UCLouvain

LOUVAIN
School of Management

Louvain School of Management

Predicting the ranking of the sales surprises in the context of active investing

Author : Romain Henneton
Supervisor : Corentin Vande Kerckhove
Year 2021-2022
Master's thesis (TFE) in order to obtain the title of
Master (60) in management
(Shift schedule)

Abstract

Quantitative investing refers to the action of trying to optimize the return of an actively managed portfolio on the stock market. It has been shown that using pure machine learning models to forecast the return of stock prices directly is a very hard task. Different mechanism have been tried to circumvent that issue. This thesis focuses on the forecasting of fundamental values. More specifically, we are forecasting the surprise on the consensus of a given fundamental value. Indeed, it has been shown that this surprise can be used to create high returns investment strategies. A few papers look at forecasting the surprise directly, even if what matters in the end in the investing heuristic that we use is the ranking of several surprises on a set of companies. Therefore, in this thesis, we try to use ranking-specific objective functions for machine learning models that try to rank the surprises on a set of given companies. We show that some improvement on the metric can be obtained using this strategy, leading to potential higher returns on the portfolio.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1602, the Amsterdam Stock Exchange was founded. Along with the creation of the Dutch East India Company (VOC). It is considered the oldest, still-functioning stock exchange in the world. The main reason for the creation of this stock comes from the fact that sailing across the sea was too risky for any single company. Using the still famous “do not put all your eggs in one basket” (ship in our case), investors could buy a share of the company that would then send several ships. Even if one ship was lost, thanks to the average success of the expeditions and the pooling of investments, the investor would still make a profit.

Nowadays, thousands of companies are listed across stock markets around the world, allowing to get funding and investor’s capital. The price of those stocks fluctuates and investors try to buy the ones that will grow the most and yield the highest return.

Studies have been made to show that forecasting the stock market return directly is a very complex task (Survey of stock market forecasting methods (1)). Prior to detailing our method, we define a few concepts that will help us understand the complete process.

1.1 Fundamental values

As defined in this article (2), *stock's fundamentals are the factors that are thought to contribute to the underlying company's value or worth as a business.* Those fundamentals are what drives the valuation of a company up or down, hence its market return. Therefore, if we can forecast those fundamentals early enough, we could estimate the return of the stock and decide whether it makes sense or not to long (bet that the stock price will increase) or short (bet that the price will decrease) the stock to maximise the profit on the best and worst performers. Financial institutions forecast those fundamentals and the average of those forecasts is called the consensus.

In 2000, *Joseph D. Piotroski* made a statistical analysis to show (3) that *the market does not fully incorporate historical financial information into prices in a timely manner*. Using that fundamental values information allows to get a better return than using a strategy that does not incorporate those fundamental values into its investment heuristic.

In 2017, *Alberg and Lipton* start to investigate the potential of forecasting those fundamental values into the future (rather than using only the available ones). They show that using an oracle[1] allows to outperform the market (4) by a huge margin. Then, they compare different machine learning model performance regarding the forecasting of those fundamental values. This 2017 article from *Alberg and Lipton* is a machine-learning-based extension of the work that has been published in 2010 and 2011 by *James M. Wahlen and Matthew M. Wieland* where the authors beat the consensus by using statistical models and therefore obtain higher returns than the ones obtained by using the consensus information (5) (6).

In this thesis, we will focus on experimenting on the fundamental value *Revenue per quarter Year over Year growth*. We look at year over year growth rather than Quarter over Quarter since this allows to remove the season peaks and lows that impact the company revenues. As

[1] An oracle means that we can simulate the fact that we would have the perfect information about the fundamentals X months prior to that fundamental value being available.

an example, if we look at the figure 1.1, which shows the revenues per quarter of the company Amazon, we can see that the Q4 revenues (that include Black Friday and Christmas) is always higher than the other quarters. Therefore, comparing the Q4 revenues with the Q3 or Q1 revenues makes less sense than comparing the evolution of Q4 revenues year over year in order to assess the growth of the company. If the yearly revenues define the valuation of a company, its growth is a proxy of the stock growth (7). Several financial institutions are producing forecast around that year on year growth, and the average of those forecasts is called *the consensus on revenues growth*. The stock price is influenced by the various consensus values that are updated throughout the year. If the revenue consensus goes up, the market valuation in the future is expected to go up (8)(9).

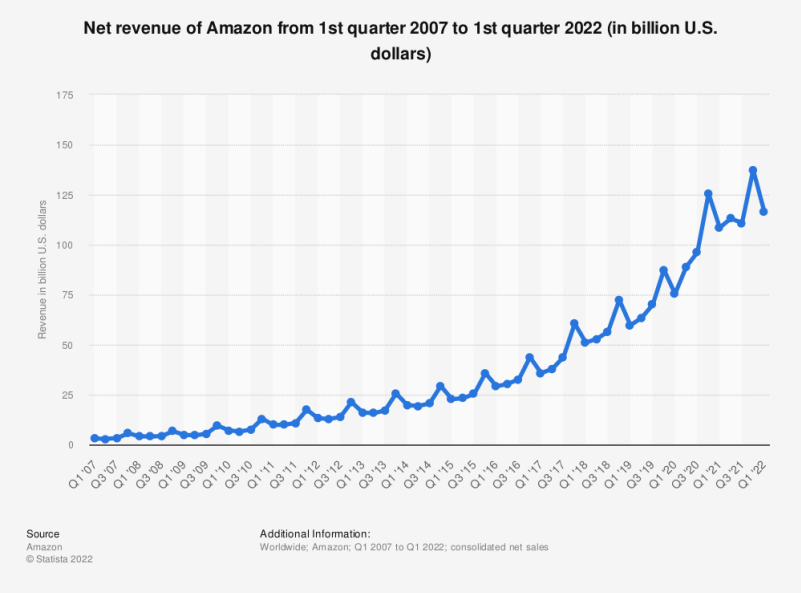

Figure 1.1 Amazon revenues by quarter. https://www.statista.com/statistics/273963/quarterly-revenue-of-amazoncom/

1.2 Focus on the surprise

Let's imagine that the consensus is wrong and the revenue for the quarter was over (under) forecasted. When the truth value is announced at the end of the quarter, the company valuation

should reflect that previous optimism (pessimism) and go down (up). If we knew in advance the truth value (oracle) of the revenues, we could predict the impact of the announcement on the stock price. Of course, it's not possible to have that oracle. But if we have a model that's good enough to beat the consensus, this extra information could be potentially used to long/short the stocks prior to the announcement. In their 1995 paper (10), *David N Dreman and Michael A. Berry* already show that the consensus performance is usually poor and that surprise values can be big. In 2001, Vasant Dhar and Dashin Chou try different model to forecast the earning surprise using machine learning models (11). One could also try to improve the investing heuristic based on the paper (12) that claims that *Post-earnings-announcement drift was found to be stronger when the revenue surprise was in the same direction as the earnings surprise*.

This notion of surprise is what the company Delphia tries to exploit as its investment mechanism. As described on their website, they are defining their portfolio allocation based on bets against the consensus (13). They forecast what is called the surprise on growth. This notion of surprise can be defined as

$$\textit{Surprise Revenues Growth} = \textit{Actuals Revenues growth - Consensus Revenues growth} \quad (1.1)$$

If the surprise is positive (negative), then the consensus was under (over) forecasting the actuals revenues growth and we should most likely make a long (short) on the stock, so that when the actual is disclosed, we make a gain.

Delphia uses the notion of hit rate (we will describe that metric later in the thesis) and Sharpe Ratio[2] to view the link between model performance and market return. The figure 1.2 is extracted from their website and shows the impact of the performance of the model (to predict the surprise) on their active investing strategy. We can see that the higher the hit rate, the higher the Sharpe Ratio.

[2]The Sharpe Ratio is defined as (14) the risk-adjusted performance of an investment.

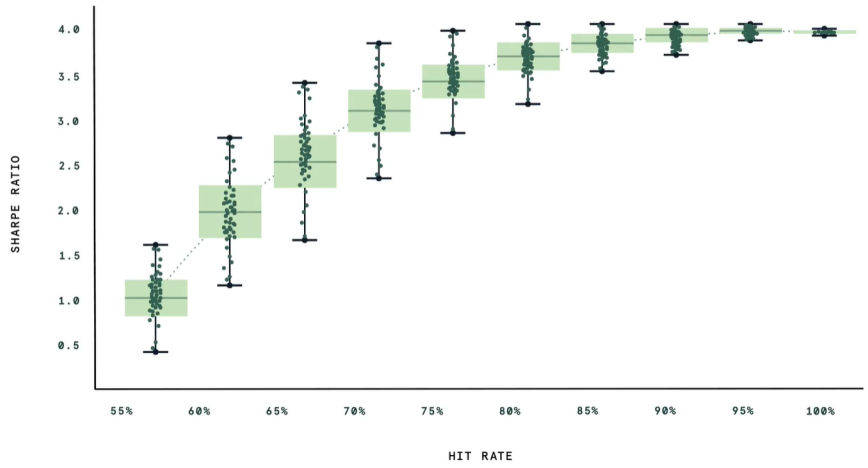

Figure 1.2 Impact of the hit rate on the surprise on the performance of the active investing strategy. https://delphia.com/algorithm

1.3 Full pipeline - Where does the ranking fit ?

As defined in the two sections above, we can see that fundamental values consensuses are not perfect. On top of that, the error brought by those consensuses has an impact on the stock price. Finally, the hit rate metric is correlated with the return on investment.

