Daniel Pace
Sara Toca

Backtesting in Financial Machine Learning

Comprehensive Assessment

Presented to Ivan T. Ivanov

Probability and Statistics
201-HTH-05

Vanier College
December 23, 2021

CONTENTS

ABSTRACT

This paper explores the uses of machine learning and artificial intelligence processes in the world of finance, namely in the practice of algorithmic trading. Machine Learning is an application of computer science and statistics that allows a computer algorithm to learn after being exposed to data. This automated approach is useful as it allows the processing of significant volumes of information at an extremely fast rate. This allows financial institutions and professional traders to act proactively and successfully participate in the market. Combined with backtesting techniques, machine learning has become an essential tool driving a large majority of today's trading decisions.

INTRODUCTION AND OVERVIEW

1. Machine Learning

Artificial intelligence, commonly known as AI, is defined as the simulation, by machines, of various behaviours associated with human intelligence. Otherwise stated, the latter can be defined as a direct application of machine learning. Combining computer science and statistical methods, machine learning allows computer programs to do specific tasks without having to program them to do so. With conventional programming, an algorithm is manually coded to generate an explicit output from an input set of data. Through this process, a computer is provided with information on the desired outcome and instructions on how to achieve it. On the other hand, machine learning consists of exposing the computer to the input and output training data from an area of interest, in order to create a program. Otherwise stated, computers self-learn by establishing patterns and analyzing sets of data in the same way that people learn through past experiences and examples. For simplicity, the process of machine learning can be illustrated by using the following example. Let's suppose that we require a computer that can tell the difference between a tomato and an apple. With the purpose of achieving this goal, various images of the two fruits can be fed to the machine. Consequently, a learning program will analyze those photos and begin to define patterns such as the shapes, sizes and colours that are specific to each fruit. By doing so, the

computer creates a sorting algorithm that will be able to recognize whether any new images shown to it belong to the tomato or apple category.

Needless to say, this a very simple example and in the real world, there are many applications of artificial intelligence and machine learning which are much more complicated. The machine learning approach is particularly useful when working with a larger variety of data or solving complex problems, tasks that cannot be completed by humans. As the program is exposed to more and more data, it continuously learns and improves, resulting in the most accurate algorithm being produced. This allows machines to make predictions and decisions more efficiently as the process requires less computing power than traditional programming and yields better results.

1.1 Machine Learning in Finance

From facial recognition to detection of credit card fraud and even the diagnosis of certain medical conditions, machine learning has become a key element in the field of data science. Nowadays, it is also widely used in the sphere of computational finance. Namely, machine learning processes play a major role in the practice of algorithmic trading. The latter is also known under the name of black-box trading and defines a method of trading that uses machine learning programs to open and close trades in the electronic financial market. These algorithms operate according to pre-defined criteria and allow for trading at a higher frequency than manual trading could ever achieve. Successful trading and making profits on the stock market requires a lot of attention given that there are numerous variables to be considered at an extremely fast rate. Human traders simply cannot process the exceedingly large volumes of information needed to make the right decisions and keep up with the ever-changing market. As a result, most of today's transactions are executed by artificial intelligence.

In finance, machine learning has many different applications for successful trading. Namely, one of its main tasks consists of simultaneously analyzing historical data sets and information from many other real-time sources in order to learn features to be able to accurately

predict the next move of the market. Combined with the experience of a professional trader, these features can be used to create trading strategies and possibly generate significant returns.

BACKTESTING

Backtesting is a very important part of successful trading as it consists of applying a trading strategy on historical data with the goal of evaluating how well it would have performed in the past. This allows traders to obtain feedback on a strategy before making trades given that it relies on the belief that if a trading technique does well in the past, it is more likely for it to perform well in the future and vice versa. Backtesting allows traders to evaluate the potential as well as the performance of a given strategy and combined to the machine learning approach, it can represent an important tool to generating profits on the stock market.

