

ΠΑΝΤΕΙΟΝ ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΟΙΝΩΝΙΚΩΝ ΚΑΙ ΠΟΛΙΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

PANTEION UNIVERSITY OF SOCIAL AND POLITICAL SCIENCES



ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΔΗΜΟΣΙΑΣ ΔΙΟΙΚΗΣΗΣ
ΤΜΗΜΑ ΟΙΚΟΝΟΜΙΚΗΣ ΚΑΙ ΠΕΡΙΦΕΡΕΙΑΚΗΣ ΑΝΑΠΤΥΞΗΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΕΦΗΡΜΟΣΜΕΝΩΝ ΟΙΚΟΝΟΜΙΚΩΝ ΚΑΙ ΔΙΟΙΚΗΣΗΣ»

Machine Learning Forecasting Oil Prices

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Καζαμίας

Αθήνα, 2020

Τριμελής Επιτροπή

Σταύρος Ντεγιαννάκης, Αναπληρωτής Καθηγητής Παντείου Πανεπιστημίου

(Επιβλέπων)

Κλάιβ Ρίτσαρντσον, Ομότιμος Καθηγητής Παντείου Πανεπιστημίου

Γρηγόρης Σιουρούνης, Επίκουρος Καθηγητής Παντείου Πανεπιστημίου



Copyright © Γεώργιος Καζαμίας, 2020

All rights reserved. Με επιφύλαξη παντός δικαιώματος.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας διπλωματικής εργασίας εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της διπλωματικής εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Η έγκριση της διπλωματικής εργασίας από το Πάντειον Πανεπιστήμιο Κοινωνικών και Πολιτικών Επιστημών δεν δηλώνει αποδοχή των γνώμων του συγγραφέα.

*Στη σύζυγο μου Γιούλη,
που παρότι δεν της αρέσει η αναφορά του τίτλου «σύζυγος» στον ζυγό και τα
προβλήματα που αντιμετωπίζουμε μαζί, υπήρξε σημαντική υποστηρίκτρια όλων των
προσπαθειών μου ακαδημαϊκών και μη, όλα αυτά τα χρόνια.*

Συντομογραφίες

AI: Artificial Intelligence

ANN: Artificial Neural Network

AR: Autoregressive

ARMA: Autoregressive – Moving Average

BNN: Binarized Neural Network

BVAR: Bayesian Autoregression

CNN: Convolutional Neural Network

CNN_HF: Convolutional Neural Network using ultra-high-frequency data

CNN_PA: Convolutional Neural Network using Palladium Realized Volatility

CNN_XX: Convolutional Neural Network using Eurostoxx 50 Index Realized
Volatility

COVID-19: Coronavirus Disease 2019

FTSE100: Financial Times Stock Exchange 100 Index

ICE: Intercontinental Exchange

LSTM: Long Short-Term Memory

LSTM_HF: Long Short-Term Memory using ultra-high-frequency data

LSTM_PA: Long Short-Term Memory using Palladium Realized Volatility

LSTM_XX: Long Short-Term Memory using Eurostoxx 50 Index Realized Volatility

MAPE: Mean Absolute Percentage Error

MAPPE: Mean Absolute Percentage Predicted Error

MIDAS: Mixed-Data Sampling framework

MIDAS-RV: MIDAS model based on an Asset's Realized Volatility

MIDAS-RV-PA: MIDAS model based on Palladium Realized Volatility

MIDAS-RV-XX: MIDAS model based on the EuroStoxx 50 index Realized Volatility

MIDAS-RV-BP: MIDAS model based on the GBP/USD exchange rate Realized
Volatility

ML: Machine Learning

MLP: Multilayer Perceptron

MLP_HF: Multilayer Perceptron using ultra-high-frequency data

MLP_PA: Multilayer Perceptron using Palladium Realized Volatility

MLP_XX: Multilayer Perceptron using Eurostoxx 50 Index Realized Volatility

MSE: Mean Squared Error

MSPE: Mean Squared Predicted Error
NYMEX: New York Mercantile Exchange
OAPEC: Organization of Arab Petroleum Exporting Countries
OPEC: Organization of Petroleum Exporting Countries
PA: Palladium
PCF: Price of Food
PCI: Price of Industrial raw materials
PCM: Price of Metals
PCO: Price per barrel of crude oil in dollars
R2: Coefficient of Determination
RFR: Random Forest Regressor
RFR_HF: Random Forest Regressor using ultra-high-frequency data
RFR_PA: Random Forest Regressor using Palladium Realized Volatility
RFR_XX: Random Forest Regressor using Eurostoxx 50 Index Realized Volatility
RNN: Recurrent Neural Network
RWM: Random Walk Model
S&P500: Standard & Poor's 500 index
USDXX: US Dollar Index
VAR: Vector Autoregression
VIX: Volatility Index.
WTI: West Texas Intermediate
XX: Eurostoxx 50 index

Ευχαριστίες

Θερμές ευχαριστίες στους κυρίους Σταύρο Ντεγιαννάκη, αναπληρωτή καθηγητή Παντείου Πανεπιστημίου, και Γεώργιο Φίλη, επίκουρο καθηγητή Πανεπιστημίου Πατρών, για την παραχώρηση των δεδομένων που χρησιμοποιήθηκαν σε αυτή την μελέτη.

Ιδιαίτερη αναφορά θα ήθελα να κάνω στην σημαντική συμβολή του κ. Σταύρου Ντεγιαννάκη, ο οποίος ήταν επιβλέπων καθηγητής για την συγγραφή της παρούσας. Οι καίριες παρατηρήσεις του και η έμπειρη επιστημονική του ματιά μου έδωσαν την αναγκαία ώθηση.

Περιεχόμενα

Συνοτομογραφίες	4
Ευχαριστίες.....	6
Περιεχόμενα.....	7
Κατάλογος πινάκων.....	9
Κατάλογος γραφημάτων, εικόνων κοκ	9
Περίληψη.....	13
Abstract	14
Introduction.....	15
The case with oil prices	17
Crude oil benchmarks.....	17
Fundamentals	19
Financialization.....	20
Volatility.....	22
Forecasting	23
Machine Learning forecasting	27
Machine Learning Definition	27
Types of Machine Learning	28
Supervised learning.	28
Unsupervised learning.....	28
Reinforcement learning.....	29
Methods and promises.....	29
ML for time-series methods.	29
Comparison to Statistic methods.	47
The baseline method.....	51
The ML methods in action.....	54
Data	54
Models used	57
Random Forest Regressor models.....	59
MLP models.	60
CNN models.....	62
LSTM models.	65
Model evaluation	68
Results	70
Conclusions.....	92

Πηγές – Βιβλιογραφία.....	94
Α' Ιστοσελίδες.....	94
Β' Βιβλιογραφία	94
Παράρτημα.....	98

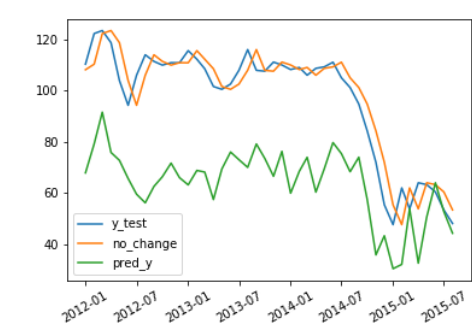
Κατάλογος πινάκων

Table 1. Formatting time-series lag for use with ML model	55
Table 2. Assets used in Degiannakis and Filis 2018 paper	56
Table 3. Models created and trained to forecast oil prices	59
Table 4. 1-month ahead forecasting horizon	71
Table 5. 12-month forecasting horizon.....	72

Κατάλογος γραφημάτων, εικόνων κλπ

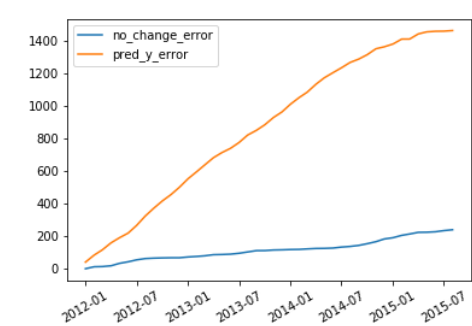
Figure 1. Crude oil front-month futures prices collapse for WTI on April 2020 due to COVID-19 pandemic	19
Figure 2. AI sector.....	29
Figure 3. Random Forest Prediction.....	33
Figure 4. MLP with one hidden layer and one output node	36
Figure 5. CNN image transformation using filters.....	39
Figure 6. CNN schematic showing the various layers	40
Figure 7. The LSTM cell anatomy	46
Figure 8. Representation of the layer connectivity in MLP_HF model	62
Figure 9. Representation of the layer connectivity in plain CNN model.....	63
Figure 10. Representation of the layer connectivity in CNN_HF model	66
Figure 11. Representation of the layer connectivity in LSTM_HF model.....	68
Figure 12. MLP 1-month forecasting.....	77
Figure 13. MLP 1-month forecasting cumulative error.....	77
Figure 14. MLP 1-month forecasting error distribution.....	77
Figure 15. MLP 1-month forecasting cumulative error distribution	77
Figure 16. MLP 1-month forecasting using Palladium	77
Figure 17. MLP 1-month forecasting cumulative error using Palladium	77
Figure 18. MLP 1-month forecasting error distribution using Palladium	78
Figure 19. MLP 1-month forecasting cumulative error distribution using Palladium.....	78

Figure 20. MLP 1-month forecasting using Eurostoxx 50



..... 78

Figure 21. MLP 1-month forecasting cumulative error using Eurostoxx 50



..... 78

Figure 22. MLP 1-month forecasting error distribution using Eurostoxx 50..... 78

Figure 23. MLP 1-month forecasting cumulative error distribution using Eurostoxx 50..... 78

Figure 24. CNN 1-month forecasting..... 79

Figure 25. CNN 1-month forecasting cumulative error..... 79

Figure 26. CNN 1-month forecasting error distribution..... 79

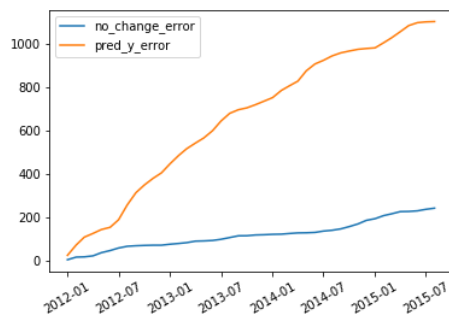
Figure 27. CNN 1-month forecasting cumulative error distribution..... 79

Figure 28. CNN 1-month forecasting using Palladium



..... 79

Figure 29. CNN 1-month forecasting cumulative error using Palladium



..... 79

Figure 30. CNN 1-month forecasting error distribution using Palladium 80

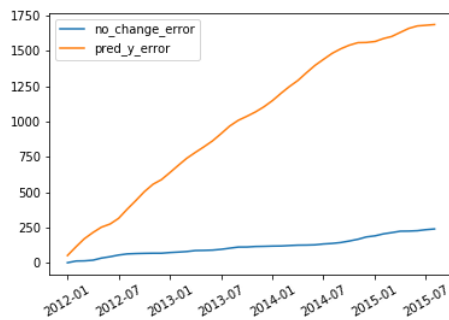
Figure 31. CNN 1-month forecasting cumulative error distribution using Palladium..... 80

Figure 32. CNN 1-month forecasting using Eurostoxx 50



..... 80

Figure 33. CNN 1-month forecasting cumulative error using Eurostoxx 50



..... 80

Figure 34. CNN 1-month forecasting error distribution using Eurostoxx 50 80

Figure 35. CNN 1-month forecasting cumulative error distribution using Eurostoxx 50..... 80

Figure 36. LSTM 1-month forecasting 81

Figure 37. LSTM 1-month forecasting cumulative error 81

Figure 38. LSTM 1-month forecasting error distribution 81

Figure 39. LSTM 1-month forecasting cumulative error distribution 81

Figure 40. LSTM 1-month forecasting using Palladium..... 81

Figure 41. LSTM 1-month forecasting accumulative error using Palladium 81

Figure 42. LSTM 1-month forecasting error distribution using Palladium..... 82

Figure 43. LSTM 1-month forecasting accumulative error distribution using Palladium 82

Figure 44. LSTM 1-month forecasting using Eurostoxx 50.....	82
Figure 45. LSTM 1-month forecasting accumulative error using Eurostoxx 50	82
Figure 46. LSTM 1-month forecasting error distribution using Eurostoxx 50	82
Figure 47. LSTM 1-month forecasting accumulative error distribution using Eurostoxx 50 ..	82
Figure 48. MLP 12-month forecasting.....	83
Figure 49. MLP 12-month forecasting accumulative error	83
Figure 50. MLP 12-month forecasting error distribution.....	83
Figure 51. MLP 12-month forecasting accumulative error distribution	83
Figure 52. MLP 12-month forecasting using Palladium	83
Figure 53. MLP 12-month forecasting accumulative error using Palladium	83
Figure 54. MLP 12-month forecasting error distribution using Palladium	84
Figure 55. MLP 12-month forecasting accumulative error distribution using Palladium	84
Figure 56. MLP 12-month forecasting using Eurostoxx 50	84
Figure 57. MLP 12-month forecasting accumulative error using Eurostoxx 50.....	84
Figure 58. MLP 12-month forecasting error distribution using Eurostoxx 50.....	84
Figure 59. MLP 12-month forecasting accumulative error distribution using Eurostoxx 50 ..	84
Figure 60. CNN 12-month forecasting.....	85
Figure 61. CNN 12-month forecasting accumulative error	85
Figure 62. CNN 12-month forecasting error distribution.....	85
Figure 63. CNN 12-month forecasting accumulative error distribution	85
Figure 64. CNN 12-month forecasting using Palladium	85
Figure 65. CNN 12-month forecasting accumulative error using Palladium.....	85
Figure 66. CNN 12-month forecasting error distribution using Palladium	86
Figure 67. CNN 12-month forecasting accumulative error distribution using Palladium	86
Figure 68. CNN 12-month forecasting using Eurostoxx 50	86
Figure 69. CNN 12-month forecasting accumulative error using Eurostoxx 50.....	86
Figure 70. CNN 12-month forecasting error distribution using Eurostoxx 50	86
Figure 71. CNN 12-month forecasting accumulative error distribution using Eurostoxx 50 ..	86
Figure 72. LSTM 12-month forecasting.....	87
Figure 73. LSTM 12-month forecasting accumulative error	87
Figure 74. LSTM 12-month forecasting error distribution	87
Figure 75. LSTM 12-month forecasting accumulative error distribution.....	87
Figure 76. LSTM 12-month forecasting using Palladium.....	87
Figure 77. LSTM 12-month forecasting accumulative error using Palladium	87
Figure 78. LSTM 12-month forecasting error distribution using Palladium.....	88
Figure 79. LSTM 12-month forecasting accumulative error distribution using Palladium	88

Figure 80. LSTM 12-month forecasting using Eurostoxx 50.....	88
Figure 81. LSTM 12-month forecasting accumulative error using Eurostoxx 50	88
Figure 82. LSTM 12-month forecasting error distribution using Eurostoxx 50.....	88
Figure 83. LSTM 12-month forecasting accumulative error distribution using Eurostoxx 50	88
Figure 84. RFR 1-month forecasting.....	89
Figure 85. RFR 1-month forecasting accumulative error	89
Figure 86. RFR 1-month forecasting error distribution.....	89
Figure 87. RFR 1-month forecasting accumulative error distribution	89
Figure 88. RFR 1-month forecasting using Palladium	89
Figure 89. RFR 1-month forecasting accumulative error using Palladium.....	89
Figure 90. RFR 1-month forecasting error distribution using Palladium	90
Figure 91. RFR 1-month forecasting accumulative error distribution using Palladium	90
Figure 92. RFR 1-month forecasting using Eurostoxx 50	90
Figure 93. RFR 1-month forecasting accumulative error using Eurostoxx 50.....	90
Figure 94. RFR 1-month forecasting error distribution using Eurostoxx 50.....	90
Figure 95. RFR 1-month forecasting accumulative error distribution using Eurostoxx 50	90
Figure 96. RFR 12-month forecasting.....	91
Figure 97. RFR 12-month forecasting accumulative error	91
Figure 98. RFR 12-month forecasting error distribution.....	91
Figure 99. RFR 12-month forecasting accumulative error distribution	91
Figure 100. RFR 12-month forecasting using Palladium	91
Figure 101. RFR 12-month forecasting accumulative error using Palladium.....	91
Figure 102. RFR 12-month forecasting error distribution using Palladium	92
Figure 103. RFR 12-month forecasting accumulative error distribution using Palladium	92
Figure 104. RFR 12-month forecasting using Eurostoxx 50	92
Figure 105. RFR 12-month forecasting accumulative error using Eurostoxx 50.....	92
Figure 106. RFR 12-month forecasting error distribution using Eurostoxx 50	92
Figure 107. RFR 12-month forecasting accumulative error distribution using Eurostoxx 50	92

Περίληψη

Η Μηχανική Μάθηση έχει αποδείξει την ικανότητα της να επιλύει καθημερινά προβλήματα και να παρέχει οφέλη στους χρήστες της. Η πρόβλεψη των μεταβολών της τιμής του πετρελαίου είναι πολύ σημαντική για πολλούς ενδιαφερόμενους, και η Μηχανική Μάθηση είναι υποψήφια για να βοηθήσει προς αυτή την κατεύθυνση. Εφαρμόζουμε διάφορα μοντέλα Μηχανικής Μάθησης και παρατηρούμε ότι παρόλο που υπάρχουν ενδείξεις για την πρακτική εφαρμογή της, υπάρχει μεγάλη απόσταση να διανύσουμε μέχρι η Μηχανική Μάθηση να μπορέσει να ανταποκριθεί σε ένα απαιτητικό, πολυμεταβλητό, πρόβλημα χρονοσειρών όπως είναι αυτό της πρόβλεψης των μεταβολών της τιμής του πετρελαίου. Στην περίπτωση μας μόνο το βασικό LSTM μοντέλο κατάφερε να δείξει απόδοση κοντά στην πρόβλεψη no-change, κάτι που το καθιστά επίσης, ακατάλληλο για την πρόβλεψη της μεταβολής των τιμών του πετρελαίου.

Λέξεις-κλειδιά: Machine Learning, Μηχανική μάθηση, Oil price movements, time-series, μεταβολή τιμής πετρελαίου, χρονοσειρές, MLP, LSTM, CNN, RandomForestRegressor

Abstract

Machine Learning algorithms have proven their ability to solve everyday problems and provide benefits to its users. Forecasting oil price movements is essential for many stakeholders, and Machine Learning is a candidate to help in that direction. We apply various Machine Learning models, and we find that although there are indications of its applicability, there is still a long way to go until Machine learning can address a challenging multivariate time-series problem like the one in forecasting oil price movements. In our case, only the plain LSTM model managed to show performance close to the no-change forecast, which renders it inappropriate for use in forecasting oil price movements.

Keywords: Machine Learning, Oil price movements, time-series, MLP, LSTM, CNN, RandomForestRegressor

Introduction

Oil is an essential commodity for the world economy. It drives many aspects of human life. Movements in its price can have a significant effect on the prices of other commodities, financial assets and macroeconomic factors like Gross Domestic Product and unemployment. It is crucial for all stakeholders involved in economic life to be able to forecast oil price movements so they can secure their investments and production cycle.

Traditionally, in the effort to forecast oil price movements, researches make use of econometric models. Various models have been under consideration for decades. Various types of data are used to fuel those models. Although there have been tons of studies on the matter, no conclusive result has been reached so far. The oil market is under pressure from various sources and is affected by global economic activity as well. More and more, oil's price is being exposed to other assets, and its financialization leads to higher Volatility.

This study employs some notable Machine Learning Models to forecast oil price movements in short- and long-term horizons. The models under consideration have proven their abilities in adapting to real-life problems and providing essential benefits from their use. Forecasting oil's price is no fit for the faint-hearted. It is a demanding task, and since oil prices get affected by other economic activity, there is no standard way to approach this problem. Machine Learning methods are promising to that sector as well. Researchers strongly believe that Machine Learning models will be able to uncover the complex correlations between the factors that affect oil price movement. This study is an effort to shed some light over this belief and check the Machine Learning Models potential.

Data is an essential factor to success for econometric models and Machine Learning models alike. This study employs a lot of time-series data that is shown in extant literature to provide forecasting accuracy. Based on Degiannakis and Filis (2018), monthly oil market's fundamentals, as well as ultra-high frequency financial data, are considered and tested to uncover their secrets.

We test the results of the Machine Learning models put in action against standard econometric models' results. The comparison of these models will uncover any hidden potential of the Machine Learning models.

The rest of this study is structured as follows.

- Section 1: The case of oil prices – reviews the literature about the oil market and its workings. It tries to uncover what drives the oil price movements and the literature about forecasting them.
- Section 2: Machine Learning forecasting – reviews the literature about Machine Learning, its methods and models, as well as it takes a closer look at some notable models that we use in this study.
- Section 3: The baseline method – reviews Degiannakis and Filis (2018) paper which we will use as a guide to our study. We compare our test results against the results found in that paper so that we conclude Machine Learning forecasting ability of oil price movements.
- Section 4: The ML methods in action – discusses the Machine Learning models used in this study, along with the data used. It makes clear what data are useful and why, and it defines the models under consideration.
- Section 5: Results – records the results found from Machine Learning model testing and compares them against the standard models and MIDAS models of the Degiannakis and Filis (2018) paper.
- Section 6: Conclusion – concludes the study and its findings.

The case with oil prices

In this section, we review the literature about crude oil and its importance in the global economy. We try to provide a perspective in what crude oil is and the inner workings of its trading tools. Crude oil is a raw natural nonrenewable resource. It is widely used in many forms around the globe. It can be refined to produce many other products to cover various needs that spread from energy to everyday use items. Some notable products based on crude oil are gasoline, diesel, asphalt and plastic.

Crude oil benchmarks

Since crude oil is such a useful commodity, it is traded in exchange markets. There are three main benchmarks in oil pricing, and there are many differences between them that affect the price of each. Those are Brent Crude (or Blend), West Texas Intermediate (WTI) crude and Dubai Crude. Their price depends on their composition like the amount of sulfur they have, the location they are extracted from and the ease of transportation to clients and thus the market they are sold (EIA, 2014).

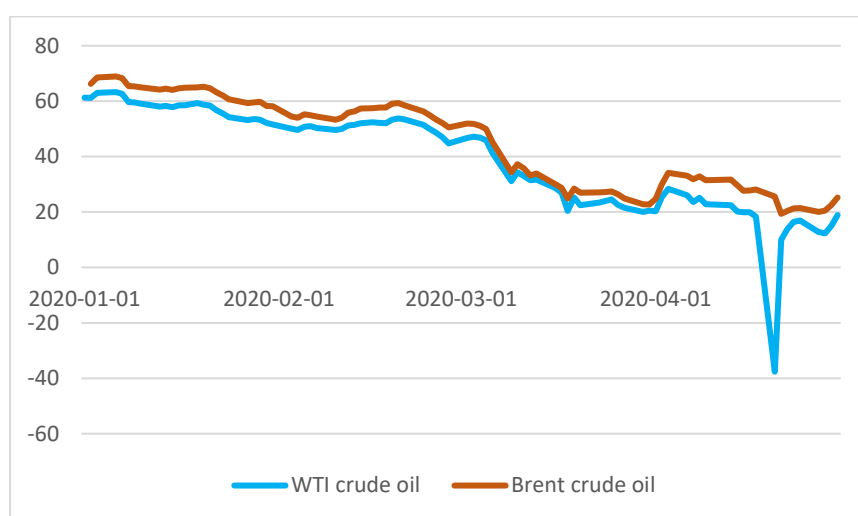
Benchmarks are used in the formation of other derivative financial tools like Futures and Options. Those tools are tied to a specific benchmark and used mainly for hedging for protection of the spot market volatility although they can be used for speculative trading as well. Each benchmark is traded in different exchange markets.

Brent Crude is extracted in the North Sea, and it is the most used type of crude. Because it is produced in the sea, it is easier and more cost-effective to transfer around the world (ICE, 2020). It is not traded directly but in the form of futures contracts in the Intercontinental Exchange (ICE) and the New York Mercantile Exchange (NYMEX). It is the most used benchmark since it is used in about 70% of the oil contracts globally (ICE, 2020). Brent is also used to price other crudes produced and traded not only in Europe but in areas such as Africa, Australia and some countries of Asia (EIA, 2014; ICE, 2020).

