-M$. We also define the loss function as C_n, $n = 1, 2, ..., N$, N being the sample size and the activation function as g. The weighted sum of inputs arriving to node j is denoted as z_j^L and the output from node j is denoted as a_j^L. To use gradient descent we need to compute the gradient of the loss function to minimize. The gradient is therefore expressed as:

$$\nabla(C) = \begin{bmatrix} \frac{\partial C_n}{\partial w_{j,k}^L} \\ ... \\ \frac{\partial C_n}{\partial w_{J,K}^{L-m}} \end{bmatrix} \tag{2}$$

We will show how to compute the gradient for one particular data point $n = 0$. We need to find the partial derivative of the loss function with respect to each weight in the network. For this example we will use the Mean Squared Error but other functions are appropriate to classification problems. This will be a two step procedure; first we compute partial derivatives with respect to weights on the last layer (output layer $L-1$). Then we generalize the results to every weight in the network. For simplicity, we assume the penultimate layer $(L-1)$ contains only one neuron. The last layer can then be represented as:

Figure 7: output layer



This diagram is inspired by the work of Grant Sanderson (3Blue1Brown) on backpropagation.

The diagram shows how the output from node 1 in layer $L-1$ is multiplied by a particular weight $w_{1,1}^L$ to obtain z_1^L, $z_1^L = \sum w_{i,j}^L * a_j^{L-1}$. z_1^L is passed throught the activation function $g(z_1^L)$ and produces the output for node 1 in layer L, a_1^L. Finding partial derivatives with respect to weights linking the last hidden layer $L-1$ to the output layer L is straightforward because we can decompose the effect of weight $w_{1,1}^L$ with the chain rule:

$$\frac{\partial C_0}{\partial w_{1,1}^L} = \frac{\partial C_0}{\partial a_1^L} * \frac{\partial a_1^L}{\partial z_1^L} * \frac{\partial z_1^L}{\partial w_{1,1}^L} \tag{3}$$

Because the loss function is a function of the output given by the network, which is a function of the activation of the previous node which in turn, is a function of the weights, the partial derivative of the loss function is equal to the product of the partial derivatives of the functions that compose it. The partial derivative of the loss function with respect to the weight $w_{1,1}^L$ is equal to the product of the partial derivative of the loss function with respect to the output of the last layer, the partial derivative of the output of the last layer with respect to the weighted sum z_1^L and the partial derivative of the weighted sum with respect to the weight $w_{1,1}^L$. We yet have to compute these derivatives.

- $\frac{\partial C_0}{\partial a_1^L}$, the loss function is equal to: $(a_j^L - y_j)^2$. Therefore its partial derivative with respect to a_j^L is: $2(a_j^L - y_j)$.
- $\frac{\partial a_1^L}{\partial z_1^L}$, the output a_1^L is the result of the activation function evaluated at z_1^L. We can also write; $a_1^L = g(z_1^L)$ Therefore its partial derivative is equal to the partial derivative of the activation function, $g'(z_1^L)$. This is the reason why we need differentiable activation functions such as the sigmoïd.
- $\frac{\partial z_1^L}{\partial w_{1,1}^L}$, z_1^L is the weighted sum of the previous output multiplied by the weight $w_{1,1}^L$, a slight change in the weight will cause z_1^L to change by the value of the previous output, a_1^{L-1}. The partial derivative of z_j^L with respect to the weight $w_{j,k}^L$ is equal to the previous output[6].

We were able to compute one element of the gradient vector, for one particular training example:

$$\frac{\partial C_0}{\partial w_{1,1}^L} = 2(a_j^L - y_j) * g'(z_i^L) * a_1^{L-1} \tag{4}$$

Recall that we are using MSE to measure loss, MSE is the average error on all data points hence we need to compute a value for each data point $n = 0, ..., N$. The gradient of the average loss is equal to the average gradient of the loss for each example therefore we repeat the process for each data point in the training set to obtain one element of the gradient vector which will be the average gradient for all points:

$$\frac{\partial \bar{C}}{\partial w_{j,k}^L} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial C_n}{\partial w_{1,1}^L} \tag{5}$$

This is only true for the partial derivatives with respect to weights on the last layer $L-1$, because the loss function is a direct function of the output from the last layer

[6]We use indices j and k as this is true for every weight

$C = C(a_j^L, y_j)$. However, it is not a direct function of the outputs from previous layers. This is where backpropagation is used, to rewrite $\frac{\partial C_0}{\partial a_j^{L-m}}$, $m \neq 0$ such that C_0 becomes a function of a_j^{L-m} using the chain rule. We want to solve:

$$\frac{\partial C_0}{\partial w_{j,k}^{L-1}} = \frac{\partial C_0}{\partial a_j^{L-1}} * \frac{\partial a_j^{L-1}}{\partial z_j^{L-1}} * \frac{\partial z_j^{L-1}}{\partial w_{j,k}^{L-1}} \tag{6}$$

$\frac{\partial a_j^{L-1}}{\partial z_j^{L-1}}$ and $\frac{\partial z_j^{L-1}}{\partial w_{j,k}^{L-1}}$ are derived exactly like if they were on the last layers as a_j^{L-1}, z_j^{L-1} and $w_{j,k}^{L-1}$ are direct functions of each other ($a_j^{L-1} = g(z_j^{L-1})$ and $z_j^{L-1} = \sum w_{j,k}^{L-1} * a_j^{L-2}$). We need to rewrite $\frac{\partial C_0}{\partial a_j^{L-1}}$ to be able to compute the gradient and identify the effect of weight $w_{j,k}^{L-1}$ on the loss function. Although the loss function is not a direct function of nodes that are not on the last layer, every node plays a role in the activation of all further neurons. Eventually, each neurons impacts the last output neuron, which in turn is a variable on which the loss function directly depends. The idea of backpropagation is to take advantage of the structure of the network to reverse engineer the propagation of the activation of neurons to be able to compute partial derivative of the loss functions with respect to all weights in the network. We would like to remind the reader that with all the partial derivatives we can calculate the gradient and optimize the weights in order to minimize the loss function. The first step to reverse engineer the network is to find $\frac{\partial C_0}{\partial a_j^{L-1}}$ by applying the chain rule on that expression:

$$\frac{\partial C_0}{\partial a_j^{L-1}} = \sum \left[\frac{\partial C_0}{\partial a_j^L} * \frac{\partial a_j^L}{\partial z_j^L} * \frac{\partial z_j^L}{\partial a_j^{L-1}} \right] \tag{7}$$

We already demonstrated how to calculate $\frac{\partial C_0}{\partial a_j^L} = 2(a_j^L - y_i)$ and $\frac{\partial a_j^L}{\partial z_j^L} = g'(z)$. We will now focus on the new term:

$\frac{\partial z_j^L}{\partial a_j^{L-1}}$, the partial derivative of the weighted sum with respect to previous output is equal to the weight $w_{j,k}^L$ and $\frac{\partial C_0}{\partial a_j^{L-1}} = 2(a_j^L - y_i)g'(z)w_{j,k}^L$. Overall we were able to calculate the impact of the weights of the previous layer by applying a chain rule inside another chain rule:

$$\frac{\partial C_0}{\partial w_{j,k}^{L-1}} = \underbrace{\frac{\partial C_0}{\partial a_j^{L-1}}}_{2(a_j^L - y_i)g'(z)w_{j,k}^L} * \frac{\partial a_j^{L-1}}{\partial z_j^{L-1}} * \frac{\partial z_j^{L-1}}{\partial w_{j,k}^{L-1}} \tag{8}$$

This is where the idea of backpropagation stems from, to compute partial derivatives with respect to previous weights, one needs to compute the partial derivatives with respect to the last weights. The gradient must be calculated on the last layers first, and then the process can be expanded backward until the beginning of the network using the chain rule. The computation of partial derivatives is repeated for each data point and

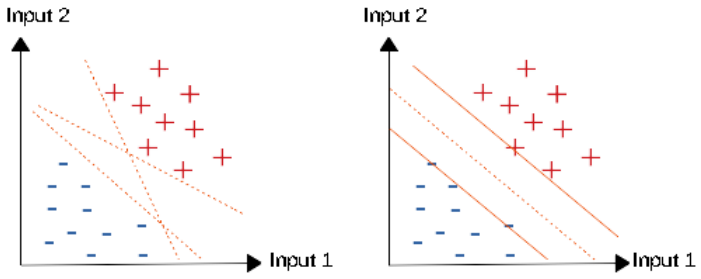
the average derivative is calculated $\frac{\partial \bar{C}}{\partial w_{j,k}^L} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial C_n}{\partial w_{j,k}^L}$ before adding it to the gradient vector. After iterating through the entire data set, the gradient vector is used to update weights such that we take a step towards a local minimum of the loss function. The same stopping rules we discussed before are also applied here to stop the gradient descent.

