LUISS

Degree Program in Economics and Finance

Course of Asset Pricing

Forecasting Financial Returns using Machine Learning Methods

Prof. Nicola Borri

SUPERVISOR

Prof. Megha Patnaik

CO-SUPERVISOR

Charbel Khazen

CANDIDATE

Academic Year 2022/2023

742991

Forecasting Financial Returns using Machine Learning methods

Charbel Khazen

Abstract

This paper examines high dimensional factor modeling using Machine Learning methods. It first delves into asset pricing theory for return forecasting and makes a case for utilizing machine learning methods in high-dimensional factor modeling. This paper thoroughly explains the mathematics and reasoning behind Ridge Regularization , Lasso Regularization, Elastic Net Regularization, Principal Component Regression, Independent Component Analysis, Generalized Additive Models, Standard Regression Trees, Boosted Trees and Random Forests. These methods, employed for stock return forecasting on high-dimensional factor models, are examined in relation to theoretical discussions and compared with traditional low-dimensional factor models. Empirical results demonstrate that Machine Learning Methods yield performance on par with, if not superior to, robust low-dimensional factor models. In ascending order, the most performing models are: Random forests, Principal Component regression , Elastic net Regression, Regression using Independent Component Analysis and finally Generalized additive models.

Keywords: Asset Pricing Theory, Statistical Learning Theory, Ridge Regularization, Lasso Regularization , Elastic Net Regularization, Principal Component Analysis, Independent Component Analysis, Information theory, Robust Estimation Methods, Generalized Additive Models, Regression Trees, Cost Complexity Pruning, Boosted Trees, Random Forests, Resampling Methods, Accelerated Proximal Gradient Descent

The paper is structured as follows: **Section 1** explores the theoretical foundations of factor models, beginning with the assumption of a stochastic discount factor and demonstrating the relevance of Machine Learning methods in this context. **Section 2** discusses the methodology used in this paper. The models rely on both theoretical and practical assumptions. I explain how my models are built and what are their limitations. **Section 3** delves into Statistical Learning theory to explain the rationale underlying Regularization and Dimension reduction techniques. **Section 4** explains $l1$, $l2$ and Elastic Net Regularizations. I argue that these methods are essential to solve both issues of overfitting and ill posedness. This section draws on concepts from statistical learning theory and linear algebra, providing a comprehensive understanding of each method's mathematical properties. **Section 5** explains both Principal Component Analysis and Independent Component Analysis. I derive PCs and ICs mathematically, discuss the properties of both methods and compare them. **Section 6** introduces the Huber Loss function as a robust alternative to the standard quadratic loss. **Section 7** discusses the Accelerated Proximal Gradient Descent Method and explains its underlying mathematical foundations. **Section 8** focuses on Generalized Additive Models, detailing their logic, properties, and comparing them with different non-linear models. **Section 9** explains regression trees, their theoretical foundations, discusses cost complexity pruning, and argues for the need of ensemble methods. I discuss both Boosted trees and Random Forests.In **Section 10** predictive metrics such as out-of-sample R-squared as well as the out of sample Mean squared error are explained. **Section 11** covers resampling methods, reviewing standard techniques and proposing a method best suited for the data. **Section 12** delves into empirical results, presenting data, comparing method performances, and benchmarking against three low-dimensional factor models. The commented code can be found in the Technical Appendix.

Contents

1 Introduction - Underlying theory and rationale

1.1 Theoretical Framework

Theoretical asset pricing relies on factor modeling to forecast returns. In this paper, I will first present the underlying logic behind these models, then I will argue for the need of machine learning methods to get more efficient results. First, much of Asset Pricing theory is based on the assumption of the law of one price, that is, the premise that all portfolios of securities with the same payoff should have the same price [1]. This is a weak assumption from an economical point of view - as it is a much less restrictive than classical utility functions assumptions for instance - yet, it entails the presence of a payoff pricing functional, i.e. a function that takes as input the payoffs [2] and returns a price in $\mathbb{R}$ [3] . LeRoy et al.[13]define the pricing functional $p : \mathbb{R}^s \to \mathbb{R}$ such that $p(X) = p$; for s the number of different states of nature; $\mathbb{X}$ a subspace of $\mathbb{R}^s$ spanned by the payoffs x and p the price of a portfolio. Notice how $\mathbb{X}$ doesn't span the whole space $\mathbb{R}^s$ as markets are assumed incomplete i.e. investors cannot buy (or synthesize) any contingent claims, and hence the payoff space is constrained; Rendering the model more realistic [4]. In addition, $p(X)$ is assumed to be linear: i.e for two payoffs x and x' we have $p(\alpha x + \gamma x') = \alpha p + \gamma p'$ for any α,$\gamma \in \mathbb{R}$.