West Texas Intermediate (WTI) refers to US extracted oil, and it is traded in the New York Mercantile Exchange (NYMEX) in the form of futures contracts. It is considered to be of better quality than Brent crude, and it should be traded at a premium, but this is not the case. They both target different markets, and thus their price reflects its market fundamentals. It is the primary benchmark for the US produced and consumed oil, and this is why its price is affected more from fluctuations in the US

economy. WTI is used for imported crude produced in Canada, Mexico and South America (EIA, 2014). WTI saw a historical collapse to its price in 2020 due to COVID-19 pandemic and the global economic shutdown. Demand disappeared, and this led WTI front-month futures price to close at -\$37.63/b on April 20, 2020, as shown in Figure 1, for the first time in a crude oil benchmark (EIA, 2020a). OPEC as well as some non-OPEC producers, following the “Declaration of Cooperation” to a stable market, agreed in production reduction from May 2020 to April 2022 (OPEC, 2020).

Figure 1. Crude oil front-month futures prices collapse for WTI on April 2020 due to COVID-19 pandemic



Notes: COVID-19 caused oil demand to fall for the first half of 2020, and oil prices followed that drop. On April 2020, WTI faced a major impact on its price for the first time in its history. WTI's price fell into negative prices. Data retrieved from investing.com (2020a,b)

Dubai/Oman Crude is the name used for crude extracted from Dubai, Oman or Abu Dhabi. It is considered to be of lower quality compared to Brent and WTI because of its higher sulfur content which makes it harder for refiners to use. Its future contracts are traded on the Dubai Mercantile Exchange under the name of “Oman Crude Oil Futures Contract”, and it is mainly used to target the Asian market. Although Dubai production has declined in the past two decades, Oman crude is used to support the use of Dubai Crude as a benchmark (EIA, 2014).

Fundamentals

There has been much discussion in the literature of what can influence the price of crude oil. As a commodity, crude oil price obeys the law of demand and supply. The markets' demand, as well as producers' supply ability, affects its price.

Demand itself is affected in many ways. In cases where a country's economy is not doing that well, production might fade, and this will drive the need for energy in the form of oil down. On the other hand, if the economy is flourishing, there will be more need for energy because production will increase and people will be more active, and they will need more energy for transportation and other uses. Also, oil can be used to produce other material products like asphalt and plastics, which in turn will be traded. In the period 2007-2008 the prices run-up because of strong demand which could not be met by the stagnated production (Hamilton, 2009a). Other reasons demand can be affected are applied policies such as the limit of carbon dioxide emissions or the taxes applied on oil products. Transportability is another reason, and it is tightly connected to regional proximity. There are significant costs involved in transporting goods altogether, and this holds to crude oil as well. The cost of transportation because of the distance between crude oil producers and refiners, can affect the final price of the products, and this will affect consumer demand. Besides the price, Hamilton (2009a) considers income to be the most critical determinant of the quantity demanded.

Supply is not stable either. There are times that producers face technical or other issues that prohibit them from reaching their potential in production. It is known that many producer countries do not reach their agreed potential while others overdo it. OPEC is an organization that forces those limits as a policy to control the prices and help stabilize the world economy. OPEC's ability to control the prices is challenged by Büyüksahin et al. (2016) due to its low spare capacity, but Razeq and Michieka (2019) disagree and they believe OPEC has still the power to affect oil prices. Transport can have a say in the case of supply ability as well. Proximity is one thing. Not all refiners can get supplies from every producer in the world because of the costs and time involved. This situation puts a practical limit to the available supply even in cases the producer can cover the needs. Even in established transport lines, for example, if there are significant changes in the hiring costs of oil tankers or new ships are introduced in the transport chain, this will affect the supply ability. Warfare and in general political unrest is another thing that can influence the transportability and the supply as a result.

In troublesome times there might be embargos established along the transport routes or even actions of war can cut those routes or production ability altogether.

In contrast to the previous examples, supply can be in excess as well. When demand drops or the producers decide to increase their production level, we face a drop in the price of oil. According to Razek and Michieka (2019), both in 1986 and 2014 oil price dropped due to excess supply. Büyükşahin et al. (2016) claim that the main difference between those two episodes is the investment in new technologies that drive the abundance of cheap oil since emerging economies sustained demand growth in 2014. Technological advancements have increased the speed at which supply can be adjusted to changes in demand (Razek and Michieka, 2019).

Political unrest has caused a debate between scholars for the effect it has on oil prices. When things get hot in oil producer regions, oil prices tend to rise. On the one hand, it is believed the main reason political unrest can influence the crude oil price is that actively it makes production fall and thus the supply reflects the consequences of war or otherwise¹ (Hamilton, 2009a,b). In contrast to this belief, there are claims that political unrest triggers precautionary mechanisms for safeguarding the availability of oil (Kilian, 2009). Concerns cause demand to increase in order to fill up the storages for future use.

Inventories can influence both demand and supply and drive the oil prices accordingly. There is a bi-directional relationship between inventories and prices (Razek and Michieka, 2019). According to expectations for future prices, refiners and storage terminals might use the stored oil products to mitigate the impact of price movements due to various events like seasonal fluctuations or weather changes. They can also use their inventories for profit in expectation to sell them later at higher prices.

Financialization

Insecurity for future oil prices led investors to create Futures Contracts to help producers and refiners secure their profit with stable prices. Each counterparty in such

¹ In his paper, Hamilton (2009a) reports of 4 incidents that caused oil price shocks. Among them, there is the Yom Kippur War which although didn't prevent oil shipments, caused OAPC (Organization of Arab Petroleum Exporting Countries) to take measures against Israel. They announced that they will cut down production by 5 percent until Israeli Forces evacuate all former Arab territories occupied.

an agreement can be sure that their activity will not face disturbances because of the volatile prices in the spot market that might be affected by unknown events. Beside producer and refiners, a lot of other consumer categories can benefit from such a security tool because their activity² is strongly related to oil (EIA, 2020b). Futures contracts are tied on a specific benchmark, and their price depends on it. In reverse causation, it is believed that the number of Futures Contracts and the price they are sold can influence the crude oil price evolution. These price movements are a result of the financialization of commodity products.

The derivatives market has vast profit potential, so investors like banks and hedge funds are looking into oil price movements, although they are not interested in physical oil trading. They use derivatives to diversify their portfolios and get benefits from speculative trading. This activity provides liquidity to futures and derivatives markets, but there are concerns that commodity trading and investment at times can lead to more vigorous price movements³ (EIA, 2020b).

Benchmarks are priced in US dollars, and because of this, profit for traders is connected to the exchange rate of currencies (EIA, 2020b). Razek and Michieka (2019) agree too that the financialization of oil has an impact on its price movements. Also, Fratzscher et al. (2014) find a bidirectional causality between the US dollar and oil prices since the early 2000s, although it did not exist the time before. They also find that both oil and US dollar are affected by changes in equity market returns and risk, and this is considered to be a result of oil's use as a financial asset. Studies in commodity price movements, including oil, found that a weaker dollar leads to higher commodity prices (Akram, 2009). Büyükşahin and Robe (2014) come to the same conclusion of a strong correlation between the rates of return on investible commodity and equity indices. Xu et al. (2019) provide evidence for a dynamic correlation between oil and currency rates. They find their dependence starting from 2004 and to dynamically change over time since then, which indicates the importance of crude oil market financialization and the effect it bears to other markets.

² Such a consumer might be an airline company that uses fuel for their airplanes or a transport company and they want to keep their costs fixed so they can plan accordingly to stay competitive in their market (EIA, 2020b).

³ Liquidity can be used up when market momentum is running in one specific direction

Akram (2009) suggests that commodity prices may have risen due to spillover effects between different commodity prices as a result of the financialization of commodities. Balcilar et al. (2019) find that there is a strong bilateral relationship between oil and gold regarding price spillovers. This finding is supported by Baffes (2007), who maintains that oil spills on many other commodities and specifically to precious metals because people turn to them as a ‘safe haven’. Also, he notes that food and fertilizers are affected, mainly because of oil’s effect on transport and production costs.

Due to the financialization of Crude oil and nonferrous materials, oil’s price has a significant effect on international copper futures prices (Chen et al., 2018). Reboredo and Ugolini (2016) support this claim in their study with six industrial metals and four precious ones. In more detail, they note that oil and metals were weakly coupled before the global financial crisis of 2008, but their coupling increased after the outbreak of the crisis. Reboredo and Ugolini (2016) ascribe the price spillover effect between oil and metal markets to oil’s price large downward and upward movements. They maintain⁴ that there are differences between the upward and downward movement with upward movement to cause more substantial spillovers.

As a result, to its heavy financialization, crude oil is missing its hedging role for the stock market as noted by Sharma and Rodriguez (2019). They find that during the post-crisis⁵ period, it is more expensive for the investors to use crude oil as a hedging tool for their portfolios. There is increased co-movement between stock and oil prices. Sharma and Rodriguez (2019) show that crude oil behaviour resembles more that of a financial asset rather than a commodity.

Volatility

Crude oil prices are incredibly volatile, and one reason for this is its financialization as already discussed. Besides the obvious use of those in material oil trade, oil is traded in exchanges for speculative purposes in the form of oil derivatives like futures and

⁴ Asymmetric spillovers are important for investors so they can strategically manage the risk involved in each oil movement (upwards or downwards) regarding their metal portfolios (Reboredo and Ugolini, 2016)

⁵ They refer to the Global Financial Crisis between 2007 and 2009

options. Those trades affect the spot price of crude oil. Albeit, Büyükşahin and Robe (2014) argue that speculative activity results because of oil's high Volatility and in contrast to the common belief.

Futures price volatility is also attributed to hedge funds, but according to Haigh et al. (2007), it is a false assumption. Hedge funds are accused of changing their positions continuously, and thus they affect Volatility. Haigh et al. (2007) find that hedge fund participants do not change their positions that much, and they are considered to be relatively inactive compared to other participants in the futures market. It seems hedge funds change their positions when they respond to price signals, providing in this way liquidity to other participants. On the other hand, speculative⁶ trading affects Volatility, according to Kaufmann (2011).

At this point, it is clear that oil's price is the result of many factors like fundamentals, speculation on its futures, oil's financialization and the effects it gets from other markets. Kaufmann (2011) maintains that the correlations between all these factors are not clear, and small changes in one factor can lead to sizeable changes in oil's price. Most important is that at times it is impossible to measure some factors like speculative expectations.

Forecasting

Many efforts are made to forecast the oil's price in various future horizons. To achieve this, researchers use statistical models applied in many time-series. It is crucial to be able to predict the oil's future prices because it has a considerable effect on the global economy as well as to a specific country. A lot of products and services are tightly connected to oil and fluctuations to its price might have an undesired effect on their function, and the price of their final product or service. Refiners are interested in protecting the crack⁷ spread to safeguard their products' prices (Murat and Tokat,

⁶ As speculators are considered all traders beside the commercial ones. That category includes hedge funds although a speculator is defined to be the trader who does not hedge, but who trades with the objective of achieving profits through the successful anticipation of price movements (https://www.cftc.gov/LearnAndProtect/EducationCenter/CFTCGlossary/glossary_s.html)

⁷ Crack spread refers to the price relationship between crude oil and the products made out of it. Crack spread is traded in the form of futures in oil markets and is found to have an effect in oil price movements (Murat and Tokat, 2009).

2009). Those price movements can affect trade, incomes, inflation, employment, house prices and in general, have a destabilizing result for the entire economy (Razek and Michieka, 2019).

Traditionally, forecasting oil prices is done by applying econometric models with varied results. As it was noted by Degiannakis and Filis (2018), not a single model can be used for the job. Although some models showed that perform better than others, no model stood out for every forecasting horizon.

A key question in forecasting the oil price is to know which data are relative for the job. Researches are trying to unfold the correlations between crude oil prices and other market trading data. Fundamentals of the oil market are considered to have still the prime impact in oil price movements, but they are not the only ones. Apart the fundamentals, the financialization of crude oil is believed to make it vulnerable in changes in other markets as well, like foreign exchange movements and specifically to the exchange value of US dollar relative to other currencies (EIA, 2020b).

Studies show that forecasting ability is improved when high-frequency data are taken into consideration (Degiannakis and Filis, 2018). They use both ultra-high and low-frequency data to forecast oil price movements on various horizons. Their monthly data consist of the oil market fundamentals such as the Global Economic Activity Index, the Baltic Dry Index, the Global Oil Production, the Global oil Stocks, the Capacity Utilization Rate of the oil and gas industry. Their ultra-high frequency data consist of tick-by-tick prices of the front-month futures contracts for three paramount exchange rates (GBP/USD, CAD/USD, EUR/USD), four stock market indices (FTSE100, S&P500, Hang Seng, Euro Stoxx 50), six commodities (Brent crude oil, Gold, Copper, Natural Gas, Palladium, Silver), the US 10 yr T-bills. They also use daily data from US Economic Policy Uncertainty Index. Degiannakis and Filis (2018) use the tick-by-tick data to construct the realized volatilities of all assets mentioned above as well as their daily log-returns.

Murat and Tokat (2009) test crack spread futures if they provide any valuable information in predicting the oil price movement with positive results. The data set includes weekly time-series of NYMEX future contract prices and weekly time-series of spot prices of WTI Crude. To convert futures price into the Crack Spread price, they use the daily settlement prices for all NYMEX-traded futures contracts related to energy products (light crude oil, heating oil and unleaded gasoline).

Akram (2009) tests the impact interest rates and exchange rate have upon commodity prices. He uses the nominal and real values of the commodity prices, as well as the real values for interest rates and the exchange rate. The commodities are PCO (the price per barrel of crude oil in dollars), PCF (the price of food), PCM (the price of metals), and PCI (the price of industrial raw materials).

Büyükaşahin and Robe (2014) use a non-public dataset of trader positions in 17 US commodity futures markets. They try to provide evidence of those markets financialization.

Fratzscher et al. (2014) test the correlation between the US dollar and oil prices. They also test how they are affected by changes in equity market returns and risk. They use daily data for US dollar exchange rate, WTI crude oil price in US dollars, Dow Jones Industrial Average Index, three-month Certificate of Deposit interest rate, Chicago Board Options Exchange Market Volatility Index (VIX). They also use weekly open interest in the NYMEX oil futures market linearly interpolated to a daily frequency.

Xu et al. (2019) test the correlation between oil prices and exchange rates. They use both daily and monthly oil futures prices, nominal bilateral exchange rates against the US dollar from 14 economies and the US Dollar Index (USDIX).

Although the above list is not exhaustive, it is a good indicator of the research that needs to be done in order to obtain useful information that can be tested and validated of their forecasting ability.

Nowadays, much effort is put in using Machine Learning models in place of econometric ones. The demanding, non-linear nature of oil price dependencies cannot be fully captured with known econometrics models. On the other hand, there are expectations for Machine Learning models to understand better the underlying correlations that affect oil price movements.

The effort put in discovering the oil price dependencies and its relation to other market indicators through econometrics studies is still relevant and valuable. Machine Learning models still need to work on provided data and learn from them. Although the different approach might provide new insights into the correlations and causalities, it is vital to take advantage of the existing knowledge. Machine Learning demands a great deal of processing power and time to train on data and learn. On the one hand, providing them with useful data will help the research process to complete sooner, while on the

other hand, it is an opportunity to revalidate the data usefulness since what is considered to be important in one method, might not be that relevant in another.

Machine Learning forecasting

Machine learning is widely used today in a plethora of applications. Finding the right method for the problem at hand is a challenging endeavour for every data scientist. Though, machine learning has proven its skill in helping people with jobs that can be automated, or filtering out data to create more up to the point reports. However, what is Machine Learning?

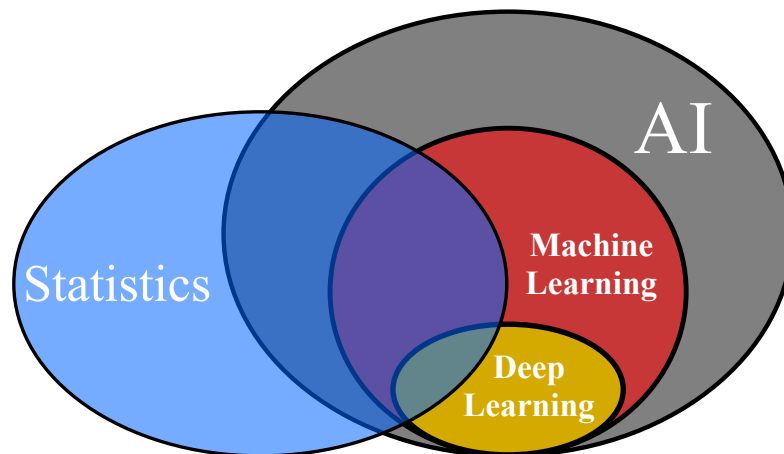
Machine Learning Definition

The term Machine Learning has been around for decades, but only recently, it made it to be part of the technical jargon. The primary reason is that modern computers can provide the so needed processing power to train a model for the job while some years back this would be a tiresome and timely process. Another reason is that a good deal of research has been made in a plethora of sectors. Some notable achievements in that research revived and reinforced the interest in those technologies (Nesreen et al., 2010).

According to Alpaydin (2004), machine learning is programming computers to optimize their performance based on collected data. Instead of explicitly programming computers how to act on each case, either because of lack of human expertise or because the acquired expertise is impossible to be put in code, learning processes can provide a solution to these problems. Another case is when the environment or the process changes with time. In that case, a machine needs to adapt its performance to the new environment and to do this, it needs some learning ability.

Machine learning is part of the broader Artificial Intelligence sector (see Figure 2). Machine Learning consists of algorithms that create a mathematical model for the sample data they are given. They might not be able to understand the process thoroughly, but they can make a good and useful approximation (Alpaydin, 2004). Those data are known as “Training data”. Machine learning algorithms can adapt using those data, and in a manner, understand and learn what the rules that the training data are based on are. Then, using this trained machine, we can make some predictions using new data. Machine learning uses statistics to build models and make inferences from a data sample (Alpaydin, 2004).

Figure 2. AI sector



Notes: AI sector and its subsectors are strongly related to classical statistics. Deep learning and machine learning are parts of the AI sector, and they are based too in statistics.

Deep Learning, is part of the Machine Learning family of algorithms, and it consists of more specialized algorithms based on the concept of artificial Neural Networks. Those algorithms are considered to be more advanced, and they can discover more complex features and non-linear relationships in the data.

Types of Machine Learning

There are different approaches in training a Machine Learning model depending on the nature of the data available to train the model and the result we need to achieve. Some major types and the first used to classify the algorithms are the following:

Supervised learning. In this type of learning approach, each set of input values has a label for the output. The purpose of the Machine Learning model is to find the mappings between input and output values (Alpaydin, 2004). This label can be either a category or a numeric value like in statistic methods. In case we are trying to make meaning out of the inputs, i.e. trying to understand if the input data indicate that a patient has some kind of disease, we talk about a classification problem. On the other hand, if we try to predict a value like the number of sales of a product, it is a regression problem.

Unsupervised learning. When a label is not available for the input data, we can still use Machine Learning methods. In unsupervised learning, there is no supervisor to provide the appropriate output values for each input value set, and the aim is to find

regularities in the inputs (Alpaydin, 2004). Such regularities, structures or patterns can be found in data more often than others.

Clustering is a process used to find such regularities. Most of the times, data can be grouped according to some feature that appears most often. In those cases that data items are grouped into clusters, they tend to have more commonalities in their features, not just one. By studying those groupings and the features they share, we can make some inferences of the reasons why, and their behaviour.

Reinforcement learning. Again, there is no label in this approach of learning because the output of the system is not an integral value or a class, but a sequence of actions (Alpaydin, 2004). The data used to train the model indicate some action to be taken from some software agent like an actor in a video game (e.g. chess). A reward is provided, when training the model, according to previous decisions and the result they had in the monitored process. That reward can use a point system where the Machine Learning model gets some point if it wins a game or if it moves a drone with precision. In this approach, there are no ready-made data, but they are created on the fly. The Machine Learning model learns by repetitive trial and error efforts. The machine repeats the same process until it achieves the desired outcome. Each trial on the process is called an epoch. Each epoch, the algorithm tries to maximize its profit according to the point system.

Methods and promises

It is clear by now that the problem we need to act upon is a supervised type of problem. We get data in the form of some time-series that will affect the price of oil which is also a time-series itself. This price is our label for the given data for each given time while the values of every other time-series are the input data. Our models have to understand the underlying structure between each time-series and the effect those relationships have on the current and future prices. We need to predict the actual price oil will be traded in the future for one or more time-steps ahead. Thus, it makes it a regression problem.

ML for time-series methods. We intend to predict a value, oil price, and this is a regression problem. We focus only on algorithms that can predict an output value for the given data. As already mentioned, Machine Learning algorithms are grouped in

categories based on the problems they target. Each category contains tens or hundreds of algorithms, and each algorithm has its pros and cons. For the needs of this paper, we focus on some of the most recognized and promising ones since it is not our purpose to test exhaustively every known method or algorithm.

Random Forest Regressor. Random forest regressor is based on decision trees to make a prediction. The idea behind Random Forest is to create decision trees from sample data. Then the regressor averages the decisions made by every tree, and this will be the prediction of the forest.

The main drive behind this idea is the notion of the “wisdom of the crowd”. In a random and diverse crowd, each individual will provide a different answer in a particular question when asked. The average of the answers, or the majority in cases of non-numeric answers, is closer to the “true” answer than most individual answers (Galton, 1907).

In the case of Random Forest, we need to split the sample data into sub-samples in a random way, and for each sub-sample, a tree will grow using a random set of features. In literature, there have been many suggested techniques with varied results concerning the generalization error. Some of those techniques include bagging (Breiman, 1996), random split selection (Dieterich, 1998) and many more. In his 2001 paper, Breiman introduced the idea of random input and random features that produce sound results for both classification and regression problems, although this method lacks a bit in regression (Breiman, 2001).

For this paper, the method that will be used is RandomForestRegressor from the scikit-learn library (Pedregosa et al., 2011), and it draws from Breiman’s (2001) paper about Random Forests. A known issue of the predictive models, in general, is that they tend to overfit the given training set. Overfitting is not a case with Random Forests due to the Law of Large Numbers (Breiman, 2001). In the same paper, Breiman notes that randomness is an essential key factor of their predictability, but there must be some care of the kind of randomness used when creating a predictive method.

During training, scikit-learn calculates feature importance as the decrease in node impurity weighted by the probability of reaching that node. The node importance for each decision tree is calculated using the Gini Importance. For a binary tree we get the function:

$$ni_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)}$$

Where:

ni_j is the importance of node j ,

w_j is the weighted number of samples reaching node j ,

C_j is the impurity value of node j ,

$left(j)$ is the child node from a left split on node j ,

$right(j)$ is the child node from a right split on node j .

Each features' importance is given using the following function:

$$fi_i = \frac{\sum_{j=1}^{FS} ni_j}{\sum_{k=1}^{NN} ni_k}$$

Where:

fi_i is the importance of feature i ,

ni_j is the importance of node j ,

ni_k is the importance of node k ,

NN is the number of all nodes,

FS is the number of splits on the feature i .

To normalize the feature importance between the values of zero and one, we use the following function:

$$normfi_i = \frac{fi_i}{\sum_{j=1}^{NF} fi_j}$$

Where:

$normfi_i$ is the normalized value of feature i importance,

fi_i is the feature i importance,

fi_j is the feature j importance,

NF is the total number of features.