2.1 Implementation

Unlike machine learning, backtesting algorithms are originally created through the process of traditional programming. Of course, nowadays with all the resources made available on the internet, one can easily find a pre-existing programming package to backtest a strategy or simply use software specifically designed for it.

2.2 Backtesting with Python

As mentioned, there are many ways of backtesting a trading strategy. One manner involves using trading platforms or specific backtesting software such as Metatrader and Amibroker. These platforms offer a simple way of doing algorithmic trading using normal indicators but require payment and do not offer the same intensive analysis as can be done using programming software. Developing custom software using a language such as python allows for a more involved approach, using uncommon analyses, like those found in Section 2.3. This more involved approach comes, of course, at the cost of resources and time, although there are packages available to alleviate the load.

Running a backtest using Python is relatively simple if using simple indicators such as moving average crossovers. It will require gathering the data to perform the backtest and establishing a strategy. Gathering the data can be done using the Pandas Data Reader package as such:

```
import pandas_datareader as pdr
import pandas as pd
df = pdr.get_data_yahoo(symbols='SPY', start = '2011-12-1', end ='2021-12-1')
```

This will pull information from Yahoo! Finance and form a data frame of the historical data containing the daily high, low, volume and adjusted close, which can then be used with the actual backtesting package. The package used below is Backtesting.py, which can be found in the Python Package Index and installed using the pip installer. For this package, a few things must be defined at first. A function must be created to calculate the exponential moving average:

```
def EMA (values, n):
return pd.Series(values).ewm(span=n, min_periods=0, adjust=False).mean()
```

Next the strategy must be created, by forming three separate moving averages with the wanted parameters, in this case using 7, 27 and 200. Next, we must also define what will trigger the buy and sell signals. For moving averages, the signals will be crossovers between the different moving averages, which indicate changes in trend directions. If the 7-day average crosses over the 200-day average, this will indicate an upward trend and trigger a buy signal. If the 200-day average and the 27-day average cross the 7-day average, it will show a downward trend and will trigger a sell signal:

```
from backtesting import Backtest, Strategy
from backtesting.lib import crossover
class EMAC(Strategy):
    def init(self):
```

```
        price = self.data.Close
        self.ema1 = self.I(EMA, price, 7)
        self.ema2 = self.I(EMA, price, 27)
        self.ema3 = self.I(EMA, price, 200)
    def next(self):
        if crossover(self.ema1, self.ema3):
            self.buy()
        elif crossover(self.ema3, self.ema1) and crossover(self.ema2, self.ema1):
            self.sell()
```

Now that we have the historical data and the strategy set up, all that is left is to do the actual backtest. Using an initial cash value of $10,000 and ignoring commission, the backtest is executed as such:

```
bt = Backtest(df, EMAC,cash=10000, commission = 0.000, exclusive_orders= True)
bt.run()
```

Running the backtest returns a table with many metrics:

Start	2011-12-01 00:00:00
End	2021-12-01 00:00:00
Duration	3653 days 00:00:00
Exposure Time [%]	99.8808
Equity Final [$]	34199.2
Equity Peak [$]	34797.9
Return [%]	241.992
Buy & Hold Return [%]	260.487
Return (Ann.) [%]	13.1007
Volatility (Ann.) [%]	18.3976
Sharpe Ratio	0.712088
Sortino Ratio	1.12375
Calmar Ratio	0.384712
Max. Drawdown [%]	-34.0533
Avg. Drawdown [%]	-1.64627
Max. Drawdown Duration	418 days 00:00:00
Avg. Drawdown Duration	19 days 00:00:00
# Trades	13
Win Rate [%]	53.8462
Best Trade [%]	52.8002
Worst Trade [%]	-3.22292
Avg. Trade [%]	9.94354
Max. Trade Duration	969 days 00:00:00
Avg. Trade Duration	281 days 00:00:00
Profit Factor	12.9689
Expectancy [%]	11.3125
SQN	1.89003
_strategy	EMAC
_equity_curve	...
_trades	Size EntryB...
dtype: object	