We have seen how Neural Networks treat the information using a simplified example involving a Perceptron. We also explained how the model finds the optimal parameters by gradually minimizing the loss function by taking iterative steps following the gradient (inverse gradient). We then generalized the results to Neural Networks and showed how backpropagation can be used to solve the same problem with more complex networks.

2.3 Support Vector Machine & the kernel trick

This section is dedicated to Support Vector Machines (SVM) and their inner workings. SVM are machine learning algorithms designed to find the best decision boundary that correctly classifies data. Once we know the equation for the decision boundary, we can use it as a decision rule for new data points that are not yet classified. Depending on which side of the boundary the data point lies, it will be associated to the corresponding class. We represent the decision boundary with an orange dotted line in the following diagram, this boundary is supposed to separate samples belonging to the positive class (red pluses) from samples belonging to the negative class (blue minuses)[7].

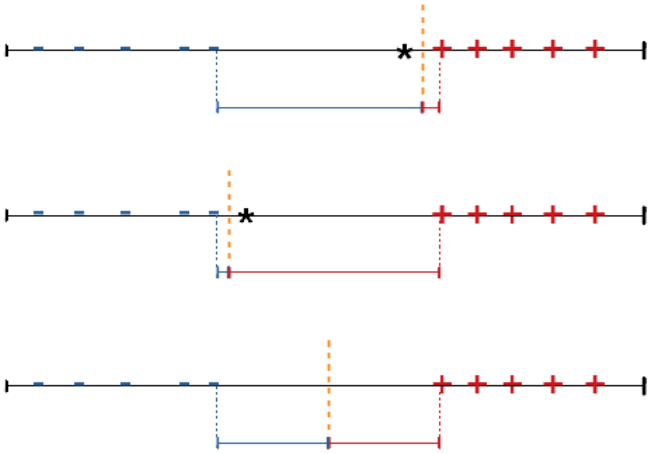
Figure 8: Support Vector Machine



As we can see on the first diagram, there are a lot of different decision boundaries that can be drawn, all of the ones represented above are valid candidates as they correctly separate the two classes. The question then arises; which one should we prefer and why ? The SVM rational suggests that the best decision boundary is the one that is as far as possible from the closest positive and negative samples such as on the second diagram. We can illustrate why this is a reasonable choice with a simple example where data is one dimensional.

[7]This representation is heavily inspired by the lecture given by P.Winston at the MIT, available online.

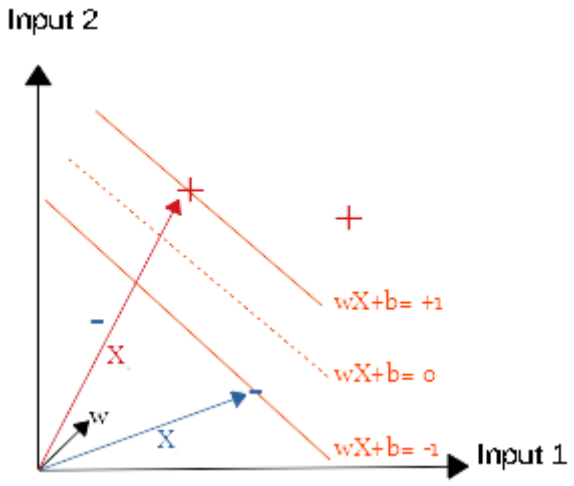
Figure 9: Intuition behind SVM



If we use a decision frontier too close to the positive samples any new observation (represented with a star) that lies slightly to the left of the frontier will be associated to the minus class whereas it is much closer to the positive samples and we can easily assume this new observation should be classified as such. On the opposite, a frontier too close to the negative samples results in the same problem, any new observation lying on the right side of the frontier will be classified as a positive sample even though it is closer to the negative sample. By maximizing the margins represented by the red and blue lines, we are maximizing the space between the decision boundary and the closest negative and positive samples. We obtain a decision frontier equidistant from the two classes that is much more likely to correctly classify new observations.

The objective is then to find the decision boundary that maximizes the margins as it will be the best suited frontier to classify data. Let us define everything in terms of vectors.

Figure 10: Vector representation



If we have a vector $\vec{w}$ perpendicular to the decision boundary, yet unknown, we now the equation for that decision boundary is $\vec{w} \cdot \vec{X} + b = 0$ where b is a constant

representing the intercept and $\vec{X}$ is a vector representing the x, y coordinates in the 2 dimensional space. For a new data point with coordinates $\vec{u}$, if $\vec{w} \cdot \vec{u} + b > 0$, we know $\vec{u}$ lies on the right side of the frontier and will then be classified as a positive sample. On the opposite, if $\vec{w} \cdot \vec{u} + b < 0$, we know $\vec{u}$ lies on the left side of the frontier and will then be classified as a negative sample.

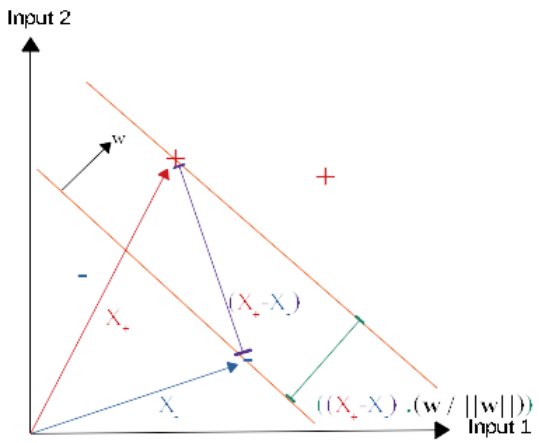
Solving for $\vec{w}$ and b would give us the equation of the decision frontier, however in the current state, we do not have enough constraints to solve for $\vec{w}$ and b. Therefore we need additional constraints to identify $\vec{w}$ and b, we are going to add that $\vec{w} \cdot \vec{X_+} + b \geq +1$ and $\vec{w} \cdot \vec{X_-} + b \leq -1$ where X_+ and X_- are vectors pointing to positive (resp. negative) samples. This new constraints adds that for any positive sample the decision function outputs +1, and -1 in the case of a negative sample. We know have enough constraints to solve for $\vec{w}$ and b.

Because we are heading towards a constrained optimization problem, we will later be in need of a Lagrangian and therefore would like to express the additional constraints as a single constraint equal to 0. In order to achieve this we declare a new variable y_i that takes a value of +1 if $\vec{w} \cdot \vec{X_i} + b \geq +1$, that is the sample is a positive sample, and -1 otherwise. This seems redundant with the constraints we added earlier, but if we know multiply each constraint but y_i we have:

- $\underbrace{y_i}_{1} \underbrace{(\vec{w} \cdot \vec{X_+} + b)}_{\geq 1} \geq +1$
- $\underbrace{y_i}_{-1} \underbrace{(\vec{w} \cdot \vec{X_-} + b)}_{\leq -1} \geq +1$

Both constraints are now the same and we are left with a single constraint expressed as $y_i(\vec{w} \cdot \vec{X_i} + b) - 1 \geq 0$ where X_i, $i = 1, ..., n$ can be either positive or negative. For all samples this constraint will output a value greater or equal to 0, we can further specify that $y_i(\vec{w} \cdot \vec{X_i} + b) - 1 = 0$ for samples on the margins, then the constraint would output 0 for all the samples that are on the positive or negative margins and values greater than 0 for points above the positive margin or below the negative margin.

Figure 11: Intuition behind SVM



If X_+ and X_- are vectors pointing to positive and negative points precisely on the margins[8], if we multiply their difference by a unit vector orthogonal to either margins we can calculate the width of the margin. We would like to remind the reader that at this point we wish to find an equation for a decision boundary such that the gap from the decision boundary to the closest positive and negative samples is maximized. We found how to express this equation as $\vec{w} \cdot \vec{X_i} + b$ and added constraints such that we can solve for $\vec{w}$ and b. As of right now, all boundaries that separate the positive and negative samples is a potential candidate because we only defined the decision function such that all samples are on their correct side of the frontier. We need to reduce this set of candidates to the one that maximizes the margin. We can make $\vec{w}$ into a unit vector by dividing it by its norm, $||\vec{w}||$, because $\vec{w}$ is orthogonal to the margins, the margin itself can be calculated as:

$$margin = (\vec{X_+} - \vec{X_-}) \frac{\vec{w}}{||\vec{w}||} \tag{9}$$

Because $\vec{X_+}$ and $\vec{X_-}$ are on the margin we have $y_i(\vec{w} \cdot \vec{X_+} + b) - 1 = 0$ and $y_i(\vec{w} \cdot \vec{X_-} + b) - 1 = 0$. From here, we know:

- $\vec{X_+} = \frac{1-b}{\vec{w}}$
- $\vec{X_-} = \frac{-1-b}{\vec{w}}$

And,

$$margin = \frac{(X_+ - X_-) \cdot \vec{w}}{||\vec{w}||} = \frac{X_+ \cdot \vec{w} - \vec{X_-} \cdot \vec{w}}{||\vec{w}||} = \frac{1 - b + 1 + b}{||\vec{w}||} = \frac{2}{||\vec{w}||} \tag{10}$$

Maximizing the margin is equivalent to maximizing $\frac{2}{||\vec{w}||}$ or equivalently, minimizing $||\vec{w}||$. For mathematical purposes, we will minimize $\frac{1}{2}||\vec{w}||^2$ instead. This minimizing problem is constrained by: $y_i(\vec{w} \cdot \vec{X_i} + b) - 1 \geq 0$ as discussed earlier. Hence we are looking to solve:

$$\begin{aligned} min : \frac{1}{2}||\vec{w}||^2 \\ \text{s.t. } y_i(\vec{w} \cdot \vec{X_i} + b) - 1 = 0 \end{aligned} \tag{11}$$

This is a standard constrained optimization problem. We will find the maximum margin using a Lagrangian defined as:

$$\mathcal{L} = \frac{1}{2}||\vec{w}||^2 - \sum \alpha_i (y_i(\vec{w} \cdot \vec{X_i} + b) - 1) \tag{12}$$

[8] And not just any point that lies above the positive margin or below the negative margin.