Finally, the feature importance on the Random Forest level is calculated using the following function:

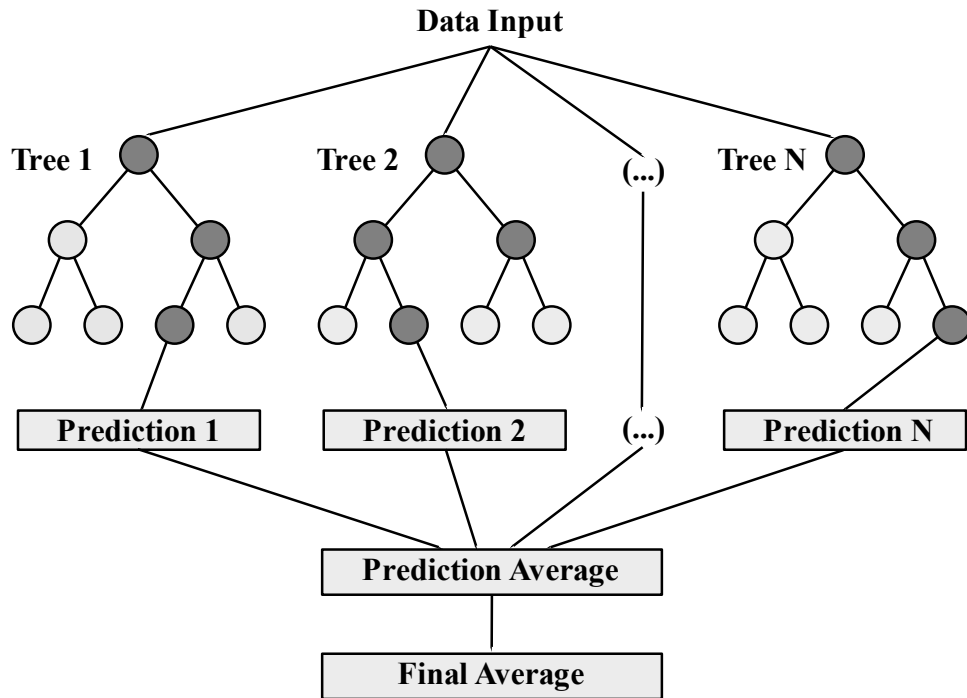
$$RFfi_i = \frac{\sum_{j=1}^T normfi_{ij}}{T}$$

Where:

$RFfi_i$ is the feature i importance as the average of all trees in the model,
 $normfi_{ij}$ is the normalized value of feature i importance in tree j ,
 T is the number of all trees in the model,

To make a prediction, the Random Forest model will run down each tree according to the feature importance it calculated during the training process. The prediction process is illustrated in Figure 3. The Random Forest model evaluates each tree branch (node) according to input data and collects the predictions made from all trees in the model. The predictions are averaged, and this value is going to be the model's output. In cases of Regression models, it will be a real number while in cases of Classification, it is going to be a class type.

Figure 3. Random Forest Prediction



Notes: Input data analyzed by every tree in the model; each tree makes a prediction; all predictions are averaged to come to a final Random Forest conclusion.

Multilayer Perceptron (MLP). A Multilayer Perceptron is a class of feed-forward artificial neural network. It consists of at least three layers of Perceptrons, being the input layer, at least one hidden layer and an output layer. The input layer reads the data, the output layer makes the prediction, and the hidden layers are the powerhouse of the device. The more hidden layers the model uses, the more complex problems it can learn to predict. Hidden layers are called so because they have no interaction with the environment but only through the inputs and outputs.

Feedforward neural network is the purest form of an artificial neural network where the information flows in only one direction, from input into the hidden layers and finally to output. Other more modern types of artificial neural networks like LSTM that will be discussed later-on contain feedback connections.

Each layer of the MLP, except the input layer, consists of many Perceptrons. The perceptron is an algorithm proposed by Rosenblatt (1957) as a learning device for binary classification. The concept for the creation of perceptron is drawn from neurology, and it is considered to be analogous to a neuron and thus comes the name of artificial neural networks, with Perceptrons acting as the neurons. It bases its functionality in statistics (Alpaydın, 2004). In a full MLP setup, each perceptron takes as inputs all the outputs of the previous layer and provides an output for the next layer. A bias input of a constant value of one (1) is also used as an input for each perceptron. For each input, a weight is calculated, including the bias, during the training of the network. The transfer function of each perceptron is:

$$y = w * x + b$$

Where:

w is the vector of weights,

x is the vector of inputs,

b the bias input weight.

Each perceptron emulates a linear function. It creates a single output value based on the combination of the input values with their respective weights, adding the also weighted bias to the result.

As we can tell from this, a single Perceptron alone cannot provide much of a predicting power since it is limited to linear classification problems. When the Back-propagation algorithm was introduced (Rumelhart et al., 1986), it was made possible to

create and train a Multilayer Perceptron network promptly. The network readjusts the weights for each perceptron input using the back-propagation algorithm, so the difference between the neural network's output and the desired output is minimized. An optimized method called Levenberg Marquardt is more efficient than basic back-propagation, and this is usually used instead (Nesreen et al., 2010).

In order for the network to be able to respond to non-linear classification problems, the output of each perceptron is passed through a non-linear activation function (e.g. Sigmoid, ReLU). The complexity of the model is controlled by selecting the number of nodes for the hidden layer (Nesreen et al., 2010). The number of output nodes is depended on the problem being solved. In the case of classification problems, there will be as many output nodes as the classes of the data. In a regression problem, it depends on how many values we need to forecast.

The output value for a three-layer MLP is calculated as follows (Nesreen et al., 2010):

$$\hat{y} = f \left(v_0 + \sum_{i=1}^{NH} v_i g(w_i^T x') \right),$$

Where:

- f is the activation function for the output node,
- NH is the number of nodes in the hidden layer,
- v_0, v_1, \dots, v_{NH} are the weights for the output node,
- g is the activation function used in the hidden layer,
- w_i is the weight vector for the ' i ' hidden node,
- and x' is the input vector x augmented with one,

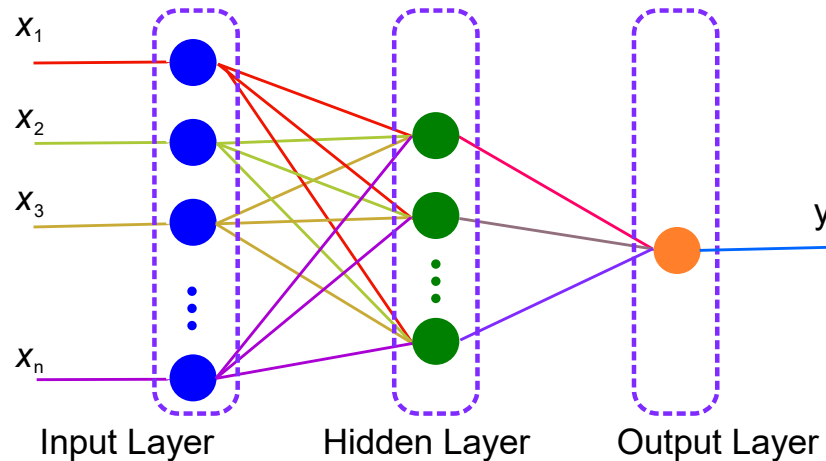
$$x' = (1, x^T)^T$$

The reason a non-linear activation function is used is that if all Perceptrons in an MLP were using a linear activation function, the existence of more layers would provide no added value to the network. According to linear algebra, any model would be reduced to a two-layer input-output model. In our case ReLU, a non-linear activation function is used, which is defined as the positive part of its argument:

$$f(x) = x^+ = \max(0, x)$$

A visual representation of an MLP with three layers (input layer, one hidden layer, output layer) is shown in Figure 4. This MLP has n inputs and one output. Inputs can either be single feature values, time-lagged values of a single feature or a combination of both.

Figure 4. MLP with one hidden layer and one output node



Notes: Each circle represents a node. The input layer has n inputs, and each input is connected to each node in the Hidden layer. There is a single output node which means only one value is predicted from the network for each set of inputs. The output node is fully connected to the Hidden layer as well.

The problem with training a neural network in detail is the following. An MLP is a feed-forward network and data are fed in its inputs to make a prediction. Those data are propagated forward through the nodes until the result reaches the outputs. When data enter each node, they are multiplied with the respective input's weight. In order for us to find the appropriate weights for each node in the network, in feed-forward style, we should try many combinations for each node while feeding the data and measuring the output. If the output was different from the expected, we should change the weights and try again. This process made training a network impossible because there are myriads of combinations for all the weights in the network. The more nodes there are, the more possible combinations to try.

In back-propagation, we work out this problem in a reverse manner. The first step is to assign random values for the weights in the network before the algorithm is applied. The process begins with the output layer. An error is calculated between the expected output and the real output for each node. This error is an indication of how this output needs to be changed. There are three ways of changing a neuron's output. We can either change the input weights, change the input values coming from the

previous nodes or change the bias. Those desired changes are recorded per node and combined with the respective desired changes coming from other nodes in the same layer. By adding together all the desired changes, we can have a set of weight changes to apply for this layer and a desired change of the inputs to this layer. Those inputs are the outputs of the previous layer, and thus the desired change is the error we need to backpropagate to the previous layer and start over this process. Working backwards layer by layer, we can calculate new weights for all network's nodes that given a specific input they can provide the desired output. That process needs to be repeated for each input set in the training data. For each set, the calculated weights are only suggestions for the changes that need to be made. Suggestions for each set of inputs are recorded, and when done, an average of the desired changes is calculated. Those changes are the ones that need to be applied to the current weights of the neural network. For computational reasons, the training data are split into batches, and this process is applied per batch.

An MLP can be applied to a supervised learning problem. It will be provided with a train set of inputs and outputs, and it will model the correlation between them.

Convolutional Neural Network (CNN). A Convolutional Neural Network is a class of deep neural networks. It is an extension of an MLP, and it consists of more layers which are not fully connected. It was invented as a means for better image recognition and classification. Lecun (1989) claimed that building into the network some a priori knowledge about the task at hand would improve its generalization performance. Image and speech recognition was proposed as good candidates to apply this idea because they are considered to be general tasks in nature. CNN's success made a massive difference in recognition of Artificial Neural Networks potential. It is the major component in modern computer vision, and it is used in various applications that demand visual feedback like autonomous car driving and medical image analysis. It can recognize multiple features in complex images and separate the significant parts of it. The use of this feature recognition and classification makes it an ideal candidate for more application types like recommender systems and even financial time-series forecasting.

CNNs use an MLP at its final stage of processing. MLP's purpose is to learn upon the features extracted from the rest of the neural network and finally make the classification of the input data once it is used in an application. The MLP has as many

output nodes as the number of classes it needs to identify. As it is made evident, the MLP does not act (learn or classify) on the original input data, but it acts on the local features extracted from them (Lecun, 1989). This feature separation is vital because it helps MLP detect complex structures irrelevant to their position in the original image or other types of datasets.

For example, an application that would use a simple MLP or some other algorithm to detect dogs and cats in user-provided pictures, it should be trained thoroughly with images containing those animals in all possible positions and all possible poses in order to minimize the error in classification operation. If during the training process, the MLP would not be fed with images that contain dogs in the upper left corner, for example, it is almost assured that it would fail to detect the existence of a dog when it was asked to do so during classification. Also, another problem would be the complexity of an image. Using a picture of a noisy environment would make it harder to train the model or apply the classification.

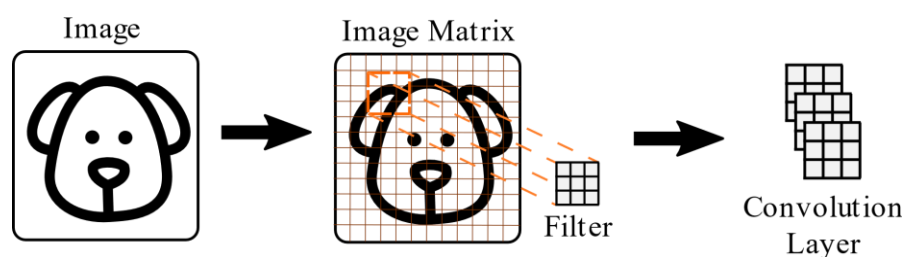
On the other hand, when CNNs are used to preprocess the input data, only features are extracted from the image and passed to the MLP. Those features might be things like vertical lines, horizontal lines, the combination of those and many more. A net of features might be associated with the object to detect like a dog or a cat. The existence of those features is irrelevant to their position in the picture. Only the existence of them is essential and how they relate to the rest of the features. As Lecun (1989), notes in his paper, we can afford to lose some position information since it is not relevant to classification. So, in case a dog appears on the upper left corner in the training data is irrelevant as long as its features are enough and appropriately clear for the detection to be made.

CNN's function is described in Lecun's (1989) paper as Net-5. The process to extract features from images (and any data) is the reason the convolution neural network gets its name. At the first stage of input data processing, there is a construction called "Convolution Layer". Those layers are built using Perceptron groups called "Filters" or "Kernels", but unlike the MLP networks, those filters in the layer are not fully connected. The filters break the input matrix that contains the image's data in smaller overlapping parts, and they check on each part separately. This process is shown in Figure 5. The size of the overlapping parts and the step from one image part to another are some of CNN's parameters. Those parameters are chosen during the construction of the CNN.

The filter runs through the entire image part by part to extract its features. Each filter extracts a single feature out of the image. In order to extract more features, many filters are used per convolution layer. The number of filters used is another parameter of a CNN model that needs to be decided during its design.

A filter tries to find a dominant feature in each image sub-part using a perceptron group connected to each part's inputs. All perceptron groups in a single filter, share their weights to simplify training (fewer weights to calculate) and make sure the filter is consistent in its entirety, meaning it detects for the same feature in each part. For each image part, the filter extrapolates a single value which describes the dominant feature that was detected. All such values extracted from the image are gathered in a matrix called "Feature map" or "Activation map". Those values are placed into the Feature map in a position relative to the one they had in the original image. This convolution process is how the CNNs get their name.

Figure 5. CNN image transformation using filters



Notes: The image is transformed into a numeric matrix, and then filters are applied through the convolution process. The image is divided into subparts, and for each part, a filter runs through it to extract a dominant feature. The whole of the filters forms the convolution layer.

In the case of an image with multiple channels (e.g. a colour image using RGB channels), a filter is applied for each channel, and the activation map is created by summing up the output of each filter plus one for the bias.

The next layer is more straightforward in its operation and comes in various names. This layer can be called "max pooling", "downsampling" or "subsampling". Its purpose is to create a new smaller matrix by downsampling the activation map it is fed. It uses a method to split the matrix into smaller overlapping parts, like the filters in the convolution layer. From each matrix part, only one value is selected, and the rest are discarded. The appropriate value can be chosen with either of two methods. One option is to use the maximum value found in the matrix (Max Pooling), and the other option is to use the average value (Average Pooling). This value is used in place of the whole

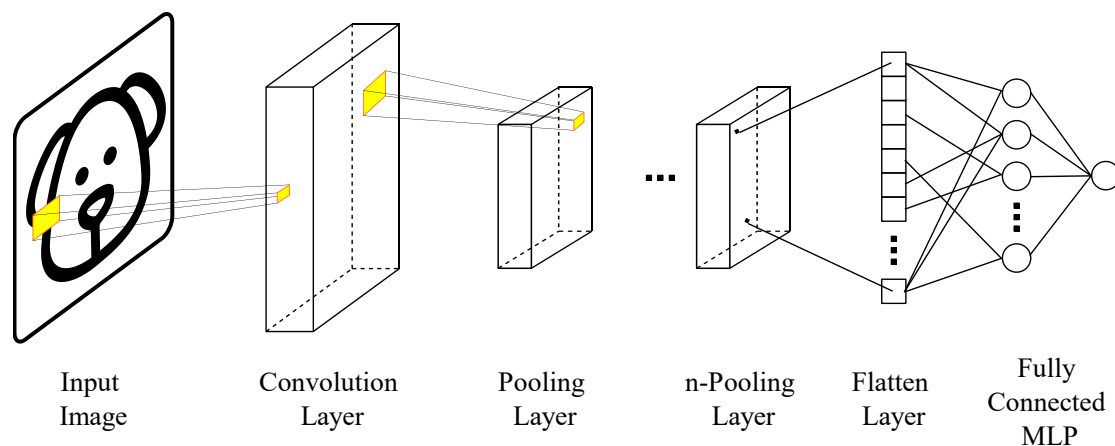
part it came from to create the new and smaller matrix. Those new matrices are the new “feature maps” exported by the layer.

This process can go on for the extraction of more complex features. By alternating convolution layers with downsampling layers, we can create a vast network that can detect more complex and advanced features but with the cost of using more processing power as well as more time needed to train such a network. Adding more layers can be critical at times, but it must not be overdone since the accuracy gain declines with every new layer.

Finally, after the feature extracting layers, comes the MLP. In order to feed the features to the MLP, all feature maps are flattened to one long column vector. This vector is used as input to the MLPs nodes. In order to train the CNN, back-propagation is used in a supervised learning scenario. The full setup of CNN is presented in Figure 6.

Although Recurrent Neural Networks like the LSTM we will examine later, in general, considered to be more appropriate for the job of time-series forecasting, recent studies have shown that CNNs can perform comparably well and better than MLP (Tsantekidis, 2017).

Figure 6. CNN schematic showing the various layers



Notes: The image is read as a matrix; it passes through the convolutional layer where the filters extract the feature map; the pooling layer then reduces the feature map; more alternating convolution and pooling layers can be added before their output is flattened; finally the fully connected MLP learns upon the extracted features.

The feature map, as already mentioned, is calculated using the convolution of the filter over the image. Both the filter and the image are matrices of numbers, and so the resulting feature map is a matrix as well. Since the filter moves over the image by

a step (also known as stride), producing a single value for each step, the resulting feature map matrix will have a smaller dimension of the image. Sometimes, in order to prevent this, and also to restore border pixels⁸ significance, padding is used. The dimensions of the feature matrix are calculated with the following function. Padding and stride are taken into account.

$$n_{out} = \left\lceil \frac{n_{in} + 2p - f}{s} + 1 \right\rceil$$

Where:

- n_{out} is the dimensions of the feature matrix,
- n_{in} is the dimensions of the input matrix (e.g. an image),
- p is the size of the padding on each side,
- s is the stride of the filter,
- f is the dimensions of the filter.

Feature map values are calculated with the following function:

$$A[m, n] = (G * H)[m, n] = \sum_{j=0}^{hj-1} \sum_{k=0}^{hk-1} H[j, k] G[m - j, n - k]$$

Where:

- A is activations/feature matrix,
- m, n are the calculated rows and columns of the activation matrix respectively,
- G is the input matrix (image),
- H is the filter matrix,
- j, k are the calculated rows and columns of the filter matrix respectively,
- hj, hk are the dimensions of the filter matrix (rows and columns respectively)

⁸ The border pixels of an image are used less times in a convolution since the filter passes over them less times than the inner pixels. This has the effect to loose information from those pixels. Sometimes this might be of importance and thus padding of the image is used. Padded pixels take the value of zero so they don't have an effect over other values of actual pixels.

Applying max-pooling to the feature matrix will create a new feature matrix with reduced dimensions. To apply max-pooling is also a convolution process, but this time instead of summing up the matrices elements, only the maximum value of the filtered input matrix is selected. The elements of the new matrix will have the maximum value found on each max-pooling filter pass.

The size of the new feature map is provided with the following functions:

$$h = \frac{n_h - f + 1}{s}$$

$$w = \frac{n_w - f + 1}{s}$$

where:

h is the height of the new feature map after max-pooling applied,

w is the width of the new feature map after max-pooling applied,

n_h is the height of the feature map,

n_w is the width of the feature map,

f is the size of the filter,

s is the stride length.

The new feature matrix is calculated with the following equation:

$$P[m, n] = \max(\{A[(m - 1) * s + j, (n - 1) * s + k]: j = 1 \dots m_f, k = 1 \dots n_f\})$$

Where:

P is the new feature matrix after max-pooling is applied,

m, n are the rows and columns of the new feature matrix respectively,

A is the input matrix (image),

s is the stride length,

m_f, n_f are the dimensions (rows and columns respectively) of the max-pooling filter.

To flatten the final feature matrix before we feed it to the MLP layer, we apply vectorization.

$$z = \text{vec}(P)$$

Where:

z is the new one-column vector,

P is the matrix that needs to be vectorized, the last feature matrix of the network.

Finally, the MLP network is using the vectorized last feature matrix to calculate the output of the CNN structure. The output of the MLP is calculated again with the following function:

$$\hat{y} = f \left(v_0 + \sum_{i=1}^{NH} v_i g(w_i^T x') \right),$$

Where:

f is the activation function for the output node,

NH is the number of nodes in the hidden layer,

v_0, v_1, \dots, v_{NH} are the weights for the output node,

g is the activation function used in the hidden layer,

w_i is the weight vector for the ' i ' hidden node,

and x' is the input vector x augmented with one,

$$x' = (1, x^T)^T$$

CNNs use the back-propagation method to be trained. Apart from the last MLP network that is trained, the filter weights are also trained. CNNs need to be trained with a large amount of data, and it is advised to be trained multiple times over the same training data. Each such training period is called an epoch. Training through various epochs help the network to understand the underlying relationships better.

Long Short-Term Memory Network (LSTM). A Long Short-Term Memory Network is a class of Recurrent Neural Network (RNN). Unlike feed-forward ANNs (artificial neural networks), the MLP we talked about is such an example, RNNs have feedback connections in such a way that the previous output is re-fed in the input. LSTMs are built upon this basic idea of RNNs.

The idea behind the feedback connection is to make the learning process depended on previous input states and thus make the output of the neural network depended on time. This connection makes the output depended on lags of unknown duration between events in a time-series. To achieve this effect, RNNs concatenate the current inputs with the network's previous output. This new input vector passes through a learnable neural network activated with a tanh function. This implementation gives the ability of memory to a neural network. Because of their ability to remember previous states and learn their relation to the current input, RNNs are good candidates for time-series forecasting and applications like speech recognition, language modelling, translation and many more.

A classic RNN setup can keep track of long past dependencies in an input sequence, but it faces the problem of Vanishing and Exploding Gradients, where in the first case nothing can be learned in acceptable time while in the second case there will be oscillating weights and the learning process will be unstable (Hochreiter and Schmidhuber, 1997). Gradients issues appear because of the way those networks are trained. RNNs use the back-propagation method to be trained as the MLP does. The gradient descent method is used to calculate the weights for each node when training the network. When the output of a node (and in extent the input weights) is passed through multiple feedback loops to the next 'state' of the calculation, it passes multiple stages of multiplications by weights with a value of less or bigger than one. Since a number multiplied numerous times with weights smaller than one, tend to vanish into zero the same thing happens with the said output value. When this happens, the memory of the network is lost. On the other hand, numbers multiplied with weights larger than one tend to explode, and this causes the network's outputs to become overwhelmed by the past memory.

LSTM was born to deal with those issues. Its name derives from the notion that the feedback loop in an RNN stores representations of recent inputs as activations which can be compared to short-term memory while information stored as input weights compare to long-term memory (Hochreiter and Schmidhuber, 1997). In order to have more control over how memory interacts with current inputs, trainable gates were introduced in the setup. This LSTM setup is also known as a memory cell, and it consists of the various gate layers as well as the trainable network (Hochreiter and Schmidhuber, 1997). The gates are activated with the sigmoid function, and they are trained on the concatenated vector of previous output and current input. Their role is to

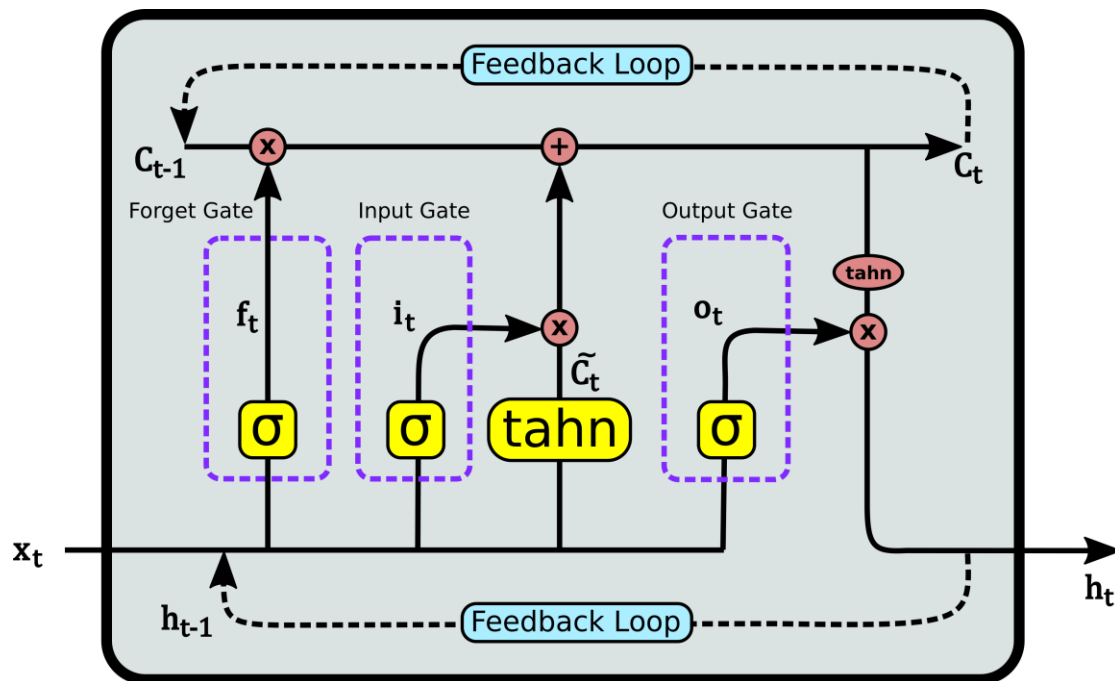
calculate the effect the signal has upon the rest of the data. The classic LSTM setup has three gate layers, while a lot of other setups were tested at times introducing, removing or combining the existing gates. Research in other setups concluded that they performed similarly.

