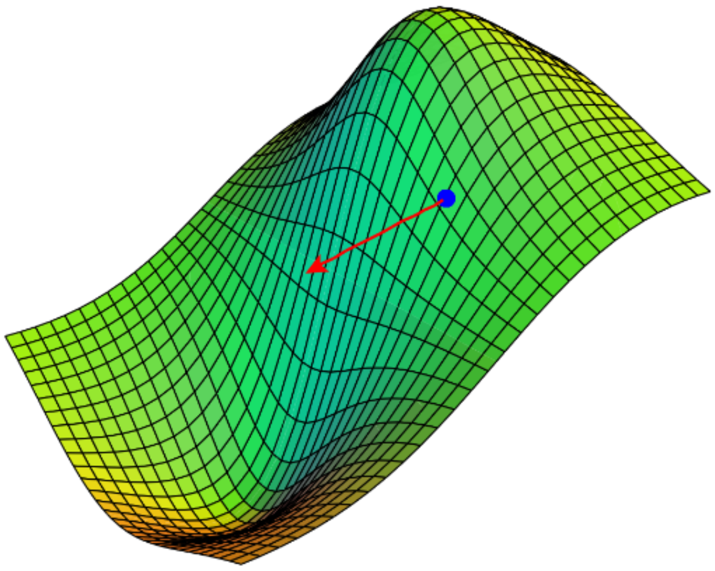
UNIVERSITÉ DE NANTES

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
27/06/2021

Contents

Abstract

We explore the Efficient Market Hypothesis and its implication on returns predictability. We use technical indicators commonly employed in practice to try and predict future price movements. Using four different stocks, we find significant correlation between technical indicators and future returns. We then proceed to predict one day ahead returns with the help of different Neural Networks and Support Vector Machines and are able to achieve roughly 60% accuracy. Findings are consistent across training and testing data sets and are economically scalable. We achieve greater return and less risk than a benchmark strategy using the S&P 500 as a proxy for the market portfolio, as suggested by the modern asset pricing theory. Economic scalability is consistent across two measures of risk, volatility and semi-deviation, we further achieve promising Sharpe and Sortino ratios. Results, although encouraging, need to be proven consistent across more periods and assets if we want to argue that markets are not efficient.

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 returns, with less risk. Eugene Fama states that the only way to expect returns greater than the market is to take additional risks. We are now able to ask the following question: does the EMH holds ? Is trading futile ? Can you achieve greater returns than the market for a corresponding market risk ? 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, and has down-to-earth application in real life. 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. Artificial Intelligence is a very powerful tool when a problem comes to identifying patterns and regularities in the data. One fact often played down is that Machine Learning requires less human input than other methods, 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, they can sometimes be interchangeable and the frontier between the two is often blurry. Here we will almost exclusively refer to Machine Learning which consists in an ensemble of algorithms designed to find complex patterns in the data without the need of explicit a priori hypothesis on the behavior of the data. 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. Models will be trained on technical indicators, commonly used by practitioners. Technical indicators are measures supposed to identify price patterns and are assumed to be able to predict future price movements. We will first gaze over the Machine Learning methods we will be using from Neural Networks to Support Vector Machines. In a second time we will try to use machine learning to predict future returns on four different assets and evaluate if the model can generate significant economic results, we also describe what modifications have

[1]Fama 1970

been made to improve the data processing[2].

This project is coded in Python 3.8, code is available on GitHub.

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 Machine Learning

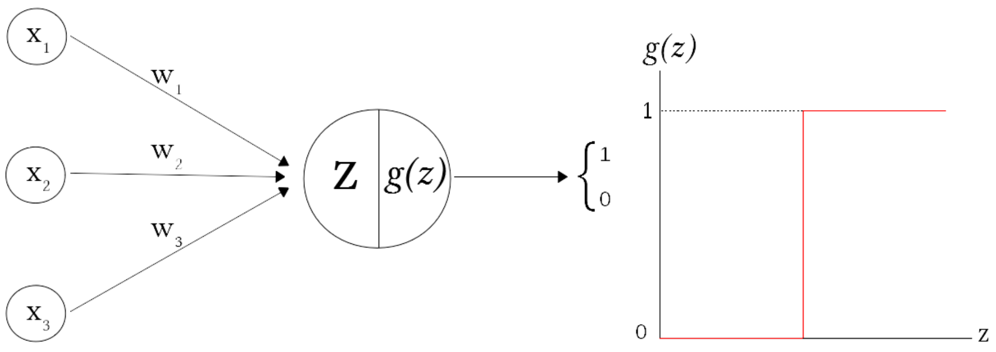
2.1 Neural Networks

Artificial Neural Networks (Neural Nets or NN for short) are Machine Learning models designed to replicate how information is processed in the brain 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 within which the information transits, starting from a particular point these branches, along with the neurons that activated, create a path from the nerve impulse to an final neuron which outputs an interpretation of the original impulse. 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.

2.1.1 The Perceptron

In this section, we will focus on the Perceptron, a particular case of Neural Networks which consists of only one neuron. It is easier to understand the logic behind NN using a Perceptron, the results can later be generalized to more complex networks with more than one neuron. 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



Perceptron is a binary classifier; we wish to produce a binary output taking values of either 0 or 1. In our case, the output represents the return on a certain day: 1 if returns are positive (price increased) and 0 if they are negative (price decreased). We begin by giving the model inputs x_p[3] representing the variables we wish to consider. Each input is multiplied by a weight w_j[4] and all weighted inputs are summed. The weighted sum, Z goes through an activation function $g(Z)$, that outputs either 1 if the sum exceeds a certain threshold, or 0 if Z falls below the threshold.

In our case, we can think of inputs as values for technical indicators such as the

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

[4] $j = 1, ..., J$, where J is the total number of weights in the model

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".

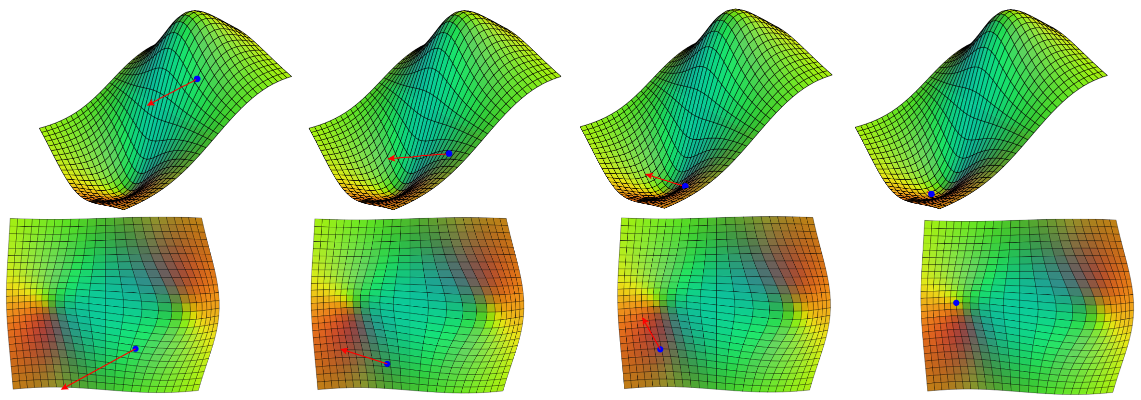
To learn how to accurately classify data, the Perceptron will update each weight such that we eventually reach a point where every data point is associated to the correct class. If the Perceptron outputs "buy", and the correct position was indeed buying, we can repeat the procedure for another data point. 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 of the network that can be manipulated, by constantly updating them we will eventually come to the right combination, 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 overall accuracy of the model at each update. In order to do so, we need a metric to assess the accuracy of the model at each step. 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.

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 Neural Nets, 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 designed to 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 on 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 are then optimal.

Figure 2: Gradient Descent

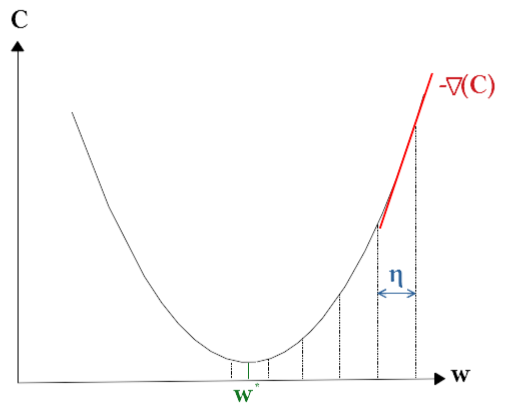


For each data point we compute the error and the gradient at that point, take the inverse and increment the weights in the direction pointed by the inverse gradient. By always choosing the direction of steepest descent we will eventually converge to a local minimum, given that the step size is adapted. 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 $\nabla(C) = \begin{bmatrix} \frac{\partial C}{\partial w_j} \\ \dots \\ \frac{\partial C}{\partial w_J} \end{bmatrix}$ 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 optimal point, 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.

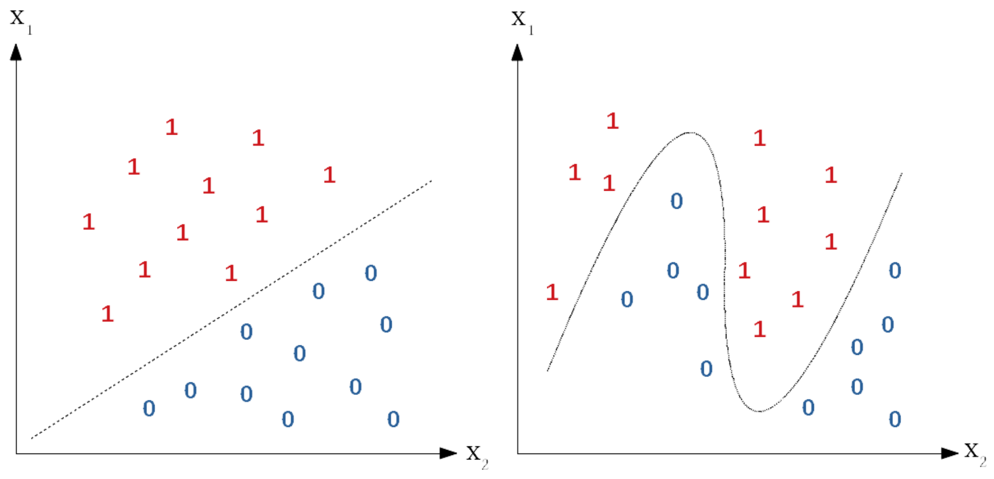
Figure 3: Overshooting the local minimum



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 weights are greater than w^*, the gradient points towards the left and because the step size is too large, we land beyond the minimum. The weight are then below w^*, 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, and also choosing appropriate step size. Large η helps converging faster but bears the risk of overshooting the minimum and small η limits the risk of overshooting but increases the required number of steps proportionally.