To solve a Lagrangian we start by solving the partial derivatives with respect to each variable (α_i, b and $\vec{w}$).

- $\frac{\partial \mathcal{L}}{\partial \vec{w}} = \vec{w} - \sum \alpha_i y_i \vec{X}_i = 0 \longrightarrow w = \sum \alpha_i y_i \vec{X}_i$
- $\frac{\partial \mathcal{L}}{\partial b} = -\sum \alpha_i y_i = 0 \longrightarrow \sum \alpha_i y_i = 0$

We can plug these two equalities in $\mathcal{L}$:

$$\mathcal{L} = \underbrace{\frac{1}{2}(\sum \alpha_i y_i \vec{X}_i) \cdot (\sum \alpha_j y_j \vec{X}_j) - \sum \alpha_i y_i \vec{X}_i \cdot (\sum \alpha_j y_j \vec{X}_j)}_{=\frac{1}{2}a - a = -\frac{1}{2}a, \text{with a} = \sum \alpha_i y_i \vec{X}_i} - \underbrace{\sum \alpha_i y_i}_{=0} + \sum \alpha_i$$

We arrive at the final result:

$$\mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{X}_i \cdot \vec{X}_j \tag{13}$$

This result tells us that the optimization problem depends on the dot product of pairs of data points $\vec{X}_i \cdot \vec{X}_j$. To find the optimum, we know have to solve for α_i. As we will discuss, this indicates that finding which $\alpha_i \neq 0$ is equivalent to finding support vectors. Once we find the support vectors (points on the marginal hyperplanes, X_+ and X_-), we can easily find the intercept b and the slope of the decision boundary $\vec{w}$.

At this point we would like to take a step back to illustrate the importance of the α_i term. Before we express w and b as sums of other variables we had:

$$\mathcal{L} = \frac{1}{2}||\vec{w}||^2 - \sum \alpha_i \underbrace{f_i(\vec{w}, b, \vec{X}_i)}_{y_i(\vec{w} \cdot \vec{X}_i + b) - 1}$$

In order to maximize this expression with respect to α_i we would like $\sum \alpha_i f_i(w, b, X_i)$ to be as small as possible. We know from previous constraints that $\sum \alpha_i f_i(\vec{w}, b, \vec{X}_i) \geq 0$ hence, to maximize $\mathcal{L}$ we need $\sum \alpha_i f_i(\vec{w}, b, \vec{X}_i) = 0$ as 0 is the smallest possible value that $\sum \alpha_i f_i(\vec{w}, b, \vec{X}_i)$ can take. This is possible in two different scenarios[9]:

1. $f_i(\vec{w}, b, \vec{X}_i) > 0$, which implies $\alpha_i = 0$
2. $f_i(\vec{w}, b, \vec{X}_i) = 0$, which implies $\alpha_i > 0$

These two scenarios tell us that for points X_i on the margin ($f_i(\vec{w}, b, \vec{X}_i) = 0$), α_i is strictly positive, for points not on the margin ($f_i(\vec{w}, b, \vec{X}_i) > 0$), $\alpha_i = 0$. Because $\vec{w}$ is defined as $\sum \alpha_i y_i \vec{X}_i$, $\vec{w}$ only exists ($\vec{w}$ is not null) for points on the margin. For every other points, α_i is equal to 0 then, $\alpha_i y_i \vec{X}_i = 0$ and $\vec{w}$ at that particular point i is null.

To maximize the margin, SVM only has to look at points on the margin, the other points will be irrelevant in the optimization problem. In other terms, once we solve

[9] Both terms cannot be equal to 0

for α_i, whenever $\alpha_i \neq 0$; the point is relevant and is on the margin. Solving for α_i is equivalent to finding points on the margin (support vectors), hence finding margins.

We then only need to differentiate $\mathcal{L}$ with respect to α_i by setting the partial derivatives equal to 0:

$$\max_{\alpha_i} \mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{X}_i \cdot \vec{X}_j \tag{14}$$

We saw that optimizing this expression required that α_i is not null only for points on the margin. Therefore, once we set partial derivatives equal to 0, we will find optimal α_i which, plugged into $w = \sum \alpha_i y_i \vec{X}_i$ will give us the w vector, orthogonal to the decision boundary with the largest margins.

We also know that $\vec{w}\vec{X}_i + b = y$, or $b = y - \vec{X}_i\vec{w}$. This strict equality is only true when $\vec{X}_i$ is on the margin. Hence for a subset s of i, $s \subset i$, for which $\vec{X}_s$ is on the margin, we have $b = y - \vec{X}_s\vec{w}$. Since we solved for $\vec{X}_s$ by solving for α_i we can compute b. The classification problem is solved, for any new point u we have:

$$\begin{aligned} &+ \text{ if } \vec{w}^* \cdot \vec{u} + b^* > 0 \\ &\text{- if } \vec{w} \cdot \vec{u} + b < 0 \end{aligned} \tag{15}$$

$\vec{w}$ and b are known so we can classify every new point according on the hyperplane we found by solving for $\vec{w}$ and b.

2.4 The kernel trick

SVM as we discussed them in the previous section are only suited to linearly separable problems and suffers from the same limits as the Perceptron. The decision boundary and corresponding margins are linear functions, they are only able to trace a linear surface to divide the sample. In cases where the data cannot be separated into strictly homogeneous groups with the help of an hyperplane, the SVM will not be able to correctly classify new observations. To solve this issue, we can map the data to an higher dimensional space where sample are linearly separable, use the SVM to find the linear decision boundary and re-scale the data back to the original dimension. The next diagram illustrates how this may be done for a problem which is not linearly separable in 1 dimension but can be linearly separated in 2 dimensions. Although mapping every single point to an higher dimension is technically possible it is not computationally efficient. Hopefully, because the SVM optimization problem relies on an inner product of two vectors, we do not need to map every point to higher dimensions.

If we have a function ϕ that maps a vector $\vec{X}$ to higher dimensional space where the samples are linearly separable, the lagrangian we had before becomes:

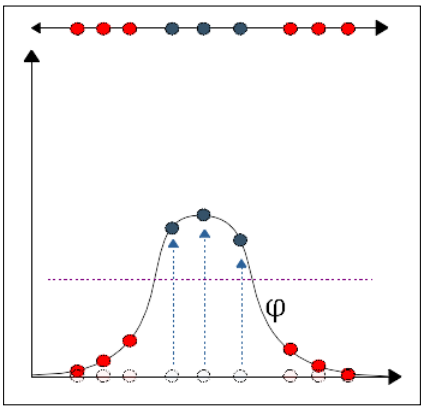
$$\max_{\alpha_i} \mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \phi(\vec{X}_i) \cdot \phi(\vec{X}_j) \tag{16}$$

If we can find a kernel function k that takes the two vectors as inputs and outputs the dot product of those two vectors in higher dimensional space such that $k(\vec{X_i}, \vec{X_j}) = \phi(\vec{X_i}) \cdot \phi(\vec{X_j})$ we can replace ϕ with k:

$$\max_{\alpha_i} \mathcal{L} = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j k(\vec{X_i}, \vec{X_j}) \tag{17}$$

This manipulation allows us to solve for the linear decision boundary in higher dimensional space without needing to map every point, instead, because the optimization depends on a dot product, we can use a kernel function that outputs the result of the dot product of the two transformed vectors. This manipulation referred to as the 'kernel trick' allows us to solve no linear problems without having to map every point to a dimension where a linear separator can be found.