Based on those findings, we can define a rough version the current pipeline used by Delphia, that we want to improve in this thesis. As the figure 1.3 shows, as input data, we have history about the different companies involved as well as other features such as stock price history, fundamental values history, ... This process is a two part process where the left part (predicting the surprise) is achieved through machine learning, while the right part is solely relying on a heuristic that takes those forecasted surprises and defines an investment strategy. This strategy can be very roughly summarized as "shorting the stocks on which we expect a negative surprise and longing the stocks on which we expect a positive surprise". We do not refine that strategy and only focus on the left part of the process.

We will assume that the hit-rate is the metric highly correlated to the investment returns. The hit rate is somehow a measure of ranking as we will see further in the thesis. Therefore, rather

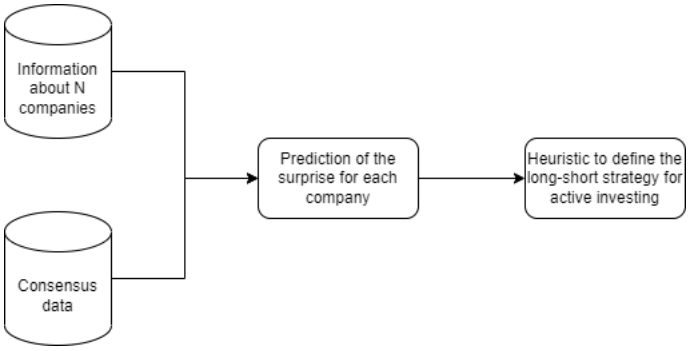

Figure 1.3 Original full investment process

than forecasting the absolute value of the surprise, we will forecast the ranking of the surprise. Indeed, since we apply a market-neutral long/short strategy, what matters is not the absolute value of the surprise, but the ordering of surprises across all the companies considered (The set of companies is referred in the literature as the *cross-section*). Having a better ranking will improve the hit-rate, which will then improve the return. The figure 1.4 shows that additional step which is hidden in the original process.

The aim of this master thesis is to use learning-to-rank tailored machine learning models

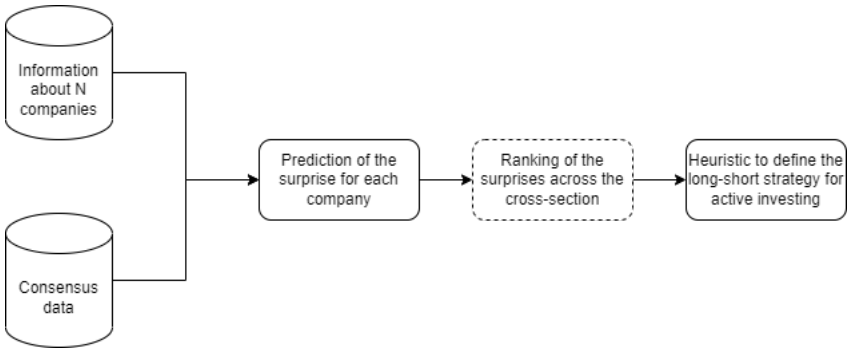

Figure 1.4 Underlying 3 step process of the investment strategy analysed

and see if they can improve the hit rate. If we can show that using learning to rank algorithm can produce a better hit rate than regression algorithms that forecast the surprise, then we show that there is potentially a further way of improving the return on investment provided by the Delphia approach. Using ranking algorithms to estimate stock returns has been tried in 2021 by *Daniel Poh Et al.* in (15) and gave encouraging results towards the use of ranking algorithms. This

thesis tries to obtain similar conclusions but with a focus on the ranking of the surprise rather than the ranking of the returns.

Chapter 2

Experimentation pipeline

This chapter describes the experimental process that we follow in this thesis. It helps understanding key concepts that makes the experiment reproducible. It is also designed in such a way that the process will yield (except if the underlying behaviour of the stock market changes dramatically) similar results for datapoints in the future.

The overall pipeline is depicted in the figure 2.1 and the sections in this chapter follow the order of processing of the pipeline.

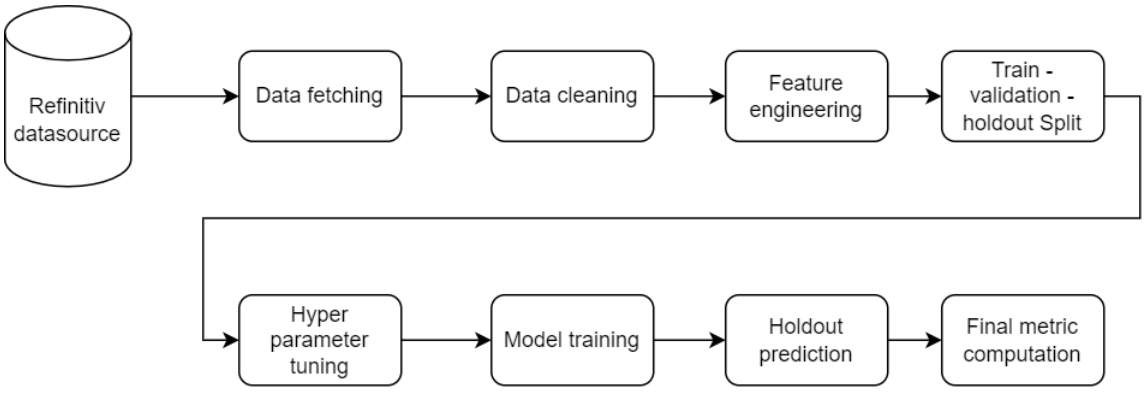

Figure 2.1 Machine learning pipeline for the experimentation

Time-series concept ; Look-ahead bias

As defined in (16), *look-ahead bias occurs by using information or data in a study or simulation that would not have been known or available during the period being analyzed. This can lead to inaccurate results in the study or simulation*.

Look-ahead bias avoidance needs to be at the center of each step of the pipeline when creating it. There will probably be no sign that some look-ahead bias happens, except potentially too-good-to-be-true results. The problem is that if the look-ahead bias also happens on the holdout set, the only moment when we will notice this bias is when the model is going to be used in production. Indeed, in production, the future information won't be available and the performance should therefore drop significantly. Then, an investigation would be needed to understand if that drop in performance is coming from a difference in the data distribution and patterns that happen on the stock market, or if the drop comes from a problem in the machine learning pipeline. Since we are working on quarter data, it could take a very long time to realize that there is some look-ahead bias that was happening in the training pipeline and the loss could be big.

Therefore, in this thesis, we will always make sure that the information used at a specific *Calc Date* (prediction date) was indeed available at that said *Calc Date*.

2.1 Data fetching

In order to compare the performance of different algorithm, the first step is to get historical data. We fetch financial market data provided by Refinitiv. More specifically, we use Refinitiv Eikon (17). Eikon is a set of software products provided by Refinitiv for financial professionals to monitor and analyze financial information. We describe in the appendix how we fetch the base data through the Eikon API.

We will describe it later, but our data spans from January 2010 til January 2022. In order

to avoid survivorship bias, we will select the top 400 companies by market cap in January 2010 from the S&P500 Index. We limit ourselves to 10 years of data and 400 companies in order to keep the size of the dataset reasonable so that we can run the algorithm on our local machine with limited RAM (16Gb or RAM in this case).

The actual revenues (called *actuals* in this thesis) are produced at a quarterly frequency. They are associated with a period end date and a report date (a few days/weeks after the period end date. This report date represents the day at which the financial information has been made public). The consensus on the other end is updated daily (We call that date the *Calc Date*). We must be cautious to use the report date when computing the features, so that there is no look-ahead bias.

2.2 Data Cleaning - Target creation

In our case, the data cleaning is fairly simple. We start by applying a forward filling on the data and target. Each day does not always have a consensus value. If we do not perform forward filling, those days that do not have a consensus value for a given company will have less then 400 datapoints. As we want to consider the full cross-section every day, we forward-fill the data. We assume that the consensus value is equal to the oldest consensus that was available prior to the missing datapoint for an instrument.