The information within the table can be very useful for determining the performance of the strategy implemented. It shows the percent return, which is the total amount of profit made as a percentage of the initial cash value. The next statistic is the annualized volatility, which is the measure of the variance of the returns over time. A higher volatility means increased risk. An important statistic for many traders is the Sharpe Ratio. This is used to quantify the return of an investment as compared to its risk. The ratio is calculated as:

$$Sharpe\ Ratio = \frac{R_x - R_f}{\sigma_x},$$

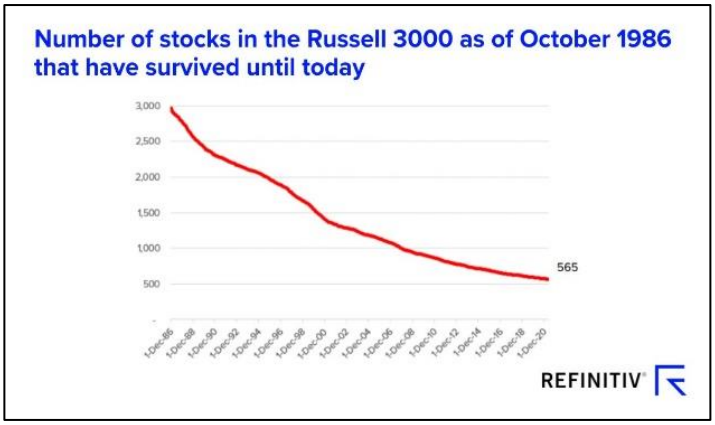
Where R_x is the average return, R_f is the risk-free rate return and σ_x is the standard deviation of the portfolios returns. The risk-free rate is the return expected from a security which has minimal volatility such as bonds.

While backtesting is an important tool for a quantitative investor, there are important factors that many people overlook. All backtest are hypothetical and so must not be taken at their face value and great care must be taken to avoid falling for common mistakes.

2.3 Problems with Backtesting

2.3.1 Survivorship Bias

In the year 1943, the American military was suffering heavy losses of its Bombers, which were continuously being gunned down by the German anti-aircraft defenses. To minimize these losses, the government knew that they needed to add armor to the planes, but could not reinforce them entirely, or they would be too heavy to fly. The military began to examine the damage incurred by the planes returning to base and reinforcing the areas with the most frequent damage. A mathematician named Abraham Wald that was asked to aid the military in this endeavor realized the logical error that was being committed. The military failed to forget one key aspect; the planes that were being examined managed to make it back to base, while all the planes that didn't were being excluded from the sample. The areas of the planes that the military was initially planning to reinforce were the exact opposite areas that required it. These could sustain bullet fire and still function and all the areas where the planes returned undamaged were to be armored[1]. This story of survivorship bias applies largely to the process of backtesting and is something that many people overlook in their tests. When backtesting, practitioners tend use companies that are currently in business, leaving out companies that were delisted due to bankruptcy or acquisitions. On average, companies that survive tend to perform better, so failing to account for equities that did not survive could yield some falsely high-performing results. In fact, looking at the components of the Russell 3000 in 1986 and comparing it to the present-day components, under 20% of them survived. The best way to account for this, if using an index like the S&P 500 is to use a list of stocks which were in the index at a given point in time, excluding companies that were added at a later date. While this might not be a be problem for professional portfolio managers and investors, access to historical data on



[1] Miller, Brendan. "How 'Survivorship Bias Can Cause You to Make Mistakes." *BBC Worklife*, BBC, 28 Aug. 2020, https://www.bbc.com/worklife/article/20200827-how-survivorship-bias-can-cause-you-to-make-mistakes.

delisted securities is hard to come by for the average person. Such data is not listed on common financial data sources such as Yahoo or Bloomberg. The easiest way would be by accessing databases which store the information, but they require subscriptions, the prices of which are steep.