Figure 12: Kernel Trick

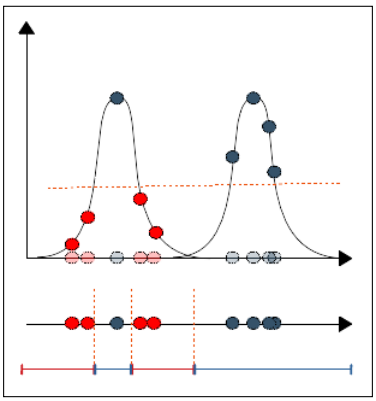


The data in its original 1 dimension is not linearly separable, trying to train a SVM on such data will not yield satisfying results (often, the model will classify all of the data as either one of the classes). We then find a function ϕ that maps all of the points to a different dimension, here for instance we map the data to a second dimension represented by the vertical axis. This function ϕ can be of different forms, here we chose a gaussian function, but other functions such as $\phi(x) = x^2$ would also work in these circumstances. Once every data point has been fed through ϕ, the problem becomes linearly separable in higher dimension and we can find a linear decision boundary represented by the purple dotted line. We can then inverse the ϕ function to bring the data back to its original dimension. As we discussed, the kernel trick performs the same manipulation without having to map every point to an higher dimensional space and saves computational time, but this representation allows us to clearly understand the benefit of expanding the data to higher dimension when it is not linearly separable in its original dimension.

In this example, $k = \phi(\vec{X_i}) \cdot \phi(\vec{X_j}) = e\left(\frac{||\vec{X_i} - \vec{X_j}||^2}{2\sigma^2}\right)$. The value of k depends on the value of σ which is a user-defined constant. The efficiency of the kernel trick depends on σ which can be seen as the width of the bell curve drawn above. If σ is too large, the transformation may not yield linearly separable data because the curve is very flat. On the other hand if σ is too small, it may induce over-fitting which is the detection of non-generalizable patterns in the data. One example is repented below, small σ yields

very steep curves.

Figure 13: Kernel Trick



The blue dot on the left is likely to be an outlier or even an error, when training the model we wish to ignore this data point as it is not reliable. We could find a linear decision boundary in the original 1 dimensional data, considering this point is an outlier, and that decision boundary is likely to represent the general behavior of the data. However if we use the gaussian kernel function and choose a small value for σ, we will consider the occurrence of the left blue dot as a significant pattern in the data which leads to a non-linear decision boundary. We can imagine how a linear decision boundary would be more efficient, every observation falling on the right side will be associated to blue and all observations falling on the left side will be considered as red. With the non-linear decision boundary however, we will have an interval on the left side inside which new observations are categorized as blue because the model considered the occurrence of the blue dot as a significant pattern in the data.

3 Application

In previous sections we discussed Neural Networks and Support Vector Machines, two models designed to classify data[10]. We are know going to put those models in practice to try to predict stock prices. The model will have to determine if future prices are either above, or below current price, which leads to a buying or selling decision. We will note whether or not our model trained on technical indicators is able to predict future prices, which would encourage us to think the markets are not totally efficient. We start with an exploratory analysis of the data and we proceed to model the information to hopefully capture deterministic patterns in stock prices.

3.1 Data analysis

The data is obtained from the Yahoo Finance API. We are using daily close prices adjusted for stock splits and dividends. We make the assumption that the market is not

[10]We can use ANN for regression but we did not cover this part as we will be addressing a classification problem.

unique and different assets have different behaviors. Volatile assets are often considered as more reactive to technical indicators than very stable assets[11]. We can impute this empirical phenomenon to the fact that volatile assets attract speculative investors whom are prone to use technical indicators to time their trades. Whereas stable assets attract long term investors who might prefer large scale macro-economic variables to build their portfolio. We can already see how, considering this hypothesis, different variables are used for different markets.

For this particular reason, we think it is reasonable to test our models on assets derived from different markets to asses whether or not these different behaviors can be highlighted. Nonetheless we will focus on US and Europe based companies. We start by listing all the assets we are going to focus on in this study:

1. **Air France-KLM (AF.PA)** is a Franco-Dutch airline holding company born from the merge of two national airlines; Air France and KLM Royal Dutch Airline. Air France-KLM is one of the largest airline in the world and both French and Dutch governments are shareholders with approximately 14% each which makes it a very sound company. During the Covid-19 pandemic the company suffered from substantial losses due to the heavy decrease in air travel, funds have then been injected by the French government to help maintain Air France-KLM afloat.

2. **Tesla (TSLA)** is currently the most famous electric car manufacturer in the world. Tesla generates a lot a attention from technology enthusiasts and investors. The newly born company[12] directly issued from the silicon valley has seen its market value skyrocket since 2012. Some consider the valuation of the company does not reflect the true economic performance of Tesla, which is, along with speculative trading, one of the reason why Tesla's stock is famously volatile.

3. **Apple (AAPL)** combines both aspects, the company is innovative and currently has a market capitalization exceeding 2 trillion dollars. But Apple is also considered as a sound investment because of its seniority and its deep implantation in the market for new technologies.

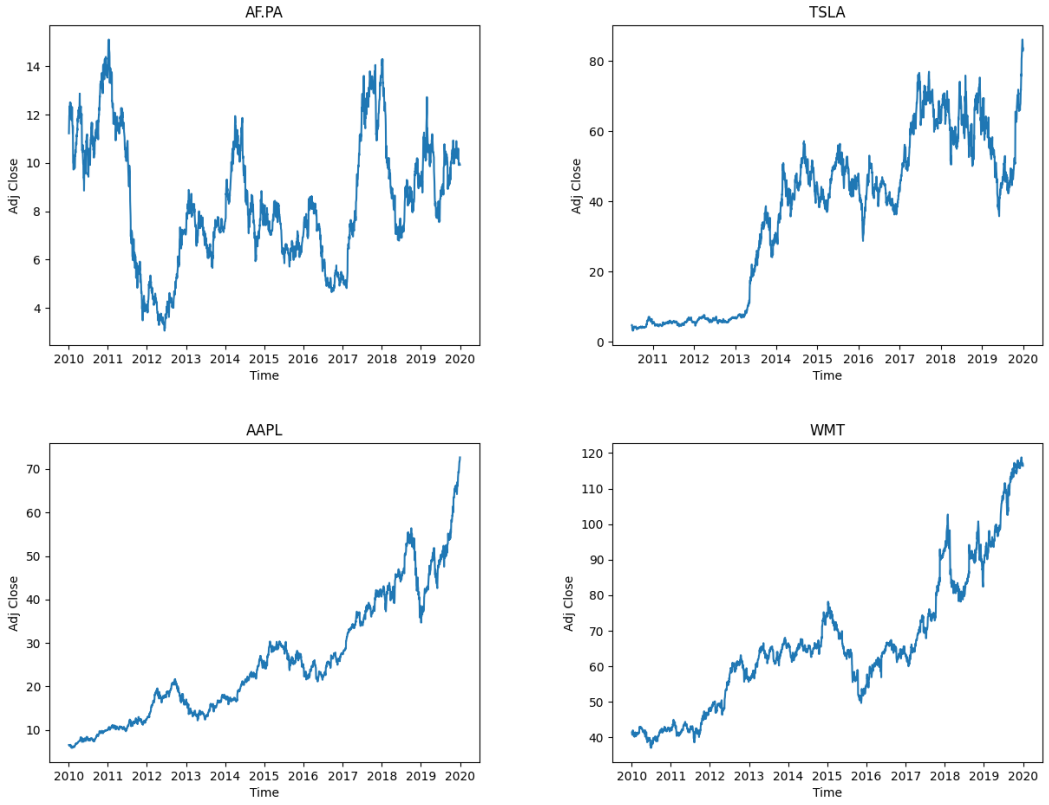
4. **Walmart (WMT)** is the largest retail corporation by revenue in 2020. Walmart is one the most notorious US-based firms in the world which makes it a good source of comparison with other assets.

We tried to choose these assets with respect to sectors, overall market sentiment on the valuation of the companies and volatility to reflect some of the disparities one can observe on financial markets. The data is gathered from 2010 August 8^{th} to 2019 December 31^{st}, for a total of 2371 observations per asset. The period is chosen to reflect normal market conditions, post 2008 crisis and pre 2020 crisis. Although choosing this particular period help us get rid of abnormalities which makes the training more efficient, it does not reflect the entirety of the market conditions. Our model will therefore be incapable of predicting this kind of rare black swans events, which constitutes a major drawback to our approach.