These loose assumptions are the building blocks of asset pricing factor modeling, as they guarantee the existence of a stochastic discount vector (SDF). In fact, by the Riesz Representation theorem: for a linear functional $L(.)$ on some Hilbert space $\mathcal{H}$, $\exists v \in \mathcal{H}$ such that $\forall u \in \mathcal{H},\ L(u) = \langle u, v \rangle$. Riesz theorem applies in our framework: given that $p(X)$ is a linear functional and $\mathbb{E}[XY]$ (for some some random X, Y in the vector space) is an inner product[5], then, for $p(X)$ s.t $p : \mathbb{R}^s \to \mathbb{R}$ then $\exists$ a unique $m \in \mathcal{X}$ which is defined as the stochastic discount factor , such that $p(X) = \langle x, m \rangle = \mathbb{E}[mx]$. Simply stated we can represent the linear pricing functional by an inner product with a unique vector, the SDF. This fact is more evident when illustrated geometrically:

[1] With a sufficient and necessary condition that every portfolio with no payoff has a price of zero

[2] At all different states of nature

[3] A functional is a mapping from some vector space to real numbers.

[4] To understand this ; here is a concrete example: Assuming two states of natures; we can imagine a simple market in which there is a single security that either pays 1.1 or .9. The payoff space is "constrained" in the sense that we cannot synthesise the security to get any payoff we want : we either get 1.1 or .9 or some scaled version of them; the codomain $\mathbb{X}$ is thus a 1 dimensional plane defined by a scaled version of the payoff in this state space in R^2. This logic is generalizable to realistic markets by assuming high dimensional state space representations.

[5] $\mathbb{E}[XY]$ is an inner product as it respects the three defining axioms of inner products: symmetry, bi-linearity and positive definiteness)

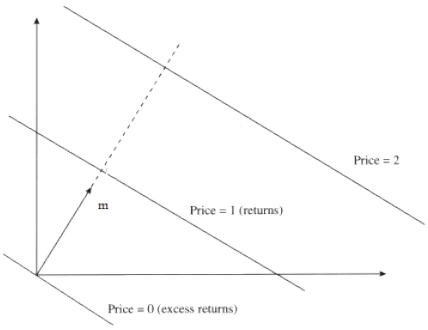


Figure 1: Geometric proof of the existence of a unique SDF m s.t. $E[mX] \equiv p(X)$. To prove it we visualize the SDF in the payoff space $R^{\mathbb{X}}$ - Not to be confused with the space of states of nature R^s

Geometrically (see Figure 1 [3]), This equivalence between $E[mX]$ and $p(X)$ can be represented in a state space representation in $\mathbb{X}$, the span of payoffs in an incomplete market [6]. I will consider for simplicity a two-dimensional payoff plane $\mathbb{X}$, but the same logic is generalizable to any dimension of X. Since each price is represented by a linear function of their corresponding payoffs (Due to the linearity of $p(.)$), we can draw parallel prices hyperplanes [7] linearly in $R^{\mathbb{X}}$ .We pick some vector m orthogonal to the price hyperplanes; then for any payoff belonging to some price plane $= a$, s.t. a represents some fixed price, the inner product between any payoff on p $= a$ and m is the same [8] .Formally: $< m, x_{\text{on price hyperplane a}} > = cst_{\text{for each hyperplane plane a}}$ implies the existence of a vector m orthogonal to all the price hyperplanes in $R^{\mathbb{X}}$ such that $E[mX]$ and $p(X)$ are equivalent.

In addition to the existence of a unique SDF, assuming the absence of arbitrage (along the law of one price) we get that the stochastic discount factor is always positive. In fact, $p(x) = \mathbb{E}(mx) = \int m(s)x(s)\pi(s)ds$ for s the number of States of Nature and $\pi(s)$ the probability of some state of nature. Assuming that prices and payoffs are always positive, and since probabilities are always positive; then $m(s)$ is always positive $\forall$s .