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. Every data 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 more complex models in order to solve non linear problems such as the one presented on the diagram below.

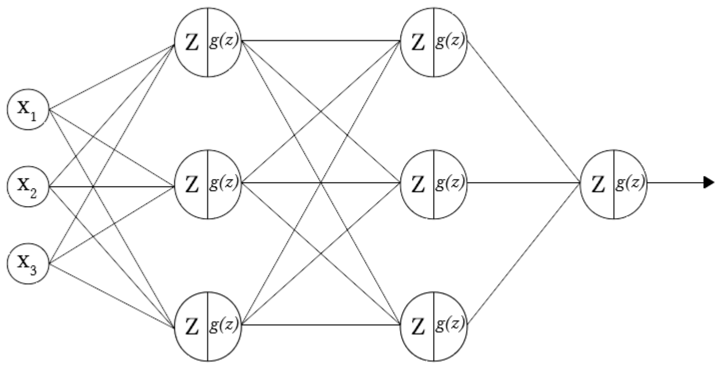
Figure 4: Linear separability



2.1.2 Networks & Backpropagation

The use of Neural Networks allows us to find non-linear decision boundaries and separate classes that could not be identified 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.

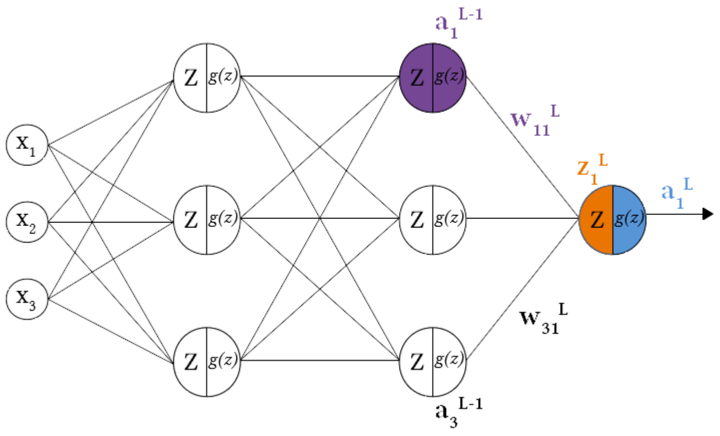
Figure 5: Neural Network with two hidden layers



The core principles are identical to those of a 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.

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

Figure 6: simple network

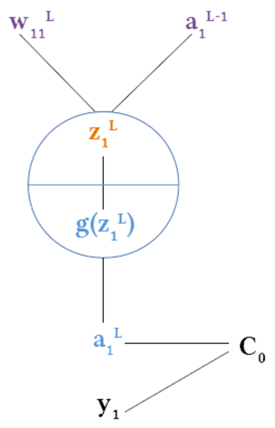


We associate each weight w to the two nodes it is connecting such that w_{jk}^{L} connects node j in the previous layer to node k in the current layer. The current layer is denoted as L and previous layers are denoted as $L-1$, $L-2$, ...,$L-M$. We also define the loss function as C_n, $n = 1, 2, ..., N$, N being the sample size and the activation function as $g(z)$. The weighted sum of inputs arriving to node j in the current layer L 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:

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

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 $(L-1)$ to output layer. 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 two last layers 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 $w^L_{1,1}$ to obtain z^L_1, $z^L_1 = w^L_{1,1} * a^{L-1}_1$. z^L_1 is passed throught the activation function $g(z^L_1)$ and gives an output in layer L: a^L_1. Finding partial derivatives with respect to the weight between the last hidden layer $L-1$ to the output layer L is straightforward because we can decompose the effect of weight $w^L_{1,1}$ with the chain rule:

$$\frac{\partial C_0}{\partial w^L_{1,1}} = \frac{\partial C_0}{\partial a^L_1} * \frac{\partial a^L_1}{\partial z^L_1} * \frac{\partial z^L_1}{\partial w^L_{1,1}} \tag{3}$$

Because the loss function (C_0) is a function of the output given by the network (a^L_1), which is a function of the weighted sum of previous inputs (z^L_1) which in turn, is a function of the weights between the last ($L-1$) and output layer ($w^L_{1,1}$), the partial derivative of the loss function is equal to the product of the partial derivatives of previous functions in the network. The partial derivative of the loss function with respect to the weight $w^L_{1,1}$ is equal to the product of the partial derivatives 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^L_1 and the partial derivative of the weighted sum with respect to the weight $w^L_{1,1}$.

- $\frac{\partial C_0}{\partial a^L_1}$, the loss function is equal to the difference between the output of the last neuron and the target value for that data point, y_0: $(a^L_1 - y_0)^2$. Therefore its partial derivative with respect to a^L_j is: $2(a^L_1 - y_0)$.
- $\frac{\partial a^L_1}{\partial z^L_1}$, the output a^L_1 is the result of the activation function evaluated at z^L_1. We can also write; $a^L_1 = g(z^L_1)$ Therefore its partial derivative is equal to the derivative of the activation function, $g'(z^L_1)$. This is the reason why we need differentiable activation functions. Sigmoïd, hyperbolic tangent and ReLU functions are some examples of differentiable activation functions.

- $\frac{\partial z_1^L}{\partial w_{1,1}^L}$, z_1^L is the weighted sum of the previous output multiplied by the weight $w_{1,1}^L$. We know, $z_1^L = w_{1,1}^L * a_1^{L-1}$ therefore 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 a_1^{L-1}.

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

$$\frac{\partial C_0}{\partial w_{1,1}^L} = 2(a_1^L - y_0) * g'(z_1^L) * a_1^{L-1} \tag{4}$$

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 function is equal to the average gradient of the loss function, therefore we repeat the process for each data point in the training set to obtain one element of the gradient vector:

$$\frac{\partial \bar{C}}{\partial w_{1,1}^L} = \frac{1}{N} \sum_{n=0}^{N} \frac{\partial C_n}{\partial w_{1,1}^L} \tag{5}$$

From here, we can generalize the results to weights on deeper layers of the networks and introduce the notion of backpropagation. Previous findings only hold for 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 $C = C(a_1^L, y_n)$, which allowed us to use the chain rule to capture the impact of $w_{1,1}^L$. For the partial derivatives with respect to weights between the previous $(L-2)$ to penultimate layer $(L-1)$, weights $w_{j,k}^{L-1}$, 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 neurons that are not on the last layer, every neuron 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 neurons activation to be able to compute partial derivative of the loss functions with respect to every weights in the network. We would like to remind the reader that with all partial derivatives with respect to each weight, we can calculate the gradient and update 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 $\frac{\partial C_0}{\partial a_j^L} = 2(a_j^L - y_0)$ and $\frac{\partial a_j^L}{\partial z_j^L} = g'(z_j^L)$. 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 in the output layer with respect to previous output in layer $L-1$ is equal to the weight $w_{j,k}^L$ because $z_j^L = \sum w_{j,k}^L * a_j^{L-1}$. Therefore $\frac{\partial C_0}{\partial a_j^{L-1}} = 2(a_j^L - y_0)g'(z)w_{j,k}^L$. We can plug this equality into the previous derivatives:

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

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. Here we showed how to compute derivatives up to 2 layers, the same process is repeated until we eventually reach the input layers.

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 all 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.

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.

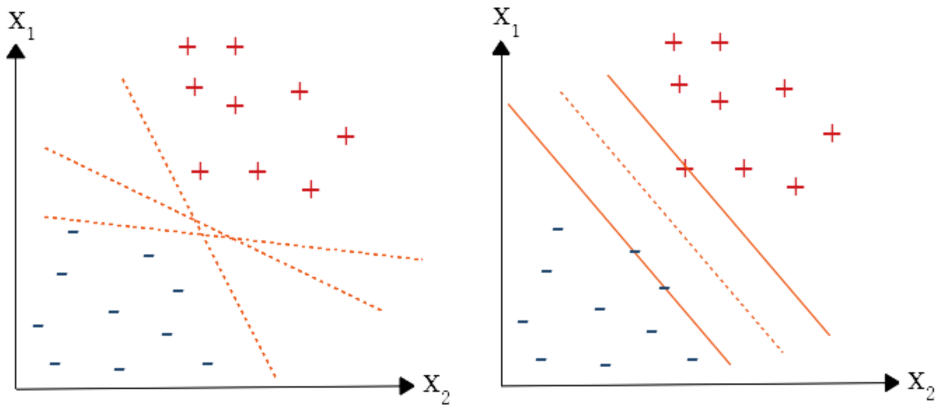
2.2 Support Vector Machines

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.

2.2.1 Linear Support Vector Machines

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) in two dimensions, each dimension representing one of the two variables we consider in this example[6].

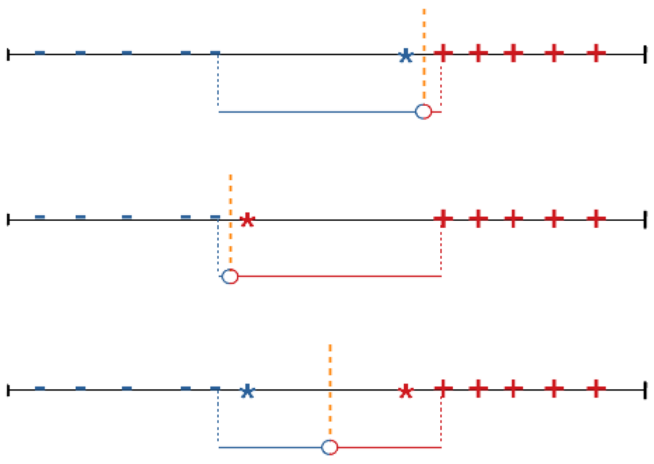
Figure 8: Maximum Margin Boundary



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. Which one should we prefer and why ? The SVM rational suggests that the best decision boundary is the one that lies 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 assumption with a simple example with only one dimension.