The following data cleaning method that we apply is a process called winsorization to input features and to the target for the training set. We apply this winsorization on the cross-section dimension, day-by-day. The aim is to avoid strong outliers which could have a too big impact on the learning metric and bias the model, giving a bigger importance to those datapoints and features. The winsorization process is a process where we truncate the smaller value to the .x percentile and the biggest value to the 1-(.x) percentile. For example, if x is 0.05, we will truncate all the values smaller than the 5% percentile to the value of the 5% percentile and all the values bigger than the 95% percentile to the value of the 95% percentile. This computation is

performed day by day to avoid look-ahead bias issues.

In order to compute the target, we first need to compute the year over year growth on the actuals. This can be done via a join on the actuals dataset merged with the actuals dataset where the financial period is shifted by a year (fperiod). This allows us to get a column called *Actuals shifted by 1 year*. Then, we can compute the year over year growth and the consensus growth as:

$$\text{Act\_growth(C, Y, Q)} = \frac{\text{Act(C,Y,Q) - Act(C,Y-1,Q)}}{\text{Act(C,Y-1,Q)}}$$

$$\text{Cons\_growth(C,Y,Q,t)} = \frac{\text{Cons(C,Y,Q,t) - Act(C,Y-1,Q)}}{\text{Act(C,Y-1,Q)}}$$

where

- C is a given company
- Y is the fiscal year as integer
- Q is a quarter as integer
- t is the Calc date at which the consensus was produced
- Act_growth is the actual revenue growth year over year
- Act is the revenue actuals
- Cons_growth is the revenue growth year over year estimated by the consensus at a given Calc date
- Cons is the revenue consensus estimate for a fiscal quarter at a given Calc date

Finally, we can compute the surprise (target) using the formula defined in equation 1.1

2.3 Validation Metric

The validation metric in a learning task is the mathematical expression that allows to measure the performance of a model. If that metric is sound, then having Model A getting a higher metric value than Model B allows to claim that Model A is better at learning the target than model B. In our case and as described in the introduction, we have seen that Delphia bases its analysis of the performance of a fundamentals forecasting model on a hit-rate metric.

We first start by computing the sign of the surprise on the cross section (for both the predicted value and the actual surprise).

$$sign\_fcst(C,t) = sign[fcst\_surprise(C,t) - median\_fcst\_surprise(t)] \tag{2.1}$$

$$sign\_act(C,t) = sign[act\_surprise(C,t) - median\_act\_surprise(t)] \tag{2.2}$$

where

- sign(x)= $\begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if} x = 0 \\ -1 & \text{if} x < 0 \end{cases}$
- fcst_surprise(C,t) is the forecasted surprise for a given company (C) at a given Calc date (t).
- median_fcst_surprise is the median of the forecasted surprises on the cross section at a given Calc date (t)
- act_surprise(C,t) is the truth surprise for a given company (C) at a given Calc date (t).
- median_act_surprise is the median of the actual surprises on the cross section at a given Calc date (t)

Then, we can compute a hit for a given company on a specific day regarding its ranking in the cross section as

$$Hit(C,t) = \begin{cases} 1 & \text{if } fcst\_surprise(C,t) = act\_surprise(C,t) \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Finally, we can compute the hit rate metric on the whole cross section (CS) for all the forecasted dates (Ts) as :

$$\text{Hit Rate}(Ts,Cs) = \frac{\sum_{t \in Ts} \sum_{C \in Cs} Hit(C,t)}{len(Ts) \cdot len(Cs)} \tag{2.4}$$

We can see that this metric is a very basic version of a ranking metric. Indeed, if we have a perfect ranking, the hit rate is going to be close to 1. If we have a random ranking, it's going to be close to 0.5. In this thesis, we focus solely on this metric as it has been shown in the introduction that it can be linked to the portfolio performance. However, we can see that this metric is quite simplistic as it only measure partially how good the ranking is (above or below the median). Also, we can see that in figure 1.2, the increase in hit rate linked to an increase in sharpe ratio is associated with a high variance. Maybe using a more complex ranking metric could provide the same kind of link but with less variance in the sharpe-ratio performance. But this is a focus on the right part of the investment process, which is outside of the scope of this thesis.

2.4 Feature engineering

Features are what drives machine learning models. They allow the model to understand more complex patterns in the data. Models cannot compute automatically lag features for example, which can be very useful when doing time-series modeling. The full list of features used by our models is in the Appendix.

2.4.1 Lag/Shift features

We describe in this section what a lag feature is and the precautions we must take in order to avoid *look-ahead bias*. Let's imagine that we take a datapoint having a Calc Date of 2020-01-01. The features available for this specific datapoint should all have been available prior to that date (2020-01-01). So for example, if the End Date of the previous quarter is 2019-12-31, but the report date of that same information is 2020-01-15, we cannot get "Last quarter Revenue" for this datapoint. Its value will then be NULL.

We must be very careful that for each lag feature that we bring, a report date is available and the lag feature is only set for a datapoint if the report date is <= than the *Calc Date* of that datapoint.

Now that we have this safeguard in place in our shifting mechanism, we can define a simple function *shift_feature* that is going to shift a feature (an actual value) by Y years and Q quarters. This shifting function also takes the report date and Calc Date into account. Then, we call that *shift_feature* function for different combinations of Y and Q.

```
def shift_feature(feature: str,
                  years_shift: int,
                  quarter_shift: int)
```

In our case, those are the combinations used to shift the revenue feature

- Y $\in$ [0,1,2]
- Q $\in$ [1,2,3]

2.4.2 Growth features based on shift features

However, shifting those revenues is going to give us the absolute value of the revenue and not the growth. In order to work purely with percentages rather than absolute values, we compute

the year-over-year growth features based on the *shift_feature* computed above. The growth can be computed over several year-over-year and in that case, the growth is annualized (taking the geometric average of that growth over several years). Let's take some examples to make it clearer

Annualized Growth increase Revenue - Actual 1Y1Q - Shift

This computes the growth between the revenue shifted by 1 quarter and the revenue shifted by 1 year and 1 quarter. So if the current quarter FY2020Q3, its value is going to be (assuming that all shift values used are available for that datapoint regarding the *Calc Date*/*Report Date* constraints).

$$\frac{\text{Act(C, FY2020Q2) - Act(C, FY2019Q2)}}{\text{Act(C ,FY2019Q2)}} \tag{2.5}$$

Annualized Growth increase Revenue - Actual 2Y1Q - Shift

This computes the annualized growth (in this case, the geometric average is a square root) between the revenue shifted by 1 quarter and the revenue shifted by 2 year and 1 quarter. So if the current quarter FY2020Q3, its value is going to be (assuming that all shift values used are available for that datapoint)

$$\sqrt{\frac{\text{Act(C, FY2020Q2) - Act(C, FY2018Q2)}}{\text{Act(C, FY2018Q2)}}} \tag{2.6}$$

Now that we have defined the methodology to compute those lag features and growth features based on actuals, we can use Refinitiv to extract other fundamental values and feed them in the lag/shift + growth feature computation mechanism. There is no need to scale them as we compute the growth percentage on the absolute values. The fundamental values that we extracted from Refinitiv are

- EBIT : Represents the earnings of a company before interest expense and income taxes paid.
- EBITDA : Gauges the raw earnings power of a company before debt servicing, corporate taxes and any allowances made for depreciation and amortization costs the company faces.

- Net Income : Measures the corporation's after-tax income.
- Goodwill : Represents the value of intangible assets such as a strong brand name, good customer relations, good employee relations and any patents or proprietary technology.
- Capital Expenditure : Represents the funds used by a company to acquire or upgrade physical assets such as property, industrial buildings, or equipment or the amount used during a particular period to acquire or improve long term assets such as property, plant or equipment.

2.4.3 Other shift features

On top of those fundamental actual shift features, which are relative to the quarter granularity and need to be computed taking the *Report Date* into account, we can also compute shift features at a day granularity. In our experimentation, we did it on 2 features that are easily available through the Refinitiv API. We picked those features as they should be impacted by the growth of the above fundamental. Hopefully, merging the extra fundamental information from above and the variable that should be affected by those fundamental change should bring useful information to the machine learning model.

- Stock Price
- Market Cap

For each *Calc Date* and instrument, we can get the corresponding Stock Price or Market Cap. Then we can compute the Stock Price/Market Cap value 1 year ago and compute the year-on-year growth on those features too. Therefore, there was no need for scaling as we use the growth rather than their absolute value.

2.5 Train-Validation-Holdout

In this section, we detail how we can perform cross-validation on a time series to assess the performance of a modeling pipeline without touching the holdout set at all. Note that the holdout

set is data put on the side at the beginning of the process. That data can never be used, except at the very end in order to measure the performance of the full pipeline on never seen/used before data. In our case, the holdout is going to be between January 1st 2021 and January 1st 2022.

In order to perform cross-validation in a time-series modeling process, we need to follow the logic depicted on the figure 2.2, which is called backtesting (18). In this case, we have a 4-fold.