Importantly, one can derive from $p(x) = \mathbb{E}(mx)$ - Also called the "Euler" (or "basic") model - a single Factor model by Simple algebraic manipulations[9] such that

$$E(R^i) - R^f = \beta_{\text{i,m}}\lambda_m \tag{1}$$

That is, the expected premium return for each security is determined by a price risk λ_m ,common to all securities, and the regression coefficient from regressing this security's return on some stochastic

[6] which is a subset of R^s

[7] Or "planes" - like in Figure - depending on the dimensions of the payoff space.

[8] This is implied by construction and is seen clearly in the figure above, since the inner product between x and m is $|\text{proj}\,(x \mid m)| \times |m|$

[9] The payoff R corresponds to a unit price; One can thus write the Euler equation and decompose it as such $1 = \mathbb{E}(mR) = \text{cov}(m, R) + \mathbb{E}(m)\mathbb{E}(R)$. We divide both sides by $E[m]$ and get $\mathbb{E}(R) = \frac{1}{\mathbb{E}(m)} - \frac{\text{cov}(m,R)}{\text{var}(m)} \cdot \frac{\text{var}(m)}{\mathbb{E}(m)}$ with $R^f = \frac{1}{E(m)}$ and with $\frac{\text{Cov(m,R)}}{\text{Var(m)}}$ the linear regression coefficient from regressing R on m with both R and m demeaned variables.

discount factor m.

This approach to asset pricing this Powerful**:** By assuming the law of One price and the absence of arbitrage, which are mild assumptions from an economical perspective, we were able to derive the Euler equation and represent expected Returns with respect to some unique stochastic discount Factor as a single Factor model. This is also generalizable to multiple Factor models by assuming factors which are linearly related to the SDF. Hence, for any vector of factors f s.t. $m = a + b'f \iff \mathbb{E}\left(R^i\right) = R^f + \lambda'\beta i$ With β_i Coefficient vectors of the regression of R^ion f . This is the underlying Theory upon which Factor models are built and Asset Pricing Theory (APT) is essentially concerned with constructing an adequate SDF and relating it to the data. In other words, APT begins with the following assumption $p(x) = \mathbb{E}(mx)$, and ultimately aims to comprehend how $m = f(\text{data})$.

"Traditionally", economically motivated stochastic discount factors (and hence Factor models) obey to two different logics: The Arbitrage Pricing Theory And the General Equilibrium logic. They both differ in their economic assumptions and their inspiration for factors. From one side, general equilibrium models are absolute pricing models derived by expressing the stochastic discount factor with respect to marginal utility - for some assumed $u(.)$ - and restricting the model by imposing economic assumptions [10]such that the SDF is a linear function of some factor(s). These models are generally sub models of the broad Intertemporal Capital Asset Pricing model. On the other hand, APT models are relative asset pricing models derived by picking factors that have a small $R2$ when regressed on return. The APT relies on the assumption of an upper bound on the Sharpe ratio, suggesting that a small $R2$ of the regression of expected return on the picked factors implies that the intercept is small and hence that the factors are well specified, implying a factor model. [11] However, ultimately, both logics express the same representation of Returns as linear function of factors. These economically motivated models do not perform well out of sample for evident reasons[12] .This phenomenon is extensively documented. For instance, Fama French disproved ICAPM models by introducing new variables to the model and testing their coefficients; Mehra and Prescott (1985) [17] demonstrated that the widely used consumption-based models (i.e. what I define as "General equilibrium logic" models) do not align with empirical data. Additionally, APT models also exhibit out-of-sample failure. Rapach and Zhou (2013)[21], in their survey on return predictability, clearly highlight the shortcomings of well-known factor models in out-of-sample tests. Furthermore, Bossaerts and Hillion (1999) [1]and Goyal and Welch (2003, 2008)[31] [6]argued that, despite the high in-sample forecastability of some well-known factor models, many popular models fail to outperform naive benchmarks on out-of-sample data. Roll

[10]Those are assumptions on wealth, on time periods, on salary etc...

[11]The ICAPM is considered to be an "absolute" model because it provides a framework that explains the price of any security, while the APT is "relative" because it lacks a fundamental rationale for all assets as it explains returns relative to other returns (without explaining them).

[12]Economic theory is restrictive. And if not tested out of sample, we will always be able to find some model that mimicks the data; but this is not forecasting anymore

(1977)[22] formalized another issue with economically motivated factors in his critique of the validity of empirical tests, arguing that factors are equivalent to mean-variance frontier returns, and that it is always possible to construct overfitting models by picking in-sample mean-variance efficient returns. In other words, many commonly used factor models do not generalize well to new, unseen data; as they tend to overfit on the training data.