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

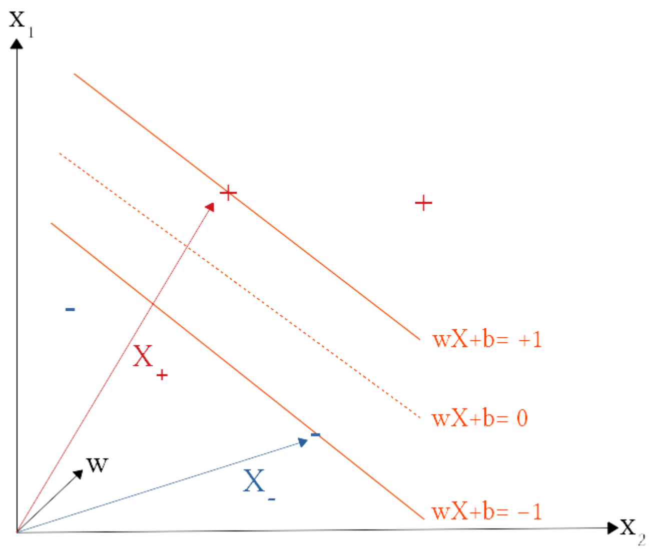
Figure 9: Intuition behind SVM



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 samples. 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. SVM treats this problem in vector form, where each data point has particular (x_1, x_2) coordinates in space we can represent with 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_1, x_2) 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. We are going to add $\vec{w} \cdot \vec{X_+} + b \geq +1$ and $\vec{w} \cdot \vec{X_-} + b \leq -1$ where X_+ and X_- are vectors pointing to positive (resp. negative) samples as shown on the diagram above. This new constraints adds that for any positive sample the decision function $(\vec{w} \cdot \vec{X} + b)$ outputs a value greater or equal to 1, and less or equal to -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 set 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 previous constraints, but if we know multiply each constraint by the newly introduced variable y_i we have:

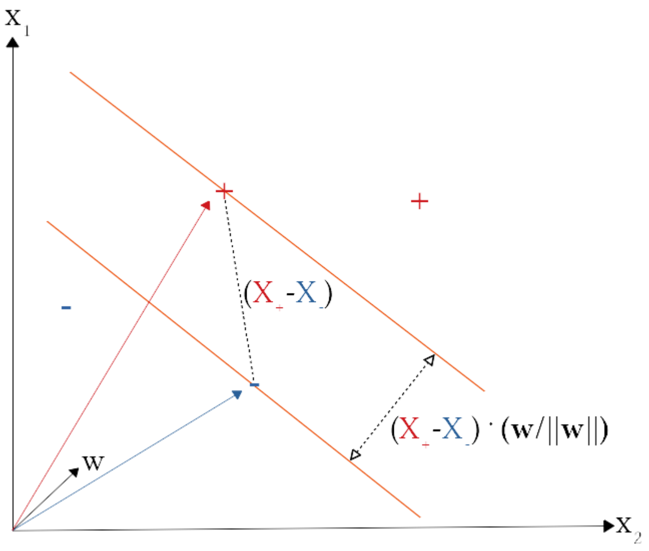
- $\underbrace{y_i}_{+1} \underbrace{(\vec{w} \cdot \vec{X_+} + b)}_{\geq 1} \geq +1$
- $\underbrace{y_i}_{-1} \underbrace{(\vec{w} \cdot \vec{X_-} + b)}_{\leq -1} \geq +1$

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 that lies exactly 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.

If X_+ and X_- are vectors pointing to positive and negative points precisely on the margins[7], 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 normal vector by dividing it by its norm, $||\vec{w}||$, because $\vec{w}$ is orthogonal to the margins, the margin itself can then be calculated as:

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

Figure 11: Finding the margin



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

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: $y_i(\vec{w} \cdot \vec{X_i} + b) - 1 = 0$ as discussed earlier[8]. Hence we are looking to solve:

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

[8]Here we use $y_i(\vec{w} \cdot \vec{X_i} + b) - 1 = 0$ as we consider points on the margins.

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

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

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 (points on the marginal hyperplanes, X_+ and X_-). Once we find the support vectors 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. In equation 12 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 $\alpha_i > 0$

[9]Both terms cannot be equal to 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 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 optimal 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:

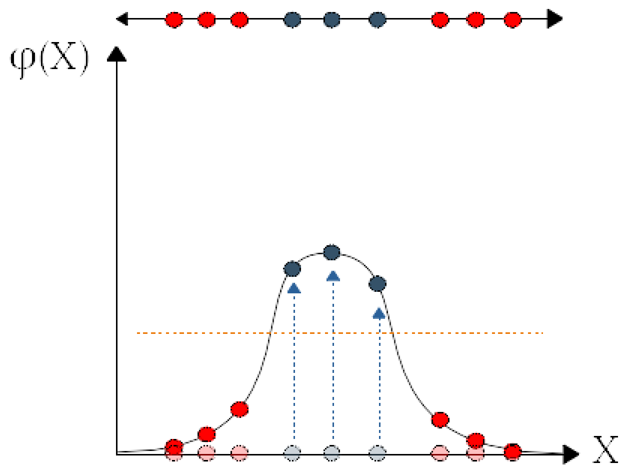
$$\begin{aligned} &+ \text{ if } \vec{w}^* \cdot \vec{u} + b^* > 0 \\ &- \text{ if } \vec{w} \cdot \vec{u} + b < 0 \end{aligned} \tag{15}$$

$\vec{w}$ and b are known so we can classify every new point..

2.2.2 Non-Linear Support Vector Machines

SVM as we discussed them in the previous section is 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.

Figure 12: Kernel Trick



The data in its original 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 points to a different dimension, here for instance we map the data to a second dimension represented by the vertical axis. Once every data point has been passed through ϕ we can find a linear decision boundary represented by the orange 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 initially not linearly separable.

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:

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

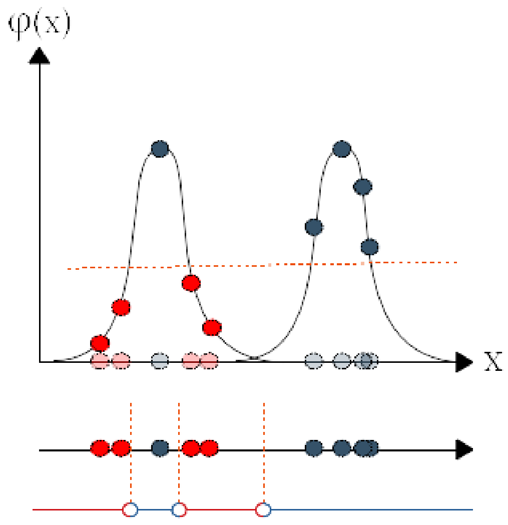
If we can find a kernel function k that takes the two vectors as inputs and outputs the dot product of those two vectors in higher dimensional space such that $k(\vec{X}_i, \vec{X}_j) = \phi(\vec{X}_i) \cdot \phi(\vec{X}_j)$ we can replace ϕ with k:

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

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.

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 not large enough, 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.

We have discussed Neural Networks and Support Vector Machines, we will know try and apply both models on financial data. We hope the models trained on technical indicators will be able to correctly classify positive and negative returns. If so, we can use the models to predict future returns and elaborate an investment strategy based on the predictions given by the models.

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 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 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 company 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

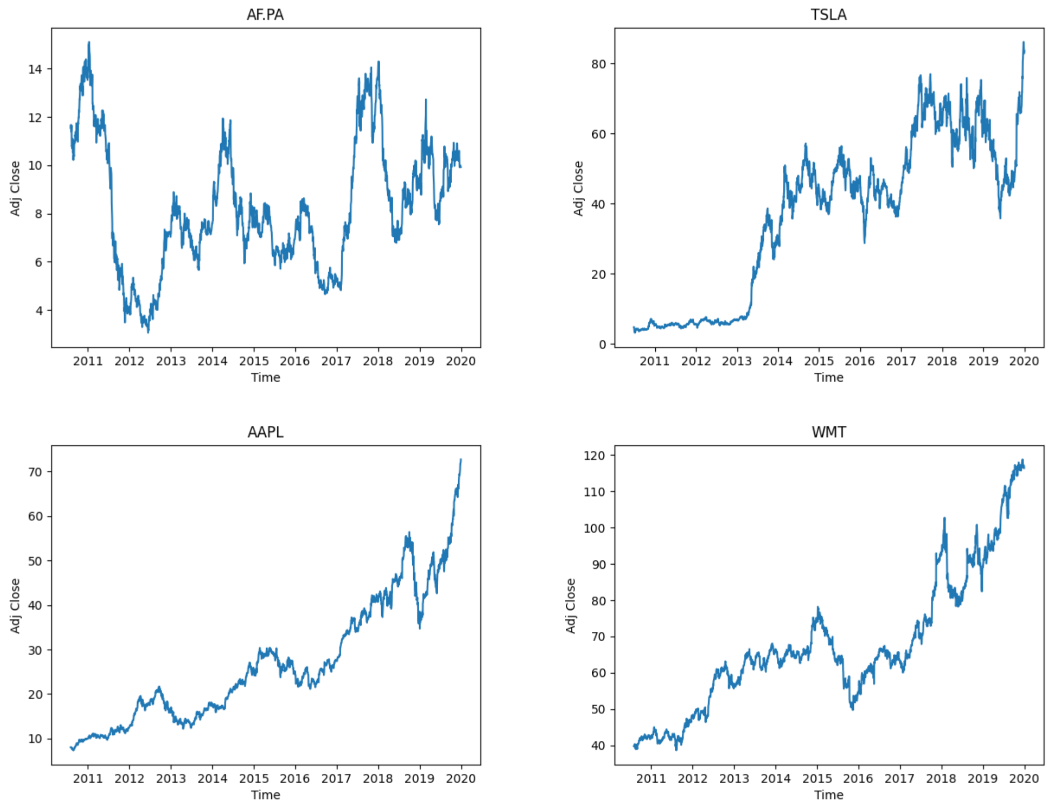
[10]We can use ANN for regression but we did not cover this part as we will be addressing a classification problem.
[11]Ma, Yang, and Su 2021
[12]Tesla was founded in 2003

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 market conditions. Our model will therefore be in case of rare black swans events, which constitutes a major drawback to our approach.