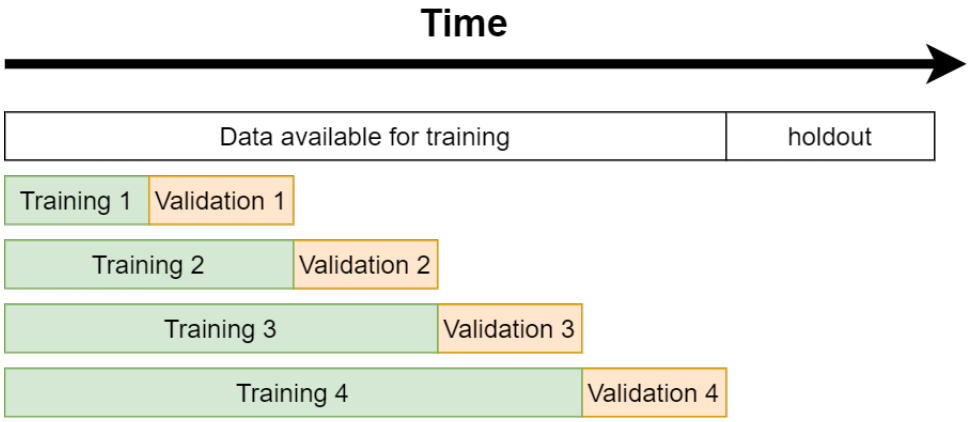

Figure 2.2 Visualisation of the train-validation split for cross validation in a time-series environment

How to take care of Calc Date and Report Date when splitting ?

On the figure 2.2, we see that we make a split on the dataset based on time. However, should we split on the Report Date or on the Calc date ? Because we cannot use information that was not available prior to the split (Report Date), nor use datapoints that fall after the split date in the training (I cannot use a consensus from the future to train my model), for the training set, we should split based on both the Report Date and on the Calc Date. Note that the Calc Date condition is enforced by default as Calc Date is always smaller than Report Date (There is no point in making a forecast on a value that has already been published).

For the validation set, it is a bit more complex. In theory we could evaluate on everything remaining. However, since companies have all different report dates, it could mean that some

Calc Date in the validation are before the split date. But that would not make any sense when we think about a concrete use case.

As an example, imagine that the split date is 2020-01-01. Some company has a report date on 2020-01-05. The datapoint having a Calc Date of 2019-12-15 is not included in the training set, because on the 2020-01-01, the fundamental was not reported yet. But is it part of the validation set ? If that datapoint was included in the validation set, it would mean that the training will contain other datapoints having a Calc date of 2019-12-25 for example (from other companies). It would mean that we have a validation point prior to a training datapoint, which cannot happen (look-ahead bias).

With this simple example, we show that the split should be

- Training set
  - Report Date < Split Date
  - Calc Date < Split Date (redundant condition because Calc Date < Report Date)
- Validation (or holdout) set
  - Calc Date >= Split Date
  - Report Date >= Split Date (redundant condition because Report Date > Calc Date)

Figure 2.3 shows that datapoint on a timeline. We can see that if the split date is smaller than the calc date (a), then the datapoint will be assigned to the validation set. If the split date is bigger than the report date (c), the datapoint belongs to the training set. Otherwise (b) the datapoint is neither in the training set nor the validation set.

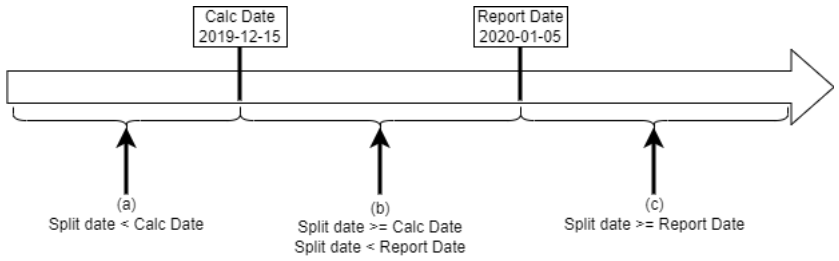

Figure 2.3 Graph helping to understand if a datapoint belongs to the training set or validation set depending on the split date

2.6 Hyperparameter tuning

Models receive as configuration a set of parameters that are going to define how this model is being built/optimised. Those parameters can have a huge impact on the final efficiency of the model. Therefore, one common step in machine learning is to try different parameters for the model and see which set of parameters is the best fit for the use case at hand. Usually, there are 2 nested loops in the training of models.

- The inner loop performs a K-fold split and is used to view which parameter is the best
- The outer loop is used to compute the expected performance of the whole pipeline using another K-fold mechanism

In our case, we are going to simplify the process and just use one loop to find the optimal parameters. Once those parameters are obtained, we will train a model on the full dataset using those optimal parameters and evaluate it on the holdout set as the final estimation of performance. This is going to simplify the computation time and will not bring a big difference in the analysis of the results.

There are various methods to optimise the hyperparameters and find which one is the best. Indeed, let's imagine that we have 5 parameters and each parameter can take 5 values. On top of that, we perform a 4 folds to measure the performance of the hyperparameter set. It would mean creating $4 * 5^5 = 12500$ models. We made a quick experimentation and observed that training 1 model takes 20 seconds on our machine. It means that the full hyperparameter

tuning would take approximately 250.000 seconds, or roughly 4 days. With this simple example, we can see that making a complete search would provide the global optimum but would take too much time to compute. There are lots of methods available to find a close to optimal set of hyperparameters (19). In this thesis, we are going to take a simple and greedy approach, described by the pseudo-code in the Listing 2.1. The aim is to loop through each parameter and pick the optimal parameter value. Then we freeze that optimal parameter value and move on to the next parameter.

```
optimal_hp = dict()
for hp_name, hp_value_list in hyperparameters_to_tune:
    optimal_performance = min_performance_possible
    optimal_parameter_value = None
    for hp_value in hp_value_list:
        performance = k_fold(hp=default_hp.union(optimal_hp))
        if performance is better than optimal_performance:
            optimal_parameter_value = hp_value
            optimal_performance = performance
    optimal_hp[hp_name] = optimal_parameter_value
```

Listing 2.1 Greedy hyperparameter tuning pseudo-code

Note that using this algorithm, rather than having 12500 models to train, we get the number of model trained down to $4 * 5 * 5 = 100$. If we keep the training time close to 20 seconds, we reach a total time of 2000 seconds to tune our parameters. For each model in the section below, we are going to define the hyperparameters tried in the experimentation (if hyperparameters are involved).

2.7 Models

Now that we have defined a dataset, the features associated and the hyperparameters, we can describe how the models work. That model will be able to generate a ranking of the surprises on the cross-section every day. We will build those models on the training data, then we will assess their performance on the holdout.

2.7.1 Random

The first strategy is what we call the *dummy model*. It is a model that is very easy to describe and implement. If any other model has a similar or worst performance than the random model, it means that the model does not learn anything and that doing nothing would be better than trying to do this modeling. Because our metric is a ranking metric, the random model can just predict a surprise between 0 and 1 for each datapoint using a uniform distribution. This method should bring a hit-rate value of 0.5 on average. Indeed, by using a uniform distribution, a datapoint has a 50% chance of being bigger than the median and 50% chance of being smaller than the median. And 50% of the actual surprise will fall above the actual median and 50% will fall below the actuals median. The figure 2.4 shows that through a visualisation where we can see that with this random approach, we have a hit (1) on 50% of the datapoints on average and a miss (0), leading to an average hit rate of 0.5.

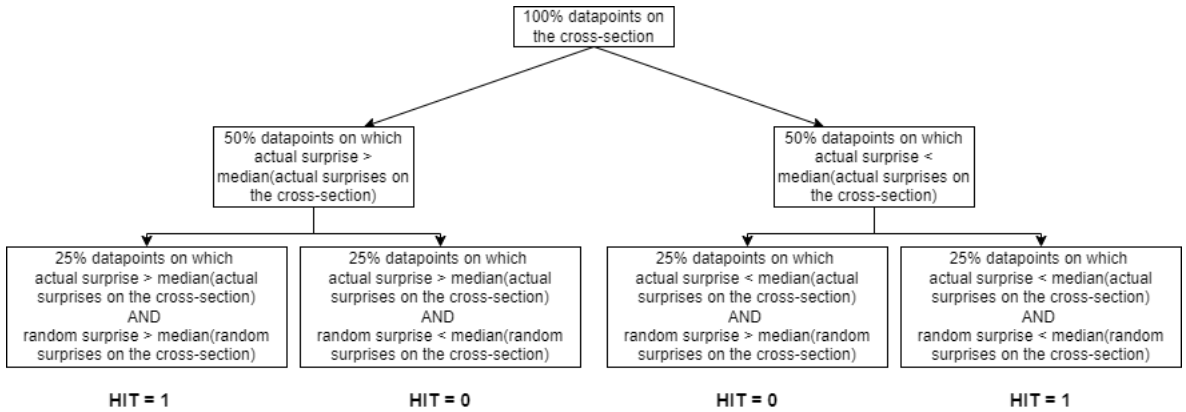

Figure 2.4 Visualisation of the random ranking.