2.3.2 Strategy Selection

When strategies have been developed, they will be backtested to measure their performance on historical data, and the best one will be selected for real-world use. While the selection process might seem evident; simply choosing the strategy that has the best return, it is not that simple. When developing models and algorithms for machine learning, they are trained on a data set called the 'Training Set'. If the model is tweaked or modified in any to yield a better result on the training set, it will be overfit. Overfitting is a concept in statistics where a model fits exactly on a data set. If this is done, the model will not optimally perform on future data, what is called the 'Training Set'. This renders the model useless for real world application. For a financial investment model, this means that a strategy is made to produce the best performance on a backtest, but this would not mean that it will work for the future. If the model is made to outperform on a specific timeframe, that means that it will be using mostly random historical patterns which will most likely never occur again. Addressing ways to avoid overfitting is no short task, but some actions could be done to alleviate the possibility. A very important thing is to realize that backtesting is to be used simply for testing the models, not for improving them. The models should be developed before the backtesting process. Adjusting a model to fit a backtest will nullify any performance it seems to have. This will significantly lower the chances of falsities but will never be zero. Since the number of strategies that can be implemented, with varying parameters can be in the thousands, even if a strategy has not been modified post-backtest, it can still be overfit, due to the yield of a false positive. Methods for testing for overfitting have been developed, but each has its own advantages and disadvantages.

2.3.2.1 The T-Statistic

As the number of tests increases, so does the chance of obtaining a seemingly significant value. To declare something an actual discovery, it must be a certain number of standard deviations removed from the null hypothesis of zero profitability. This is given by the t-statistic:

$$T - statistic = \ Sharpe\ Ratio \times \sqrt{Number\ of\ years}$$

Using the Sharpe Ratio given by the backtest in Section 2.2 (0.712088), we obtain a statistic of 2.25. The probability of this occurring, using $Number\ of\ Years - 1$ as the degrees of freedom, yields just over 2.5%. This percentage is the chance that the Sharpe Ratio was a fluke. This would be well justified if the backtester was simply running an independent test, but this is rarely the case. Multiple testing is needed for any kind of research, and so the chances of false positives will also be very large. Correcting for multiple testing can be done through many different methods.

2.3.2.2 The Bonferroni Test

The family-wise error rate (FWER) defines the probability for false discoveries multiple testing, where any false positives would be unacceptable. The most popular, and simplest, test for using this method is the Bonferroni test. It simply states that as the number of tests increases, so the level of significance will decrease. The general formula is given as:

$$100\% - \frac{\alpha}{m},$$

Where α is the level of significance and m is the number of trials. If an initial confidence interval was 95% for an independent test, this leaves a 5% chance for false discovery. For 10 tests, the adjusted interval would be $100\% - \frac{5\%}{10} = 99.5\%$. Obtaining a significant test using this confidence interval from a t-statistic would require a value of over 2.8. Anyone can see that this value goes up drastically if the number of tests goes into the thousands. This could be seen as too

stringent a test, leaving space for missed discoveries. If a less stringent test is needed, other methods can be used.

2.3.2.3 The Benjamini-Hochberg-Yekutieli Test

The Benjamini-Hochberg-Yekutieli (BHY) test falls under the umbrella of a false discovery rate (FDR) approach to multiple testing. This approach is much less severe than the FWER and is also fairly easy to implement. The BHY test begins by running the individual *p*-values in ascending order and assigning ranks to them. In this case the smallest value would have rank 1, the second smallest rank 2 and so on. The formula is given as:

$$p_k = \frac{k \times \alpha}{M \times c(M)},$$

Where k is the rank, α is the level of significance, M is the number of tests and c(M) is the normalizing constant. This constant is a function that is increasing in M and is given as:

$$c(M) = \sum_{i=1}^{M} \frac{1}{i}$$

Starting with the largest, the p-values are compared with their calculated thresholds. As soon as one is below its threshold, it and all subsequent values are said to be significant.