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

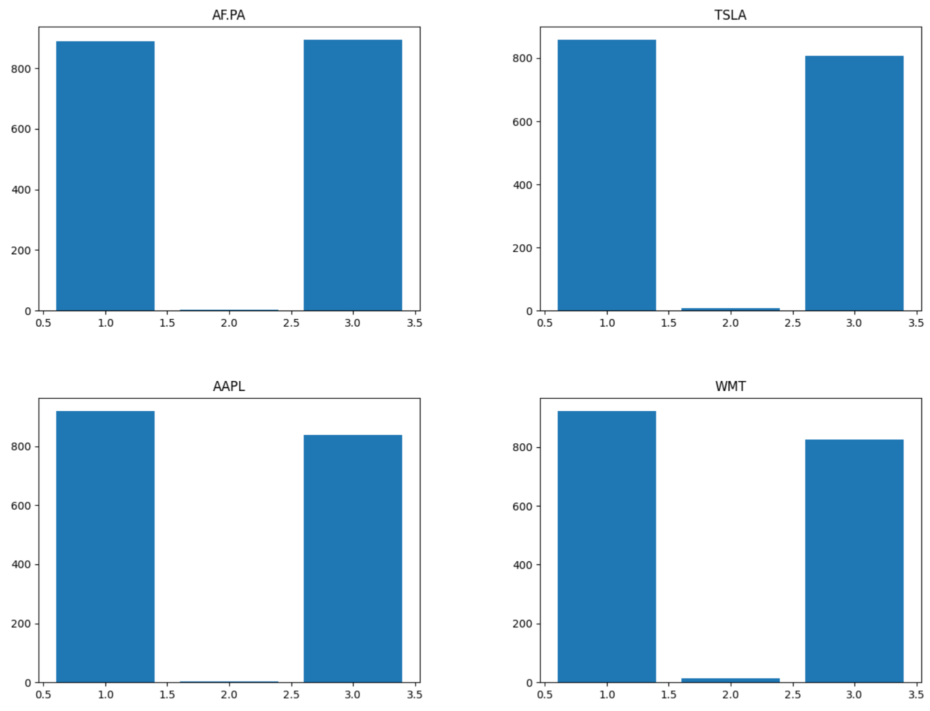


AAPL and **WMT** exhibit very clear upward trends and **AF.PA** and **TSLA** are more volatile. **AF.PA** close price heavily decreased in 2011 due to an increase in oil prices and fewer passengers. It reaches historic lows during 2012 and fully recovers in 2014. The price finally reaches previous values in 2018 after a year of increased demand and low oil prices. **TSLA** close price is very stable at the beginning of the period, the price begins to increase in 2012 after the company launched the Model S in the United States and Europe. We also notice two other significant breaks in 2017 and 2020 after the release of the Model 3 and Y as well as the inclusion if Tesla in the S&P 500 during 2020. **AAPL** and **WMT** record heavy losses in 2018 due to the pressure caused by the China-United States trade war. Except **AF.PA**, all assets have seen their close price increase between 2011 and 2020.

As we discussed in 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. We will later consider removing the second case such that we are able to use binary classifiers, which will be easier to interpret and 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 wrongly 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 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.

As for the features, we will select a range of technical indicators. Technical indicators are calculations based on price, volume and volatility, they are assumed to detect price patterns such as trend, momentum, reversals and others. Technical analysis relies on such indicators to anticipate future prices and profit from opportunities when the price is either increasing or decreasing. Our objective here is to build a machine learning model capable of conducting technical analysis on an asset and open trades depending on what the model predicts for future prices. There are more than a hundred technical indicators used in technical analysis. Depending on markets, period, investment horizon or objectives, not all of them are relevant and we therefore need to choose the ones we think are the most appropriate. We chose 6 of the most followed indicators. Our choice is motivated by several reasons, first; we chose a range of indicators where each represents a different behavior (trend, momentum, reversal, price-volume reactions and volatility) to maximize the amount of information we can extract from the price series, and to avoid redundancy among two or more indicators. Furthermore, we also chose the most followed indicators as they are the ones with the highest probability of being correlated with returns. If we suppose the EMH holds, technical indicators should not be correlated with returns. However even in these conditions, if all investors on the market base their trades on such indicators, even if they are initially uncorrelated with market returns, the collective impact of investors' decisions on the market will introduce some form of dependency between indicators and market conditions. Let us briefly present the indicators we considered:

1. **Relative Strength Index (RSI)**: RSI is a measure of overbought/oversold stocks. RSI is calculated as follow: $RSI = 100 - \frac{100}{1+\frac{U}{D}}$, where U is the average of positive returns over n days and D is the average of negative returns over the same period. RSI will lie between 0 and 100, it will increase if positive returns exceed negative returns and decrease if negative returns exceed positive returns. There is a direct relation between the RSI and the ratio of positive and negative returns. The idea is to assume that if this ratio is close to extreme values, the market is not efficiently pricing the asset which is overbought (very large and/or consecutive positive returns coupled with very small and/or rare negative returns) or oversold. We generally choose 70 and 30 as cutoff values, that is if the RSI is above 70 the asset is overbought and we should sell and if the RSI is below 30 the asset is oversold and we should

buy.

2. **Moving Average Converge Divergence (MACD)**: MACD is a trend indicator, the objective is to hopefully predict future price trends. MACD takes the difference between a moving average over a long period (generally 26 days) and a moving average over a shorter period (generally 12 days). We also compute a moving average of the MACD itself which constitutes a signal line. If MACD crosses its moving average upward it indicates a positive trend and is MACD crosses it signal line downward it signals a negative trend. The absolute value of the difference between MACD and the signal line represents the strength of the trend. Based on the MACD we want to buy when its value is greater that the one of its moving average and sell when its value is less than one of its moving average.

3. **Bollinger Bands**: Bollinger bands are a volatility indicator. They are the positive and negative limits of a 2 standard deviations confidence interval centered around a moving average of n days (usually 20). Approximately 95% of the prices should lie between the two Bollinger Bands, assuming returns are normally distributed. When prices cross the upper limit of previous days confidence interval, it suggest increasing volatility over the next periods and positive trend, when prices cross the lower limit it suggests a following negative trend. It is well know in the literature that returns are often leptokurtic and sometimes slightly skewed. Bollinger Bands might be more efficient assuming another distribution, such as a Student distribution.

4. **Average Directional Index (ADX)**: ADX measures the strength of a price trend. First we compute $M_t^+ =$ current highest price - highest price last period and $M_t^- =$ lowest price last period - current lowest price. We can then calculate the Directional Movements, DM_t^+ and DM_t^- such that:

   $$DM_t^+ = \begin{cases} M_t^+, \text{ if } M_t^+ > M_t^- \text{ and } M_t^+ > 0 \\ 0, \text{ if } M_t^+ < M_t^- \text{ and } M_t^+ < 0 \end{cases} \quad \text{and}$$

   $$DM_t^- = \begin{cases} 0, \text{ if } M_t^- < M_t^+ \text{ and } M_t^- < 0 \\ M_t^-, \text{ if } M_t^- > M_t^+ \text{ and } M_t^- > 0 \end{cases}$$

   If the difference between current and past highest price is superior to the difference between current and past lowest price and not equal to 0 (current highest price and past highest price are not equal), then DM_t^+ is equal to the difference between current highest and past highest price. Otherwise, DM_t^+ is equal to 0. DM_t^- follows the same logic when we consider the difference between current and past lowest price. The Directional Movements indicates the direction of the price movement and the strength of this movement is represented by the difference in highest and lowest prices. M_t^+ very large means the price is currently higher than previous highs, price movement is positive and well-defined (price is increasing). M_t^- very large means the price is currently lower than previous lows, price movement is negative and well-defined (price is decreasing). We now compute the Directional Index (DI) by taking the $n - days$ moving averages of DM_t^+ and DM_t^- and scale it with respect to the volatility of the day proxied by the True Range (TR),

   $$TR_t = max \begin{cases} |\text{current highest price - current lowest price}|; \\ |\text{current highest price - previous close price}|; \\ |\text{current lowest price - previous close price}| \end{cases}$$

$DI^+ = \frac{MovingAverage(DM_t^+)}{TR_t}$ and $DI^- = \frac{MovingAverage(DM_t^-)}{TR_t}$. The Average Directional Index is finally a ratio of the positive and negative Directional Indices:

$ADX = 100\frac{|DI^+ - DI^-|}{DI^+ + DI^-}$

And we usually smooth the ADX value by taking its moving average over n days (generally 14 days). ADX tells us if in recent periods prices have been increasing or decreasing, as given by the Directional Indices and the relative strength of this movement given by $|DI^+ - DI^-|$, the difference of differences of past and current highest and lowest prices. $|DI^+ - DI^-|$ very large indicates prices have been consistently exceeding past highest prices or consistently falling behind past lowest prices. $|DI^+ - DI^-|$ very small means prices are staying relatively still.

5. **Stochastic Oscillator**: Stochastic Oscillators are momentum indicators, they indicate if the current trend is excepted to reverse. If a stock price is currently in an upward trend, we expect prices to close higher than previous periods. If this momentum is somehow broken by a large negative return we can assume the trend is fading and will likely reverse in future days. The Stochastic Oscillator is defined as:

   $K = \left(\frac{C - L_k}{H_k - L_k}\right) * 100$

   Where C is the current closing price, L_k and H_k are respectively the lowest and highest closing price over a $k - days$ period. If prices where increasing and they suddenly fall under the lowest price recorded in a $k - days$ period, the positive momentum is slowing down and price might continue falling. In practice the Stochastic Oscillator is often smoothed by taking its $3 - days$ moving average. As for the RSI, Stochastic Oscillators are bounded by threshold values; when it reaches values above 80, the positive momentum is about to reverse. When it reaches 20, the negative momentum is supposed to reverse.

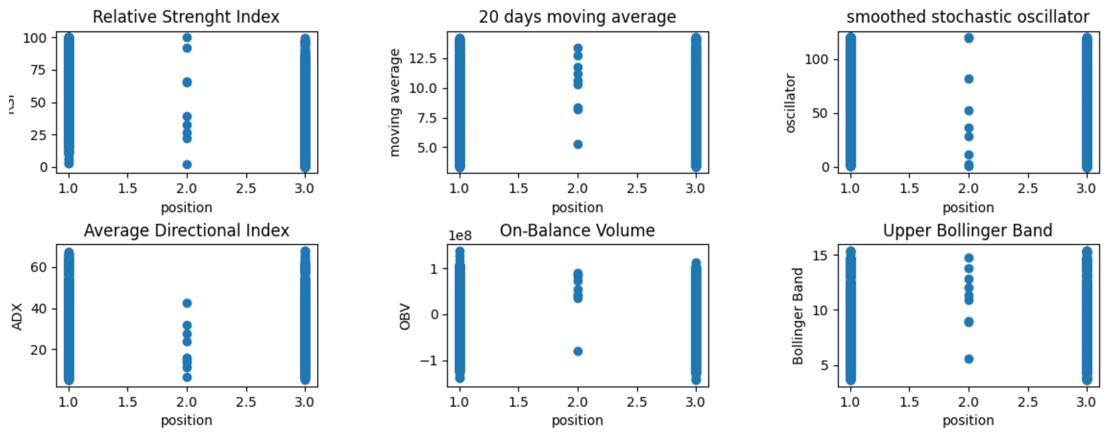
6. **On-Balance Volume (OBV)**: OBV is a volume indicator. Volume indicate the amount of transactions happening on a particular asset on a certain day. The idea is that volume measures the incentive of the market to participate in price movements. During trends, positive or negative, volume is high indicating many investor are taking action on the market. If volume starts decreasing; the asset is generating less enthusiasm and the trend will eventually end. OBV for period t is defined as:

   $$OBV_t = OBV_{t-1} + \begin{cases} \text{volume, if } C_t > C_{t-1} \\ \text{0, if } C_t = C_{t-1} \\ \text{-volume, if } C_t < C_{t-1} \end{cases}$$

   Where C_t is the closing price at period t.