[11] *Stock return predictability: Evidence from moving averages of trading volume* - Pacific-Basin Finance Journal 65 (2021) - Yao Ma , Baochen Yang, Yunpeng Su

[12] Tesla was founded in 2003

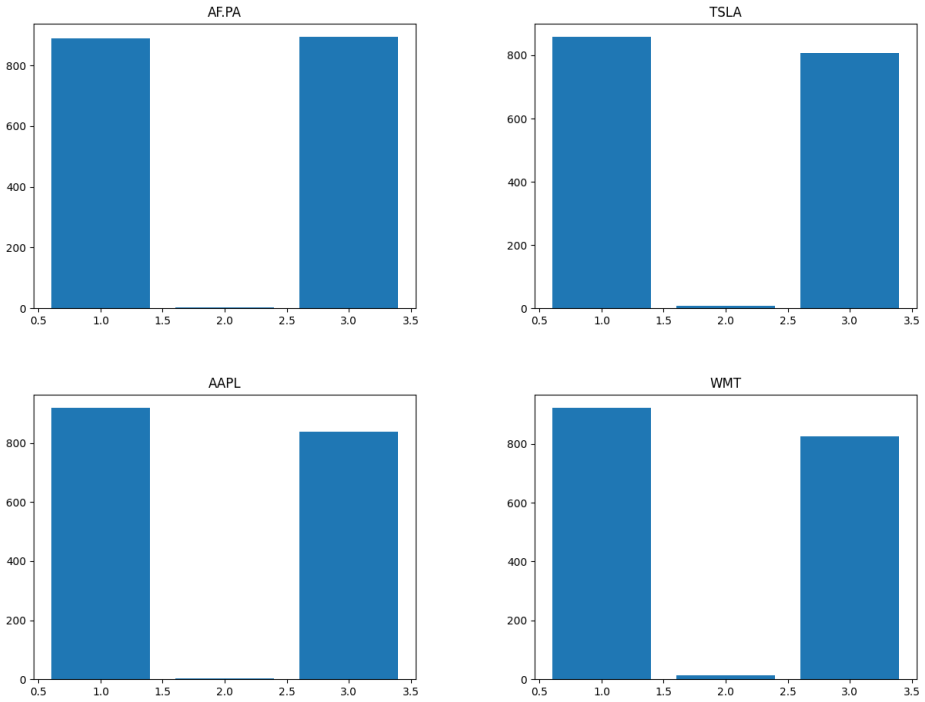
Figure 14: Series from 08/02/2010 to 31/12/2019



As we discussed in the previous sections, we will be using classification models. Therefore, we need to encode the close price into a categorical variable. We can create a new variable that takes a value of 1 when then next day price is superior to the current price, 2 if both prices are equal and 3 if the future price is below the current price. Although occurrences of the second case are very rare, we need to consider it. We will see later how if we can transform the problem such that we are able to use binary classifiers, which will be easier to interpret and more probably more efficient. The next figure represents the frequencies of appearance of each case for all four series. We will later refer to this variable as **position**.

The data set will be split into a training set and a testing set. We conserve 70% of the data for training and assign the remaining 30% to testing. The models will be trained on data spanning from 08/02/2010 to 02/03/2017 for a total of 1640 observations. The testing set starts on 02/04/2017 and ends on the 31/12/2019 and contains 703 observations.

Figure 15: Buy and Sell frequencies



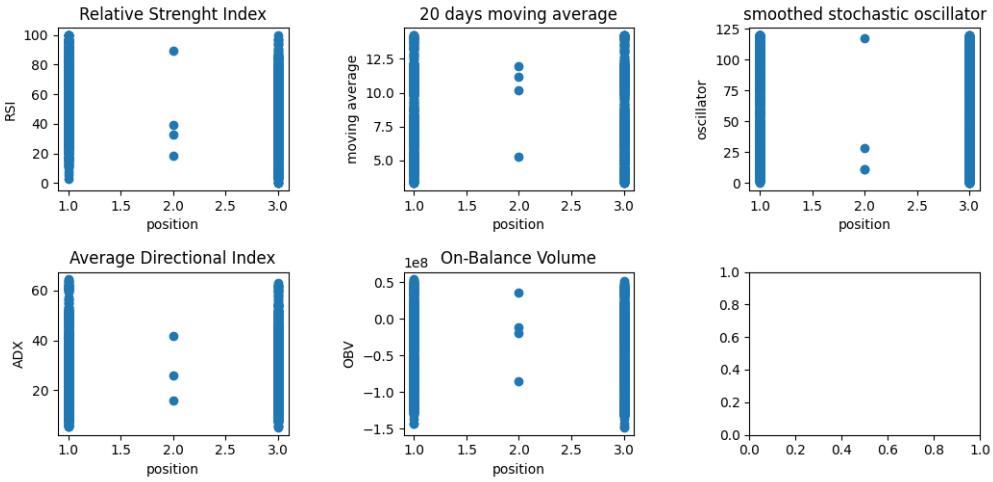
We can see how days where price is identical to the price the prior day are very rare. We fear that having this third category carries more noise than it carries useful information, that is the reason why we might consider transforming the problem into a 0/1, binary problem. Furthermore, positive returns and negative returns are represented equivalently in all four series with approximately 50% of the total number of days where returns where positive and the other 50% where returns where negative. This is one desirable property, as imbalanced categories induce a natural bias in terms of precision. If 90% of the sample belong to a certain class and 10% to the opposite class, even if the model is not able to classify data and assumes all observations belong to the first class, it will attain a 90% accuracy. Even with 90% accuracy, we can see how the model is not able to differentiate both classes. By only considering the percentage of correctly classified observations, we are encouraging the model to behave badly while giving a false impression of efficiency. Depending on how the model is trained, what loss functions are used and overall which error metric we choose, we can obtain very different results. In our case the buy (1) and sell (3) classes are very well balanced, which carries convenient properties, and further encourages us to treat days where the return is exactly 0.

Among all assets, we notice more positive returns than negative returns for **WMT** and **AAPL**. Although very well balanced we think it is interesting to note. We saw earlier how **WMT** and **AAPL** have the strongest positive trends whereas **TSLA** and **AF.PA** have a more subtle upward trends and are more volatile. The slight asymmetry in **WMT** and **AAPL** returns is in part responsible for the positive trends the two assets exhibit.

présenter les variables ici

In order to start our analysis, we visualize the **position** with respect to the predictors in a two dimensional figure. This will help us determine whether or not the distribution of position is conditional on the predictors, which means the variable will change depending on the value of the predictor considered. In the other scenario, the **position** is independent from the predictor, meaning the value of the predictor on a certain day does not provide information on future price. Visual assertion of a relation between multiple variables is not sufficient to assume that variables are linked, we further need to assess the statistical significance of the relation.

Figure 16: Conditional distributions



Note: We only present a subset of the variables for one of the assets. The remaining figures are presented in the annexes.

At first glance, it seems the variables are uncorrelated with **position**. If we look at the RSI for example, we assumed that high values are associated with a 'sell' signal whereas small values are associated with 'buy' signals. Conversely, if **position** $= 1$ we should have small RSI values, and high RSI values if $position = 0$. However the RSI spans on the same range of values when we consider days where the price increased and days where the prices decreased, there is no distinct difference in the RSI depending on the right position. The same observations can be applied to the other predictors and can also be generalized to all four assets we are considering. This finding coincides with the EMH which states that since all the available information is already incorporated in the prices, technical indicators cannot anticipate future prices. Also, indicators are not supposed to be used on a daily basis, the theory behind it suggests that we should look for crossing points to detect trends, patterns. We will try to transform the variables such that instead of considering the absolute values we evaluate certain conditions to extract information from the indicators as it is being used by practitioners on the markets.

3.2 Modelling

However we see no correlation between our variables vis a vis **position**, we need to statistically test for absence of significance of the variables to confidently establish that there is

no residual information in the indicators. to fulfill this aim, we will need a model capable of generating hypothesis tests, which we cannot obtain with machine learning models. We will then try to classify **position** using a logit model. One important note is that logit is a linear model. It is possible to have no linear dependencies between the indicators and **position** while still having non-linear dependencies between the variables. Although logit model will help us determine the nature of the relationships in our data set, it is limited by its linearity. We report the results is the table below

Table 1: Logit Regression

Asset : WMT		
LLR : $1.49e^{-16}$		
Variable	Coeff.	p-value
RSI	-0.0390	0.125

Dep. Variable:	position		No. Observations:		1742	
Model:	MNLogit		Df Residuals:		1726	
Method:	MLE		Df Model:		14	
Date:	Thu, 08 Apr 2021		Pseudo R-squ.:		0.1007	
Time:	18:18:36		Log-Likelihood:		-1144.4	
converged:	False		LL-Null:		-1272.5	
Covariance Type:	nonrobust		LLR p-value:		1.491e-46	
position=2	coef	std err	z	P>|z|	[0.025	0.975]
RSI	-0.0390	0.025	-1.536	0.125	-0.089	0.011
D	0.0230	0.013	1.726	0.084	-0.003	0.049
MA	-1.718e+10	9.24e+05	-1.86e+04	0.000	-1.72e+10	-1.72e+10
boll_up	8.591e+09	4.61e+05	1.86e+04	0.000	8.59e+09	8.59e+09
boll_dw	8.591e+09	4.6e+05	1.87e+04	0.000	8.59e+09	8.59e+09
MACD_short	-0.0463	0.015	-2.993	0.003	-0.077	-0.016
MACD_long	0.0213	0.007	2.992	0.003	0.007	0.035
adx	-0.0282	0.027	-1.036	0.300	-0.082	0.025
OBV	-7.177e-09	3.69e-09	-1.948	0.051	-1.44e-08	4.59e-11
position=3	coef	std err	z	P>|z|	[0.025	0.975]
RSI	-0.0648	0.005	-13.526	0.000	-0.074	-0.055
D	0.0250	0.002	10.174	0.000	0.020	0.030
MA	-1.0882	2.19e+04	-4.97e-05	1.000	-4.29e+04	4.29e+04
boll_up	0.5701	1.09e+04	5.21e-05	1.000	-2.14e+04	2.14e+04
boll_dw	0.5459	1.09e+04	4.99e-05	1.000	-2.14e+04	2.14e+04
MACD_short	0.0009	0.003	0.358	0.720	-0.004	0.006
MACD_long	-0.0004	0.001	-0.360	0.719	-0.003	0.002