The gates used in a classic LSTM cell are the input gate and the output gate (Hochreiter and Schmidhuber, 1997). Later, the forget gate was introduced to protect the network break down of an indefinitely internal state growth (Gers et al., 1999). In their study, they found that LSTM failed to learn to process specific continual time-series that was not segmented into training subsequences. Now, the forget gate is considered to be an integral part of the LSTM cell. The main difference with a simple RNN is that the LSTM cell has a separate state other than the output, known as cell state. The cell state is passed to the next lag of calculations along with the output.

At this point, it must be made clear that the LSTM cell is a full neural network, but as an entity, it can be considered to be a neural network neuron such as the perceptron, with a memory feature. LSTM cells can be combined to form layers of bigger neural networks like perceptrons do. An LSTM cell has an input and an output that connect to other neurons, like any other type of neural network neuron. For its inner workings though, the cell uses two more outputs that are fed back to its inputs, and those outputs are its memory known as cell state and its actual output. The cell is going to be trained on the combination of its previous output and its current input. The combination of previous output and current input is used on each gate as the actual input signal. The previous cell state is used to form the new cell state after the gates manipulate it in their way. The anatomy of the LSTM cell is shown in Figure 7.

Each gate is used to learn the importance of its input signal using a sigmoid activated neural network during the training process. Later on, the gate will apply a weight to its input signal according to this learned importance, before it allows the input signal to propagate to the next processing step. This learned importance of the input signal, in the form of neural network weights, is the long term memory of the cell because it is kept unchanged during the whole time the cell is active. All gates use the combined current cell input and previous cell output as their input signal, and this makes the cell's functioning depended on previous states as already mentioned before. This dependency on previous outputs along with the dependency on the previous cell state is the short term memory of the cell. The need for each gate is described bellow.

Figure 7. The LSTM cell anatomy



Notes: A lot more designs were proposed but with no significant performance improvement.

The forget gate was introduced so the cell can have control over its previous memory. As the name implies, this gate controls the amount of information that will be preserved from the previous cell state. Sometimes this memory needs to be preserved, while in other times it needs to be forgotten. The gate weighs the importance between zero and one. When a zero is used, the last cell-state is forgotten. On the other, when a value of one is used, the cell's state is preserved in its total. Values between zero and one indicate the level of importance of the state (memory) to preserve. The previous cell state is multiplied with that weight, for it to be transformed according to its importance. Then, this weighted previous cell state is used in the input gate to create the new cell state.

The input gate is used to classify the importance of the current input signal. The cell learns on the input signal using a tanh activated neural network, which is the actual learning component of the cell. The result of this learning process has a direct impact on the current cell state. Before it is added to the current cell state, it must be weighted according to its calculated importance through the input gate. The output of the tanh activated neural network is multiplied with the input gate's calculated weights, and then the outcome is added to the cell state. The new cell state combines the old state with the new one.

The output gate is responsible for weighting the current cell state before it is outputted as the actual cell output. The weights calculated by the gate are multiplied with the current cell state, and the result forms the new cell's output. Before the output gate uses the cell state, the cell state runs through a squashing tanh function to limit its values between minus one and one ($-1 \leq h_t \leq 1$).

The output of the forget gate is calculated as:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

The output of the input gate is calculated as:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

The output of the output gate is calculated as:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

The learned output is calculated as:

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

The new cell state is calculated as:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

Finally, the cell output is calculated as:

$$h_t = o_t \circ \tanh(c_t)$$

Where:

- x_t is the cell's input vector,
- f_t is forget gate's activation vector,
- i_t is input gate's activation vector,
- o_t is output gate's activation vector,
- \tilde{c}_t is the cell's input activation vector,
- h_t is the cell's output vector,
- c_t is the cell's state vector,
- σ is the sigmoid activation function,

\tanh is the tanh activation function,
 W is the input weight matrix that needs to be learned,
 U is the last output weight matrix that needs to be learned,
 b is the bias vector that needs to be learned,
 \circ denotes the Hadamard product

The sigmoid activation function is a mathematical function producing an S-shaped curve. There are a lot of sigmoid functions. The one used in the LSTM cell is the logistic function, and it produces values between zero and one. The logistic function is defined as:

$$s(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

The tanh or Hyperbolic Tangent activation function is a mathematical function that produces an S-shaped curve as well, but the values are limited between minus one (-1) and one. The tanh function is defined as:

$$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

LSTMs use the back-propagation method to be trained. Apart from the core neural network that is trained, the gates are also trained. Gates are trained to achieve an understanding of the current and past inputs and their importance. A problem with LSTMs is that they are prone to overfitting, which will result in pure performance. They need to be trained with a large amount of data, and it is advised to be trained multiple times over the same training data. Each such training period is called an epoch. Training through various epochs help the network to understand the underlying relationships better.

Comparison to Statistic methods. Machine Learning and specifically Deep Learning have proven their value in a variety of applications, so they are considered to be ideal for time-series forecasting applications. Their ability to recognize and learn upon non-linear relationships is a privilege over classical statistical methods for forecasting. Time-series data and the correlation they have with other time-series is overwhelmingly complicated. Even in more straightforward cases where only one time-

series is used to predict future values, it is not a simple task since the future is never the same with the past.

A vital fit, along with new model development, is the empirical validation of the numerous models and the comparison of their performance (Nesreen et al., 2010). Much effort has been put to studies to check Machine Learning methods' predictive skills focusing on the classification domain (Nesreen et al., 2010) and less so for time-series applications. There are not many large-scale studies focusing on comparing various models, Machine Learning and statistical methods, altogether (Nesreen et al., 2010; Makridakis et al., 2018). Most studies focus on a single or a couple of models with varied results, but mostly neural networks tended to outperform classical linear techniques (Nesreen et al., 2010). Although it is generally admitted that Machine Learning models show some predictive skill, there are concerns about their performance. When compared to classical statistical methods in a 2018 study by Makridakis et al., (2018) they are outperformed in all of the occasions.

According to Makridakis et al., (2018), studies that claim to find improved accuracy on new Machine Learning models, lack objective evidence since no benchmark was used to compare the results of those new algorithms. Instead, those studies only mention their findings of their model's performance, and most of the times, the information provided is vague and impossible to reproduce. Other reasons mentioned for those faulty claims is the usage of limited time-series data, and the evaluation of their method for one-step-ahead only, ignoring long term horizons. Those reasons can both lead to specific cases with small significance for general applications.

An important topic in academic discussion is the need to preprocess data before applying them to the machine learning algorithms. It is well known that in statistical forecasting, a preprocess of the data is needed for the model to generalize better on the provided data and improve its forecasting accuracy (Nesreen et al., 2010). In general, it is considered a minor issue since deep learning algorithms has shown that they are not affected negatively by the lack of stationarity or the existence of trend and seasonality. Models trained on non-preprocessed data have shown that their accuracy remained intact, but Nesreen et al., (2010) argue that the preprocessing method used can have a significant impact on performance. A possible advantage mentioned by Nesreen et al., (2010) in using, for example, a moving average as a preprocessing method, is that moving averages smooth out the noise allowing the model to focus on the global properties of the series.

On the other hand, in the same study, they mention that the use of differencing the time-series is not always a good strategy for non-stationary series. Thus, it is suggested the use of some kind of preprocessing, either with one or multiple techniques applied because this will lead to the development of a simpler model and the reduction in training time and resources required (Makridakis et al., 2018). An important thing to remember when developing a Machine Learning model is that a lot of trial and error attempts must be made in training the models, to find the best-suited hyperparameters for the model at hand. In those tests, using different forms of data might be useful as well, and it is advised to use unprocessed data as well as preprocessed data using different techniques every time (Nesreen et al., 2010; Makridakis et al., 2018) since there is no way to tell what form of data might work better for each problem. What would be of interest to study is the ability of future Machine Learning models to find on their own the best way to preprocess their data.

An argument about Machine Learning superiority is their sophistication by design. It is believed that the thought put in their design and their complexity alone is reason enough for them to have better skill in forecasting than statistical methods. This claim is backed by Machine Learning models' previous achievements in other fields of study, like computer vision and speech recognition. The truth is that studies have shown that even in statistical methods a lot of the times, simple methods outperform more complex ones, or at least they perform on the same level (Nesreen et al., 2010; Makridakis et al., 2018). That study comes to the same conclusion about deep learning algorithms where MLP and BNN achieve better performance than the rest.

On the other hand, Machine Learning algorithms have shown their skill to fit better the training data and their incompetence to post-sample forecasting accuracy. An example of this case is LSTM which is considered to be the best candidate in forecasting time-series. In Makridakis et al. (2018) study, when forecasting post-sample, the LSTM had the best fit in the training sample data but performed poorly. A study by Gers et al. (2001) came to the same conclusion when LSTM was outperformed by simple methods for univariate time-series forecasting applications.

Statistical methods outperform Machine Learning methods in multistep forecasting as well (Makridakis et al., 2018). In their study, three methods were considered for multiple step-ahead forecasting for Machine Learning methods.

- Iterative forecasting, where the step $(t + i)$ forecast was used to forecast step $(t + i + 1)$ in a one-step-ahead forecast scenario.

- Direct forecasting, where the neural network had as many output nodes as the steps required to predict (in the example it was 18), and each output was directly forecasting the appropriate step-ahead.
- Multi-neural network forecasting; where multiple neural networks were trained, and each neural network was trained to forecast a step-ahead horizon.

In all cases, Machine Learning methods were outperformed by simple statistical methods as well.

Makridakis et al. (2018) conclude that Machine Learning theorists need to work to improve the accuracy of their models and to achieve this, they need to use benchmark methods to test their findings. If a simple benchmark method performs better than the ML model, then more work needs to be done on the ML algorithm. Empirical testing must also be followed by research which will provide a better understanding of the inner mechanics of the algorithm. A final important thing suggested to add to a Machine Learning algorithm is the ability to provide the level of confidence and uncertainty of each prediction.

The baseline method

The purpose of this paper is to study the performance of Machine Learning models when applied in a multivariate time-series problem like the one in forecasting crude oil prices for various horizons. In order to check on their performance, tests need to be made against some baseline models. Those models can be some simple benchmarks or proved statistical models suggested by the extant literature. The Machine Learning models need to perform better than the benchmark to at least consider them as useful and having some predictive skill for the problem at hand. If this first milestone is achieved, their competitiveness will be measured against the statistical methods' results. If Machine Learning models cannot compare to statistical methods, this will be an indication of their immaturity and that more research needs to be done to improve their performance for multivariate time-series problems. If they manage to outperform the statistical models, this will be an indication that their non-linear function is more suited for multivariate time-series problems and that they manage to recognize complex time-series correlations besides the fact that they fail to do so in univariate problems according to Makridakis et al. (2018).

The statistical models that are considered for this paper will be taken from Degiannakis and Filis (2018) paper that studies the importance of high-frequency financial data in forecasting oil prices. In that paper Degiannakis and Filis (2018) employ 29 MIDAS models using monthly fundamental oil time-series data along with ultra-high frequency data to forecast oil prices for 1-, 3-, 6-, 9- and 12-months ahead horizons. For each MIDAS model, they use one asset's Volatility or return at a time.

To test the importance of using ultra-high frequency data in forecasting monthly oil price movements, Degiannakis and Filis (2018) employ other models suggested by the extant literature to provide superior forecasting results. Their purpose is to compare the results and see which model performs better and what combination of data used helps those results to improve. More importantly, the question that needs to be answered is, does the use of ultra-high frequency data improve the predictive ability? Those models use only monthly data. The models used are AR(1), AR(12), AR(24), and ARMA(1,1), unrestricted VAR models and BVAR models with three and four endogenous variables. All these models are denoted as standard models.

As a necessary measure for the models' performance, a benchmark is used. The benchmark that will be used is the Random Walk Model (RWM) which according to

Coppola (2008) considers it to be a consistent benchmark for oil market since oil market efficiency implies that prices fully reflect all information available. Degiannakis and Filis (2018) use the random-walk model as the no-change forecast. This benchmark model is one of the simplest tests that need to be applied to a forecasting model. It is used to evaluate the forecasting performance of the model against this simple forecast. If a forecasting model fails to underperform this simple benchmark, then the model has no forecasting skill whatsoever.

Another essential test is the directional accuracy of the models. Although the model might not be able to predict the exact price, it may predict the direction the price is going to move. Degiannakis and Filis (2018) in order to measure the directional accuracy, use the success ratio of the times the direction of price movement was predicted correctly against the no-change forecast. The importance of this ratio is calculated for every model.

In their paper, Degiannakis and Filis (2018) find that the best model is indeed a MIDAS model and the set of data that provides the most meaningful information in predicting the oil price movements are the Realized volatilities set. In particular, the models that distinguish are the MIDAS-RV-XX (MIDAS model based on Euro Stoxx 50 index Realized Volatility), MIDAS-RV-BP (MIDAS model based on GBP/USD exchange rate Realized Volatility), and MIDAS-RV-PA (MIDAS model based on Palladium Realized Volatility). Those models perform the best for all horizons except the 9-month ahead. For the 9-month ahead horizon, the best model is the trivariate BVAR model with 12 lags (3-BVAR(12)). As already mentioned, their performance is tested against the no-change forecast. Another critical thing noticed during their study is the fact that for each forecasting horizon, a different asset was eminent as the most important and the one that contributes the most in forecast accuracy. Concerning the directional accuracy, the things are a bit different. MIDAS-RV are those with the highest directional accuracy for up to 3-months ahead forecasting horizon but then VAR and BVAR models surpass them. MIDAS-RV models retake head for the 12-months ahead forecasting horizon.

The data used in Degiannakis and Filis (2018) study, will be used in this study as well, so the results are comparable. The data contain monthly and daily tick-by-tick time-series for the period that spans from August 2003 to August 2015. Monthly data consist of the primary oil market fundamentals data (the Global Economic Activity Index, the Baltic Dry Index, the Global Oil Production, the Global oil Stocks, the

Capacity Utilization Rate of the oil and gas industry), and the ultra-high frequency data consist of four different asset classes (Forex, Stocks, Commodities and Macro). Ultra-high frequency data are used to create the weakly realized volatilities of the assets and their daily log-returns. Those assets are the following: the front-month futures contracts for three paramount exchange rates (GBP/USD, CAD/USD, EUR/USD), four stock market indices (FTSE100, S&P500, Hang Seng, Euro Stoxx 50), six commodities (Brent crude oil, Gold, Copper, Natural Gas, Palladium, Silver), two macroeconomic assets (the US 10 yr T-bills and the US Economic Policy Uncertainty Index).

The ML methods in action

In this section, we study the Machine Learning models that will be tested against the models employed in Degiannakis and Filis (2008) paper. Their performance results will be compared to the ones we produce using Machine Learning models employing various setups. We need to come to an understanding of Machine Learning models performance and know which, if any, can be used in forecasting oil price movements. Also, we need to understand which data combination can provide better information to such a model.

Data

The data used in Degiannakis and Filis (2018) study, will be used in this study as well, so the results are comparable. In this study, we do not question the importance of each time-series used in Degiannakis and Filis (2018) paper. We only need to compare the performance of Machine Learning models in a challenging multivariate time-series problem against some proven, and skilful statistical models and their paper provides a decent framework to do so. Using the same data will help us focus on the importance of the model used and its forecasting ability instead of the importance of using a specific time-series. Besides, this process will provide new insights on the importance of each time-series, and specifically, it will make prominent which time-series work better with the Machine Learning model at hand.

There will be a small differentiation in the usage of training data used by the Machine Learning models. Training data will be split into two groups. Those groups are the actual training data which consists of 80% of the original training data, and the rest 20% of them will be the validation set, used to configure the optimal hyperparameters for each model. The test set will be the same as proposed by Degiannakis and Filis (2018).

A considerable difference between using Machine Learning models and an econometric model is the data format. The packages that are used for econometric analyses use ordered tables with columns representing each time-series and a lag value on each row. Those packages do the preparations needed on the data, so they feed each model during the training process. They are specialized in working with time-series data, and they can be configured in the way they treat lags in the data.

In contrast, there are no such automated processes in Machine Learning because Machine Learning models are not aware of the specifics of the time-series data. Because of the generalized form of Machine Learning processes, there is no specialized approach to time-series problems, and there is no way to configure how the model needs to treat those lags. Thus, for that reason, data need to be prepared manually to match the models' input. For example, in a RandomForestRegressor model, each lag must be set as a new column. The Machine Learning model at hand will try to find the correlations between those columns and by definition, the correlation between lags. The same applies to MLP as well. The following example demonstrates this.

Table 1. Formatting time-series lag for use with ML model

Date	TimeSeries_A	TimeSeries_A_Lag_1	TimeSeries_A_Lag_2
4/1/2020	1	2	3
3/1/2020	2	3	4
2/1/2020	3	4	...
1/1/2020	4

Notes: Lags must be provided as features in a machine learning model since it is not aware of the notion of time lag.

The example above demonstrates how a time-series with two lags needs to be prepared in order to feed a Machine Learning model. For each lag, a new column is created, and for each lag, the rows are shifted up one place from the previous lag.

We can choose which data lags we provide as columns in case we have some information on the data significance, so we help simplify the model. For example, we can skip the fifth lag if we know from other studies that it does not provide any significant forecasting information. In any other case we provide as many lags we can so the model can train on them and decide which one is important.

At this point, we need to point out that the known Machine Learning models behave more like the standard models mentioned by Degiannakis and Filis (2018). They can only use one frequency data. For that reason, high-frequency data and their lags must be provided as new columns as well. So, each row can have tens or even hundreds of columns, which is going to have an impact on the resources and time need for training each model.

In Degiannakis and Filis (2018), a lot of time-series data are used in combination. Those time series are the oil market fundamentals, ultra-high-frequency realized volatilities and ultra-high-frequency returns.

Those assets are presented in Table 2 of Degiannakis and Filis (2018) paper, and they are the following.

Table 2. Assets used in Degiannakis and Filis 2018 paper

Name	Acronym	Description/frequency
Global Economic Activity Index	GEA	Proxy for the global business cycle. Monthly data.
Baltic Dry Index	BDI	Proxy for the global business cycle. Monthly data.
Global Oil Production	PROD	Proxy for oil supply. Monthly data.
Global Oil Stocks	STOCKS	Proxy for global oil inventories. Monthly data
Capacity Utilization Rate	CAP	Proxy for oil demand in relation to economic activity. Monthly data
Brent Crude Oil	CO	Tick-by-tick data of the front-month futures prices
GBP/USD exchange rate	BP	Tick-by-tick data of the front-month futures prices
CAD/USD exchange rate	CD	Tick-by-tick data of the front-month futures prices
EUR/USD exchange rate	EC	Tick-by-tick data of the front-month futures prices
FTSE100 index	FT	Tick-by-tick data of the front-month futures prices
S&P500 index	SP	Tick-by-tick data of the front-month futures prices
Hang Seng index	HI	Tick-by-tick data of the front-month futures prices
Euro Stoxx 50 index	XX	Tick-by-tick data of the front-month futures prices
Gold	GC	Tick-by-tick data of the front-month futures prices
Copper	HG	Tick-by-tick data of the front-month futures prices
Natural Gas	NG	Tick-by-tick data of the front-month futures prices
Palladium	PA	Tick-by-tick data of the front-month futures prices
Silver	SV	Tick-by-tick data of the front-month futures prices
US 10 yr T-bills	TY	Tick-by-tick data of the front-month futures prices
Economic Policy Uncertainty Index	EPU	Proxy for the US macroeconomic volatility. Daily data.

The assets that were used are the following:

- Fundamentals set. Contains all of GEA, PROD, STOCKS, CAP
- Realized volatilities. A time-series at a time was used of the following: CO, BP, CD, EC, FT, SP, HI, XX, GC, HG, NG, PA, SV, TY, EPU.
- Returns. A time-series at a time was used of the following: CO, BP, CD, EC, FT, SP, HI, XX, GC, HG, NG, PA, SV, TY.

EPU is only used in realized volatilities because it is a proxy of macroeconomic Volatility itself.

We do not use every single of those time-series for the sole reason that not all of them are important for our case. In order to test every single of them the way Degiannakis and Filis (2018) do it, we would be forced to train and evaluate hundreds of models. Fortunately, we are in the position to know which set of data can provide the best forecasting ability to our models. As mentioned in Degiannakis and Filis (2018), there is no single set of data that provides the best forecasting ability for all forecasting horizons. For each forecasting horizon, a different time-series rises to the top. As shown, the best models for horizons 1-, 3- & 6-, 12-months ahead, are MIDAS-RV with Eurostoxx 50, GBP/USD and Palladium volatilities respectively. For the 9-month ahead horizon, the BVAR model with 12 lags is seen to perform the best of them. So, we can use only the best of them depending on the forecast horizon we test so we can get an idea of their performance against the best models in Degiannakis and Filis (2018).

For this paper and to save on time and resources, we use a limited amount of data. We use the monthly oil prices and all of the four monthly fundamentals time series () as well as a couple of ultra-high-frequency series. The ultra-high-frequency series we use is the Euro Stoxx 50 index Realized Volatility and the Palladium Realized volatility. We use these two series because they provide the best forecasting information for 1- and 12-months ahead horizons according to Degiannakis and Filis (2018).

For each monthly series, 12 lags are used to feed the models. This number of lags was chosen because the BVAR model with 12 lags showed a superior forecasting ability in Degiannakis and Filis (2018) study. This finding is a good indication that there is important information in those lags, and it might be valuable for the Machine Learning models as well. Those models have to decide if and in what way they can use this information.

For the ultra-high-frequency series, 22 lags are used as in Degiannakis and Filis (2018). Those lags are daily, and they represent the Realized Volatility of the asset for the past month. The models use only one asset Realized Volatility set at a time.

Models used

For this paper, we create models using the Machine Learning model classes already mentioned. Each model class has its pros and cons, and since they are so different in design, it is interesting to test each one of those.

It is made clear that a lot of different models need to be created, trained and tested to cover for any possible combination of ‘Model class’ – ‘data combination used’ – ‘forecasting horizon’. In order to test any combination of the previously mentioned models, data and horizons, we would need to create more than 600 models. To train so many models demands a lot of resources and time. Such an effort is beyond the purposes of this study, so we limit our options to the following.

To simplify things a bit, and to make better use of the available resources and time, we are going to create models to forecast oil price movements only for the 1-month ahead and 12-month ahead horizons for each of the four proposed models. Those two forecasting horizons are enough to get an idea of the forecasting ability of the Machine Learning models.

For each horizon, three different combinations of data are used based on the results of Degiannakis and Filis (2018). We use fundamentals only dataset, and the combination of the fundamentals only dataset with each of the two Realized Volatility sets we chose earlier.

So, all in all, we create twelve models in total, which result in 24 forecasts. In detail, one model is created for each model class that uses the only fundamentals dataset and one model that uses a combination of the fundamentals dataset with the ultra-high-frequency datasets. Each of those models is used to predict for each of the two selected forecasting horizons.

The models are trained using the train data set, which in turn is split into two other sets. The first 80% of this data set is the factual training set, and the rest 20% is used to tune the hyperparameters of each trained set in order to optimize the model’s generalization ability.

Bellow, we study each set of models. For each model, two versions are presented. The first version is the one that uses only the fundamentals dataset, and it is the purest form of the model. The second version is the more complex construct that uses the ultra-high-frequency data as well. The second version is going to be fed the two realized volatilities series in two different runs. Those are the Palladium realized volatility, and the Eurostoxx 50 index realized volatility. Those models are presented in Table 3.