1.2 Necessity for Machine Learning Methods

Because economic Theory imposes restrictive and non-realistic assumptions, there has been a growing literature that rather focuses on a pure linear algebra interpretation of stochastic discount factors (and Factor models in general) . The initial problem remains the same from this perspective, the Euler equation remains the foundational structure of the model, but, the stochastic discount factors' construction is tackled differently. Accordingly, many "modern" academic asset pricing papers study the problem as a linear algebra problem and interpret the Stochastic discount factor as factor loadings accordingly: Principal component analysis for example - among other dimension reduction techniques - has been extensively documented in asset pricing literature: Stock and Watson (2002) [24] use principal component to summarize the number of macro economic predictors in macroeconomic forecasts, Ludvigson and Ng (2007)[15]use both Factor Analysis in correspondence with Principal Component Analysis to summarize the feature space, and gets statistically significant out-of-sample results and Nagel (2021)[19], for example, argues for the need of dimension reduction methods for better out of sample performance... References to dimension reduction techniques in empirical asset pricing academia are ubiquitous, reflecting the abundance of dimension reduction methods (and candidate regularization methods associated to each of them). Hence, Machine learning emerges in factor modeling though the prism of linear algebra.

In addition, because of the growing availability of huge financial data, machine learning emerges also as a practical alternative to standard classical models which fail statistically under high dimensional factor modelling . By introducing l_p regularization, dimension reduction and non linearity to the standard factor modeling framework, the model is now able to adapt to high dimensional data and hence incorporate complex dynamics. Accordingly, many academicians tackled the problem as such: Lewellen (2015)[14], for instance, uses 15 features, Freyberger et al. (2020)[5] use 36, Gu et al. (2020b) [9] use approximately a thousand factor and Nagel (2021)[19] argues that the adoption of high-dimensional factor modeling is inevitable and highlights the growing prevalence of factors in literature as evidence supporting this trend.

In addition, machine learning methods, focus on practical out of sample measures to evaluate a model, this is in contrast with standard asset pricing testing, which rely on in-sample goodness of fit

tests. Typically, for a generic factor model, $E[R^{\text{ei}}] = \beta_{im}\lambda_m$, one first runs a time series regression to find the beta, regresses the expected excess return on the beta, and tests the significance of the model using GLS, Wald...or some other in sample test. This typical Procedure, is not robust, as it neglects the out of sample performance; while the essence and strength of machine learning lie in its ability to provide reliable out-of-sample performance metrics through the utilization of diverse resampling methods.

2 Methodology

Thus, for all these reasons, Machine learning methods emerge naturally when building predictive models. In this paper, I build my predictive models upon the Euler equation using different machine learning methods. The exact framework I use is the following: $\mathrm{E}_t\left(r_{i,t+1}|X_t\right) = g\left(\beta_{i,t}\lambda_t\right)$ with $g(.)$ some flexible function (Note:I will refine this model at the end of this paragraph). This model incorporates Euler equation, but permits its generalization to non linear models. I build upon Euler equation as it is the least restrictive and thus the most adaptable to machine learning methods, and because - as I argued - economically interpreted asset pricing models fail in out-of-sample forecasting. This equation , is however , slightly different than the one presented formerly. In fact, the formula derived at 1 is an unconditional model for which the returns and the βs of asset i are independently sampled at each time from their respective distribution for each firm; I instead condition the equation using a large condition set without loss of generality[13] ; the conditioned Euler equation upon which I will work is rather : $\mathrm{E}\left[R_{i,t+1} \mid \mathcal{X}_t\right] = \beta_{i,t}\lambda_t$ with X_t the set of all conditioning information at time t, which I choose to be extensive. The particularity of this model is twofold : It is very flexible and conditions on a large set of information; rendering it ideal for machine learning applications. The information set I use contains both firm characteristics features and fundamental macroeconomic indicators; the covariance matrix is thus a combination of both types of factors ; hence, factors are written as: $X_{i,t} = (m_t) \otimes \mathbf{c_{i,t}}$; defined as the Kronecker product between the m_t , the macro variable and $c_{i,t}$ the characteristics for each firm i at time t . Conditioning on an information set is essential for building predictive models , as the returns at different times are not independent and identically distributed (as in unconditioned models) . I consider $g(.)$ to be a general function of the p-dimensional set of factors X_i, t, relating the Euler equation to different non linear non parametric models; notably Generalized additive models, and trees. This function is neither time dependent nor security dependent; it describes a general relationship that is always the same across the cross section and the time series, and provide stable estimations across the panel. Panel modeling used here, differs from the standard time series or cross sectional[14] models used in classical asset pricing, as it attempts to explain both the time variation and cross section of returns. It is relevant in the context of using machine learning methods, because exploiting different stock levels observations for different time stamps provides a vast number of observations to work with and introduces complex patterns, that can eventually be learned using machine learning methods. In addition, it is common practice to focus on returns rather than prices; because, assuming efficient

[13]Note: Passing from an unconditioned to a conditioned model is without loss of generality as the conditioned model is true $\forall$ time period; however, the converse is not true.