Among all the assets, we were only able to estimate a logit regression for **WMT**. Other assets yielded NaN values for the coefficients and their standard error. We suppose this is due to very high multicolinearity among the predictors. Bollinger bands, which are represented by **MA**, **boll_up** and **boll_dw**, represent the positive and negative limits of a two standard deviations interval centered around a moving average, they are constructed as a function of the moving average, hence, a completely dependent. Because of such multicolinearity, we cannot estimate the logit regression. Nonetheless, the estimation works with **WMT** data, but the MLE does not converge, and the multicolinearity is still present therefore the coefficients are biased and their standard error along with corresponding statistical tests are also biased. The first panel indicates the increasing or decreasing probability of switching from a buy position to a hold position. We can see how coefficients for **MA**, **boll_up** and **boll_dw** are very large with very small standard errors leading to a reject of the null hypothesis of no explanatory power[13]. These results further indicate presence of multicolinearity especially between these three variables. The smoothed stochastic oscillator **D** is significant at a 10% confidence level, both 20 days and 6 days moving averages **MACD_long** and **MACD_short** are significant at a 1% confidence level. Multicolinearity might also play a role in these two variables being significant. The On Balance Volume **OBV** is significant to a 10% confidence level and almost significant to a 5% confidence level. Relative Strength Index **RSI** is not significant neither is the Average Directional Index **ADX**. The Likelihood Ratio indicates the model is better fitter than the null model (p-value ¡ 0.01), telling us our regression has at least some explanatory power on the position. The second panel shows the increasing or decreasing probability of switching from a buy position to a sell position. Here, **RSI** and **D** are the only two statistically significant variables. Both are significant to a 0.01% confidence level. The bollinger bands variables are not significant with very large standard errors. The moving averages are not significant. The different results further indicate the regression is biased, especially the fact that standard errors inverse when considering different positions. Despite the pitfalls in our model, we can believe that the stochastic oscillator and the RSI are still the two indicators that carry the most information on future prices as their values and values for their standard errors are the most rational.

As we saw earlier,technical indicators are often used as signals rather than as they are. Common practice is to establish thresholds which, once crossed upward or downward, send a buy or sell signal. Transforming the variables to replicate this behavior would first coincide with common practice and second would be more appropriate to our problem. As we saw, the raw values of the indicators are not dependent on the **position**, and carry very little if no information on future prices. Transforming the variables might restore some explanatory power. We will transform the variables and run another logistic regression to see if the estimation gives better results, and if the so transformed variable are better suited. In that case, we will try to correctly classify buy and sell opportunities using SVM and ANN and compare it to the logistic regression.

The transformations are consistent with existing practice, we briefly detail our procedure before moving on to the estimation.

- The RSI is divided by four different thresholds; crossing 70 upward, downward and crossing 30 upward and downward. We expect that crossing 70 upward send a sell signal and crossing 30 upward send a buy signal.

[13] H_0: $\beta_i = 0$ against H_a: $\beta_i \neq 0$, with β_i the coefficient of the $i^{th} variable$

- The stochastic oscillator follows the same rules expect thresholds are replaced with 80 and 20 respectively.
- The bollinger bands are transformed such that crossing the upper bound sends a signal and crossing the lower bound sends another signal.
- The MACD are transformed such that when the short 12 days moving average crosses the long 20 days moving average upward the variable send a signal and when the short moving average crosses the long moving average downward sends another signal.
- OBV and ADX are kept as raw values.

We also transformed **position** into a binary variable to get rid of the second class with very limited representation across the dataset. We expect that removing this class will help the different classifiers as the data is not polluted by unnecessary information.

Figure 17: Logit regression with transformed variables

MNLogit Regression Results

Dep. Variable:	position	No. Observations:	1666
Model:	MNLogit	Df Residuals:	1660
Method:	MLE	Df Model:	5
Date:	Sat, 10 Apr 2021	Pseudo R-squ.:	0.007704
Time:	15:44:13	Log-Likelihood:	-1145.9
converged:	True	LL-Null:	-1154.8
Covariance Type:	nonrobust	LLR p-value:	0.003219

position=1	coef	std err	z	P>|z|	[0.025	0.975]
RSI	-0.0938	0.043	-2.174	0.030	-0.178	-0.009
D	-0.1345	0.058	-2.308	0.021	-0.249	-0.020
MACD_long	0.3055	0.196	1.560	0.119	-0.078	0.689
adx	0.0005	0.004	0.148	0.882	-0.007	0.008
OBV	2.606e-09	1.04e-09	2.506	0.012	5.68e-10	4.64e-09
boll	0.1627	0.137	1.189	0.235	-0.106	0.431

MNLogit Regression Results

Dep. Variable:	position	No. Observations:	1640
Model:	MNLogit	Df Residuals:	1634
Method:	MLE	Df Model:	5
Date:	Sat, 10 Apr 2021	Pseudo R-squ.:	0.008317
Time:	15:42:52	Log-Likelihood:	-1126.1
converged:	True	LL-Null:	-1135.6
Covariance Type:	nonrobust	LLR p-value:	0.002015

position=1	coef	std err	z	P>|z|	[0.025	0.975]
RSI	-0.0895	0.041	-2.162	0.031	-0.171	-0.008
D	-0.0122	0.058	-0.210	0.834	-0.126	0.102
MACD_long	-0.2704	0.184	-1.473	0.141	-0.630	0.089
adx	0.0056	0.003	1.852	0.064	-0.000	0.012
OBV	9.823e-12	7.2e-11	0.136	0.892	-1.31e-10	1.51e-10
boll	0.4255	0.124	3.426	0.001	0.182	0.669

MNLogit Regression Results

Dep. Variable:	position	No. Observations:	1640
Model:	MNLogit	Df Residuals:	1634
Method:	MLE	Df Model:	5
Date:	Sat, 10 Apr 2021	Pseudo R-squ.:	0.009676
Time:	15:46:10	Log-Likelihood:	-1123.7
converged:	True	LL-Null:	-1134.7
Covariance Type:	nonrobust	LLR p-value:	0.0005333

position=1	coef	std err	z	P>|z|	[0.025	0.975]
RSI	-0.1012	0.044	-2.287	0.022	-0.188	-0.014
D	0.0062	0.056	0.111	0.911	-0.104	0.116
MACD_long	-0.2709	0.188	-1.440	0.150	-0.640	0.098
adx	0.0099	0.003	3.259	0.001	0.004	0.016
OBV	2.864e-11	1.53e-11	1.870	0.062	-1.38e-12	5.87e-11
boll	0.2515	0.133	1.895	0.058	-0.009	0.512

MNLogit Regression Results

Dep. Variable:	position	No. Observations:	1640
Model:	MNLogit	Df Residuals:	1634
Method:	MLE	Df Model:	5
Date:	Sat, 10 Apr 2021	Pseudo R-squ.:	0.006962
Time:	15:45:08	Log-Likelihood:	-1126.2
converged:	True	LL-Null:	-1134.1
Covariance Type:	nonrobust	LLR p-value:	0.007471

position=1	coef	std err	z	P>|z|	[0.025	0.975]
RSI	-0.0829	0.041	-2.001	0.045	-0.164	-0.002
D	0.0591	0.053	1.109	0.267	-0.045	0.164
MACD_long	-0.3365	0.195	-1.726	0.084	-0.719	0.046
adx	0.0007	0.004	0.183	0.855	-0.007	0.008
OBV	4.932e-11	3.36e-10	0.147	0.883	-6.09e-10	7.07e-10
boll	0.4284	0.134	3.208	0.001	0.167	0.690

From the previous table we notice a first improvement compared to previous attempt as the logit regression was estimated for all assets and all converged to the maximum likelihood. The likelihood ratio tests indicates all models have at least some explanatory power on their respective asset. However the pseudo R^2 indicate the models explain a very small fraction of the total variation of the **position** variable. Less than 1% for all models. From these first observations, we can assume technical indicators have very little correlation with future prices. The RSI is significant to a 5% level for all assets, the Bollinger bands are significant to a 1% level for **WMT** and **TSLA** and to a 10% level for **AAPL**. As mentioned in the earlier sections, we expected higher significance when considering "young" companies such as **AAPL** and **TSLA**, especially as both operate in

the technological sector, usually more prone to speculative and short-term trading. The results seem to partially contradict our initial hypothesis. The ADX is significant for **AAPL** and **TSLA** (to a 5% and 10% respectively) and not significant for **WMT** and **AF.PA**, which coincides with our initial statement. The stochastic oscillator is only significant for **AF.PA**, at a 5% level, and is not significant for every other remaining assets. The MACD is only significant to a 10% for **WMT** and is almost significant to a 10% level for **AF.PA**, **TSLA** and **AAPL**. The OBV is significant for **AF.PA** and **AAPL** (to a 5% and 10% levels respectively). Overall, results indicate that different assets react to different indicators, the Bollinger bands, the MACD and the RSI are the thre indicators with highest significance overall. On reason to explain this would be that those particular indicators are among the most commonly considered by traders. It is responsible to assume that if a substantial fraction of investors on a market follow one indicator to place trades, the resulting supply and demand will cause the price to move accordingly. ADX seems to be significant when used on more volatile markets. The likelihood ratio also suggests similar observations, the fit of the model is best for **AAPL** and **TSLA** which also both have the two smallest likelihood ratio. We partially confirm that indicators play an higher role in relatively newer and unstable markets, but some observations contradict this hypothesis. Finally, even with very little explained variance, the models are all delivering useful information on future prices. With linear models, such as logistic regression, the predictions might not be salable to substantial economic gains, but maybe the inclusion on non-linearities is necessary to emcopass the relationship between technical indicators and future prices.