Table 3. Models created and trained to forecast oil prices

Model Name	Description
Plain_CNN	Convolutional Network used with Fundamentals only dataset
CNN_PA	Convolutional Network used with Fundamentals dataset in combination with Palladium Realized Volatility
CNN_XX	Convolutional Network used with Fundamentals dataset in combination with Eurostoxx 50 Index Realized Volatility
Plain_LSTM	Long Short-Term Memory used with Fundamentals only dataset
LSTM_PA	Long Short-Term Memory used with Fundamentals dataset in combination with Palladium Realized Volatility
LSTM_XX	Long Short-Term Memory used with Fundamentals dataset in combination with Eurostoxx 50 Index Realized Volatility
Plain_MLP	Multilayer Perceptron used with Fundamentals only dataset
MLP_PA	Multilayer Perceptron used with Fundamentals dataset in combination with Palladium Realized Volatility
MLP_XX	Multilayer Perceptron used with Fundamentals dataset in combination with Eurostoxx 50 Index Realized Volatility
Plain_RFR	Random Forest Regressor used with Fundamentals only dataset
RFR_PA	Random Forest Regressor used with Fundamentals dataset in combination with Palladium Realized Volatility
RFR_XX	Random Forest Regressor used with Fundamentals dataset in combination with Eurostoxx 50 Index Realized Volatility

Random Forest Regressor models. One version of this model is implemented. Unlike deep learning models that their inputs must match the number of features in the data, Random Forest models can adapt to any number of features in the input data. In contrast to this, two versions are used in this paper for the clarity of reference. Each version refers to the data used, like the fundamentals dataset and the ultra-high-frequency dataset.

Plain RFR is the first version of the Random Forest Regression model, using only the fundamentals dataset. The data has 60 features because it has 12 time-lags for each of the five features it uses. Namely, they are oil prices, cap, prod, stocks and gea.

RFR_HF is the second version of the Random Forest Regression model, using the same dataset as the first version, plus the ultra-high-frequency dataset for one realized volatility series each time. The data has 82 features because it gets 60 features from the fundamentals like the plain RFR model and 22 extra features for the lags in the Realized Volatility series.

MLP models. Two versions of this model are implemented, one for fundamentals dataset and one for the combination of fundamentals and ultra-high-frequency dataset, respectively.

Plain MLP is the first version of the model in its purest form, having one hidden layer. The output calculation of this model is explained in section Multilayer Perceptron (MLP).

The output value for the MLP model is calculated as follows:

$$\hat{y} = f \left(v_0 + \sum_{i=1}^{NH} v_i g(w_i^T x') \right),$$

The input layer has 60 inputs because it has 12 time-lags for each of the five features it uses. Namely, they are oil prices, cap, prod, stocks and gea.

MLP_HF is the second version of the MLP model, using the same dataset as the first version, plus the ultra-high-frequency dataset for one realized volatility series each time. To achieve this, we implement two hidden layers in parallel, each feeding on different inputs. The first hidden layer feeds on the fundamentals dataset as before, and the second hidden layer feeds on the ultra-high-frequency data. Finally, we concatenate their node outputs, and we calculate the output value. Layer connectivity is presented in Figure 8.

The output value for MLP_HF model is calculated as follows:

$$\hat{y} = f \left(v_0 + \sum_{i=1}^{NHX} v_i g(w_i^T x') + \sum_{k=1}^{NHZ} b_k g(m_k^T z') \right),$$

Where:

f is the activation function for the output node,

NHX is the number of nodes in the fundamentals hidden layer,

NHZ is the number of nodes in the ultra-high-frequency hidden layer,

v_0 is the weight for the output node's bias,

v_1, \dots, v_{NHX} are the output node's weights coming from the fundamentals hidden layer,

b_1, \dots, b_{NHZ} are the output node's weights coming from the ultra-high-frequency hidden layer,

g is the activation function used in the hidden layers,

w_i is the weight vector for the ‘ i ’ fundamentals hidden node,
 m_k is the weight vector for the ‘ k ’ ultra-high-frequency hidden node,
 x' is the fundamentals input vector x augmented with one,

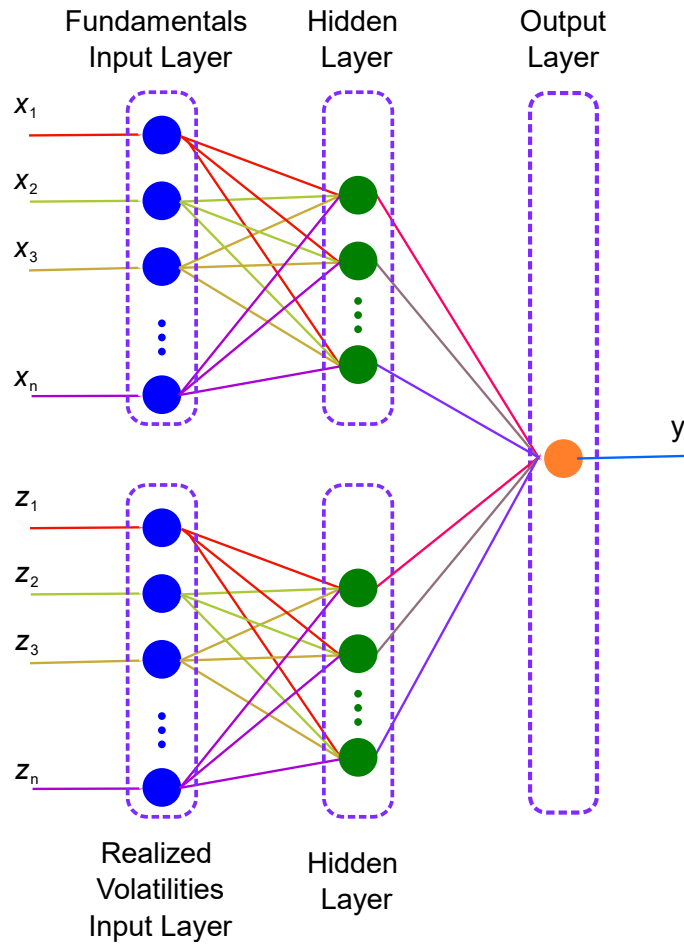
$$x' = (1, x^T)^T$$

and z' is the ultra-high-frequency input vector z augmented with one,

$$z' = (1, z^T)^T$$

The input layer has 82 inputs because it gets 60 inputs from the fundamentals like the plain MLP model and 22 extra inputs for the lags in the Realized Volatility series.

Figure 8. Representation of the layer connectivity in MLP_HF model

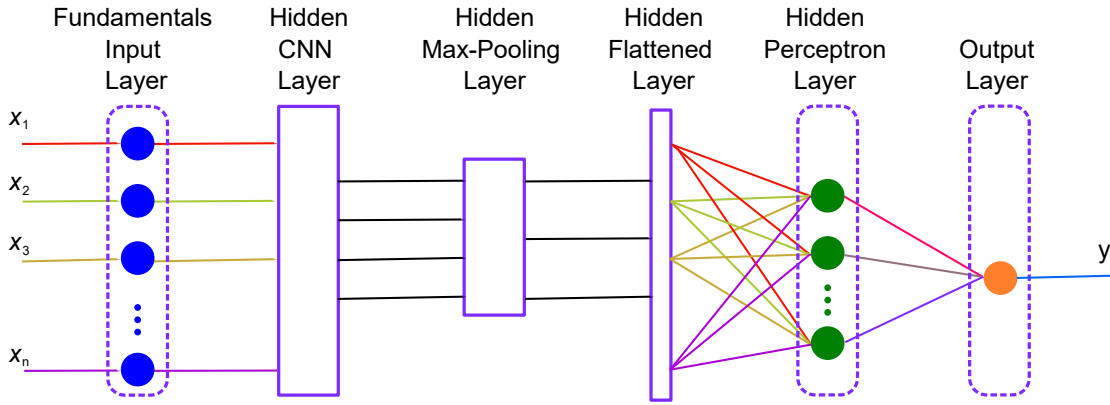


Notes: Two branches of perceptron layers are used. One layer gets input from fundamentals dataset; the other layer gets inputs from the ultra-high-frequency dataset. The output concatenates both branches.

CNN models. Two versions of this model are implemented, one for fundamentals dataset and one for the combination of fundamentals and ultra-high-frequency dataset, respectively.

Plain CNN is the first version of the model in its purest form, having one hidden convolution layer and one max-pooling layer. The output passes through an MLP with one hidden layer as usual. Figure 9 shows the layer connectivity used in the plain CNN model. The output calculation of this model is explained in section Convolutional Neural Network (CNN).

Figure 9. Representation of the layer connectivity in plain CNN model



Notes: A single convolution layer is used; followed by a max-pooling layer; which in turn is flattened and fed to an MLP network to calculate the output.

The input layer has 60 inputs because it has 12 time-lags for each of the five features it uses. Namely, they are oil prices, cap, prod, stocks and gea.

CNN_HF is the second version of the CNN model, using the same dataset as the first version, plus the ultra-high-frequency dataset for one realized volatility series each time. To achieve this, we implement two sets of hidden layers in parallel, each feeding on different inputs. The first hidden layer set feeds on the fundamentals dataset as before, and the second hidden layer set feeds on the ultra-high-frequency data. Finally, we flatten and concatenate their outputs before we drive them to an MLP with one hidden layer. Finally, we use one output node to get the result. Layer connectivity is presented in Figure 10.

The output value for CNN_HF model is calculated as follows. First, we calculate the two feature maps A1 and A2, using the following equation:

$$A_x[m, n] = (G * H)[m, n] = \sum_{j=0}^{hj-1} \sum_{k=0}^{hk-1} H[j, k]G[m - j, n - k]$$

Where:

A_x is activations/feature matrix for the x convolution set (A_1 and A_2),
 m, n are the calculated rows and columns of the activation matrix respectively,
 G is the input matrix (image),
 H is the filter matrix,
 j, k are the calculated rows and columns of the filter matrix respectively,
 hj, hk are the dimensions of the filter matrix (rows and columns respectively)

We apply max-pooling to each feature matrix using the following equation:

$$P_x[m, n] = \max(\{A_x[(m - 1) * s + j, (n - 1) * s + k]: j = 1 \dots m_f, k = 1 \dots n_f\})$$

Where:

P_x is the new feature matrix after max-pooling is applied on the x convolution set,
 m, n are the rows and columns of the new feature matrix respectively,
 A_x is the input matrix (image) for the x convolution set,
 s is the stride length,
 m_f, n_f are the dimensions (rows and columns respectively) of the max-pooling filter.

We flatten the final feature matrices before we feed them to the MLP layer, we apply vectorization.

$$z_x = \text{vec}(P_x)$$

Where:

z_x is the new one-column vector for the x convolution set,

P_x is the matrix that needs to be vectorized, the last feature matrix for the x convolution set.

Finally, the MLP network is using the vectorized last feature matrix to calculate the output of the CNN structure. The output of the MLP is calculated again with the following functions:

$$r = g \left(v_0 + \sum_{i=1}^{NHX} v_i z_{1,i} + \sum_{k=1}^{NHZ} b_k z_{2,i} \right),$$

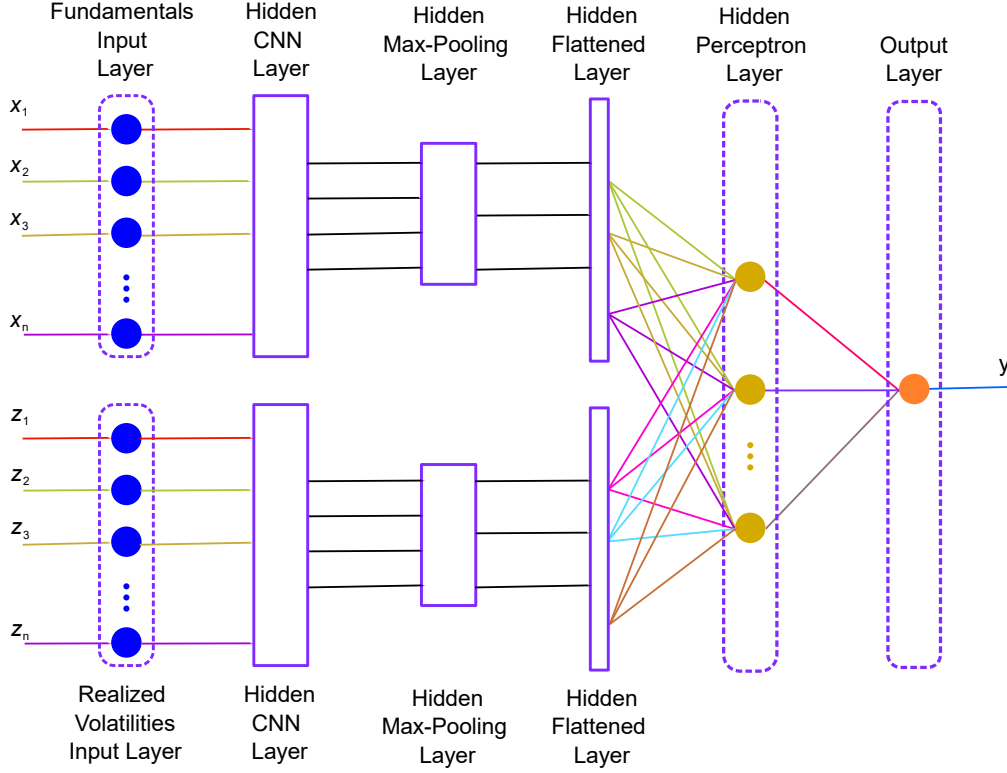
$$\hat{y} = f \left(c_0 + \sum_{i=1}^{NH} c_i (r_i) \right),$$

Where:

g is the activation function used in the hidden perceptron layer,
 NHX is the number of feature values that come out of the flattened max-pooling matrix in the hidden fundamentals CNN layer,
 NHZ is the number of feature values that come out of the flattened max-pooling matrix in the hidden ultra-high-frequency CNN layer,
 $z_{1,i}$ is the output vector of the hidden fundamentals CNN layer for the ' i ' feature value,
 $z_{2,i}$ is the output vector of the hidden fundamentals CNN layer for the ' i ' feature value,
 v_1, \dots, v_{NHX} are the perceptron hidden layer node's weights coming from the fundamentals hidden CNN layer,
 b_1, \dots, b_{NHZ} are the perceptron hidden layer node's weights coming from the ultra-high-frequency hidden CNN layer,
 f is the activation function for the output node,
 NH is the number of nodes in the hidden perceptron layer,
 c_0, c_1, \dots, c_{NH} are the weights for the output node,
 r_i is the output vector of the hidden perceptron layer for the ' i ' node,

The input layer has 82 inputs because it gets 60 inputs from the fundamentals like the Plain CNN model and 22 extra inputs for the lags in the Realized Volatility series.

Figure 10. Representation of the layer connectivity in CNN_HF model



Notes: Two branches of convolution layers are created, one branch for fundamentals and one branch for ultra-high-frequency data. The outputs of both branches are flattened, concatenated and fed to an MLP network.

LSTM models. Two versions of this model are implemented, one for fundamentals dataset and one for the combination of fundamentals and ultra-high-frequency dataset, respectively.

Plain LSTM is the first version of the model in its purest form, having one hidden layer of LSTM cells and one perceptron output node. The output calculation of this model is explained in section Long Short-Term Memory Network (LSTM). Figure 4 can as well be used to describe this setup since it is the same, but instead of perceptrons, the hidden layer consists of LSTM cells.

The output value for the MLP model is calculated as follows:

$$\hat{y} = f\left(v_0 + \sum_{i=1}^{NH} v_i h_i\right)$$

Where:

f is the activation function for the output node,

NH is the number of LSTM cells in the hidden layer,

v_0, v_1, \dots, v_{NH} are the weights for the output node,

h_i is the output vector of the LSTM layer for the ' i ' hidden LSTM cell.

The input layer has 60 inputs because it has 12 time-lags for each of the five features it uses. Namely, they are oil prices, cap, prod, stocks and gea.

LSTM_HF is the second version of the LSTM model, using the same dataset as the first version, plus the ultra-high-frequency dataset for one realized volatility series each time. To achieve this, we implement two hidden layers in parallel, each feeding on different inputs. The first hidden layer feeds on the fundamentals dataset as before, and the second hidden layer feeds on the ultra-high-frequency data. Finally, we concatenate their outputs before we drive them to one more hidden layer of perceptrons. Finally, we use one output node to get the result. Layer connectivity is presented in Figure 11.

The output value for MLP_HF model is calculated as follows:

$$p = g \left(v_0 + \sum_{i=1}^{NHX} v_i hl1_i + \sum_{k=1}^{NHZ} b_k hl2_i \right),$$

$$\hat{y} = f \left(c_0 + \sum_{i=1}^{NH} c_i(p_i) \right),$$

Where:

g is the activation function used in the hidden perceptron layer,

NHX is the number of LSTM cells in the hidden fundamentals LSTM layer,

NHZ is the number of LSTM cells in the hidden ultra-high-frequency LSTM layer,

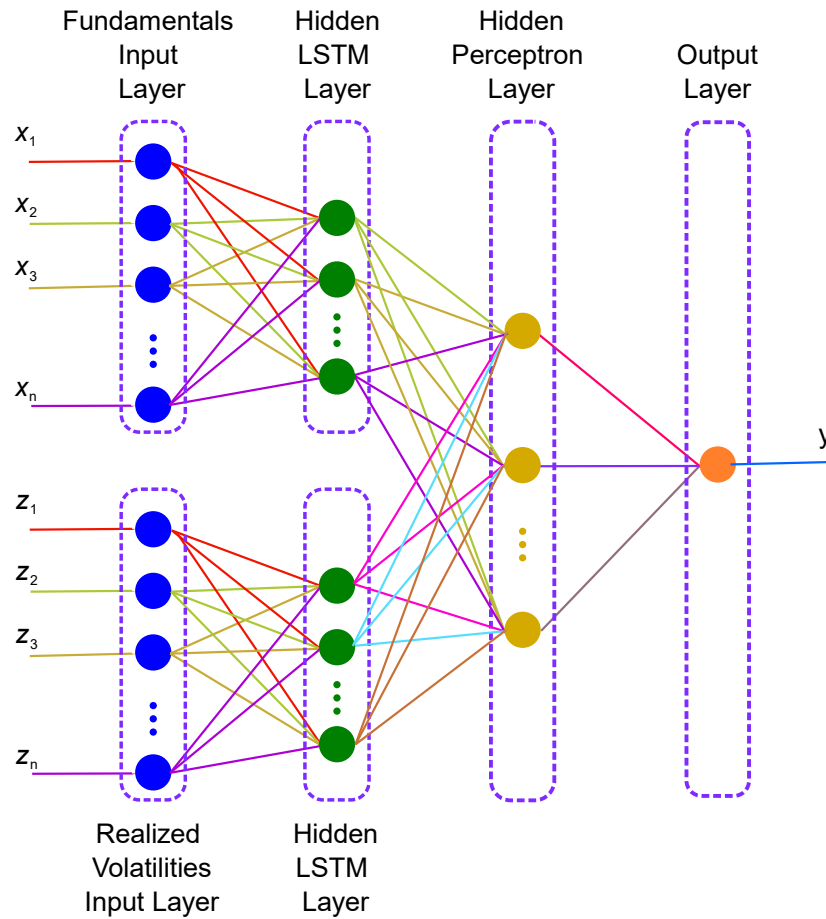
$hl1_i$ is the output vector of the hidden fundamentals layer for the ' i ' LSTM cell,

$hl2_i$ is the output vector of the hidden fundamentals layer for the ' i ' LSTM cell,

v_1, \dots, v_{NHX} are the perceptron hidden layer node's weights coming from the fundamentals hidden LSTM layer,
 b_1, \dots, b_{NHZ} are the perceptron hidden layer node's weights coming from the ultra-high-frequency hidden LSTM layer,
 f is the activation function for the output node,
 NH is the number of nodes in the hidden perceptron layer,
 c_0, c_1, \dots, c_{NH} are the weights for the output node,
 p_i is the output vector of the hidden perceptron layer for the 'i' node,

The input layer has 82 inputs because it gets 60 inputs from the fundamentals like the Plain LSTM model and 22 extra inputs for the lags in the Realized Volatility series.

Figure 11. Representation of the layer connectivity in LSTM_HF model



Notes: Two branches LSTM layers are created, one branch for fundamentals and one branch for ultra-high-frequency data. The outputs of both branches are flattened, concatenated and fed to an MLP network.

Model evaluation

During the model evaluation, we calculate the predictive accuracy using two loss functions. The Mean Squared Predicted Error (MSPE or MSE) and the Mean Absolute Percentage Predicted Error (MAPPE or MAPE) are employed for this purpose, according to Degiannakis and Filis (2018). Then, the results are compared to the no-change forecast, and they are also compared against those found by Degiannakis and Filis (2018) to tell the predictive accuracy of the Machine Learning models.

MSE is calculated using the following function:

$$M = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2$$

Where:

- M is the Mean Square Error,
- n is the number of samples tested,
- y_i is the actual value for the i sample,
- \hat{y}_i is the forecasted value for the i sample.

MAPE is calculated using the following function:

$$M = \frac{1}{n} \sum_{t=1}^n \left| \frac{A_t - F_t}{A_t} \right|$$

Where:

- M is the Mean Absolute Percentage Error,
- n is the number of samples tested,
- A_t is the actual value for the t sample,
- F_t is the forecasted value for the t sample.

Another test is the directional accuracy of the models. We calculate the directional accuracy using the success ratio of the times the model predicted the direction oil price would move in the future. It is shown in the following function:

$$a_t = \begin{cases} 1, & (y_{t-1} > y_t) = (y_{t-1} > \hat{y}_t) \\ 0, & (y_{t-1} > y_t) \neq (y_{t-1} > \hat{y}_t) \end{cases}$$

$$ar = \frac{1}{n} \sum_{t=1}^n a_t$$

Where:

- a_t is the success in predicting the direction of the price movement,
- y_t is the actual value for sample t,
- y_{t-1} is the actual value for sample t-1,
- \hat{y}_t is the forecasted value for sample t,
- n is the number of samples tested.

Those two tests can provide an insight into each model's forecasting ability and how they compare to the classic statistical models.

Results

Our study showed that the Machine Learning models tested for this paper, lack of any predictive ability. They fail to make good use of the information available in the provided time-series, although those time-series are already proven from the extant literature to be of importance for classic statistical models like MIDAS and BVAR. The results are shown in Table 4 and Table 5, for 1- and 12-month forecasting horizons, respectively.

For the horizon of 1-month ahead, we notice that only the plain LSTM model with the fundamentals dataset exhibits some performance similar to the no-change forecast. Both test MSE and test MAPE show a slightly better performance for the test dataset over the no-change forecast, although in essence, they are almost equal with an improvement of less than 1%. Still, the rest of the models for the 1-month ahead forecasting horizon, exhibit a way worse behaviour with some values being up to 6 times larger for MAPE and 38 times larger for MSE compared to no-change forecast metrics.

One other metric to check for the model performance is the coefficient of determination (R^2). In our case, this metric shows some good fit on the test data ($R^2=0.9045$) well on this metric as well, but they produce way higher errors than the no-change forecast. All models, despite the R^2 metric, are rendered incompetent in forecasting oil price movements.

Finally, checking the directional success ratio of the LSTM model, we see it has the highest success ratio of the models used for this forecasting horizon, with a value of 65.12%. Still, this metric indicates quite poor performance as well. Once more, the rest of the models fail to this metric too. Although some indicate to have a similar success ratio, they cannot be trusted because of their error-prone behaviour.

Table 4. 1-month ahead forecasting horizon

Model Type	Training MSE	Valid MSE	Test MSE	Training MAPE	Valid MAPE	Test MAPE	Training R ²	Valid R ²	Test R ²	Success Ratio
CNN	0.0016	18.4986	20.6660	0.0381	4.2530	4.2873	0.9998	-2.5956	-0.9938	0.5581
CNN_PA	0.0274	12.5771	17.5612	0.1763	3.4874	3.8082	0.9964	-1.4446	-0.6943	0.6279
CNN_XX	0.0012	31.0681	36.2651	0.0339	5.5396	5.6806	0.9998	-5.0387	-2.4988	0.6279
LSTM	0.3494	1.4243	0.9903	0.5981	0.9729	0.9950	0.9546	0.7232	0.9045	0.6512
LSTM_PA	0.0000	2.3683	1.7984	0.0016	1.4055	1.2127	1.0000	0.5397	0.8265	0.6047
LSTM_XX	0.0000	7.4299	7.2804	0.0032	2.7121	2.4871	1.0000	-0.4441	0.2976	0.5581
MLP	0.0001	11.2350	37.9764	0.0110	3.4534	6.4762	1.0000	-1.1837	-2.6639	0.6279
MLP_PA	0.0000	3.2027	9.8095	0.0053	1.7517	3.2435	1.0000	0.3775	0.0536	0.6279
MLP_XX	0.0000	8.9203	26.9456	0.0047	3.0590	4.9790	1.0000	-0.7338	-1.5997	0.6047
RFR	0.8093	0.8145	1.4110	0.6874	0.8009	1.2097	0.8949	0.8417	0.8639	0.5349
RFR_PA	0.7200	0.7024	1.3698	0.6998	0.8440	1.1761	0.9064	0.8635	0.8678	0.6279
RFR_XX	0.2544	0.7195	1.7796	0.4592	0.8384	1.3134	0.9669	0.8601	0.8283	0.6047

Notes: All MSE and MAPE ratios have been normalized relative to the monthly no-change forecast. Boldface indicates predictive gains relative to the no-change forecast.