[14]Subtlety: many models labeled as cross-sectional in the literature are, in fact, panel data models. This conflation of panel modeling with cross-sectional analysis stems from historical reasons. Initially, "cross-sectional" analysis referred to unconditional modeling of single returns. Over time, this transitioned into a panel prediction problem, encompassing various returns. However, the term "cross-section" persisted. (Kelly, 2023) [11]

markets, prices can be seen as an $AR(1)$ process, and return as white noise [15] . In practice, the $AR(1)$ assumption often falls short, and the stationarity of returns is not guaranteed. Considerable attention has been devoted to this issue, with extensive literature exploring the optimal process models for prices from one side and the challenges of testing for stationarity from the other. However these topics are beyond the scope of this paper.I nonetheless utilize returns instead of prices in my models as a heuristic.

In addition, in reality, it is important to recognize that the underlying distribution of asset prices is very complex as prices are set by a colossal number of - both dependent and independent - real agents, each having they own preferences, reacting - both dependently and independently - to different sets of information. [16] Hence, even though the model employed in this paper is economically flexible and broad from an economical point of view - as it does not rely on equilibrium assumptions,or agent-related assumptions - This is not the case from a practical point of view. In fact, building a predictive model using a single $g(.)$ function that incorporates all the complexity of price dynamics, can be considered far fetched from a practical point of view. Hence, I do not expect my model to serve as a practical forecasting tool for practitioners; the primary aim of investigating this topic is to assess its empirical results and compare them to benchmark models commonly employed in the literature. Accordingly, I use "robust" factors documented by Lewellen (2015) [14] , to form my benchmark factor models. Those factors are robust in the sense that their statistical performance has been rigorously assessed through Fama-MacBeth Regressions across various time windows and against diverse models. Specifically, these factors consist of two distinct sets of firm characteristics, grouped differently, both exhibiting robust predictive power. Those factors encompass size, book to market ratio, working capital accruals, return on equity, asset growth, dividend to price ratio , growth in common shareholder equity, 36 and 12 month momentum, beta , return volatility, share turnover, sales to price and leverage. I review the benchmark models in detail in theEmpirical Analysis chapter.

In the upcoming sections of this paper, I introduce the foundational theory behind machine learning methods, specifically focusing on statistical learning theory. This will establish the theoretical basis for the statistical methods utilized in this paper. Following this, robust linear estimation is explained, and the discussion extends to $L_1$and L_2 regularization, clarifying their rationale in solving ill-posed problems and controlling over fitting, and discussing their relevance to my work. Subsequently, dimension reduction methods, such as Principal Component Analysis and Independent Component Analysis, are

[15]Considering prices as AR(1) processes , we can write$P_{t+1} = P_t + \epsilon_t$, with ϵ_t i.i.d. . $P_{t+1} - P_t$ is mean-independent, however, the magnitude of the percentage changes may increase with the level of the stock price, dividing, by P_t , gives us a stationary process. This is the formula for Returns $R_t = \frac{P_{t+1}-P_t}{P_t}$

[16]Taleb (2022)[26], in his work on the statistical consequences of fat tails, even posits that returns are not predictable as they exhibit fractal properties. This argument is even stronger than the Efficient Market Hypothesis which does not preclude the existence of return predictability (Rapach,Zhou[21]) . What Taleb actually suggest, is that, by exhibiting power law behavior, returns' essential statistical properties are not clear anymore

explored, emphasizing their significance and application in high-dimensional return prediction models. A detailed examination of Generalized Additive Models follows, succeeded by an exploration of Regression Trees, Gradient Boosted Trees and Random Forest and their application to this research. Then, I rigorously explain the optimization method employed in this paper , notably the accelerated Proximal Gradient Descent Method, as well as the resampling method used ; and finally conclude with an empirical analysis of the results.