We start our analysis by visualizing the **position** variable with respect to the features. This will help us determine whether or not the distribution of **position** is conditional on the features, which means the **position** value will change depending on the value of the feature considered. In the other scenario, the **position** is independent from the feature, meaning the value of the feature does not provide information on future price. Visual assertion of a relation between multiple variables is not sufficient to assume that variables are correlated, 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.

At first glance, it seems the features are uncorrelated with **position**. If we look at the RSI for instance, we assumed that high values are associated with a 'sell' signal (**position** = 3) whereas small values are associated with 'buy' signals (**position** = 1). Conversely, if **position** = 1 we should have small RSI values, and high RSI values if **position** = 3. However the RSI takes similar values when we consider days where the price increased and days where the prices decreased. 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 should not be correlated with returns.

3.2 Modelling

3.2.1 Logistic Regression

However we did not observe dependence between our features and **position**, we need to statistically test for absence of significance of the features to confidently establish that indicators do not carry information on future returns. We will therefore 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 logistic regression. One important note is that logistic regression is a linear model. It is possible to have no linear dependencies between the indicators and **position** while still having non-linear dependencies, resulting in non-significant logistic regression but usefull machine learning models.

We were unable to estimate a logistic regression on current data. We suppose this is due to very high multicolinearity in the features, this leads to singular matrices which are non-invertible. In such conditions, it is impossible to estimate a logistic regression model. For instance, Bollinger bands are upper and lower limits of a 2 standard deviations interval centered around a moving average. They are technically a function of the moving average, therefore the moving average and the two bollinger bands are completely dependent and colinear. These features, among others, are the reason we were not able to estimate the logistic regression.

As we previously discussed, 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.

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.
- 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 moving average crosses the signal line upward the variable sends a signal and when the moving average crosses the signal line downward the indicator sends another signal.
- OBV and ADX are kept as raw values.

Categorical features are then One Hot encoded in order to be able to feed them to Keras and Sickit-Learn. We also transformed **position** into a binary variable to get rid of the second class with very limited representation across the data set. We expect that removing this class will help the different classifiers as the data is not polluted by unnecessary information.

Table 1: Logistic Regression

Asset	**AF.PA**	**TSLA**	**AAPL**	**WMT**
Feature	Coef.			
ADX	-	+*	+**	+
OBV	+**	+	+*	+
RSI_0	+***	+***	+***	+***
RSI_1	-***	-***	-***	-***
RSI_2	+***	+***	+***	+***
RSI_3	-***	-***	-***	-***
D_0	+***	+	+	+
D_1	-**	-.	-*	-***
D_2	+***	+***	+*	+.
D_3	-*	-*	-	-**
boll_0	-*	-***	-***	-***
boll_1	+**	+	+*	+
MACD_2	+	+	+**	+
Likelihood Ratio p-value	$3.1e^{-44}$	$3.8e^{-42}$	$1.4e^{-40}$	$2.3e^{-50}$
$Pseudo - R^2$	0.103	0.101	0.097	0.12

*Note: confidence levels are: . = 0.1, * = 0.05, ** = 0.01, *** = 0.001*

Results indicate the increased or decreased probability that next day returns are positive which incentives us to buy. All models have a larger likelihood than the null models. In other words our models fit the data better than a model with no parameters, technical indicators seem to be able to predict future returns to a certain extent. The additional information is not trivial as the likelihood ratio test is highly significant. We explain approximately 10% of the variance of future returns, the best performing model based on pseudo R^2 is the one trained on **WMT** data. The asset for which the model is worse fitted is **AAPL**. Because we are using logistic regression, we would rather interpret the signs of the coefficients rather than their value. Their significance level is given by the Wald test, with null hypothesis H_0: $\beta_i = 0$ and alternative H_a: $\beta_i \neq 0$, where β_i is the coefficient associated with the i^{th} feature. Most coefficients are significant to a 5% confidence level, signs of the coefficient remains the same for most assets, which is a good sign. It indicates the same model gives the same results on different data, which shows stability over different samples. When **RSI** decreases below 70 (RSI_1), the models suggests a decreased probability that future day returns will be positive, and further suggests to sell, which coincides with common practice. The asset was overbought and is currently experiencing a correction. When **RSI** crosses 30 upward (RSI_2) the model suggests increased probability that returns will be positive, which was expected; the asset was oversold and crossing 30 signals the market has stopped selling the asset below its fair price and price should start to increase. We were expecting buy (resp. sell) signals when **RSI** increases over 70 (resp. decreases below 30) but we observe the opposite, it is a possibility that when entering overbought (resp. oversold) conditions, the market will conserve its momentum for a few periods. This would explain why signals when exiting such phases have the effect we expected whereas signals when entering overbought/ oversold phases predict returns opposite to what we were expecting. However, we often observe that overbought/ oversold phases last more than one period and even if results are counter-intuitive they are not disconnected from the reality of the markets. The stochastic oscillator, **D**, shows the same pattern than **RSI** with less significance overall. **D** also shows when the asset might be overbought/ oversold, the difference with **RSI** is 1- the calculation 2- the cutoff values (80 and 20 instead of 70 and 30 for the RSI). It is interesting to us that **D** and **RSI** have the same behavior, they explain the same phenomenon hence their impact on future returns should be of the same sign. **D** is significant overall, considering all four thresholds, D_1 and D_2 are the most significant signals on every asset whereas D_0 and D_3 are only significant on certain assets. We observe the same pattern than with **RSI**; signals when entering overbought/ oversold phases bring less information on future returns than signals when exiting the same phases. Overall, **RSI** seems to be better fitted to identify overbought/ oversold assets than **D**. **ADX** and **OBV** which were kept as raw values are only sparsely significant. This was expected, we know the relation between these two features and future returns are non-linear, trying to identify them with a linear model will yield non-significant coefficients. high **ADX** values mean strong positive or negative trend, hence very high chance of positive or negative returns, because **ADX** does not specify the direction of the trend, its values are correlated with returns[13], but the relation is non-linear. **OBV** is a cumulative sum of volume and returns, it could potentially expand to infinity, therefore high values are not directly correlated with high returns, once again if there is a relation between **OBV** and future returns, we know this relation is non-linear. This is why having non-significant linear correlation between **ADX** and returns and **OBV** and returns is not surprising. SVMs and ANNs might be able to make sense of the non-linear relations between these variables. Our transformation of **MACD** does not allow us to detect signals and the variable is only significant when considering **AAPL**. When price cross the upper Bollinger band (boll_1), or when prices increase above two standard deviations of the moving average, the model suggests a decreased probability that future returns will be positive. We were expecting boll_1 to signal the beginning of a positive trend as prices breaks the upper limit of past recent days and boll_0 to signals the start of a negative trend. The models suggests otherwise and the correlation between the Bollinger bands and future returns are inverse of what we were expecting. One explanation would be that Bollinger bands act has an overbought/ oversold indicators, if prices increase (resp. decrease) over a rational level the market begins to sell and prices experience a correction. boll_0 is significant at 5% for all assets and at 0.1% for **TSLA**, **AAPL** and **WMT** whereas boll_1 is only significant for **AF.PA** and **AAPL**.

Overall, the results from the logistic regression are positive. We have significant models, pseudo R^2 are relatively high with respect to the rest of the literature indicating the models explain a non negligible part of the returns generating process. The coefficients and respective significance levels are rational and coincide with common practice. We have confidence in the fact that technical indicators can, to a certain extent, predict future returns. These results are a first step to disproving the EMH. 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 off as we explain less of the variance. The inclusion of aforementioned types of variable definitely is a reasonable improvement to consider.

[13] Assuming returns can be predicted with the **ADX**.

3.2.2 Neural Networks and SVMs

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 simple network of two hidden layers of 10 neurons with ReLU activation functions and a final output layer of 2 neuron coupled with a softmax activation function. The first final output neuron will output the probability that the correct position is "buy" and the second final output neuron will give the probability that the correct position is "sell". 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.

Precision and Recall give us a comprehensive measure of the performance of the model. 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 Negatives. If the model is unable to classify the data 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 the model indeed identified. 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. In that case, we can use the f1-score, which is simply the harmonic mean of Precision and Recall. f1-score is then calculated as: $f_1 score = \frac{Precision*Recall}{Precision+Recall}$.

[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

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

Table 2: Confusion matrices

AF.PA

		True 0	True 1
Pred	0	580	335
Pred	1	254	497

AAPL

		True 0	True 1
Pred	0	275	117
Pred	1	504	744

TSLA

		True 0	True 1
Pred	0	293	116
Pred	1	496	735

WMT

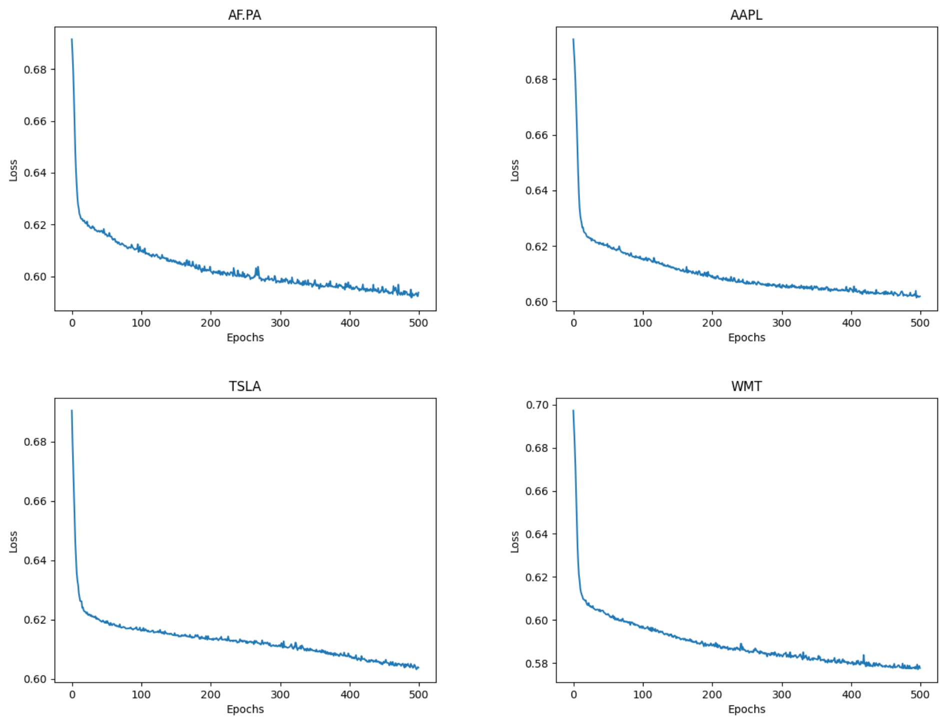
		True 0	True 1
Pred	0	354	145
Pred	1	419	722