We would also like to note that macroeconomic variables play a role in stock prices modeling, as well as many other variables such as weather, period of the year, business cycles etc... the omission of all determining variables is a pitfall in our model. First the coefficient of the selected variables are therefore biased (omitted variable bias), and second, our predictions will consequently be worse as we explain less of the variance. The inclusion of aforementioned types of variable definitely is a reasonable improvement to the model to consider.

We can now train a Neural Network and a Support Vector Machine to classify "buy" and "sell" positions based on the transformed technical indicators. We will begin with a relatively network with two hidden layers of 10 neurons with relU activation functions and a final output layer of 2 neurons and a softmax activation function. Each of the final neurons will output the probability that the correct is "buy" first the first one and "sell" for the second, the final prediction then is the maximum of these two values. We use Sparse Categorical Cross Entropy as the loss function to optimize and the Adam optimizer[14]. All parameters can and will be updated to improve the model. The performance will be assessed using the confusion matrix, precision, recall and f1-score. For clarity the results for each assets are presented separately.

Table 2: Confusion matrix

		True	
		0	1
Pred	0	576	398
	1	258	434

[14]Adam is a stochastic gradient descent algorithm with adaptive learning rate which helps converge to global minimums more accurately and limits risk of overshooting

Table 3: Caption

Standard Neural Net				
	TSLA	AF.PA	AAPL	WMT
P	0.60	0.61	0.58	0.56
R	0.69	0.61	0.60	0.56
F_1	0.56	0.60	0.58	0.56
Optimized Neural Net				
P	0.65	0.68	0.66	0.66
R	0.66	0.69	0.67	0.66
F_1	0.65	0.67	0.66	0.66
Linear SVM				
P	0.53	0.58	0.56	0.57
R	0.56	0.59	0.59	0.60
F_1	0.47	0.58	0.54	0.55
Kernel (non linear) SVM				
P	0.65	0.62	0.61	0.65
R	0.67	0.63	0.67	0.69
F_1	0.64	0.62	0.58	0.63

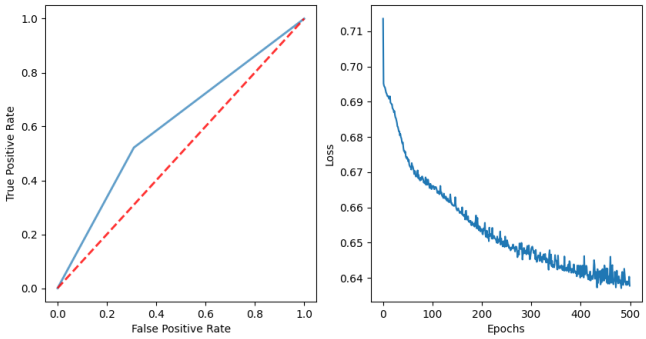
The first Neural Network is able to classify the data, although it is far from perfect. For this asset (**AF.PA**) we have a total of $576 + 434 = 1010$ correctly classified observations against $258 + 398 = 656$ misclassified ones. This translates to 60.62% of the sample correctly classified against 39.38% misclassified. The Recall is 61%, among all days were we should have bought, the model identified 61% of those days as buy. The Precision is also 61% which means among all observations classified as buy, 61% of them where actually buy. Precision and Recall give us a comprehensive measure of the model performance. They are calculated as $Recall = \frac{TP}{TP+FN}$ and $Precision = \frac{TP}{TP+FP}$ where TP is the number of True Positives, FP is the number of False Positives and FN is the number of False Negative. If the model is unable to classify the date and associates 100% of the sample to one class at random the overall accuracy is 50%[15]. By associating all the observations to one class, if this class represents 50% of the sample, then 50% of the observations are correctly classified and 50% are misclassified, they belong to the other class but were associated with this first class. Precision and Recall offer a way to counteract this false sentiment of accuracy by using a ratio of cases where the model correctly classified the data and cases where the model did not correctly classify the data. Precision help us see among all the observation associated to one class, how many are actually belonging to that class whereas Recall tells us among all the observations that should have been associated to one class, how many did the model indeed identify. Recall and Precision are calculated for each class and then averaged to obtain the overall Precision and Recall. Sometimes, we want to concatenate Precision and Recall into one single metric to ease the analysis of the performance of the model. In that case, we can use the f1-score, which is simply an harmonic mean of Precision and Recall. f1-score is then calculated as: $f_1 score = \frac{Precision * Recall}{Precision + Recall}$.

For this asset, we obtain 61% Precision and Recall, meaning on average (considering buy and sell), the model is right more times than it is wrong. Among all the

[15]This is true if and only if classes are perfectly balanced.

observations classified as buy or sell, 61% where actually buy or sell. And among all the true buy or sell 61% where identified by the model. The f1 score is 60% suggesting the model is able to separate buy and sell with help of technical indicators.

Figure 18: Results



The ROC curve (Receiver Operating Characteristic curve) indicates the ratio of True Positives divided by False Positives for different thresholds. The threshold is the probability at which an observations switches from one class to the other. If the threshold is 0.5, if the modeled probability that observation i belongs to the first class exceeds 0.5, it will be associated with the first class. Varying thresholds result in different classification schemes, independent of the model itself which is simply designed to predict the probability. ROC indicates the performance of the model for each possible threshold between 0 and 1. The bisector is the line where, for any threshold, the number of True Positives is equal to the number of False Positives. If we are above the bisector, the model is more often right than it is wrong (the number of True Positives is superior to the number of False Positives). If we are below the bisector, the model is more often wrong than it is right (the number of True Positives is inferior to the number of False Positives). For this reason we also look at the value of the are between the ROC curve and the bisector, also called the ROCauc. The larger the ROCauc, up to 1, the more accurate the model is, the smaller the ROCauc is, down to -1, the worst the model is. If we have a 0 ROCauc, the ROC and bisector are confounded and the model is not able to classify data. Note that not being able to classify (ROCauc = 0) is different from being wrong 100% of the time (ROCauc = -1). Here we have a slighlty positive ROCauc, indicating the model is able to correctly classify the data to some extent. This is not sufficient to disprove the EMH, first, we need to make sure the model is able to classify data it was not trained on, as the results presented above are obtained from the training data. Second we need to make sure this accuracy is economically scalable, meaning we can achieve greater returns by trusting the model instead of buying and holding a large index such as the S&P 500.

The last figure shows the loss with respect to the number of epochs. As we explained in the first section, we should see a decrease in the loss functions as the number of epochs increases. This illustrates the convergence of the gradient descent towards a local minimum. If the loss functions does not decrease as the number of epochs goes up, it means we are constantly updating weights but never such that we make smaller and smaller errors. Here, the loss steadily deacreases in a standard exponential manner, which is what we usually see. The optimization takes steeper steps down the loss function at the begining, and as we approach the minimum, it starts taking smaller steps, leading to

smaller decreases in the loss funciton. We can also notice the loss function tends to become more erratic towards the end besides decreasing slower. This is due to the fact that we are using stochastic gradient descent, due to noise in the data, instead of a smooth path down the curve, the loss function is going back and forth and we can sometimes even take a step up the loss curve.