Table 5. 12-month forecasting horizon

Model Type	Training MSE	Valid MSE	Test MSE	Training MAPE	Valid MAPE	Test MAPE	Training R ²	Valid R ²	Test R ²	Success Ratio
CNN	0.0000	1.4005	0.9308	0.0055	1.1006	1.0492	0.9999	-22.2691	-0.0790	0.4688
CNN_PA	0.0000	0.7939	1.3723	0.0015	0.8237	1.3081	1.0000	-12.1900	-0.5908	0.4375
CNN_XX	0.0000	1.9089	1.3318	0.0007	1.3670	1.2451	1.0000	-30.7168	-0.5438	0.5000
LSTM	0.2158	1.3475	6.8427	0.3998	1.0517	3.2296	0.5418	-21.3892	-6.9322	0.3438
LSTM_PA	0.0000	0.7260	11.7315	0.0056	0.7214	4.1317	0.9999	-11.0617	-12.5995	0.3438
LSTM_XX	0.0113	1.1514	11.0964	0.0888	0.9993	4.0334	0.9759	-18.1304	-11.8632	0.3438
MLP	0.0000	1.3144	1.9568	0.0041	0.9928	1.6039	1.0000	-20.8386	-1.2684	0.4063
MLP_PA	0.0000	0.8253	1.9065	0.0012	0.7938	1.5602	1.0000	-12.7120	-1.2101	0.4063
MLP_XX	0.0000	1.0932	1.3074	0.0055	0.8934	1.2556	0.9999	-17.1638	-0.5156	0.4375
RFR	0.0345	1.7580	1.7899	0.1371	1.2677	1.2428	0.9268	-28.2098	-1.0749	0.5313
RFR_PA	0.0390	1.3857	1.5928	0.1474	1.2015	1.3690	0.9172	-22.0234	-0.8464	0.5000
RFR_XX	0.0486	1.6856	1.6634	0.1571	1.3312	1.2886	0.8968	-27.0069	-0.9283	0.5000

Notes: All MSE and MAPE ratios have been normalized relative to the monthly no-change forecast. Boldface indicates predictive gains relative to the no-change forecast.

Moving to the 12-month ahead forecasting the case is even worst. Again, no model can be trusted for use in predicting oil price movements. Their performance is inferior compared to the no-change forecast. All of them have a coefficient of determination (R^2) less than zero, and this is an indication that the model's predictions cannot fit well with the test data. Their success ratio is also too small. At best, it gets close to 0.5, which is an indicator of random behaviour.

The MSE and MAPE metrics of those models also indicate that their performance is inferior. They are outperformed by the no-change forecast with the sole exception of the CNN model. This model has a lower MSE than the no-change forecast, but with no significant improvement. Its MAPE is still worse than the no-change forecast. Since its coefficient of determination (R^2) indicate a terrible fit on the test data and its accuracy ratio is lower than 0.5, this model cannot be the right candidate to forecast oil price movements either. Its mediocre performance on MSE alone is not strong enough evidence for a good model.

One thing we notice for both forecasting horizons is that many models were overfitting during the training. We can conclude to this by checking the rest of the columns of the results tables. We have two clues as to an indication of this behaviour. The first one is the training MSE column. When there is a value close or equal to zero in that column, it means that there was a perfect fit for the model on the training data. Otherwise, the error would be somewhat larger than zero. The training R^2 column can lead to the same deduction. In that column, we see many values close or equal to one. A value of one in the R^2 column means a perfect fit on the data used as well.

The issue with the overfitted models is that they cannot generalize well on test data. Those models cannot adapt to new signals from test data, and so they exhibit that lousy performance. Three reasons cause overfitting to a Deep Learning model.

The first reason is the lack of data. If training data are not enough, the model tends to fit with ease on the few samples, especially if there is not much diversity in them. Using more data forces the neural network to adapt to a more general position. Also, the more data there is, the more information available for the system under investigation.

The second reason is structural. It depends on the number of components used to implement the Neural Network. As an example, the case of an MLP built with many nodes will lead to overfitting since the most nodes it has, the easiest it is for it to memorize more data states.

The third reason is the number of epochs used to train the model. When using many epochs to train the model, it gets the opportunity to learn better on the data by the continuous repeat of them.

In our case, the primary reason for poor performance is the lack of training data. Having about 77 monthly samples available to train the models is not enough. For that reason, a simpler model could have performed better. During training, many tests were made using various epochs and node numbers to find a model that can perform decently. Those model parameters are the so-called hyperparameters. Properly configuring the hyperparameters demands more resources and time as already mentioned.

As it becomes clear, those Machine Learning models are outperformed from classic statistical methods and in most cases from the simple benchmark of the no-change forecast. Applying more hyperparameter tuning could lead to better models.

Finally, we present the models' outputs in the form of graphical plots. Those plots are presented in figures from Figure 12 to Figure 105. Four plots are created for each model.

The first plot puts in comparison the oil's actual price (y_{test}) along with the no-change forecast (no_change) and the model's prediction ($pred_y$). That type of plots helps us visually form a better understanding of the relation between the model's prediction and the actual oil price. In most cases, it makes clear how far worst those models perform compared to the no-change forecast.

The second plot draws the cumulative error through time. The error is calculated as the absolute difference between the actual oil price and the prediction. Two errors are presented in parallel. The first one is the no-change forecast error (no_change_error), and the second one is the model's prediction error ($pred_y_error$). Again, comparing those two plots visually, we see the effect the model's performance has. The expected value of a superior prediction should be about zero for all times, but this is not the case in our models' tests.

The third plot draws a box and whiskers representation of the error distribution. The error is calculated as the absolute difference between the actual oil price and the prediction. Again, we put in comparison the two errors, no-change forecast error (no_change_error), and the model's prediction error ($pred_y_error$). This plot also shows that the LSTM model is performing similar to the no-change forecast. All features drawn in the plot have about the same value. The expected plot of a superior

prediction should have all features (median, quartiles and whiskers) be about zero. Again this is not the case in our models' tests.

The fourth plot draws a box and whiskers representation of the cumulative error distribution. The error is calculated as the absolute difference between the actual oil price and the prediction. The two errors again are the no-change forecast error (`no_change_error`), and the model's prediction error (`pred_y_error`). This plot shows how errors evolve. We can again notice that the no-change forecast has smaller boxes and whiskers in all horizons. The expected plot of a superior prediction should also have all features (median, quartiles and whiskers) be about zero.

Some notable models using those plots are the MLP_PA and the CNN_PA for 12-months ahead forecasting horizon. Although they exhibit inferior performance, their box and whiskers plots show that their performance is superior to the rest of the models and close to the no-change forecast. Maybe there is some space for improvement there.

Figure 12. MLP 1-month forecasting

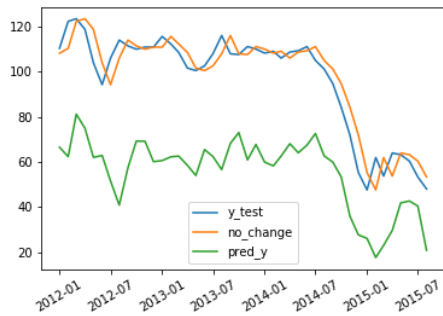


Figure 13. MLP 1-month forecasting cumulative error

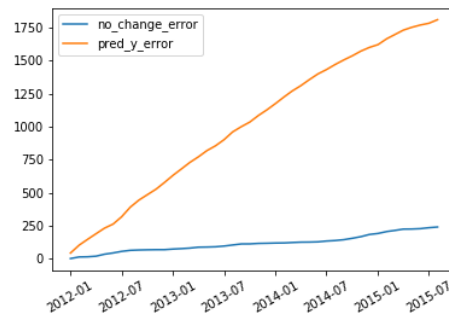


Figure 14. MLP 1-month forecasting error distribution

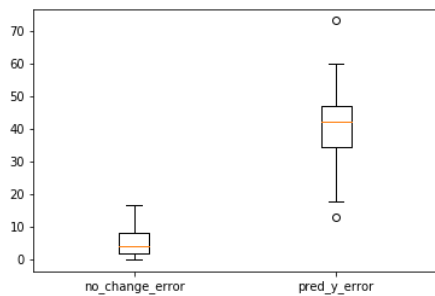


Figure 15. MLP 1-month forecasting cumulative error distribution

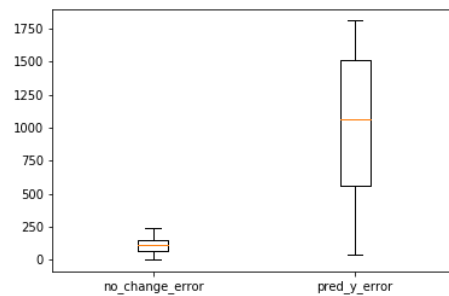


Figure 16. MLP 1-month forecasting using Palladium

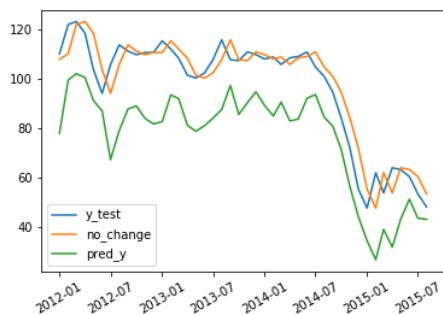


Figure 17. MLP 1-month forecasting cumulative error using Palladium

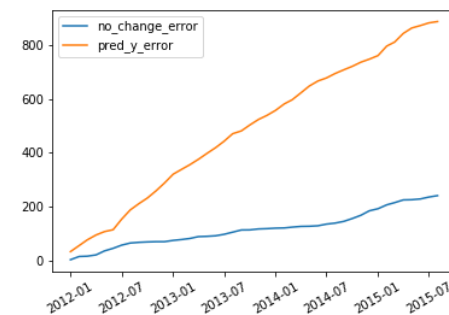


Figure 18. MLP 1-month forecasting error distribution using Palladium

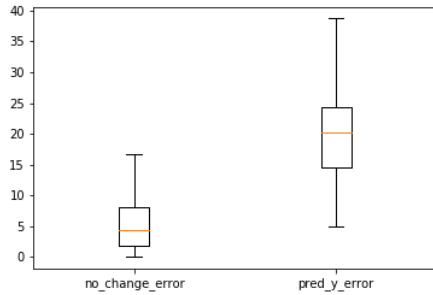


Figure 19. MLP 1-month forecasting cumulative error distribution using Palladium

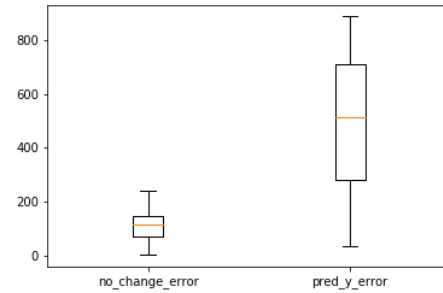


Figure 20. MLP 1-month forecasting error distribution using Eurostoxx 50

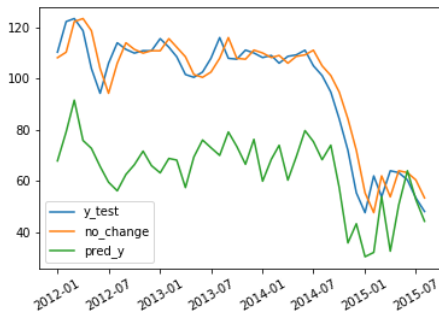


Figure 21. MLP 1-month forecasting cumulative error using Eurostoxx 50

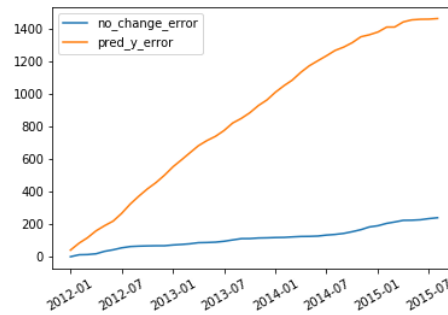


Figure 22. MLP 1-month forecasting error distribution using Eurostoxx 50

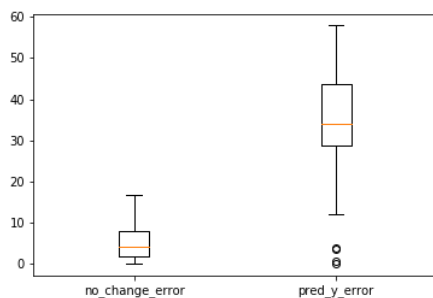


Figure 23. MLP 1-month forecasting cumulative error distribution using Eurostoxx 50

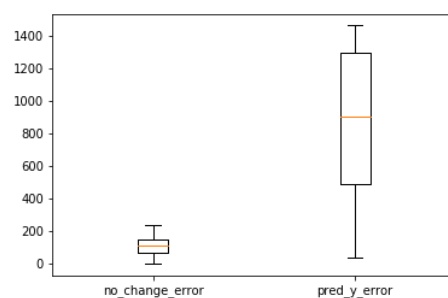


Figure 24. CNN 1-month forecasting



Figure 25. CNN 1-month forecasting cumulative error

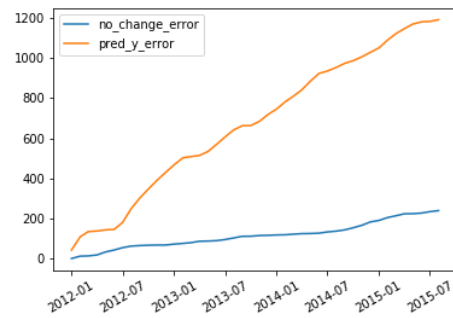


Figure 26. CNN 1-month forecasting error distribution

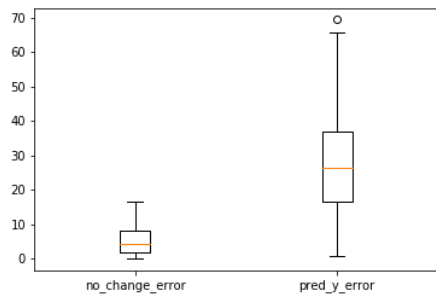


Figure 27. CNN 1-month forecasting cumulative error distribution

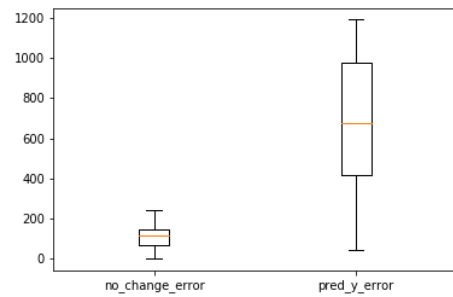


Figure 28. CNN 1-month forecasting using Palladium

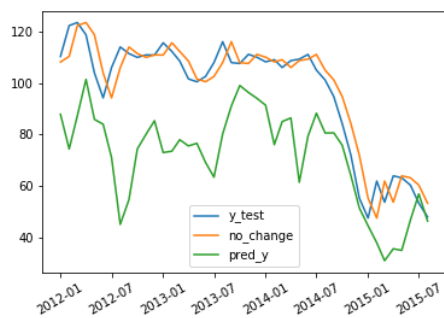


Figure 29. CNN 1-month forecasting cumulative error using Palladium

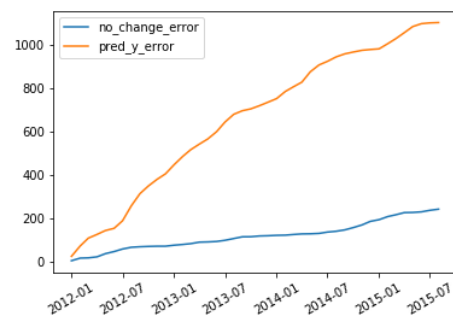


Figure 30. CNN 1-month forecasting error distribution using Palladium

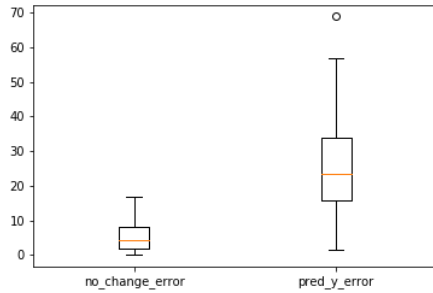


Figure 31. CNN 1-month forecasting cumulative error distribution using Palladium

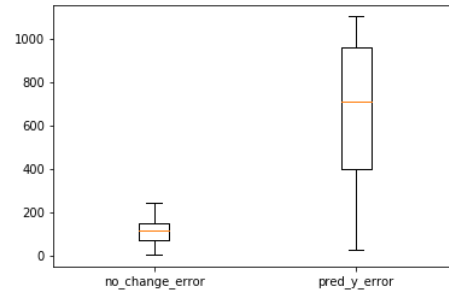


Figure 32. CNN 1-month forecasting error distribution using Eurostoxx 50



Figure 33. CNN 1-month forecasting cumulative error using Eurostoxx 50

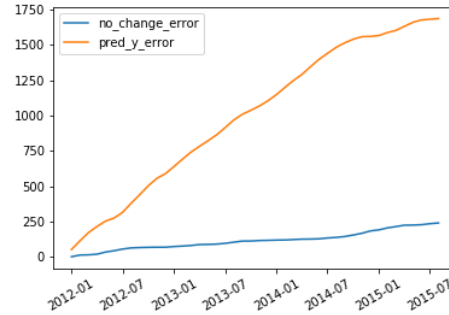


Figure 34. CNN 1-month forecasting error distribution using Eurostoxx 50

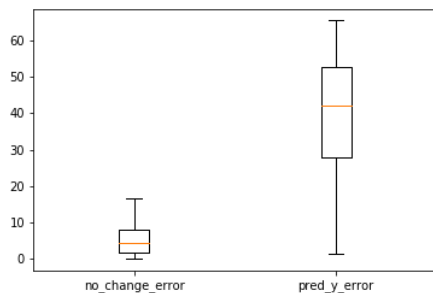


Figure 35. CNN 1-month forecasting cumulative error distribution using Eurostoxx 50

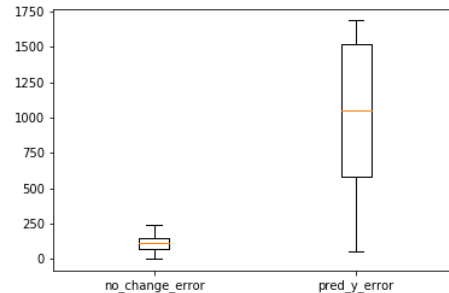


Figure 36. LSTM 1-month forecasting

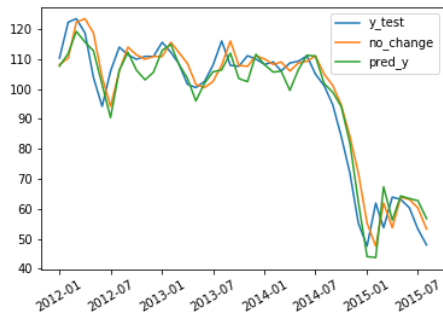


Figure 37. LSTM 1-month forecasting cumulative error

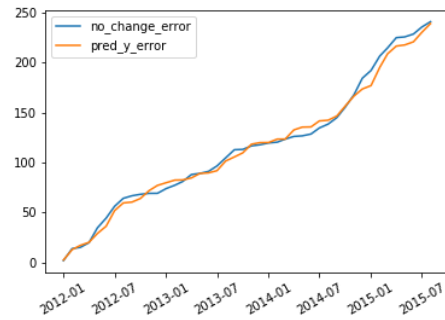


Figure 38. LSTM 1-month forecasting error distribution

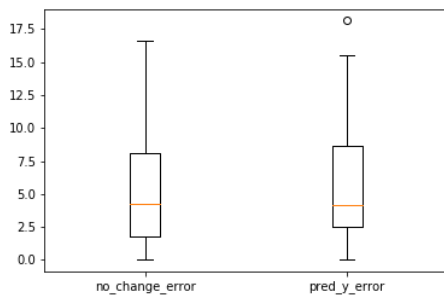


Figure 39. LSTM 1-month forecasting cumulative error distribution

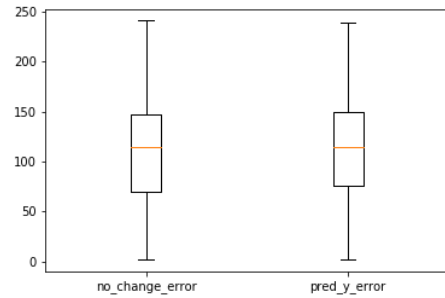


Figure 40. LSTM 1-month forecasting using Palladium



Figure 41. LSTM 1-month forecasting accumulative error using Palladium

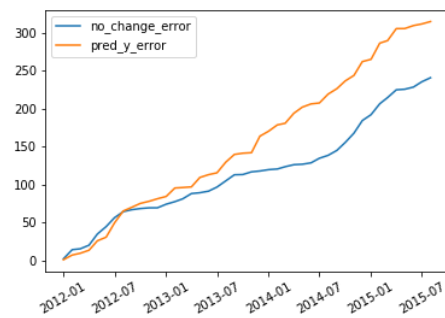


Figure 42. LSTM 1-month forecasting error distribution using Palladium

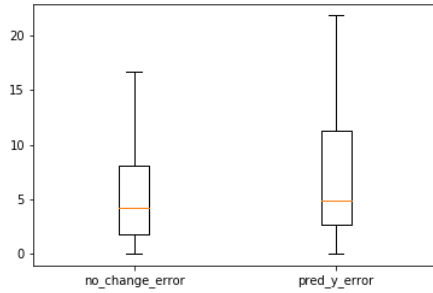


Figure 43. LSTM 1-month forecasting accumulative error distribution using Palladium

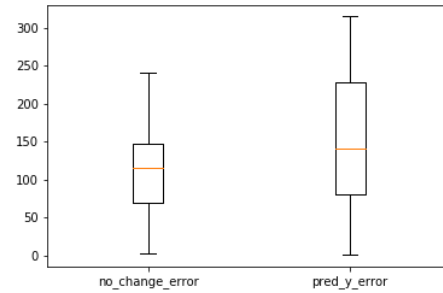


Figure 44. LSTM 1-month forecasting using Eurostoxx 50

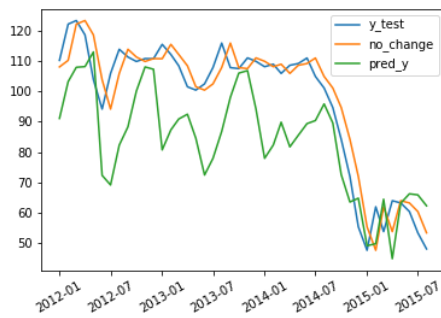


Figure 45. LSTM 1-month forecasting accumulative error using Eurostoxx 50

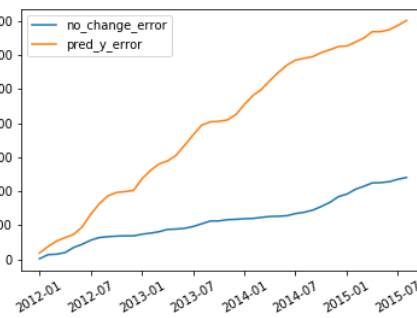


Figure 46. LSTM 1-month forecasting error distribution using Eurostoxx 50

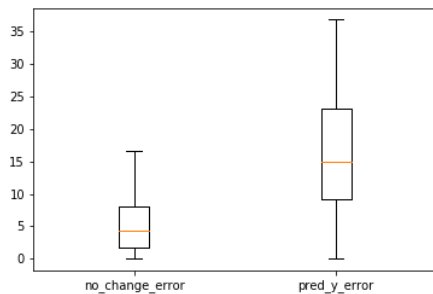


Figure 47. LSTM 1-month forecasting accumulative error distribution using Eurostoxx 50