2.7.2 Naive

The naive approach that we chose is quite basic too but serves as a baseline for comparison. This baseline will help to put some context and see by how much the machine learning modeling can help to increase the performance and the metric. Note that the naive surprise forecast value is going to be put as input feature to the machine learning models as well.

The naive methodology that we use in this model is the following

Assume that the Revenue year over year growth will be equal to the latest (previous quarter) available year over year growth

What is important is to remember that because of the difference between the end date of a quarter and its *Report Date*, and also because the horizon could be to forecast N quarters into the future, sometimes the year on year growth of the previous quarter won't be available.

Let's take a toy problem to make it clearer (synthetic data). The base data is shown on table 2.1.

Quarter	End date	Report Date	Revenue
FY18Q4	2018-12-31	2019-01-15	40
FY19Q1	2019-03-31	2019-04-15	10
FY19Q2	2019-06-30	2019-07-15	20
FY19Q3	2019-09-30	2019-10-15	30
FY19Q4	2019-12-31	2020-01-15	40
FY20Q1	2020-03-31	2020-04-15	11
FY20Q2	2020-06-30	2020-07-15	22
FY20Q3	2020-09-30	2020-10-15	33
FY20Q4	2020-12-31	2021-01-15	44

Table 2.1 Simple Naive modeling base data example

We are going to show some scenarios to help the reader get a better feeling of how that naive feature is computed.

Calc Date = 2020-07-01 and horizon is 0 (*Current quarter*)

We are then forecasting FY20Q2 (the quarter is finished, but the numbers are not public yet). In order to do that, we can look at the yoy growth of the previous available quarter. We can see that FY20Q1 data is available and FY20Q2 data is not available. Therefore, we compute the yoy growth between FY19Q1 and FY20Q1 (10%) and assume that it will be the same for the growth between FY19Q2 and FY20Q2 (and we are very lucky, it will be the exact same growth). Let's assume now that the consensus was 21.5 for the quarter on a given Calc Date. In that case, since we expect a growth of 10%, our expected revenue is 22. The expected surprise on growth can be computed.

$$\text{Consensus yoy growth} = \frac{21.5-20}{20} = 7.5\%$$
$$\text{Naive forecast expected yoy growth} = \frac{22-20}{20} = 10\%$$
$$\text{Naive Forecasted surprise} = 10\% - 7.5\% = 2.5\%$$

Calc Date = 2020-02-01 and horizon is 2 (Forecasting FY20Q3)

In this case, the last revenue available is FY19Q4 and the naive growth is computed as growth between FY19Q4 and FY18Q4 (From 40 to 40). The consensus forecasts 42. The expected surprise is computed like this

$$\text{Consensus yoy growth} = \frac{42-40}{40} = 5\%$$
$$\text{Naive forecast expected yoy growth} = \frac{40-40}{40} = 0\%$$
$$\text{Naive Forecasted surprise} = 0\% - 5\% = -5\%$$

In this case, we forecast a negative surprise (we forecast that the consensus over-estimates the real revenue).

What matters with this example is that the naive method is not computing FY20Q3 yoy growth based on FY20Q2 yoy growth nor FY20Q1 yoy growth, since that information is not available at the time of making the forecast. This naive method is safe from look-ahead bias issues.

2.7.3 Regress-then-rank

The first models that we are going to build are models whose objective is to forecast the absolute value of the surprise. Those are regression models. Then, based on the surprise forecasts, we can compute the ranking and the hit-rate.

XGBoost

XGBoost (Extreme Gradient Boost) is a famous and efficient machine learning algorithm that was designed in 2016 by Tianqi Chen and Carlos Guestrin (20). Its success is very impressive and the original paper has more than 17.000 citations. It has been made popular by being a very competitive algorithm on websites like Kaggle (21). As well, it has an integration with scikit learn(22). Without going too much into details of how the model works, XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

In order to solve the regression problem of forecasting the surprise, we are going to define the objective function being used as *reg:squarederror* which is the default objective function for regression problems and minimizes the square loss. During the hyperparameter tuning phase, we are going to try and tune 4 key parameters :

- eta : Also called learning rate ; Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features.
- max_depth : Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit.
- min_child_weight: Minimum sum of instance weight (hessian) needed in a child.
- num_boost_round : Number of boosting iterations.

LightGBM

LightGBM is another gradient boosting framework that uses tree based learning algorithms. It was designed by a team at Microsoft and the original paper was published in 2017 (23). It is considered to be the competitor of XGBoost and an improved version of it, that scales more and can process bigger datasets more efficiently.

LightGBM allows to specify the objective as being *regression*, which when looking in the documentation corresponds to optimizing the RMSE (l2 loss), which is equivalent of what XGBoost is optimising the model for. The hyperparameters that we will tune are

- learning_rate: Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features.
- max_depth: Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit.
- min_data_in_leaf: Minimal number of data in one leaf. Can use this to deal with over-fit.
- num_iterations: Number of boosting iterations.

2.7.4 Ranking algorithms

From an implementation perspective, the ranking algorithms that we are going to use are very similar to the Regress-then-rank algorithms tried. Indeed, XGBoost and LightGBM both support different objective functions that are suited for the Learn to Rank task. The following options are available :

- LightGBM (24)
  - lambdarank, lambdarank objective (25).
  - rank_xendcg: faster than and achieves the similar performance as lambdarank.
- XGBoost (26)

  - rank:pairwise: Use LambdaMART to perform pairwise ranking where the pairwise loss is minimized
  - rank:ndcg: Use LambdaMART to perform list-wise ranking where Normalized Discounted Cumulative Gain (NDCG) is maximized
  - rank:map: Use LambdaMART to perform list-wise ranking where Mean Average Precision (MAP) is maximized

In order to limit the options tried, we are going to try the following 3 algorithms to try and see the impact of different objective function. We discard the LightGBM rank_xendcg as it is only a faster version of LightGBM lambdarank. We also discard XGBoostrank:ndcg as it is it also a list-wise ranking objective, as XGBoost rank:map. The various objective functions are described in the following section.

- LightGBM lambdarank
- XGBoost rank:pairwise
- XGBoost rank:map

Note that in order to be able to use those models, we need to modify the label so that they are integers that will denote the rank of each datapoint. We are going to generate 10 rank categories, based on the rank of the surprise on a given day. We could try and tune this number of categories as a hyperparameter using different values [1]. Also, the 2 API need to integrate a grouping which is going to be represented by the Calc Date, turned into an integer representing the day (day_id feature). The hyperparameters tuned are going to stay the same as for the two models above (but will be recomputed for the new objective function).

Point wise vs Pairwise vs Listwise objective

There are 3 main categories when considering Learn To Rank objective functions (27).

[1] This was tried quickly in a Jupyter notebook and did not seem to have a big impact on the performance of the model. So we leave this out of this thesis report

The first ranking metric category is the pointwise ranking metric, where we define the ranking problem like any other machine learning task, where we try to predict the labels by using regression. This is equivalent to the method that we have used in the Regress-then-rank models.

The following metric that we try to optimise for is called the pairwise ranking metric. This pairwise metric is defined in (28). *We consider models where the learning algorithm is given a set of pairs of samples [A,B] (...) with target probabilities P_{AB} that sample A is to be ranked higher than sample B*. Basically, under that setup, we train a classifier on pairs of sample and that classifier returns the likelihood of one sample being ranked higher than the other.

Finally, the listwise ranking metric was introduced in 2007 by *Z Cao et al.* in (29). The technique is an improvement of the pairwise metric as it considers multiple instances at the same time rather than pairs of datapoints. The gradient is computed based on those set of instances. As the listwise ranking metric is an improvement of the pairwise ranking, it should in theory provide better results in our experimentation phase.

We can see that we are trying a pairwise and a listwise objective through XGBoost (rank:map is a listwise objective), while LighGBM uses lambdarank as a objective, which is a listwise objective. We also try the pointwise metric with XGBoost and LightGBM as regressors.

Chapter 3

Experiment results - Horizon 1 quarter

In this section, we analyse the results of the different models and their hit-rate performance against the holdout set. Remember that the holdout set is from 2021-01-01 until 2022-01-01. This experiment ran with a forecast horizon of 1 quarter. We start by analysis the learning curves of a model to see the evolution of the training hit rate and the evolution of the validation hit rate, in order to verify that some learning happens. It also allows us to see if overfitting happens quickly or not. Then, we look at the feature importance of a model to see if what is learned seems to make sense. Finally, we have a look at the results on the holdout.

Learning curve

The learning curve on figure 3.1 is coming from the LightGBM model and shows that there is a quick fit followed by some quick overfitting. Indeed, we can see that the hit rate on the training set keeps increasing, while after 6 boosting iterations, the performance on the validation set starts to drop. In order to avoid that, we could implement some regularisation in the algorithm or try to have a smarter feature engineering.