The first Neural Network is able to classify the data, although it is far from perfect. For the first asset (**AF.PA**) we have a total of $580+497 = 1077$ correctly classified observations against $254 + 335 = 589$ misclassified ones. This translates to 64.65% of the sample correctly classified against 35.35% misclassified. Recall, Precision and F_1 are presented in Table 3. The Recall is 64%, the model correctly identified 64% of the buying and selling positions. The Precision is 65% which means among all observations classified as buy or sell, 65% of them where actually buy or sell. For **AAPL** we have a total of $275+744 = 1019$ correctly classified observations against $117+504 = 621$ misclassified ones. This translates to 62.13% of the sample correctly classified against 37.87% misclassified. Recall is 0.63 and Precision is 0.69, we correctly identified 63% of the buying and selling opportunities and 69% of the identified opportunities were correct. F_1 is 0.65, slightly above the F_1 for **AF.PA**, but we cannot say the model is better fitted on **AAPL** than **AF.PA**, due to randomness in the algorithm, the confusion matrix and resulting metrics can slightly vary each time the data is fed to the network. This 0.01 difference in F_1 can be attributed either to sampling distribution or better performance of the model on the second asset. For **TSLA** we have a total of $293 + 735 = 1028$ correctly classified observations against $496 + 116 = 612$ misclassified ones. This translates to 62.82% of the sample correctly classified against 37.32% misclassified. Precision is 0.76, definitely higher than for **AF.PA** and **AAPL**, the model is more precise i.e. among the observations classified as either buy or sell, more observations were correctly classified than with **AF.PA** or **AAPL**. Recall is 0.62 which leads to a 0.65 F_1, overall the model is not better suited for **TSLA** than for other assets. It is more parsimonious, and chooses to classify observations more carefully at the cost of not being able to classify observations for which the position is less clear. For the last asset (**WMT**) we have a total of $354 + 722 = 1076$ correctly classified observations against $419 + 145 = 564$ misclassified ones. This translates to 65.61% of the sample correctly classified against 34.39% misclassified. Recall is 65% and Precision is 68% for a resulting F_1 of 66%.

We are not able to detect an asset for which the model is strictly better suited although we notice a trade-off between Precision and Recall. Assets where Recall is higher have less Precision and assets where Precision is higher have less Recall. Depending on the behavior of the asset the model either detects a few but very clear buy and sell signals (hence more Precision) or more but less clear signals (increasing Recall). The goal is then to improve the model in order to conciliate Recall and Precision and increase both metrics if possible.

Figure 17 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 decreases in a standard exponential manner, which is what we usually observe. The optimization takes steeper steps down the loss function at the beginning, and as we approach the minimum, it starts taking smaller steps, leading to smaller decreases in the loss function. One thing to note is that with 500 epochs, the loss function does not seem to completely converge. In other words, the model requires to update weights more than 500 times to reach a minimum.

Figure 17: Learning Curves



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 different loss function suited to our problem. We will then be using the

Binary Cross Entropy (BCE) to try to improve our model's performance.

As seen in Table 3 after optimizing the model we achieve higher accuracy overall. For all the assets except **TSLA**, Precision, Recall and F_1 increased. The f1-score lies between 0.70 and 0.71 with the new model against [0.64:0.66] with the former model and increased for all assets. Precision ranges from 0.70 and 0.71 against [0.65:0.76] and increased for **AF.PA**, **AAPL** and **WMT**. Recall ranges from 0.70 and 0.71 against [0.62:0.65] and increased for all assets. For **TSLA** Precision went from 0.76 before optimization to 0.71 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 were able to reduce the Precision Recall trade-off and increase both metrics. The latest Neural Network seems much more stable across assets which might indicate a better fit, it is constantly more precise than the previous model.

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. The linear SVM performs as well as the first Neural Network on the training data. Except for **AF.PA** we achieve far greater Precision than both Neural Networks: 0.81 for **TSLA** as opposed to 0.76 before, 0.81 for **AAPL** as opposed to 0.70 and 0.83 for **WMT** instead of 0.71. The counterpart is lesser Recall compared to the second Network leading to F_1 slightly above what we obtained with the first Network. Compared to Neural Networks, linear SVM is equally efficient in signal detection but more precise on the signals it detects. 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][16]

Adding a kernel function and allowing for non linearity in the SVM did not improve the results. We achieve similar accuracy than with the linear SVM. f1-score lies between 0.64 and 0.69, Precision and Recall are very similar to the ones obtained with either the linear SVM or the first Neural Network. Overall, it seems higly non linear Neural Networks are able to detect more signals and SVM, linear or non-linear detect less signals but classifies them more accurately (it only detects very clear signals).

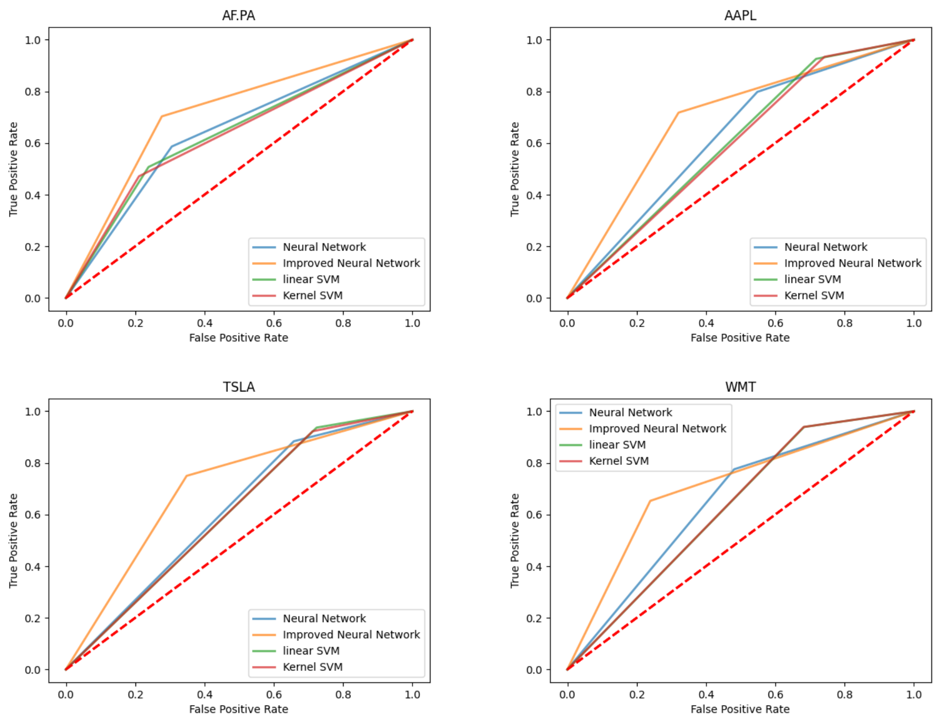
[16] $\gamma = \frac{1}{\sigma}$ and C represents the acceptable misclassification rate.

Table 3: Model Performance on training data

	TSLA	AF.PA	AAPL	WMT
Standard Neural Net				
P	0.76	0.65	0.69	0.68
R	0.62	0.64	0.63	0.65
F_1	0.65	0.64	0.65	0.66
Optimized Neural Net				
P	0.71	0.71	0.70	0.71
R	0.70	0.71	0.70	0.70
F_1	0.70	0.71	0.70	0.70
Linear SVM				
P	0.83	0.67	0.81	0.83
R	0.62	0.63	0.62	0.65
F_1	0.67	0.64	0.67	0.69
Kernel (non linear) SVM				
P	0.81	0.68	0.83	0.83
R	0.62	0.63	0.61	0.65
F_1	0.66	0.64	0.67	0.69

The ROC curves (Receiver Operating Characteristic curve) in Figure 18 indicates the ratio of True Positives divided by False Positives for different thresholds. The threshold is the probability at which an observation 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 area 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 less accurate 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).

Figure 18: ROC for training data



All models havce a positive ROCauc, and are able to classify future returns better than random. We can identify overperforming models based on the ROCauc, the figure seems to indicate the Optimized Neural Network performs better on training data than any other model. Linear and non-linear SVM are almost identical as we previously noticed and perform as well as or slightly better than the first Network even though the linear SVM is a simpler model. One interesting result is the fact that both SVMs and the simpler Neural Network consider very different thresholds than the Optimized Network. Moreover, the threshold is stable across all assets for the second Network whereas it oscillates for the other models. This illustrates the Precision Recall trade-off we noticed earlier. For very high thresholds, the model only detects very clear signals and will exhibit high Precision and low Recall meanwhile for very low thresholds the model detects more signals even if they are less clear resulting in high Recall and less Precision. We noticed how the Optimized Neural Network had the smallest trade-off among all the models, this is illustrated on the ROC curves by the different thresholds. Sampling distributions put aside, we suspect the relation between the technical indicators and future returns have a linear component and a highly non-linear component, this would coincides with earlier assumption about the nature of the relationship between our features and future returns. Another possible explanation is that we are not taking time into account, our models are not time series models. When looking at returns which are typically time series data, time series models might be more appropriate. Fitting recurrent or Long Short Term Memory Networks might improve the results.

According to the EMH, we should not have been able to detect significant relations between technical indicators at date t and returns at date $t + 1$. With logistic regression, we prove technical indicators carry information, although very sparse, on returns leading us to assume all the available information on the market is not instantly reflected in 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. We found all models considered we able to classify part of the data correctly. The Optimized Neural Network seems to be the best model so far, as it is capable of reducing the Precision Recall trade-off observed with other models. We were expecting models to perform better on 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 models perform better on assets with the clearest trend and less volatility **AAPL** and **WMT**. Model constantly under-perform on **AF.PA** which is the asset with the less significant trend and over-perform on **WMT** which steadily increases over the entire period, the second best suited asset is **AAPL** which exhibits a clear trend but is more volatile than **WMT**. It is often mentioned in the literature that technical indicators have more predictive power in bull markets than in bear markets, what we are witnessing probably is a manifestation of this phenomenon. The relationship between trend, volatility and the Precision-Recall trade-off is also very interesting and should be investigated.