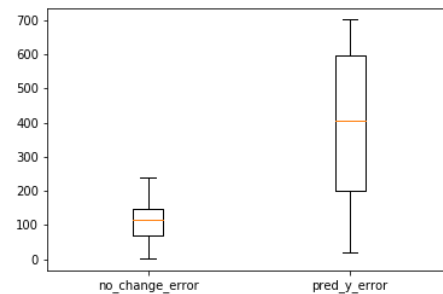


Figure 48. MLP 12-month forecasting

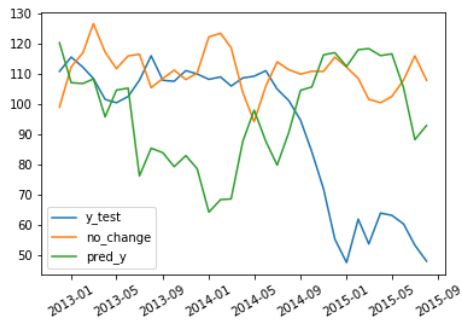


Figure 49. MLP 12-month forecasting accumulative error

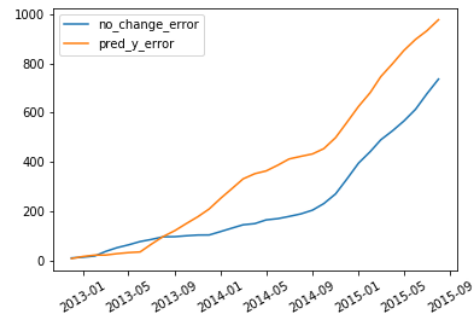


Figure 50. MLP 12-month forecasting error distribution

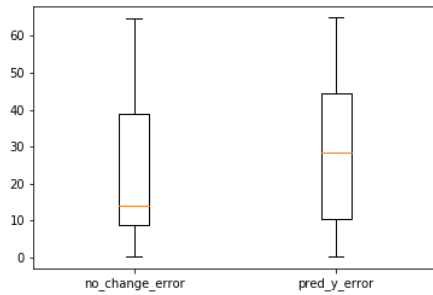


Figure 51. MLP 12-month forecasting accumulative error distribution

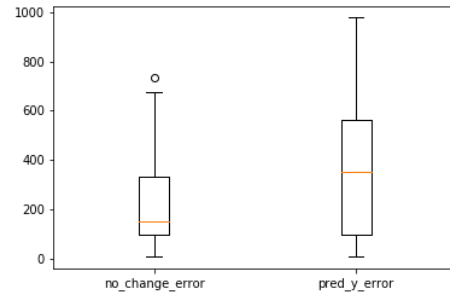


Figure 52. MLP 12-month forecasting using Palladium

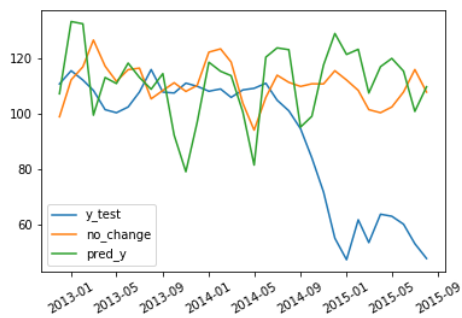


Figure 53. MLP 12-month forecasting accumulative error using Palladium

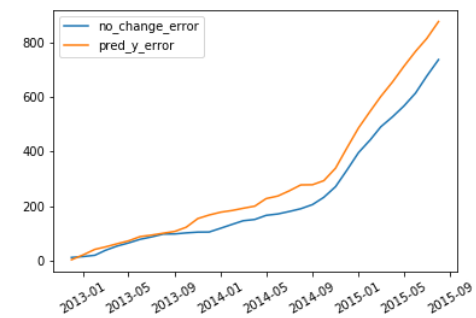


Figure 54. MLP 12-month forecasting error distribution using Palladium

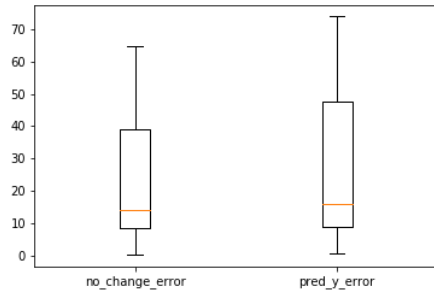


Figure 55. MLP 12-month forecasting accumulative error distribution using Palladium

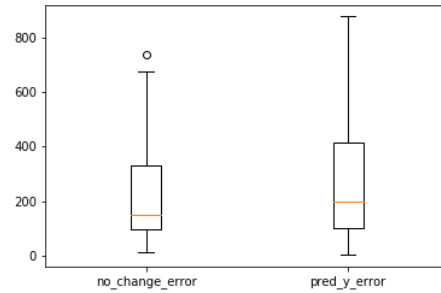


Figure 56. MLP 12-month forecasting error distribution using Eurostoxx 50

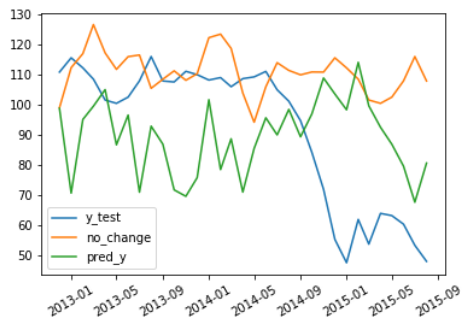


Figure 57. MLP 12-month forecasting accumulative error using Eurostoxx 50

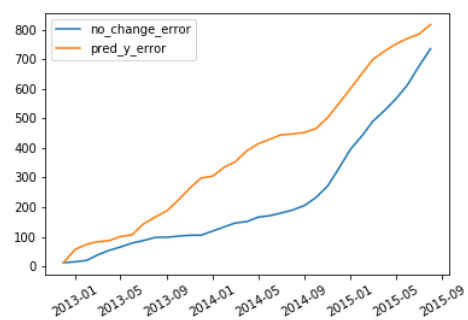


Figure 58. MLP 12-month forecasting error distribution using Eurostoxx 50

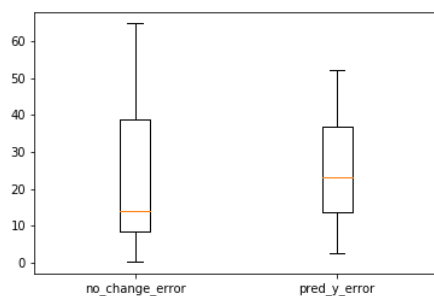


Figure 59. MLP 12-month forecasting accumulative error distribution using Eurostoxx 50

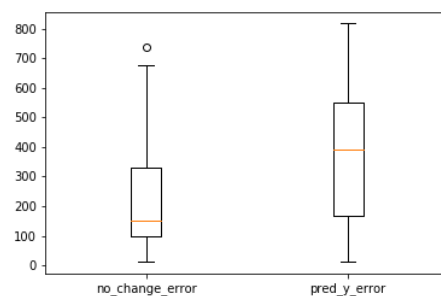


Figure 60. CNN 12-month forecasting

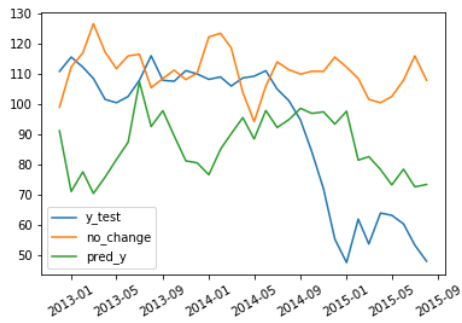


Figure 61. CNN 12-month forecasting accumulative error

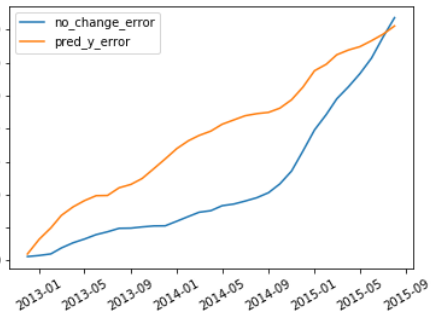


Figure 62. CNN 12-month forecasting error distribution

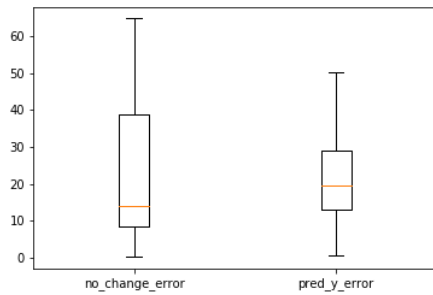


Figure 63. CNN 12-month forecasting accumulative error distribution

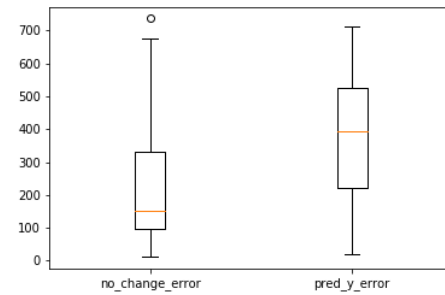


Figure 64. CNN 12-month forecasting using Palladium

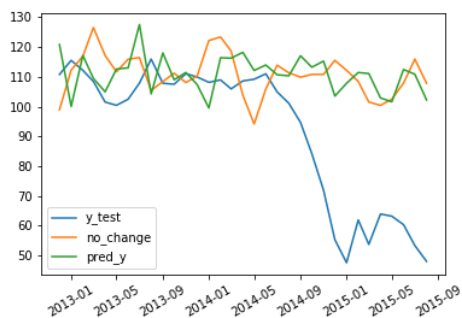


Figure 65. CNN 12-month forecasting accumulative error using Palladium

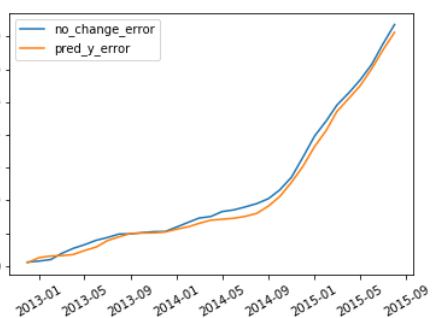


Figure 66. CNN 12-month forecasting error distribution using Palladium

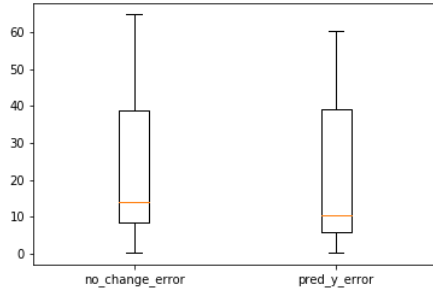


Figure 67. CNN 12-month forecasting accumulative error distribution using Palladium

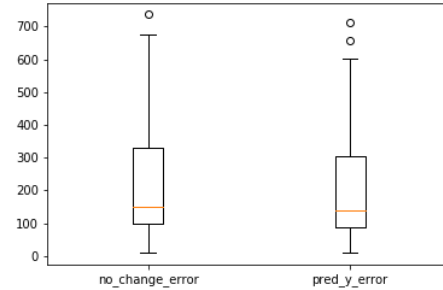


Figure 68. CNN 12-month forecasting error distribution using Eurostoxx 50

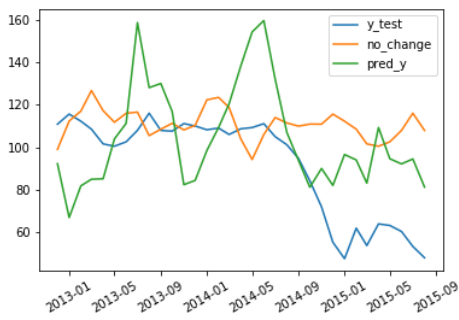


Figure 69. CNN 12-month forecasting accumulative error using Eurostoxx 50

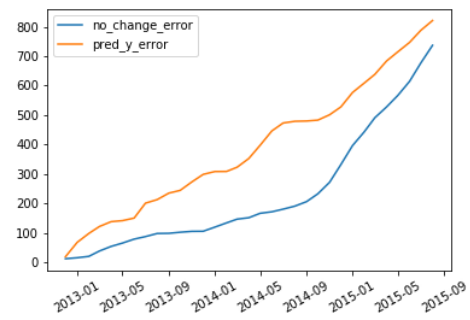


Figure 70. CNN 12-month forecasting error distribution using Eurostoxx 50

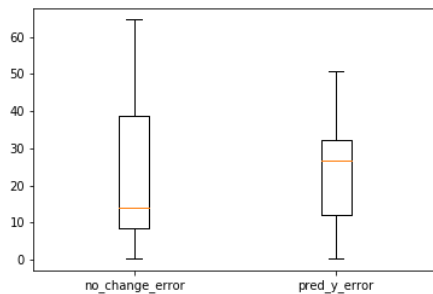


Figure 71. CNN 12-month forecasting accumulative error distribution using Eurostoxx 50

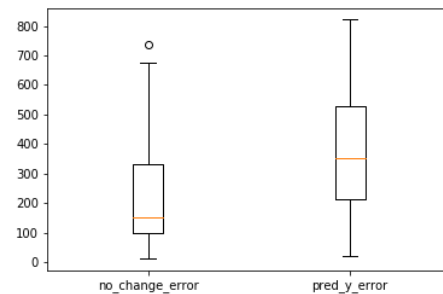


Figure 72. LSTM 12-month forecasting

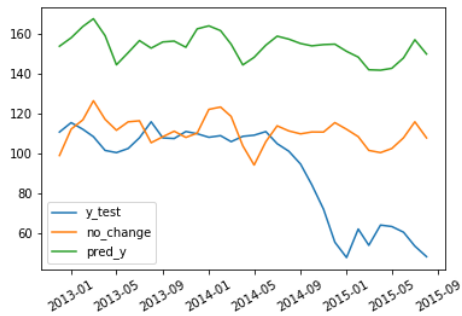


Figure 73. LSTM 12-month forecasting accumulative error

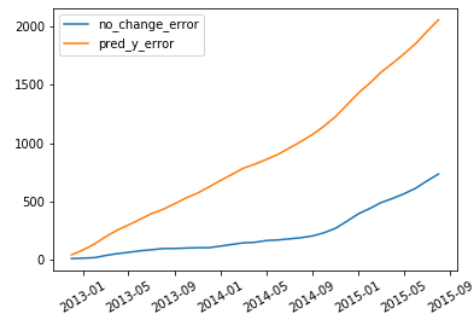


Figure 74. LSTM 12-month forecasting error distribution

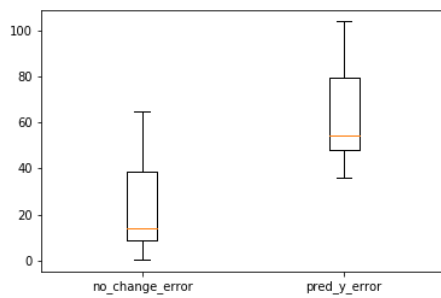


Figure 75. LSTM 12-month forecasting accumulative error distribution

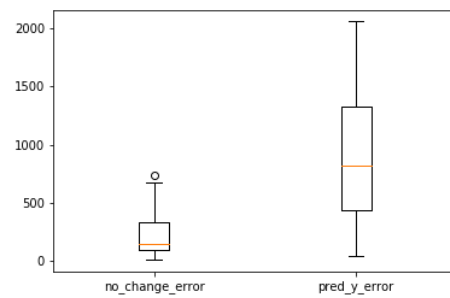


Figure 76. LSTM 12-month forecasting using Palladium

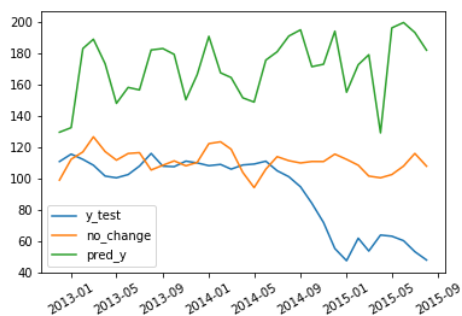


Figure 77. LSTM 12-month forecasting accumulative error using Palladium

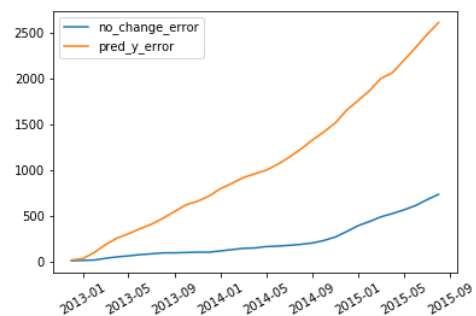


Figure 78. LSTM 12-month forecasting error distribution using Palladium

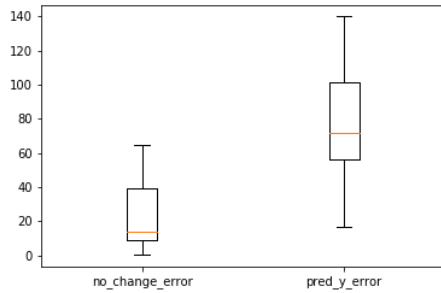


Figure 79. LSTM 12-month forecasting accumulative error distribution using Palladium

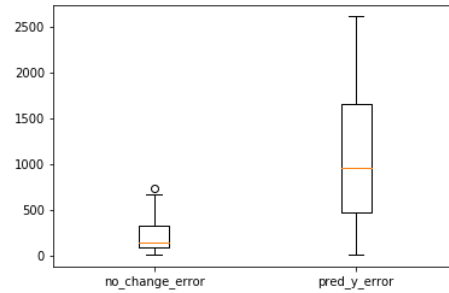


Figure 80. LSTM 12-month forecasting error distribution using Eurostoxx 50

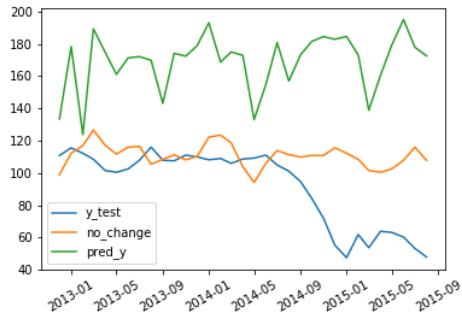


Figure 81. LSTM 12-month forecasting accumulative error using Eurostoxx 50

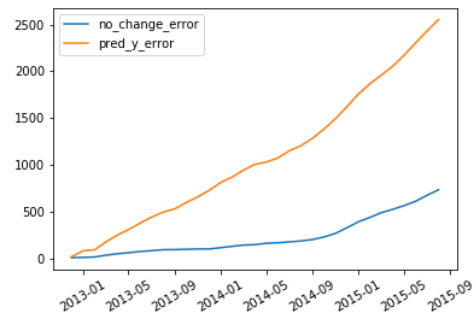


Figure 82. LSTM 12-month forecasting error distribution using Eurostoxx 50

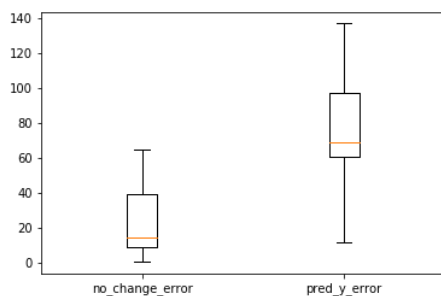


Figure 83. LSTM 12-month forecasting accumulative error distribution using Eurostoxx 50

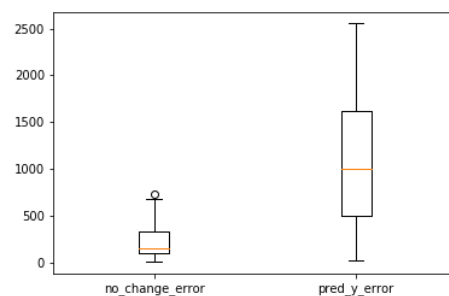


Figure 84. RFR 1-month forecasting

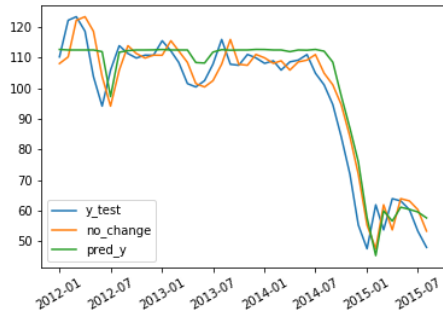


Figure 85. RFR 1-month forecasting accumulative error

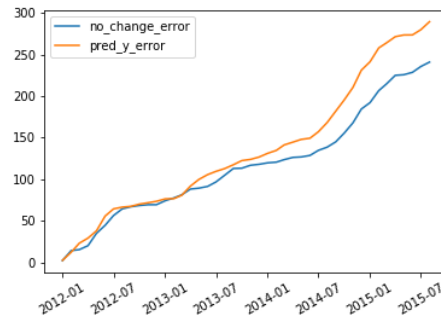


Figure 86. RFR 1-month forecasting error distribution

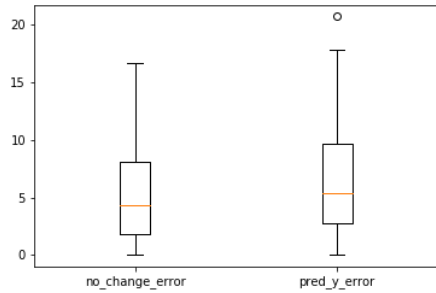


Figure 87. RFR 1-month forecasting accumulative error distribution

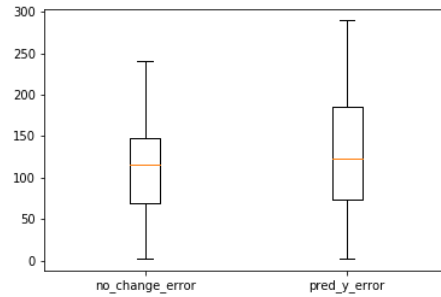


Figure 88. RFR 1-month forecasting using Palladium

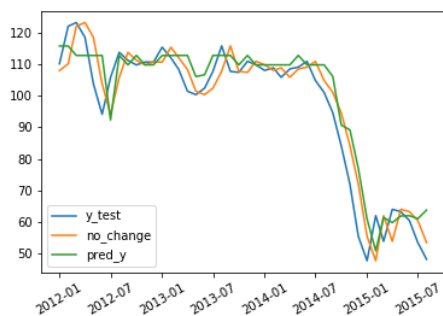


Figure 89. RFR 1-month forecasting accumulative error using Palladium

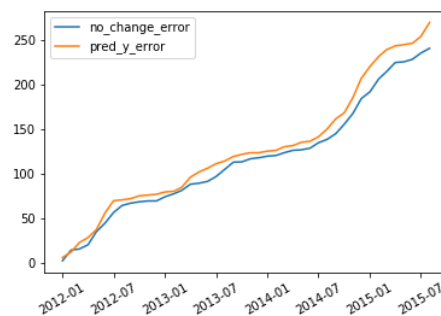


Figure 90. RFR 1-month forecasting error distribution using Palladium

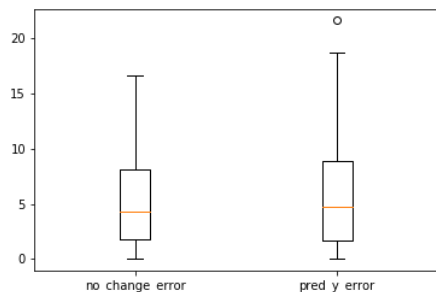


Figure 91. RFR 1-month forecasting accumulative error distribution using Palladium

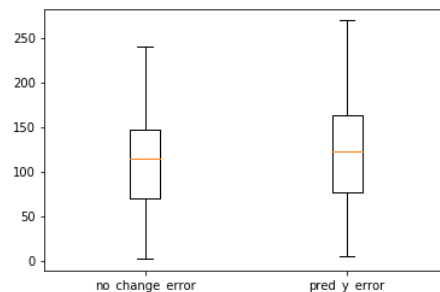


Figure 92. RFR 1-month forecasting error distribution using Eurostoxx 50



Figure 93. RFR 1-month forecasting accumulative error using Eurostoxx 50

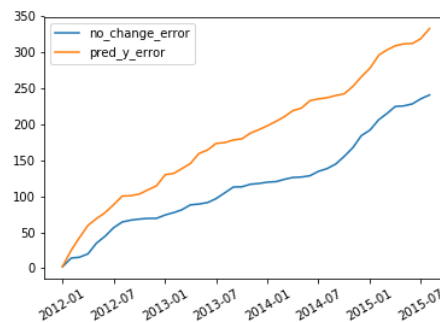


Figure 94. RFR 1-month forecasting error distribution using Eurostoxx 50

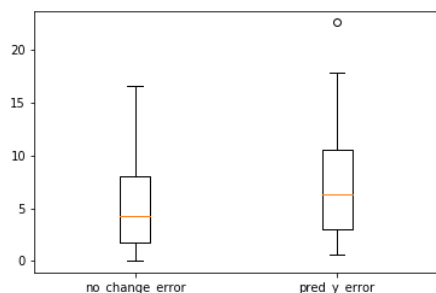


Figure 95. RFR 1-month forecasting accumulative error distribution using Eurostoxx 50