Feature Importance

The feature importance shown on figure 3.2 is the importance coming from the same LightGBM model described above. Note that the day_id being the most important feature seems to be one of

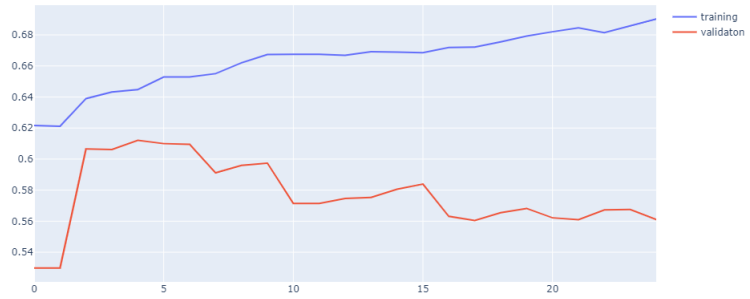

Figure 3.1 Learning curve for a LightGBM model. The metric used is the hit rate

the reason of this overfitting. Indeed, this feature is representing the distance in day from the 1st of January 2000 and should not be as relevant for the training. We can see that the naive surprise is quite relevant, but also that the increase in Revenue is relevant. The Revenue - Mean represents the consensus. This feature importance seems to show that there is something interesting being learned and that the performance is not purely due to overfitting. However, we can see that some features are not relevant and should probably be filtered if we wanted to build more robust models.

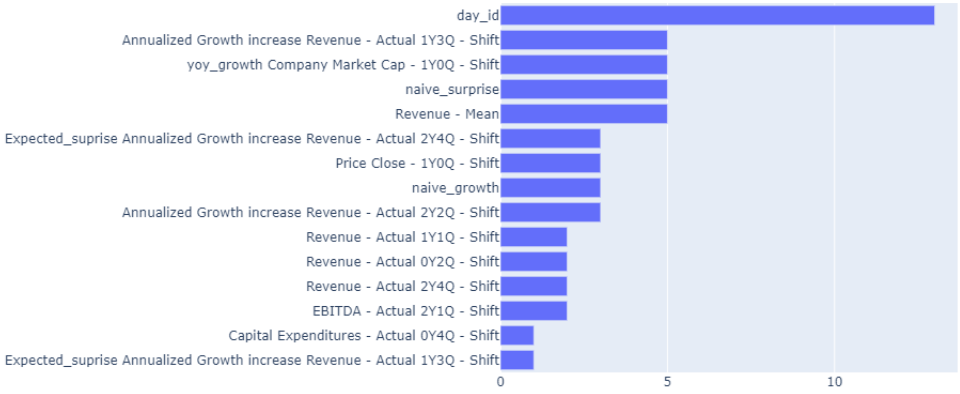

Figure 3.2 Feature importance for a LightGBM model predicting the surprise

Hit rate results on the holdout set

The table 3.1 shows the mean hit rate of the various algorithm, while the figure 3.3 shows the distribution of daily hit rates over the full year of 2021.

Learning Method	Hit rate
Random	0.4996
Naive (baseline)	0.5234
Regress-then-rank : XGBoost	0.5753
Regress-then-rank : LightGBM	0.5735
Learn-to-rank : LightGBM : lambdarank	0.5720
Learn-to-rank : XGBoost : rank:pairwise	0.5915
Learn-to-rank : XGBoost : rank:map (listwise)	0.5180

Table 3.1 Hit-rate results on the holdout set for 1 quarter horizon learn to rank task

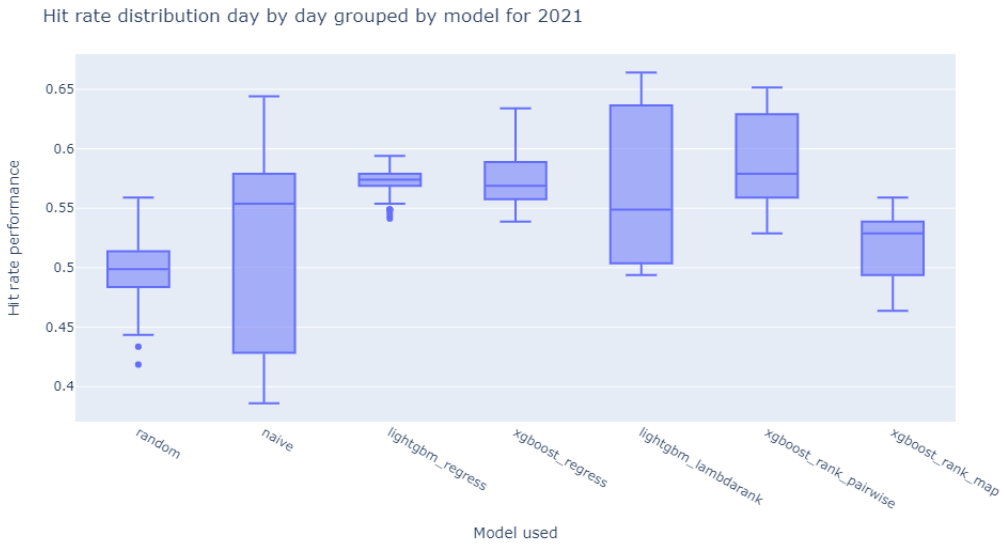

Figure 3.3 Hit rate per modelling method

- We can see that as expected, the random model is giving a hit rate of 0.5.
- The naive method (which is based on forecasting the year on year growth using the latest available year on year growth) already yields a hit rate higher than 0.5, even if we can see that the variance is quite important. This means that beating the consensus could be

done by simply looking at the most recent and available past year on year growth. It is interesting to see that such a naive approach can provide a hit rate higher than 0.5. This could potentially be explained by the fact that those forecasts of revenues are sometimes made by humans and can be tweaked to avoid future issues. For example, this paper (30) shows that there is an incentive for the management to avoid negative surprises.

- The 2 following results come from the regress-then rank models, where we are framing the problem as the pure forecasting of the absolute value of the surprise. We can see that on average there is a clear improvement from the baseline/naive method, even if when taking the median, the results are quite close (line in the middle of each blue box on the box-plot). An interesting aspect of those 2 models is that they provide a low variance in terms of performance metric. This is an important aspect of the metric as well, since in finance, variance is associated with risk and lowering the variance in the metric means lowering the risks.
- Then we have the 3 ranking models. Those models are the focus of this thesis and the aim was to see if the performance would increase.
  - When we look at the mean results and at the boxplots, there is a clear outlier, the *XGBoost : rank:map*, which has a really poor performance, which is even worse than the naive model.
  - The lambdarank model on the other end seems to give results that are very similar to the regress-then-rank models but with a higher variance. Making it less stable than the regress-then-rank methods.
  - The only model that seems to perform a bit better than all the others is the XGBoost:rank:pairwise, which has a higher mean hit rate and a higher median hit rate than all the models. However, it does not seem to be significant enough to be considered as a clear improvement.

It would be interesting to analyse in the details of the implementation of those ranking objective functions in XGBoost and LightGBM and the implication that it has on the learning. However,

as we saw earlier, there is a quick overfitting effect and it would be interesting to bring more features that could potentially be more correlated to the surprise. This could potentially stabilise the learning and help us reach more concrete conclusions on the effect of using ranking models.

It seems fair to conclude that there could be an improvement brought by using ranking objective functions rather than pure regressions. Unfortunately, the cut is not clear and we cannot see that there is an obvious improvement coming from that methodology.

3.1 Discussion ; To go further

In order to be able to improve the hit rates or the Sharpe Ratio, one could try various modification to the experimentation process. We could start by improving the data provided to the model by having more complex feature engineering to find features that are more closely correlated to the surprise. Then we could also try to focus on other modelling mechanisms which might be better suited to learn a ranking rather than a single score, such as deep learning Learn-To-Rank models. On the data aspect, we could also look at modifying the cross-section. For example, focus on smaller companies for which less experts are producing fundamental values forecasts. Hopefully the hit rate on the surprise can be higher for those companies. Another approach could be to look at custom objective functions that would be more tailored for this specific problem.

Another interesting field of study would be to analyse which fundamental surprises are the most easy to forecast and how that extra precision maps to extra returns on the whole portfolio. One could use as a start the paper published by *Yonca Ertimur and Joshua Livnat* which shows that *investors value more highly a dollar of revenue surprise than a dollar of expense surprise* (31).

Chapter 4

Conclusion

The field of active investing based on fundamental values forecasting allows to link machine learning forecasting to higher risk-adjusted returns. Having an edge on the average fundamental value forecast (the consensus) can bring an increase in the Sharpe Ratio as shown on figure 1.2. Therefore, finding ways to improve the performance of machine learning models that predict the surprise on fundamental forecasts equates to finding ways to improve the return on investment of actively managed portfolios.