3.3 Results

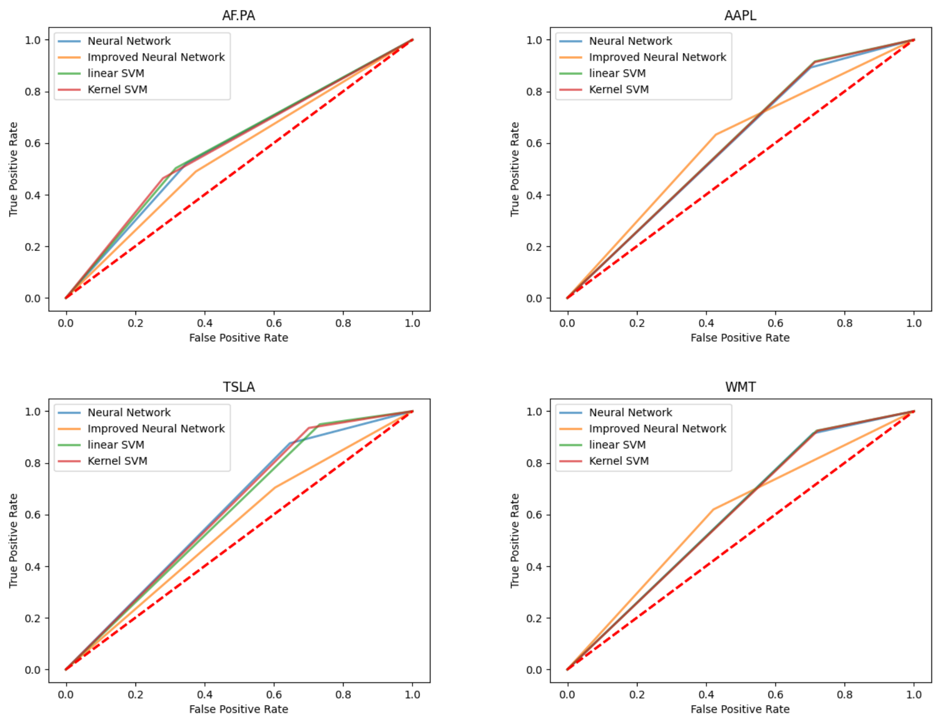
3.3.1 Accuracy on Testing Data

We can now evaluate the different models on testing data to evaluate their respective predictive power. We report the same metrics as before; Recall, Precision and f1-score and illustrate the results with ROC curves presented in Figure 19 and Table 4. 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.

Table 4: Model Performance on testing data

	TSLA	AF.PA	AAPL	WMT
Standard Neural Net				
P	0.75	0.59	0.70	0.81
R	0.62	0.58	0.62	0.64
F_1	0.65	0.58	0.64	0.69
Optimized Neural Net				
P	0.60	0.57	0.60	0.60
R	0.55	0.55	0.59	0.60
F_1	0.56	0.55	0.60	0.60
Linear SVM				
P	0.84	0.61	0.80	0.82
R	0.61	0.59	0.63	0.64
F_1	0.66	0.59	0.67	0.69
Kernel (non linear) SVM				
P	0.82	0.62	0.82	0.82
R	0.62	0.58	0.62	0.64
F_1	0.66	0.59	0.66	0.69

Figure 19: ROC for testing data



As expected, models are less precise on the testing data set compared to the training data set. Nonetheless we achieve satisfying predictions, Precison, Recall and F_1 are all greater than 0.5 for all assets and all models. The Optimized Neural Network is the worst model and also the model that experienced the greater decrease in accuracy between both data sets, showing signs of over-fitting. **AF.PA** is the asset for which precision metrics are the smallest and also the asset for which the decrease in accuracy between both data sets is the greatest. We already observed that the series exhibits no clear trend and models tends to fit better on series with a continuous upward momentum. On the opposite **WMT** has the highest F_1 among all assets for all models. Linear and non-linear SVMs achieve very similar accuracy, as it already was the case on training data. The non-linear SVM is not over-fitting the data because the transformation applied to the data is negligible. We discussed how the σ hyperparameter determines the intensity of the transformation and how very small σ can lead to over-fitting. Here σ is very large, hence the non-linear SVM is relatively close to the linear SVM. This explains both why the two models have similar accuracy and why the non-linear SVM does not suffer from over-fitting. The results are encouraging, the stability of the model over different data and proof of absence of over or under-fitting seem to indicate it is possible to obtain legitimate predictions of future returns with technical indicators.

The ROC for the standard Neural Network and the two SVMs are almost identical suggesting the three models have approximately the same accuracy. We observe the same phenomenon we observed on training data where thresholds are very different from these three models compared to the optimized Neural Network. The major difference is the under-performance of the latest model compared to the three others. The positive ROCauc further suggests the models are capable of predicting future returns to some extent.

Our results seem to disprove the EMH, we are able to build models capable of predicting future price direction better than random. On the other hand, we would like to remind the reader that we need economical scalability to fully state the EMH can be rejected.

3.3.2 Economic Scalability

To test whether or not our models yield better returns we are going to look at two risk-reward ratios of our strategy against the benchmark strategy. The benchmark is a buy & hold strategy on the S&P 500, this is based on the CAPM theory developed by W.Sharpe and H.Markowitz, where S&P 500 is a proxy for the market portfolio.

The Sharpe ratio is a risk-adjusted measure which allows us to compare strategies based not only on returns but also on the risk generated by these strategies. The risk-free rate is assumed to be 0, we justify this assumption because in reality the risk-free rate in very low, almost null (the rate of return of 30-years treasury bills is 0,007 on 05/14/2021). Furthermore assuming risk-free rate equal to 0 is equivalent to comparing absolute returns instead of excess returns and does not change the relative measure of performance between two strategies. The Sortino ratio is an extension of the Sharpe ratio which only accounts for the downside volatility. While volatility is a measure of the magnitude of price movements during a certain period, downside volatility (also called semi-deviation) is a measure of the negative price movements on the same period. It can be argued that downside volatility is a better measure of risk on financial markets. Because upside volatility measures the stretch of positive returns, it is difficult to consider it as risk for the investor. On financial markets, risk is associated with a probability of loss therefore downside volatility is a better proxy for the idea of financial risks as it excludes the positive price movements and focuses on losses.

The Sharpe ratio is then calculated as: $Sharpe = \frac{E[R_i]}{\sigma_i}$ where $E[R_i]$ is the annualized expected return of strategy i and σ_i is the annualized standard deviation of daily returns from strategy i. We assume 250 trading days to annualize returns and standard deviation. $Sharpe$ increases with $E[R_i]$ and decreases with σ_i, we therefore wish to achieve the highest possible ratio corresponding to a strategy with high reward and low risk. Independently of the value of $Sharpe$ we wish that $Sharpe_i > Sharpe_m$ where $Sharpe_i$ is the Sharpe ratio of the strategy i and $Sharpe_m$ is the Sharpe ratio of the benchmark strategy[17]. If $Sharpe_i > Sharpe_m$, the risk-reward ratio of our strategy is favorable against the benchmark buy & hold strategy. In other words we achieve greater returns than the market with the same level of risk, or the same returns as the market with a lower risk. This is assumed to be impossible if the CAPM theory and the EMH holds.

The Sortino ratio is defined as: $Sortino = \frac{E[R_i]-T}{\sigma_i^-}$ where T is a threshold below which we consider price movements as part of the downside volatility. T is set to 0, any return below 0 (every negative return) is considered to be part of the downside volatility. T can be chosen such that it is strictly positive or strictly negative depending on risk aversion, horizon, investing objectives and other factors. σ_i^- is the annualized standard deviation of returns below the threshold T, 0 in our case.

[17] m stands for market.

Our strategy consists in opening a long position whenever the model predicts a positive return for the next day and closing the position whenever next day returns are predicted to be negative. Position are kept opened as long as the prediction for future returns is positive. We are not shorting the market and returns are not compounded, each time our capital exceeds the initial investment, excess capital is secured and not reinvested. We can either assume inflation is null or that excess capital is invested with interest rates equal to the inflation rate. Once again either hypothesis does not change the measure of the relative performance of the two strategies. The benchmark strategy consists in opening a long position in the S&P 500 at the beginning of the period and secure excess capital whenever we are profitable. We are not considering the profit (or loss) generated by the sale of the asset at the end of the period as it is independent from the performance of the model. We report the main metrics in Table 5

Table 5: Economic results

Null model (buy & hold)					
	TSLA	AF.PA	AAPL	WMT	S&P 500
$E[R_i]$	0.16	0.10	0.22	0.18	0.10
σ_i	0.29	0.27	0.23	0.21	0.18
σ_i^-	0.26	0.20	0.21	0.19	0.17
Sharpe	0.55	0.36	0.98	0.86	0.54
Sortino	0.62	0.40	1.07	0.94	0.57
Standard Neural Net					
$E[R_i]$	0.63	0.51	0.39	0.40	
σ_i	0.18	0.22	0.21	0.19	
σ_i^-	0.25	0.2	0.21	0.16	-
Sharpe	2.34	2.26	1.87	2.06	
Sortino	2.53	2.58	1.88	2.46	
Optimized Neural Net					
$E[R_i]$	0.43	0.48	0.37	0.29	
σ_i	0.28	0.22	0.21	0.19	
σ_i^-	0.26	0.22	0.21	0.17	-
Sharpe	1.58	2.19	1.77	1.54	
Sortino	1.68	2.29	1.78	1.73	
Linear SVM					
$E[R_i]$	0.57	0.51	0.44	0.38	
σ_i	0.28	0.22	0.22	0.20	
σ_i^-	0.25	0.20	0.20	0.17	-
Sharpe	2.02	2.32	2.00	1.95	
Sortino	2.24	2.63	2.15	2.28	
Kernel (non linear) SVM					
$E[R_i]$	0.59	0.50	0.44	0.38	
σ_i	0.28	0.22	0.22	0.20	
σ_i^-	0.25	0.20	0.20	0.17	-
Sharpe	2.11	2.29	2.00	1.94	
Sortino	2.32	2.57	2.15	2.27	

From 07/02/2017 to 12/31/2019 the expected annualized return of the S&P 500 was 10% with a annualized volatility of 18%. The resulting Sharpe ratio is 0.54, the annualized semi-deviation is 0.17 and the Sortino ratio is 0.57. The downside volatility is less than the total volatility which indicates the negative price movements are less substantial or less frequent than positive price movements. In order for our models to yield greater risk-adjusted returns we need a Sharpe ratio greater than 0.54 and a Sortino ratio greater than 0.57. For all assets, every model yields greater expected returns than the S&P 500 but the volatility is also greater than the volatility of S&P 500. **TSLA** yields greater returns than other assets the majority of the time, bearing more risk represented by higher volatility and higher semi-deviation. On the opposite **WMT** yields less significant returns than most assets but bears less risk. As we previously noticed, the standard Neural Network, the linear SVM and the non-linear SVM are the more accurate models and they yield greater return and less volatility than the Optimized Neural Network. The standard Network seems to be the best model among all, but the over-performance might not be significant as the model relies on stochastic procedures. All Sharpe ratios are greater than 0.54 and all Sortino ratios are greater than 0.57, we were able to generate better risk reward trade-offs than the market. Technical indicators and machine learning seem to be capable of predicting future returns with enough accuracy to beat the market. Furthermore, for all assets at least one model yields a Sharpe ratio greater than 2 which is empirically considered to be a very acceptable risk-reward ratio. The smallest Sharpe ratio obtained is for **WMT** and the optimized Neural Network (1.54) and the largest Sharpe ratio is obtained with **TSLA** and the standard Neural Network. Sortino ratios are all greater than 0.57 going from 1.68 for **TSLA** and the optimized Neural Network to 2.63 for **AF.PA** and the linear SVM.