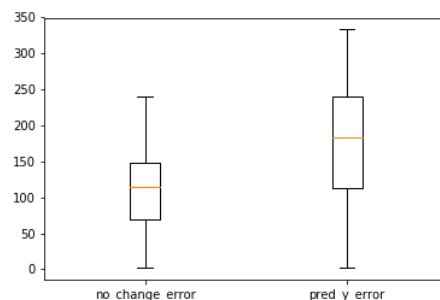


Figure 96. RFR 12-month forecasting

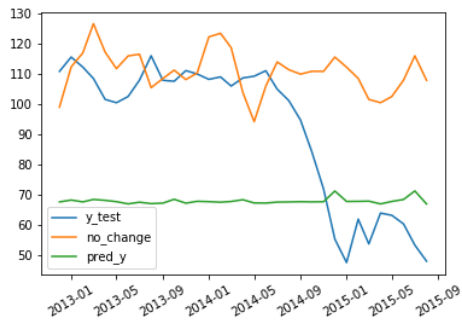


Figure 97. RFR 12-month forecasting accumulative error

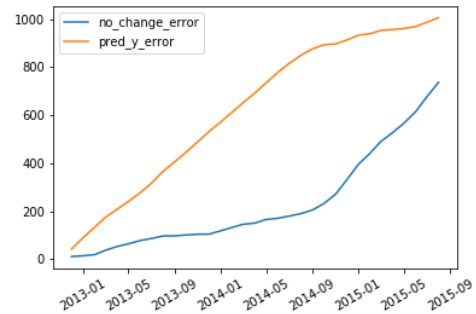


Figure 98. RFR 12-month forecasting error distribution

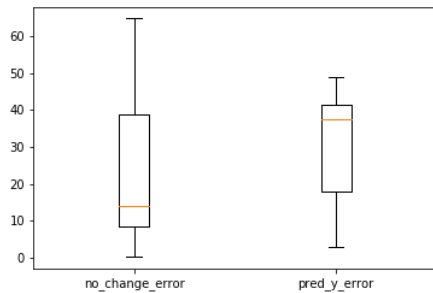


Figure 99. RFR 12-month forecasting accumulative error distribution

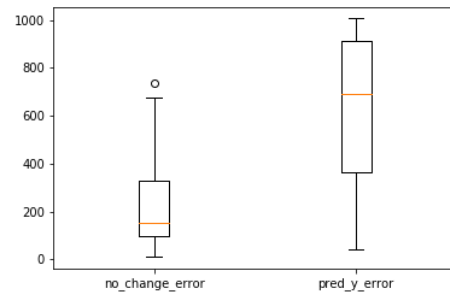


Figure 100. RFR 12-month forecasting using Palladium

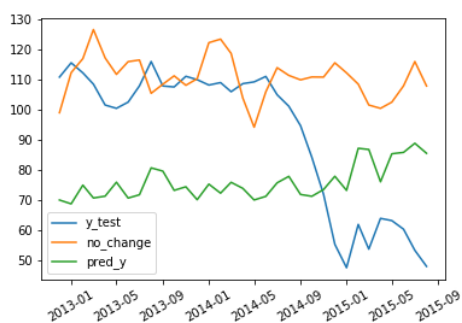


Figure 101. RFR 12-month forecasting accumulative error using Palladium

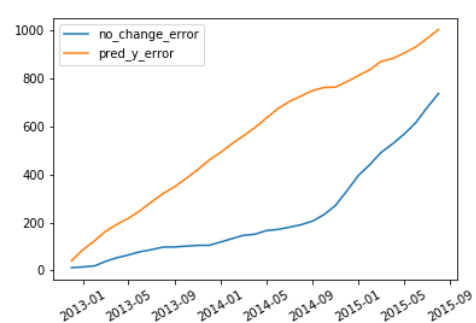


Figure 102. RFR 12-month forecasting error distribution using Palladium

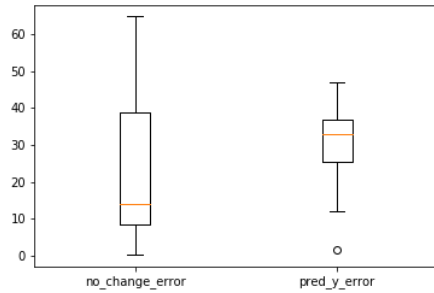


Figure 103. RFR 12-month forecasting accumulative error distribution using Palladium

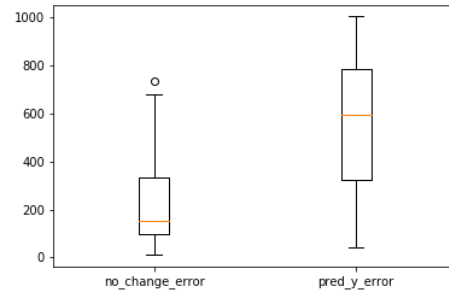


Figure 104. RFR 12-month forecasting error distribution using Eurostoxx 50

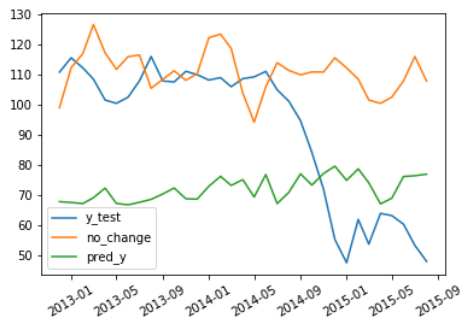


Figure 105. RFR 12-month forecasting accumulative error using Eurostoxx 50

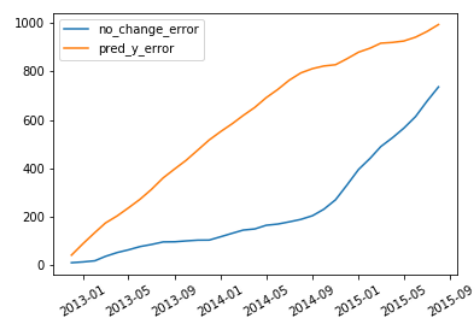


Figure 106. RFR 12-month forecasting error distribution using Eurostoxx 50

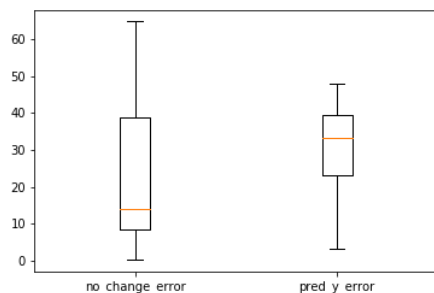
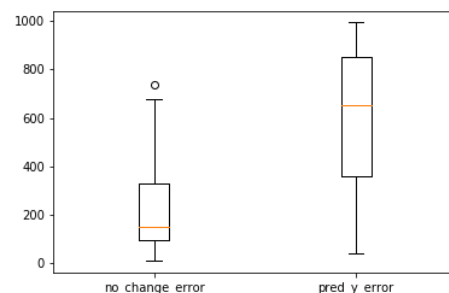


Figure 107. RFR 12-month forecasting accumulative error distribution using Eurostoxx 50



Conclusions

Machine Learning models failed to forecast oil price movements. It seems they are not able to adapt to such a demanding multivariate time-series problem. This finding comes inline with the extant literature and provides evidence that Machine Learning models are not appropriate for time-series problems in general. Classical statistic methods, as well as basic benchmark methods like the no-change forecast, can outperform Machine Learning models. There are many reasons for this, and work needs to be done to find a proper model.

Machine Learning models and Deep Learning models are dynamic non-linear mechanisms. They depend on many hyperparameters in order to be appropriately configured for the problem at hand. This process requires a lot of tests, resources and time, and the scarcity of those can lead to poorly defined and configured models. The results are prominent. In our case, only one model managed to come close to the no-change forecast, which in its own right is considered to exhibit poor performance, while the rest of the models showed even worse adaptability.

Another reason is the lack of data. Problems like the one in forecasting oil price movements are quite demanding with lots of interdependencies between assets. The underlying relationships are not always transparent. To overcome those issues, Machine Learning models require lots of data so they can train and recognize the hidden patterns. It is known that the more data available, the better the ML model will be trained and the better it will behave when it is put into action.

As Makridakis et al. (2018) state, time-series problems are not easy to handle. Unlike other problems like games which have a set of clearly defined rules, in time series problems, usually this is not the case. Many factors can influence the behaviour of the system under investigation, like in the case of the oil price collapse. In problems like that, there are not many fixed rules that Machine Learning models can learn.

Machine Learning applications exhibit superior performance in other sectors where the environment is stable, and the predictions cannot influence the future itself (Makridakis et al., 2018). Financial markets are known to create expectations, and those expectations can turn on their heads as easy, creating this way a precarious and insecure environment. An environment that Machine Learning algorithms, as well as classical statistic methods, have a hard time to work out.

The results of this study indicate that for the time being, there is not much place in time-series problems for Machine Learning models applications. Their poor performance indicates that more work needs to be done before we claim Machine Learning models can adapt to a time-series problem.

Future studies can work on improving the data used to train the models. We used only a limited amount of the data known from the extant literature. Better data can guarantee better performance. Other than using more sources of data, the data themselves can be improved as well. It is believed that there is no need to prepare the data, like taking care of seasonality or trend, before feeding them into a Neural Network or other algorithms, but this is something to investigate in future studies. Still, some light needs to be shed on the quality of data needed to train a Machine Learning algorithm properly.

Besides data, the algorithms themselves need improvement. We just started realizing the potential, and the abilities Machine Learning applications have. In the case of Deep Learning, the models are quite flexible, and one can use various setups to address the problem. It is up to the researcher to choose the number of nodes and the connections used on each neural network. More sophisticated neural networks will show completely different behaviour. Sometimes though, a sophisticated neural network might exhibit overfitting tendencies. A simpler model might be more appropriate.

Future is hard to be predicted, but Machine Learning has still a say on this.

Πηγές – Βιβλιογραφία

A' Ιστοσελίδες

- Büyükşahin, B., Ellwanger, R., Mo, K., & Zmitrowicz, K. (2016). *Low for Longer? Why the Global Oil Market in 2014 Is Not Like 1986*. Retrieved from <https://www.bankofcanada.ca/wp-content/uploads/2016/07/san2016-11.pdf>
- EIA. (2014). *Benchmarks play an important role in pricing crude oil*. Retrieved from <https://www.eia.gov/todayinenergy/detail.php?id=18571>
- EIA. (2020a). *SHORT-TERM ENERGY OUTLOOK – Crude Oil*. Retrieved from <https://www.eia.gov/outlooks/steo/marketreview/crude.php>
- EIA. (2020b). *WHAT DRIVES CRUDE OIL PRICES? An analysis of 7 factors that influence oil markets, with chart data updated monthly and quarterly*. Retrieved from https://www.eia.gov/finance/markets/crudeoil/financial_markets.php
- Fratzscher, M., Schneider, D., Van Robays, I. (2019). *Oil Prices, Exchange Rates and Asset Prices. ECB Working Paper 189*. Retrieved from <https://www.econstor.eu/handle/10419/154122>
- ICE. (2020). *Brent*. Retrieved from <https://www.theice.com/brent-crude>
- Investing.com. (2020a). *Brent oil futures historical prices*. Retrieved from <https://www.investing.com/commodities/brent-oil-historical-data>
- Investing.com. (2020b). *Crude Oil WTI oil futures historical prices*. Retrieved from <https://www.investing.com/commodities/crude-oil-historical-data>
- OPEC. (2020). *The 10th (Extraordinary) OPEC and non-OPEC Ministerial Meeting concludes*. Retrieved from https://www.opec.org/opec_web/en/press_room/5891.htm

B' Βιβλιογραφία

- Akram, Q. F. (2009). Commodity prices, interest rates and the dollar. *Energy Economics*, 31(6), 838-851.
- Alpaydin, E. (2004). *Introduction to Machine Learning*. Cambridge, MA: MIT Press.

- Baffes, J. (2007). Oil spills on other commodities. *Resources Policy*, 32(3), 126–134.
doi:10.1016/j.resourpol.2007.08.004
- Balcilar, M., Ozdemir, Z. A., & Ozdemir H. (2019). Dynamic return and volatility spillovers among S&P 500, crude oil, and gold. *International Journal of Finance & Economics*. 1–18. doi:10.1002/ijfe.1782
- Breiman, L. (1996). Bagging predictors. *Machine Learning*, 26(2), 123–140.
- Breiman, L. (2001). Random Forests. *Machine Learning*, 45, 5–32.
- Büyükkşahin, B., & Robe, M. A. (2014). Speculators, commodities and cross-market linkages. *Journal of International Money and Finance*, 42, 38-70.
- Chen, J., Zhu, X., & Zhong, M. (2019). Non-linear effects of financial factors on fluctuations in nonferrous metals prices: A Markov-switching VAR analysis. *Resources Policy*, 61(April), 489–500.
<https://doi.org/10.1016/j.resourpol.2018.04.015>
- Coppola, A., 2008. Forecasting oil price movements: exploiting the information in the futures market. *The Journal of Futures Markets*, 28 (1), 34–56.
doi:10.1002/fut.20277
- Degiannakis, S., & Filis, G. (2018) Forecasting oil prices: High-frequency financial data are indeed useful. *Energy Economics*, 76, 388-402.
doi:10.1016/j.eneco.2018.10.026
- Dietterich, T. (1998). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization, *Machine Learning*, 1–22.
- Galton, F. (1907). Vox Populi. *Nature*, 75, 450–451.
<https://doi.org/10.1038/075450a0>
- Gers, F. A., Eck, D., & Schmidhuber, J. (2001). Applying LSTM to Time Series Predictable through Time-Window Approaches. *Lecture Notes in Computer Science*, 669–676. doi:10.1007/3-540-44668-0_93

- Gers, F. A., Schmidhuber, j. & Cummins, F. (1999). *Learning to forget: continual prediction with LSTM*. 9th International Conference on Artificial Neural Networks: ICANN '99
- Hamilton, J. D. (2009a). Causes and Consequences of the Oil Shock of 2007–08. *Brookings Papers on Economic Activity*
- Hamilton, J. D. (2009b). Understanding Crude Oil Prices. *The Energy Journal*, 30(2), 179-206
- Hochreiter S. & Schmidhuber j. (1997). Long short-term memory, *Neural Computation*, 9(8), 1735–1780.
- Kaufmann, R. K. (2011). The role of market fundamentals and speculation in recent price changes for crude oil. *Energy Policy*, 39(1), 105–115.
doi:10.1016/j.enpol.2010.09.018
- Kilian, L. (2009). Not All Oil Price Shocks are Alike: Disentangling Demand and Supply Shocks in the Crude Oil Market? *American Economic Review*, 99 (3), 1053-1069.
- Lecun, Y. (1989). Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, & L. Steels (Eds.), *Connectionism in perspective*. Elsevier.
- Makridakis, S., Spiliotis. E. & Assimakopoulos, V. (2018). Statistical and Machine Learning forecasting methods: Concerns and ways forward. *PLoS ONE*, 13(3).
doi:10.1371/journal.pone.0194889
- Murat, A., & Tokat, E. (2009). Forecasting oil price movements with crack spread futures. *Energy Economics*, 31(1), 85-90.
- Nesreen K. A., Amir F. A., Neamat E. G., & Hisham E. S. (2010). An Empirical Comparison of Machine Learning Models for Time Series Forecasting, *Econometric Reviews*, 29(5-6), 594-621. doi:10.1080/07474938.2010.481556
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, j., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay E. (2011), Scikit-

- learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830
- Razek, N. H. A., & Michieka, N. M. (2019). OPEC and non-OPEC production, global demand, and the financialization of oil. *Research in International Business and Finance*, 50, 201–225. doi:10.1016/j.ribaf.2019.05.009
- Reboredo, J. C., & Ugolini, A. (2016). The impact of downward/upward oil price movements on metal prices. *Resources Policy*, 49, 129–141. doi:10.1016/j.resourpol.2016.05.006
- Rosenblatt, F. (1957). *The Perceptron: A Perceiving and Recognizing Automaton, Report 85-460-1*. Buffalo: NY, Cornell Aeronautical Laboratory
- Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning internal representations by backpropagating errors. In D. Rumelhart and J. McClelland (Eds.), *parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 1, pp. 318–362). MIT Press
- Sharma, S., & Rodriguez, I. (2019). The diminishing hedging role of crude oil: Evidence from time varying financialization. *Journal of Multinational Financial Management*, 52–53. doi:10.1016/j.mulfin.2019.100593
- Tsantekidis, A., Passalis, N., Tefas, A., Kannianen, A., Gabbouj, A., & Iosifidis, A. (July 2017). *Forecasting Stock Prices from the Limit Order Book Using Convolutional Neural Networks*. 2017 IEEE 19th Conference on Business Informatics (CBI). Thessaloniki, Greece: IEEE: 7-12. doi:10.1109/CBI.2017.23
- Xu, Y., Han, L., Wan, L., & Yin, L. (2019). Dynamic link between oil prices and exchange rates: A non-linear approach. *Energy Economics*, 84. doi:10.1016/j.eneco.2019.104488

Παράρτημα

To create the models, Python was used along with the libraries numpy, pandas, matplotlib, keras, and others. Here we provide some sample code that creates, trains and tests some basic deep learning models as a starting point for someone who needs to delve into deep learning.

```
from numpy import array
from keras.models import Sequential
from keras.models import Model
from keras.layers import Dense
from keras.layers import Input
from keras.layers import Flatten
from keras.layers import LSTM
from keras.layers.merge import concatenate
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D

import pandas as pd
import numpy as np
import datetime

%matplotlib inline
import matplotlib.pyplot as plt

import pickle

models_to_train = {
    "model1": {
        "filename": "model_mlp",
        "type": "MLP",
    },
    "model3": {
        "filename": "model_cnn",
        "type": "CNN",
    },
    "model5": {
        "filename": "model_lstm",
        "type": "LSTM",
    },
}

def split_train_valid_test(X, Y, test_count, validation_split=0):
    if validation_split < 0:
        validation_split = 0
    elif validation_split > 1:
        validation_split = 1

    x_train = X[: test_count]
    y_train = Y[: test_count]

    x_valid = []
    y_valid = []

    if validation_split > 0:
```

```

    test_split_count = int(test_count * (1 - validation_split))

    x_valid = x_train[test_split_count:]
    y_valid = y_train[test_split_count:]

    x_train = x_train[:test_split_count]
    y_train = y_train[:test_split_count]

    x_test = X[test_count:]
    y_test = Y[test_count:]

    return x_train, y_train, x_valid, y_valid, x_test, y_test

def split_sequences(sequences, n_steps, n_step_out = 1):
    if n_step_out < 1:
        n_step_out = 1

    X, y = list(), list()

    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        out_pos = end_ix + n_step_out - 1

        # check if we are beyond the dataset
        if end_ix > len(sequences):
            break

        # check if we have an output in the dataset
        if out_pos > len(sequences) - 1:
            break

        # gather input and output parts of the pattern
        seq_x = sequences[i:end_ix, :]
        seq_y = sequences[out_pos, 0]

        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

def get_oil_data(n_timesteps, n_outputsteps):
    # Read data from file
    oildata = pd.read_csv('../dataset/oil.csv',
        parse_dates=['_date_'])

    oildata["date"] =
    pd.to_datetime(oildata["_date_"]).dt.strftime("%Y%m%d").astype(int)

    # select columns and create array
    data = array(oildata[['oil', 'cap', 'prod', 'stocks', 'gea']])

    # split into X, Y with delays
    X, Y = split_sequences(data, n_timesteps, n_outputsteps)

    dates_x = np.array(oildata['_date_'][n_timesteps-1:-
n_outputsteps])

    dates_y = np.array(oildata['_date_'][-len(Y):])

    return X, Y, dates_x, dates_y

```

```

def count_train_data(dates, limit_date):
    if type(limit_date) == datetime.datetime:
        limit_date = np.datetime64(limit_date)

    return len(dates[dates < limit_date])

def flatten_array(X):
    ret_list = list()

    if type(X) == list or type(X) == tuple:
        for item in X:
            if type(item) == list or type(item) == tuple:
                print('enclosed')
                enclosed_list = flatten_array(item)
                ret_list.append(enclosed_list)

            elif type(item) == np.ndarray:
                if len(item.shape) == 3:
                    n, re_item = flatten_input_vectors(item)
                    ret_list.append(re_item)

                else:
                    ret_list.append(item)

            else:
                ret_list.append(item)

        return ret_list

    elif type(X) == np.ndarray:
        if len(X.shape) == 3:
            n, re_item = flatten_input_vectors(X)
            return re_item

        else:
            return X

    else:
        print('\x1b[31m' + 'SoftAPIError: Unknown type to flatten' +
              '\x1b[0m')
        return X

def flatten_input_vectors(data):
    sh = data.shape

    if len(sh) == 2:
        return data.shape[1], data

    elif len(sh) == 3:
        n_input = data.shape[1] * data.shape[2]
        return n_input, data.reshape((data.shape[0], n_input))

    else:
        raise ValueError("Wrong shape. It should be of length 2 or 3
")

def create_model(model_type, steps, features):
    if model_type == "MLP":
        return model_mlp_mv(steps * features)
    elif model_type == "CNN":

```

```

        return model_cnn_mv(steps, features)
    elif model_type == "LSTM":
        return model_lstm_mv (steps, features)
    else:
        raise ValueError(f'Wrong model type: "{model_type}"')

def model_mlp_mv(n_input):
    model = Sequential()
    model.add(Dense(10, activation='relu', input_dim = n_input))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')

    return model

def model_cnn_mv(n_steps, n_features):

    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=2, activation='relu' ,
input_shape=(n_steps, n_features)))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(10, activation='relu' ))
    model.add(Dense(1))
    model.compile(optimizer='adam' , loss='mse' )

    return model

def model_lstm_mv(n_steps, n_features):
    model = Sequential()
    model.add(LSTM(10, activation='relu' , input_shape=(n_steps,
n_features)))
    model.add(Dense(1))
    model.compile(optimizer='adam' , loss='mse' )

    return model

def show_graph(dates, lines, show=True, save=False, filepath=''):
    fig, ax = plt.subplots()

    for k, v in lines.items():
        ax.plot(dates, v, label=k)

    ax.legend()

    plt.xticks(rotation=30)

    if show:
        plt.show()

    if save:
        plt.savefig(filepath)

    plt.close()

def train_all():
    for key in models_to_train:
        model_train = models_to_train[key]

        filename = model_train['filename']
        model_type = model_train['type']
        n_steps = 12

```

```

n_features = 5
forecast_horizon = 1
n_epochs = 100

aX, aY, aDx, aDy = get_oil_data(n_steps, forecast_horizon)

# Split the data
x_train, y_train, x_valid, y_valid, x_test, y_test \
    = split_train_valid_test(aX, aY, count_train_data(aDx,
datetime.datetime(2011, 12, 1)), 0.2)

    if model_type == "MLP":
        train_data, validation_data, test_data =
flatten_array([x_train, x_valid, x_test])
    elif model_type == "CNN" or model_type == "LSTM":
        train_data, validation_data, test_data = [x_train,
x_valid, x_test]

# create a model to train
model = create_model(model_type, n_steps, n_features)

# Fit the model
# validate on validation data
print(f'=== Training model -> {key} ===')
ini_time_for_model = datetime.datetime.now()
model.fit(train_data, y_train, epochs=n_epochs, verbose=1,
validation_data=(validation_data, y_valid))

#calculate run time
run_time = datetime.datetime.now() - ini_time_for_model
print (f'Run time: {run_time}')

# save the model for latter use
model_filepath = f'{filename}.pkl'
pickle.dump(model, open(model_filepath, 'wb'))

# Predict on test data in order to evaluate the models
performance
y = model.predict(test_data)
pred_y = y.reshape(y.shape[0])

# Save the graph
img_filepath = f'{filename}.png'
show_graph(aDy[-len(y_test):], {"y_test": y_test, "pred_y":
pred_y}, show=False, save=True, filepath=img_filepath)

train_all()

```