There has been several attempts at improving the surprise forecast, based on statistical methods, and some are based on machine learning regression (11)(8). In this thesis, we try to reuse objective functions such that the regression task is transformed into a Learn to Rank task. The aim is to try to see whether changing that objective function can lead to better results in term of surprise forecasting (15).

Unfortunately, the results that we obtained are not significant enough to conclude that using ranking models is the best approach and brings a clear edge versus using simple regression models.

Bibliography

[1] Dev Shah, Haruna Isah, and Farhana Zulkernine. Stock market analysis: A review and taxonomy of prediction techniques. *International Journal of Financial Studies*, 7(2):26, May 2019.

[2] What are stock fundamentals? definition, examples faq. https://www.thestreet.com/dictionary/f/fundamentals. Accessed : 2022-04-23.

[3] Joseph D. Piotroski. Value investing: The use of historical financial statement information to separate winners from losers. *Journal of Accounting Research*, 38:1, 2000.

[4] John Alberg and Zachary Lipton. Improving factor-based quantitative investing by forecasting company fundamentals. 11 2017.

[5] James M. Wahlen and Matthew M. Wieland. Can financial statement analysis beat consensus analysts' recommendations? *Review of Accounting Studies*, 16(1):89–115, March 2010.

[6] Matthew M. Wieland. Identifying consensus analysts' earnings forecasts that correctly and incorrectly predict an earnings increase. *Journal of Business Finance &amp Accounting*, 38(5-6):574–600, April 2011.

[7] Narasimhan Jegadeesh. Revenue growth and stock returns. *SSRN Electronic Journal*, 05 2002.

[8] Gary A. Benesh and Pamela P. Peterson. On the relation between earnings, changes, analysts' forecasts and stock price fluctuations. *Financial Analysts Journal*, 42(6):29–39, 1986.

[9] Chul W. Park and Earl K. Stice. *Review of Accounting Studies*, 5(3):259–272, 2000.

[10] David N. Dreman and Michael A. Berry. Analyst forecasting errors and their implications for security analysis. *Financial Analysts Journal*, 51(3):30–41, May 1995.

[11] V. Dhar and D. Chou. A comparison of nonlinear methods for predicting earnings surprises and returns. *IEEE Transactions on Neural Networks*, 12(4):907–921, July 2001.

[12] Joshua Livnat. Post-earnings-announcement drift: The role of revenue surprises and earnings persistence. *SSRN Electronic Journal*, 2003.

[13] Delphia website - algorithm page. https://delphia.com/algorithm. Accessed : 2022-04-23.

[14] William F. Sharpe. The sharpe ratio. *The Journal of Portfolio Management*, 21(1):49–58, 1994.

[15] Daniel Poh, Bryan Lim, Stefan Zohren, and Stephen Roberts. Building cross-sectional systematic strategies by learning to rank. *The Journal of Financial Data Science*, 3(2):70–86, March 2021.

[16] What is look-ahead bias? https://corporatefinanceinstitute.com/resources/knowledge/finance/look-ahead-bias/. Accessed : 2022-04-24.

[17] Eikon - wikipedia. https://en.wikipedia.org/wiki/Eikon. Accessed : 2022-04-23.

[18] Marcos Lopez de Prado. *Advances in Financial Machine Learning*. Wiley Publishing, 1st edition, 2018.

[19] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges, 2021.

[20] Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, August 2016.

[21] Kaggle. https://www.kaggle.com.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[23] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[24] Lightgbm parameters. https://lightgbm.readthedocs.io/en/latest/Parameters.html. Accessed : 2022-04-24.

[25] Christopher Burges, Krysta Svore, Paul Bennett, Andrzej Pastusiak, and Qiang Wu. Learning to rank using an ensemble of lambda-gradient models. *Journal of Machine Learning Research - Proceedings Track*, 14:25–35, 01 2011.

[26] Xgboost parameters. https://xgboost.readthedocs.io/en/stable/parameter.html. Accessed : 2022-04-24.

[27] Pointwise, pairwise and listwise learning to rank. https://medium.com/swlh/pointwise-pairwise-and-listwise-learning-to-rank-baf0ad76203e. Accessed : 2022-04-24.

[28] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*. ACM Press, 2005.

[29] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. volume 227, pages 129–136, 01 2007.

[30] Dawn A. Matsumoto. Management's incentives to avoid negative earnings surprises. *The Accounting Review*, 77(3):483–514, July 2002.

[31] Yonca Ertimur, Joshua Livnat, and Minna Martikainen. *Review of Accounting Studies*, 8(2/3):185–211, 2003.

Appendix A

Appendix

A.1 Fetching data using the Refinitiv Eikon API

This appendix describes the way we fetch data for our experiment. There are 2 ways to get data from Eikon.

- Going through the Eikon Excel plugin
- Using the Python eikon API

The excel plugin is quite convenient in order to explore the dataset, find which fields are available and how data can be fetched through calls. However, it does not scale and does not allow to execute complex queries/joins on the data once fetched. As we want to be able to easily modify parameters of data fetched, we are going to use the second option (Python API calls) to perform the heavy lifting of the data fetch task.

Below is a description of the different fields that we use to get the base data. Note that because we compute year over year growth, 2 years over year growth, ... we need to fetch more data than only the data between the start date and end date of the training-set/holdout set.

1. Get 500 instruments from S&P 500
    - Instrument : 0#.SPX (which corresponds to the S&P500 index in the Refinitiv API).

- Fields:
  - TR.RIC : Returns the list of sub-instruments for the input instrument provided
- Parameters:
  - SDate : 2010-01-01

2. Get actual revenue
   - Instruments : Top 400 Instruments sorted by total market cap value on January 1st 2010.
   - Fields:
     - TR.RevenueActValue : The revenue for the quarter in USD
     - TR.RevenueActValue.periodenddate : The end date of the period considered
     - TR.RevenueActValue.fperiod : The period as a string (FY2002Q4)
     - TR.RevenueActValue.announcedate : The date at which the actual revenue became public = *Report Date*
   - Frequency : 1 per quarter
3. Get consensus
   - Instruments : Top 400 Instruments sorted by total market cap value on January 1st 2010.
   - Parameters:
     - Period : An integer depicting how many quarters in the future we are forecasting the surprise. If the horizon is 0, we are forecasting the surprise on the current quarter consensus. If the horizon is 1, we're forecasting what the surprise will be on the following quarter, ...
   - Fields:
     - TR.RevenueMean : The consensus revenue for the quarter in USD (on a given day)

  - TR.RevenueMean.periodenddate : The end date of the period considered
  - TR.RevenueMean.fperiod : The period as a string (FY2002Q4)
  - TR.RevenueMean.calcdate : The date at which the consensus is computed (Also referred to as *Calc Date* or *Forecast date*).
- Frequency : 1 per day

A.2 List of features used in modeling

Revenue based features

- Revenue - Actual 0Y1Q - Shift
- Revenue - Actual 0Y2Q - Shift
- Revenue - Actual 0Y3Q - Shift
- Revenue - Actual 0Y4Q - Shift
- Revenue - Actual 1Y0Q - Shift
- Revenue - Actual 1Y1Q - Shift
- Revenue - Actual 1Y2Q - Shift
- Revenue - Actual 1Y3Q - Shift
- Revenue - Actual 1Y4Q - Shift
- Revenue - Actual 2Y1Q - Shift
- Revenue - Actual 2Y2Q - Shift
- Revenue - Actual 2Y3Q - Shift
- Revenue - Actual 2Y4Q - Shift
- Annualized Growth increase Revenue - Actual 1Y1Q - Shift
- Annualized Growth increase Revenue - Actual 1Y2Q - Shift
- Annualized Growth increase Revenue - Actual 1Y3Q - Shift
- Annualized Growth increase Revenue - Actual 1Y4Q - Shift
- Annualized Growth increase Revenue - Actual 2Y1Q - Shift
- Annualized Growth increase Revenue - Actual 2Y2Q - Shift
- Annualized Growth increase Revenue - Actual 2Y3Q - Shift
- Annualized Growth increase Revenue - Actual 2Y4Q - Shift

Revenue Surprise based features

Those features are based on the Revenue based features but also on the Consensus computed at the given *Calc Date*.