2.3.2.4 The Combinatorically Symmetric Cross-Validation

The combinatorically symmetric cross-validation (CSCV) was first proposed by Bailey et al. (2014, 2016) as a way of estimating the probability of backtest overfitting (PBO). To implement this method, we begin by collecting performance information of the multiple strategies as a function of time. A matrix **M** is formed with this information, where every column represents the performance of individual strategies, given as **N**, over the observations (time) **T.** The matrix

created will then have dimensions T x N (See Appendix). Next the matrix must be subdivided through the rows into an even number, S, of submatrices M_s , where the dimension of each submatrix will be $T/S \times N$. As such, a matrix which contained 64 rows, if subdivided into S = 16 submatrices, will have N columns, each one for a strategy, and will have $64/16 = 4$ rows. Next the submatrices will be reconstituted in every possible combination of size $S/2$. Using the previous example, this means that 8 out of the 16 submatrices will be recombined to form another matrix, which is a training set J of order $(T/2 \times N)$. This is done for every combination ${}_SC_{S/2}$, therefore is S = 16, the number of possible combinations will be 12,780. For each combination J, the testing set $\bar{J}$ is formed from its complement. The testing set is formed by the submatrices that are not found in the training set. These two sets can be seen as the in sample and out of sample data from traditional cross-validation techniques. For every possible combination of the submatrices, half of them will be found in the training set (J) and the other half will be in the testing set ($\bar{J}$). A performance vector R is created from the training set, by taking the mean performance and dividing it by the standard deviation. Another performance vector is created for $\bar{J}$, named $\bar{R}$. The best performing strategy in R is found and is compared to the other strategies in $\bar{J}$. Its percentile rank in $\bar{J}$ is determined as $\bar{\omega}_c \in (0, 1)$. A logit value is defined as $\lambda_c = \log\left[\frac{\bar{\omega}_c}{1-\bar{\omega}_c}\right]$ (See Appendix for python example of CSCV). A high logit value is indicative of a consistency between the training set and the testing set, meaning it is not very much overfit. All the logit values for every combination are collected in a histogram, where a probability density function ($f(\lambda)$) can be fit, where $\int_{-\infty}^{\infty} f(\lambda)d\lambda = 1$. Finally, the probability that the backtest is overfit can be estimated as $\emptyset = \int_{-\infty}^{0} f(\lambda)d\lambda$. This will give the probability of in sample optimal strategies that underperformed out of sample. If $\emptyset \approx 0$, then a low proportion of the in sample optimal strategies outperformed out of sample which indicates no overfitting. If $\emptyset \approx 1$, then there is significant overfitting. The general value of $\emptyset$ provides a quantitative measure of overfitting, and so any value deemed too high, such as larger than 5%, can be rejected. The estimates for the probability of overfitting can also be used to compute a weighted portfolio given by a certain scheme such as $1/PBO$.

CONCLUSION

In conclusion, artificial intelligence is a subset of machine learning that consists of a computer algorithm learning and providing accurate insights after being exposed to data. Given their efficiency in making predictions, machine learning algorithms are widely used in the finance industry, namely by professionals in the field of algorithmic trading. Among other things, the information provided by such algorithms can be used to build trading strategies which can further be used to generate profits. Before making trades, these techniques can be subjected to backtesting in order to measure their potential without risking capital. These processes involve the application of a trading strategy on historical data to gain insight on its performance relative to different factors. However, interpreting backtesting results and appropriately conducting a backtest can be very difficult. Some of the problems with backtesting include the survivorship bias which consists of only considering stocks that are available in the present while disregarding the ones that do not exist anymore but were part of the market in the past. Another known issue related to backtesting is the very possible occurrence of overfitting which renders the results unapplicable in future situations. While it has its downsides and one should not solely rely on backtesting and machine learning to participate in the market, they are both key tools heavily relied on by financial institutions as well as hedge fund managers to conduct successful trading activities.