Table 4: Confusion matrix

		True	
		0	1
Pred	0	208	50
	1	580	801

Figure 19: Results

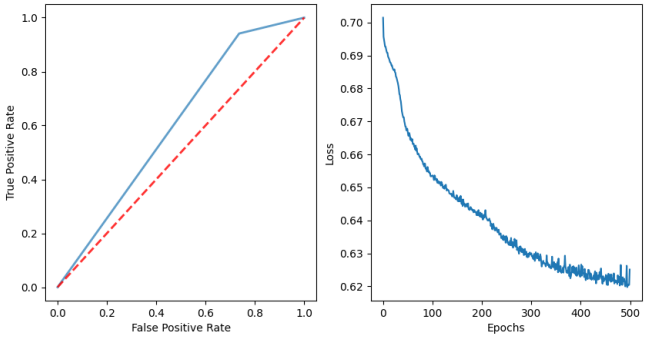


Table 5: Confusion matrix

		True	
		0	1
Pred	0	550	438
	1	229	423

Figure 20: Results

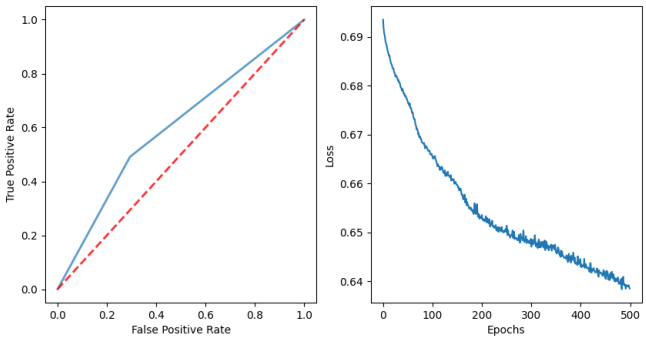
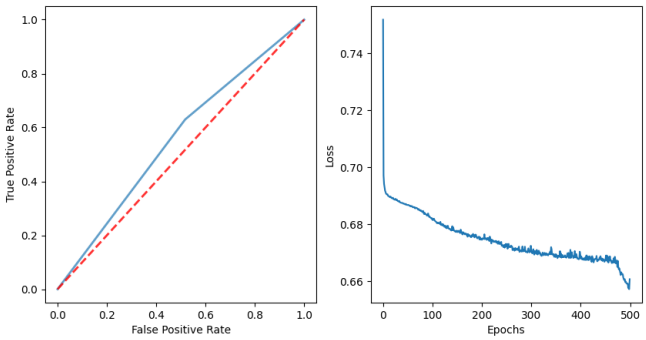


Table 6: Confusion matrix

		True	
		0	1
Pred	0	373	321
	1	400	546

Figure 21: Results



We have similar results for the other assets. We achieve 0.56 f1-score for **TSLA**, with 0.69 Recall and 0.69 Precision. For **AAPL**, we reach 0.58 f1-score, the highest of every assets considered, and 0.60 Recall and 0.58 Precision. And finally for **WMT** we achieve 0.56 f1 score and 0.56 Precision and Recall. Overall, all models were apparently able to classify part of the data correctly, the ROCauc are positives, meaning for the two classes, more observations are correctly classifies than observations are misclassified. We will later test the model on different data and see if the results are economically scalable. One thing to note is that with 500 epochs, the loss function does not seem to reach a minimum. In other words, the gradient descent needs more updates of the weights to reach the least possible error.

We can now consider different approaches to optimize the model to improve classification accuracy. First, we increase the non-linearity by adding hidden layers and neurons. We will use 3 hidden layers of 32 neurons each. We can also increase the number of iterations to 1000 to ensure convergence and train the model for a longer time, on larger amounts of data. Less is often more, increasing the number of epochs might not be necessary, but it will be interesting to see the impact on the model's accuracy. Finally, we can choose a different loss function. We are currently using the Categorical Cross Entropy (CCE) which is a multi class probabilistic loss function. This function was appropriate before, as we had three different classes. However we transformed the data to avoid pollution due to very imbalanced data and we know face a binary classification problem. We then need a new loss function suited to our problem. We will then be using the Binary Cross Entropy (BCE) to try to improve our model's performance.

- **AF.PA**

Table 7: Confusion matrix

		True	
		0	1
Pred	0	694	401
	1	140	451

- **TSLA**

Table 8: Confusion matrix

		True	
		0	1
Pred	0	447	222
	1	342	629

- **AAPL**

Table 9: Confusion matrix

		True	
		0	1
Pred	0	427	193
	1	352	668

- **WMT**

Table 10: Confusion matrix

		True	
		0	1
Pred	0	479	255
	1	294	612

After optimizing the model we achieve higher accuracy overall. For all the assets, Precision, Recall and f1-score increased. The f1-score lies between 0.56 and 0.67 with the new model against [0.56:0.60] with the former model. Precision ranges from 0.60 and 0.68 against [0.56:0.61] and Recall ranges from 0.66 and 0.68 against [0.56:0.69]. For **TSLA** Recall went from 0.69 before optimization to 0.68 and is the only metric to decrease for the entirety of the data set. Longer training, increased non-linearity and changing the loss function seem to yield more accurate results. We reckon changing the loss function has no impact on the predictions as Keras is supposed to be able to internally convert binary classification to multi class classification.

We will now train a Support Vector Machine on the same data to see if it performs better than the Neural Network. At first we will use a linear SVM with no kernel and proceed to add non-linearities with a gaussian kernel (Radial Basis Function kernel). The kernel hyperparameters will be optimized by grid search cross-validation.

- **AF.PA**

Table 11: Confusion matrix

		True	
		0	1
Pred	0	589	450
	1	245	382

- **TSLA**

Table 12: Confusion matrix

		True	
		0	1
Pred	0	137	99
	1	652	752

- **AAPL**

Table 13: Confusion matrix

		True	
		0	1
Pred	0	226	139
	1	553	722

- **WMT**

Table 14: Confusion matrix

		True	
		0	1
Pred	0	246	148
	1	527	719

The linear SVM performs as well as our first Neural Network on the training data. We achieve the lowest f1-score for **TSLA** (0.47). We observed that adding non-linearity helps improve the Neural Network, hence it seems reasonable to add a kernel function to the SVM to help improve the classification. The different hyperparameters are optimized with Grid Search Cross Validation. We are simultaneously evaluating C = [0.001, 0.01, 0.1, 1, 10, 100] and γ = [10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001]

- **AF.PA**

Table 15: Confusion matrix

		True	
		0	1
Pred	0	672	420
	1	162	412

- **TSLA**

Table 16: Confusion matrix

		True	
		0	1
Pred	0	400	221
	1	389	630

- **AAPL**

Table 17: Confusion matrix

		True	
		0	1
Pred	0	254	88
	1	525	773

- **WMT**

Table 18: Confusion matrix

		True	
		0	1
Pred	0	314	100
	1	459	767

Adding a kernel function and allowing for non linearity in the SVM improved the results. We achieve similar accuracy than with the second Neural Network. f1-score lies between 0.58 and 0.64, with higher values for **WMT** and **AF.PA**. We were expecting a better performance on **TSLA** and **AAPL** but are witnessing the opposite. The non linear SVM is slightly less accurate than the second Neural Network but still outperforms the linear SVM and the first Neural Network, which seems to coincide with the assumption that when predicting market movements, considering a non linear framework yields stronger results.

According to the EMH, we should not have been able to detect significant relations between technical indicators at date t and prices at date $t+1$. With logistic regression, we prove technical indicators carry information, although very sparse, on prices leading us to assume all the available information on the market is not instantly reflected in the prices. Running a linear regression allowed us to get a sense of the statistical significance of this information. Afterward we tried to make use of this information by training machine learning models with different configurations. It appears that considering a non linear framework yields better results. Furthermore, Neural Networks seems to slightly but constantly outperform SVM. To illustrate this, we compare the f1-scores of the best performing SVM and the best performing Neural Network in the table below:

Table 19: f1-scores

	SVM	**NN**
AF.PA	0.64	0.67
TSLA	0.62	0.65
AAPL	0.58	0.66
WMT	0.63	0.66

We were expecting models to perform better on more short term traded assets such as **TSLA** or **AAPL**. We assumed technical indicators would carry more information on such assets as they are more subject to speculative investing. Results seem to indicate the opposite, one plausible reason is that **TSLA** and **AAPL** are more volatile and risky with abrupt reversals in price patterns. They are overall less predictable than more stable assets considering all possible predictors which explains why the models under perform on these two assets.

3.2.1 Results

Our results seem to disprove the EMH, we are able to build models capable of predicting the future price direction better than random. On other hand, we would like to remind the reader that accuracy metrics presented above are obtained from the training data and also that we need economical scalability to fully state the EMH can be rejected. Measuring the accuracy of the model on the same data it was trained on leads to an upward bias, as the model may be over-fitting the data. Furthermore, if the predictive power of the model leads to returns inferior to those obtained by buying and holding a large index, the EMH cannot be practically rejected. For these two reasons, we are going to pursue our research by first testing the best model on testing data and then calculate the realized return on the testing period.

The testing data consists in the last 30% of the data set from 07/02/2017 to 12/31/2019 for a total of 703 observations. To test whether or not our model yield better returns we assume that we follow daily predictions, positions are opened and closed on a daily basis, we are taking compounded returns into account, We consider an initial $1000 investment and at each date the amount is updated based on the previous day returns, the resulting amount is invested following the prediction of the model for the next day and so on and so forth. We are focusing on the best Neural Network and the best SVM as both achieve similar performance on the training data set.

4 Conclusion