- Expected_suprise Annualized Growth increase Revenue - Actual 1Y1Q - Shift
- Expected_suprise Annualized Growth increase Revenue - Actual 1Y2Q - Shift
- Expected_suprise Annualized Growth increase Revenue - Actual 1Y3Q - Shift
- Expected_suprise Annualized Growth increase Revenue - Actual 1Y4Q - Shift
- Expected_suprise Annualized Growth increase Revenue - Actual 2Y1Q - Shift
- Expected_suprise Annualized Growth increase Revenue - Actual 2Y2Q - Shift
- Expected_suprise Annualized Growth increase Revenue - Actual 2Y3Q - Shift
- Expected_suprise Annualized Growth increase Revenue - Actual 2Y4Q - Shift

Capital Expenditure based features

- Capital Expenditures - Actual 0Y1Q - Shift
- Capital Expenditures - Actual 0Y2Q - Shift
- Capital Expenditures - Actual 0Y3Q - Shift
- Capital Expenditures - Actual 0Y4Q - Shift
- Capital Expenditures - Actual 1Y1Q - Shift
- Capital Expenditures - Actual 1Y2Q - Shift
- Capital Expenditures - Actual 1Y3Q - Shift
- Capital Expenditures - Actual 1Y4Q - Shift
- Capital Expenditures - Actual 2Y1Q - Shift
- Capital Expenditures - Actual 2Y2Q - Shift

- Capital Expenditures - Actual 2Y3Q - Shift
- Capital Expenditures - Actual 2Y4Q - Shift
- yoy_growth Capital Expenditures - Actual 1Y1Q - Shift
- yoy_growth Capital Expenditures - Actual 1Y2Q - Shift
- yoy_growth Capital Expenditures - Actual 1Y3Q - Shift
- yoy_growth Capital Expenditures - Actual 1Y4Q - Shift
- yoy_growth Capital Expenditures - Actual 2Y1Q - Shift
- yoy_growth Capital Expenditures - Actual 2Y2Q - Shift
- yoy_growth Capital Expenditures - Actual 2Y3Q - Shift
- yoy_growth Capital Expenditures - Actual 2Y4Q - Shift

Market Cap based features

- Company Market Cap
- Company Market Cap - 1Y0Q - Shift
- yoy_growth Company Market Cap - 1Y0Q - Shift

Stock Price based features

- Price Close
- Price Close - 1Y0Q - Shift
- yoy_growth Price Close - 1Y0Q - Shift

EBIT based features

- EBIT - Actual 0Y1Q - Shift
- EBIT - Actual 0Y2Q - Shift
- EBIT - Actual 0Y3Q - Shift
- EBIT - Actual 0Y4Q - Shift

- EBIT - Actual 1Y1Q - Shift
- EBIT - Actual 1Y2Q - Shift
- EBIT - Actual 1Y3Q - Shift
- EBIT - Actual 1Y4Q - Shift
- EBIT - Actual 2Y1Q - Shift
- EBIT - Actual 2Y2Q - Shift
- EBIT - Actual 2Y3Q - Shift
- EBIT - Actual 2Y4Q - Shift
- yoy_growth EBIT - Actual 1Y1Q - Shift
- yoy_growth EBIT - Actual 1Y2Q - Shift
- yoy_growth EBIT - Actual 1Y3Q - Shift
- yoy_growth EBIT - Actual 1Y4Q - Shift
- yoy_growth EBIT - Actual 2Y1Q - Shift
- yoy_growth EBIT - Actual 2Y2Q - Shift
- yoy_growth EBIT - Actual 2Y3Q - Shift
- yoy_growth EBIT - Actual 2Y4Q - Shift

EBITDA based features

- EBITDA - Actual 0Y1Q - Shift
- EBITDA - Actual 0Y2Q - Shift
- EBITDA - Actual 0Y3Q - Shift
- EBITDA - Actual 0Y4Q - Shift
- EBITDA - Actual 1Y1Q - Shift
- EBITDA - Actual 1Y2Q - Shift
- EBITDA - Actual 1Y3Q - Shift
- EBITDA - Actual 1Y4Q - Shift
- EBITDA - Actual 2Y1Q - Shift
- EBITDA - Actual 2Y2Q - Shift
- EBITDA - Actual 2Y3Q - Shift
- EBITDA - Actual 2Y4Q - Shift
- yoy_growth EBITDA - Actual 1Y1Q - Shift
- yoy_growth EBITDA - Actual 1Y2Q - Shift
- yoy_growth EBITDA - Actual 1Y3Q - Shift
- yoy_growth EBITDA - Actual 1Y4Q - Shift
- yoy_growth EBITDA - Actual 2Y1Q - Shift
- yoy_growth EBITDA - Actual 2Y2Q - Shift

- yoy_growth EBITDA - Actual 2Y3Q - Shift
- yoy_growth EBITDA - Actual 2Y4Q - Shift

GoodWill based features

- Goodwill - Actual 0Y1Q - Shift
- Goodwill - Actual 0Y2Q - Shift
- Goodwill - Actual 0Y3Q - Shift
- Goodwill - Actual 0Y4Q - Shift
- Goodwill - Actual 1Y1Q - Shift
- Goodwill - Actual 1Y2Q - Shift
- Goodwill - Actual 1Y3Q - Shift
- Goodwill - Actual 1Y4Q - Shift
- Goodwill - Actual 2Y1Q - Shift
- Goodwill - Actual 2Y2Q - Shift
- Goodwill - Actual 2Y3Q - Shift
- Goodwill - Actual 2Y4Q - Shift
- yoy_growth Goodwill - Actual 1Y1Q - Shift
- yoy_growth Goodwill - Actual 1Y2Q - Shift
- yoy_growth Goodwill - Actual 1Y3Q - Shift
- yoy_growth Goodwill - Actual 1Y4Q - Shift
- yoy_growth Goodwill - Actual 2Y1Q - Shift
- yoy_growth Goodwill - Actual 2Y2Q - Shift
- yoy_growth Goodwill - Actual 2Y3Q - Shift
- yoy_growth Goodwill - Actual 2Y4Q - Shift

Net Income based features

- Net Income - Actual 0Y1Q - Shift
- Net Income - Actual 0Y2Q - Shift
- Net Income - Actual 0Y3Q - Shift
- Net Income - Actual 0Y4Q - Shift

- Net Income - Actual 1Y1Q - Shift
- Net Income - Actual 1Y2Q - Shift
- Net Income - Actual 1Y3Q - Shift
- Net Income - Actual 1Y4Q - Shift
- Net Income - Actual 2Y1Q - Shift
- Net Income - Actual 2Y2Q - Shift
- Net Income - Actual 2Y3Q - Shift
- Net Income - Actual 2Y4Q - Shift
- yoy_growth Net Income - Actual 1Y1Q - Shift
- yoy_growth Net Income - Actual 1Y2Q - Shift
- yoy_growth Net Income - Actual 1Y3Q - Shift
- yoy_growth Net Income - Actual 1Y4Q - Shift
- yoy_growth Net Income - Actual 2Y1Q - Shift
- yoy_growth Net Income - Actual 2Y2Q - Shift
- yoy_growth Net Income - Actual 2Y3Q - Shift
- yoy_growth Net Income - Actual 2Y4Q - Shift

Other features

- day_id : A unique day identifier (distance in days from 2000-01-01). Used mainly for the metric computation (day by day)
- naive_growth : The naive growth based on the latest available year on year growth.
- naive_surprise : The naive surprise based on the latest available year on year growth and the consensus growth value.
- yoy_consensus_growth : The expected growth according to the consensus.

Abstract :
Quantitative investing refers to the action of trying to optimize the return of an actively managed portfolio on the stock market. It has been shown that using pure machine learning models to forecast the return of stock prices directly is a very hard task. Different mechanism have been tried to circumvent that issue. This thesis focuses on the forecasting of fundamental values. More specifically, we are forecasting the surprise on the consensus of a given fundamental value. Indeed, it has been shown that this surprise can be used to create high returns investment strategies. A few papers look at forecasting the surprise directly, even if what matters in the end is the ranking of several surprises on a set of companies. Therefore, in this thesis, we try to use ranking-specific objective functions for machine learning models that try to rank the surprises on a set of given companies. We show that some improvement on the metric can be obtained using this strategy, leading to potential higher returns on the portfolio.

Résumé :
L'investissement quantitatif est défini comme étant l'optimisation du rendement d'un portefeuille d'actions, géré de manière active, sur base de signaux statistiques. Différents papiers montrent que prédire le rendement futur d'une action est une tâche ardue. Afin d'éviter ce problème, diverses approches ont été tentées. Dans ce travail, nous nous concentrons sur la prévision de surprise de valeurs fondamentales. L'objectif est de prédire le mieux possible la surprise qu'il y aura sur le consensus correspondant à une valeur fondamentale spécifique. Cependant, actuellement, la plupart des expériences utilisent des modèles dont l'objectif est de prévoir la valeur de la surprise et non pas l'ordre relatif des surprises entre différentes sociétés. Nous montrons dans ce papier qu'en utilisant des fonctions objectives spécifiques aux problèmes de classement il est possible d'améliorer la métrique d'évaluation de performance de prévision, ce qui pourrait permettre d'obtenir un meilleur rendement sur la stratégie d'investissement active.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Louvain School of Management
Place des Doyens, 1 bte L2.01.01, 1348 Louvain-la-Neuve
Boulevard Emile Devreux 6, 6000 Charleroi, Belgique
Chaussée de Binche 151, 7000 Mons, Belgique
www.uclouvain.be/lsm