REFERENCES

1. Alpaydin, Ethem. *Machine Learning : The New AI.* The MIT Press, 2016. pp.16-28. *EBSCOhost,* https://bit.ly/3oobaRK.

2. N.A., "Backtesting." *Corporate Finance Institute*, 2016, https://bit.ly/3pyhqWt

3. Kuepper, Justin. "The Importance of Backtesting Trading Strategies." *Investopedia*, 31 July 2021, https://bit.ly/3yay2rf

4. Campbell R., Harvey, and Yan Liu. "Evaluating Trading Strategies." *The Journal of Portfolio Management*, vol.40, no.5, 2014, pp.108-118.

5. Bailey, David H. et al. "The Probability of Backtest Overfitting." (*Journal of Computational Finance*, vol.20, no.4, 2016, SSRN, https://bit.ly/3orDtyy.

6. Lopez de Prado, Marcos. *Advances in Financial Machine Learning.* John Wiley & Sons, 2018.

7. Miller, Brendan. "How 'Survivorship Bias Can Cause You to Make Mistakes." *BBC Worklife*, BBC, 28 Aug. 2020, https://www.bbc.com/worklife/article/20200827-how-survivorship-bias-can-cause-you-to-make-mistakes.

APPENDIX

Example of The Combinatorically Symmetric Cross-Validation Matrix M:

	SMAC	*EMAC*	*RSISystem*
0	*-164.919*	*-3.85023*	*0*
1	*-199.509*	*0*	*0*
2	*0*	*0*	*0*
3	*82.18047*	*0*	*0*
4	*498.8697*	*419.5201*	*0*
5	*-599.951*	*-196.02*	*-77.2199*
6	*23.1004*	*201.6696*	*0*
7	*440.2189*	*-1135.26*	*0*
8	*717.1005*	*393.47*	*0*
9	*0.780167*	*330.6405*	*0*
10	*0*	*0*	*0*
11	*-1336.79*	*603.8601*	*0*
12	*977.7593*	*-607.64*	*0*
13	*-842.999*	*0*	*0*
14	*-435.651*	*0*	*0*
15	*-127.361*	*171.9899*	*0*
16	*0*	*835.7099*	*1337.34*
17	*-731.521*	*0*	*0*
18	*647.9199*	*516.5602*	*989*
19	*187.7995*	*0*	*0*
20	*-199.061*	*27.36005*	*0*
21	*-75.7496*	*-253.819*	*0*
22	*-1114.2*	*-933.14*	*-660.44*
23	*-81.1602*	*199.0797*	*0*
24	*-84.9606*	*0*	*0*
25	*-14.7998*	*0*	*0*
26	*52.62079*	*-63.3396*	*725.1991*
27	*192.5*	*65.55084*	*264.601*
...	...	...	...

Combinatorially Symmetric Cross-Validation with Python

```
from itertools import combinations
import pandas as pd
import numpy as np
import math
#Create the matrix M as a data frame
df= pd.read_csv(r'Performance.csv')
submatrix = {}
#Create Submatrices Ms in a dictionary
for i in range(0, 64, 4):
   submatrix[i]= df.iloc[0+i:4+i]
#For each combination of 8 keys from dictionary
for i in combinations(submatrix, 8):
   training_set={}
   testing_set = {}
#Define the training set
   for element in (i):
      training_set[element] = submatrix[element]
#Define testing set as complement of training set
   for x in submatrix:
      if x not in training_set:
         testing_set[x] = submatrix[x]
#Form performance statistic vectors for training and testing set
   J = pd.concat(training_set)
   Jbar = pd.concat(testing_set)
   R = J.mean()/J.std()
   Rbar = Jbar.mean()/Jbar.std()
#Determine best performer in R
   best = R[['SMAC', 'EMAC','RSISystem']].idxmax()
#Determine relative rank of best performer in R̄
   Rbar['pct_rank'] = Rbar.rank(pct=True)
   rank = Rbar['pct_rank'][best]
#Define the logit
   logit = math.log(rank/(1-rank))
```