Results are very encouraging, we consistently achieve greater returns and/or less risk than the buy and hold strategy on the S&P 500. According to asset pricing theory, the risk-reward ratios for the S&P 500 should always be higher than for individual stocks, especially here as **WMT**, **AAPL** and **TSLA** are all included in the S&P 500. According to the Efficient Market Hypothesis, we should not have been able to predict future returns as all the information contained in the technical indicators should have already been reflected on prices. However, we were able to find significant parameters when predicting future returns and how machine learning models were able to correctly identify roughly 60% of the data. This finding was consistent across both training and testing and allowed us to generate greater returns and reduce risk leading to risk-reward ratios more advantageous than what was offered by the market index during the same period. We would also like to note that these results are consistent across different measures of risk (volatility and semi-deviation). Our results question both the EMH and the CAPM and further imply that financial markets are not efficient.

We can also see how the Sharpe and Sortino ratios increase when we compare the buy and hold strategy against our model's predictions for the different assets. We are able to achieve greater returns and less risk than the benchmark strategy, for all assets, leading to better ratios. It is interesting to see how we beat the performance of the market but also how the predictive power of each model also beats the benchmark strategy for each asset individually. It is especially important here as all assets expect **AF.PA** already have greater Sharpe and Sortino ratios than the S&P 500 with the benchmark strategy, excluding predictions from the various models. In other words, the model could have no predictive power, the assets would still individually beat the market. We make sure that the excess in both Sharpe and Sortino ratios compared to the S&P 500 is due to our models and not just to the fact that for the period, the assets over-perform the market on their own. This underscores how the results are sensitive to the period considered and how consistency over time is the only way to state that market are not efficient. On a different period of time our models could have under-performed the market. We can only say that we were able to show how, between, 07/02/2017 and 12/31/2019, market were not completely efficient. First because individual assets over-perform a large market index, which according to asset pricing theory should not happen because diversification supposedly allows for better risk-reward ratios. Also because for each asset, we were able to predict one day ahead returns with enough accuracy to over-perform the benchmark strategy on the S&P 500 but also over-perform the benchmark strategy on each asset individually.

Although we have proof that market were, during this time period, not completely efficient, we could not state they are completely inefficient. Machine learning has often been very inaccurate in case of black swans events, or large and sudden losses. We can imagine how in these scenarios, the models would under-perform the market and market as a whole would be considered efficient. Because we are studying a period chosen so we avoid such events (especially the 2008 subprime crisis and the 2020 Covid crisis) we are training and testing models on data that generally yields better results. Once again, consistency over extended periods of time with various market conditions, macroeconomic phenomenons and business cycles is key to prove markets are not efficient. However, we managed to select features and models that over-perform the market and create substantial economic value, which is a non negligible finding.

4 Conclusion

The goal was to explore the Efficient Market Hypothesis which states all the available information is already reflected in present prices and further implies one cannot predict future returns. In order to assess the veracity of this theory we decided to train machine learning models to predict future returns and see if models were performing enough to generate economic value. We used technical indicators as features to train our models, technical indicators are widely used by practitioners and often forsaken over macroeconomic variables or more well-founded financial measure of the performance of a particular asset. After going through the rationale and mathematics behind Neural Networks and Support Vector Machines we finally trained the model on financial data. We chose a total of four assets, originating from different markets and gathered daily data from 2010 August 8^{th} to 2019 December 31^{st}. Logistic regression indicates most of the chosen features were significant and were able to forecast one day ahead return with good accuracy; R^2 is around 0.10 for all assets and the likelihood ratio indicates the model reduces a significant part of the variance of the assets' returns. The signs of the regression coincide with common practice and are consistent across assets, leading us to think there exists a well-defined correlation between technical indicators and one day ahead returns. A total of four machine learning models were trained: a simple Neural Network which we later optimized leading to a second, more complex Neural Network, a linear SVM and a non-linear SVM using a radial basis function kernel. On training data all models achieve significant accuracy, especially the optimized Neural Network. There is no difference between the assets, although the accuracy for **AF.PA** is often lower than other assets. Simpler models are more parsimonious and detect less of the signals but all signals detected are verified, leading to superior Precision. The accuracy loss between training and testing data is very sparse expect for the optimized Neural Network indicating over-fitting. **AF.PA** remains the asset on which the models are worst suited and **WMT** is the asset on which the models are best suited. This leads us to think there is a relationship between the volatility and trend of the asset and the ability for a machine learning model to predict future returns. We are able to compute two different risk-adjusted metrics of the performance of the models: the Sharpe ratio and the Sortino ratio. All models over-perform the market with higher expected returns and less volatility than the S&P 500 index. Ratios are often above the value of 2 which indicate a substantial economic performance compared to the market. There seems to be a positive correlation between the Precision of the model and its economic performance, Precision seems to matter more when it comes to generating good economic results. This finding inspires us to further investigate the relation between the Precision-Recall trade-off and real life application of machine learning in finance. However, **WMT** which is over-performing in terms of accuracy metrics under-performs in terms of economic gains. We assume this implies machine learning is better suited for more volatile assets as the part of reduced variance is the same but it translates to more substantial gains. Two possible extensions to this work would be to assess the impact of the fundamental metrics of the returns on the performance of the models (expectation, deviation, skewness and kurtosis) and to reconcile the accuracy and the economic value of the models.

5 References

References

[AB97] Jeffrey S. Abarbanell and Brian J. Bushee. “Fundamental Analysis, Future Earnings, and Stock Prices”. In: *Journal of Accounting Research* 35.1 (1997), pp. 1–24.

[BT17] Daniel Buncic and Martin Tischhauser. “Macroeconomic factors and equity premium predictability”. In: *International Review of Economics and Finance* 51 (2017), pp. 621–644.

[CT12] Tolga Cenesizoglu and Allan Timmermann. “Do return prediction models add economic value?” In: *Journal of Banking & Finance* 36 (2012), pp. 2974–2987.

[CDK17] Amélie Charles, Olivier Darné, and Jae H. Kim. “International stock return predictability: Evidence from new statistical tests”. In: *International Review of Financial Analysis* 54 (2017), pp. 97–113.

[CG08] Rohit Choudhry and Kumkum Garg. “A Hybrid Machine Learning System for Stock Market Forecasting”. In: *World Academy of Science, Engineering and Technology* 39 (2008).

[CEP20] John Cotter, Emmanuel Eyiah-Donkor, and Valerio Poti. “Commodity Return Predictability: Economic Value and Links to the Real Economy”. Geary Institute, University College Dublin. 2020.

[DZK21] Zhifeng Dai, Huan Zhu, and Jie Kang. “New technical indicators and stock returns predictability”. In: *International Review of Economics and Finance* 71 (2021), pp. 127–142.

[Dec+01] Patricia M. Dechow et al. “Short-sellers, fundamental analysis, and stock returns”. In: *Journal of Financial Economics* 61 (2001), pp. 77–106.

[EDT12] Senol Emir, Hasan Dinçer, and Mehpare Timor. “A Stock Selection Model Based on Fundamental and Technical Analysis Variables by Using Artificial Neural Networks and Support Vector Machines”. In: *Review of Economics & Finance* 2 (2012), pp. 106–122.

[FK12] Eugene F.Fama and KennethR.French. “Size, value,and momentum in international stock returns”. In: *Journal of Financial Economics* 105 (2012), pp. 457–472.

[Fam70] Eugene Fama. “Efficient Capital Markets: A Review of Theory and Empirical Work”. In: *The Journal of Finance* 25.2 (1970), pp. 383–417.

[FGX20] Guanhao Feng, Stefano Giglio, and Dacheng Xiu. “Taming the Factor Zoo: A Test of New Factors”. In: *The Journal of Finance* 75 (2020), pp. 1327–1370.

[HSS13] Osman Hegazy, Omar S. Soliman, and Mustafa Abdul Salam. “A Machine Learning Model for Stock Market Prediction”. In: *International Journal of Computer Science and Telecommunications* 4.12 (2013), pp. 17–23.

[Kim14] Jae H. Kim. “Predictive regression: An improved augmented regression method”. In: *Journal of Empirical Finance* 26 (2014), pp. 13–25.

[MYS21] Yao Ma, Baochen Yang, and Yunpeng Su. “Stock return predictability: Evidence from moving averages of trading volume”. In: *Pacific-Basin Finance Journal* 65 (2021), pp. 1–23.

[MB15] S. Madge and S. Bhatt. “Predicting Stock Price Direction using Support Vector Machines”. In: 2015.

[MFZ20] David A. Mascio, Frank J. Fabozzi, and J. Kenton Zumwalt. “Market timing using combined forecasts and machine learning”. In: *John Wiley & Sons* 40.1 (2020), pp. 1–16.

[MR13] Phayung Meesad and Risul Islam Rasel. “Predicting stock market price using support vector regression”. In: *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*. 2013, pp. 1–6. DOI: 10.1109/ICIEV.2013.6572570.

[Nee+10] Christopher J Neely et al. *Forecasting the Equity Risk Premium:The Role of Technical Indicators*. Working Paper 2010-008H. Federal Reserve Bank of St. Lo, 2010.

[Sin+19] Sukhman Singh et al. “Stock Market Forecasting using Machine Learning: Today and Tomorrow”. In: *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*. Vol. 1. 2019, pp. 738–745. DOI: 10.1109/ICICICT46008.2019.8993160.

[WG08] Ivo Welch and Amit Goyal. “A Comprehensive Look at The Empirical Performance of Equity Premium Prediction”. In: *The Review of Financial Studies* 21.4 (2008).

[YKJ05] Paul D. Yoo, Maria H. Kim, and Tony Jan. “Machine learning techniques and use of event information for stock market prediction: A survey and evaluation”. University of Wollongong, Faculty Of Commerce. 2005.