3 Underlying Machine Learning Theory

3.1 Statistical Learning theory

Ideally, for all machine learning methods we minimize the empirical risk - hoping that it is a good surrogate for the real risk. Ultimately, one would like to get a strategy that generalizes best from training to testing data (By testing I mean, unseen independent data). That is, we do not want the model to overfit i.e. fit too well that it fails on new unseen data.

Statistical learning theory provides insights to these "over fitting"/"Generalization" phenomena in Machine learning by theorizing these concepts and by proposing solutions. In fact, there are three essential statistical learning metrics that determine Overfitting: On the first hand the "Generalization error" denoted $||\hat{r}(\hat{s}) - r(\hat{s})||_p$ for some strategy $\hat{s}$ and for some norm p, measures the difference between the in-sample and the out of sample loss for some strategy evaluated in-sample. On the other hand, statistical learning theory, defines two other metrics : The estimation and the optimization errors through risk decomposition. In fact, any risk associated to some strategy can be decomposed by simple algebraic manipulation. We can thus write the the "Estimation / Approximation" decomposition :

$$\underbrace{r\left(\hat{s}_{\text{erm}}\right) - r_0}_{\text{Excess risk}} = \underbrace{r\left(\hat{s}_{\text{erm}}\right) - r\left(s^*\right)}_{\substack{\text{Estimation} \\ \text{error}}} + \underbrace{r\left(s^*\right) - r_0}_{\substack{\text{Approximation} \\ \text{error}}} \tag{2}$$

With $\hat{s}_{\text{erm}}$ [17]the estimated strategy that empirically minimizes the loss on training data such that $\hat{s}_{erm} = \text{Arginf}_{s \in S}\, \hat{r}(s)$; s^* the strategy that minimizes the loss on unseen data for some hypothesis space $\mathbb{S}$ such that $s^* = \text{Arginf}_{s \in S_{all}}\, \hat{r}(s)$ and s_0 the strategy that minimizes the loss on unseen data on the whole hypothesis space such that:$s_0 = \text{Arginf}_{s \in S_{all}}\, r(s)$.

Note that the risk function $r(.)$ we have used in 2 is the true risk [18] - it is the loss evaluated on unseen data- and the decomposition formalizes the idea that learning is essentially about picking some strategy s belonging to some set of strategies $\mathbb{S}$ which is , in turn , a subset of the whole space of strategies: Explicitly, one can write : $s \in \mathbb{S} \subset \mathbb{S}_{all}$.[19]

Having presented these metrics; the generalization error [20] , the approximation error[21] and the estimation error [22] our goal is to ultimately minimize all three of them for the empirically minimized

[17]"Erm" is an abbreviation for Empirical Risk Minimization

[18]Not $\hat{r}(.)$ - The empirical one

[19]For example, if one chooses to modelize the data with a second order degree polynomial regression function and hence end up with some ERM fit: s is represents the ERM fit, which is contained in $\mathbb{S}$, the $P_2(x)$ polynomial, which in turn is contained in $\mathbb{S}_{all}$, the large hypothesis space of all $P_N(x) : n > 2$.

[20]The distance between model's risk evaluated on some sample and the expected value of the risk

[21]i.e. How far is the best model in a chosen hypothesis class from the true best hypothesis

[22]i.e. How far is the best model to the estimated model in some chosen hypothesis space

strategy to serve as an effective proxy to the true strategy. We thus want to study the dynamics of all three of them. From the one hand, it is easy to deduce that, the approximation error, $(r(s^*) - r_0)$ is inversely related to the complexity: That is because for $s_1^* \in S_1 \& s_2^* \in S_2$ with $S_1 \subset S_2$ (i.e. S_1 belongs to a more complex hypothesis class), $r\left(s_1^*\right) \geq r\left(s_2^*\right)$, i.e. the optimal risk can only reduce or stay the same when exposed to new strategies.Hence, we minimize the approximation error by increasing model's complexity.

However, understanding the dynamics of both the generalization error and the estimation is not trivial. Unlike the approximation error, it is not clear how a broader class of hypothesis would affect both errors (We can individually study the dynamics of some risk associated to some strategy - this was done for the approximation error whereby only one risk matters, the other , r_o being fixed. However, studying the dynamics of the difference between the 2 varying risks is not trivial).

Classical statistical learning theory (Vapnik [29]) proposes to bound both errors measures in order to study their dynamics. The rationale behind the derivation of these bounds is the following: Probability theory offers many methods to derive bounds on deviations from expectations [23], which we express as $P(|x - \mathbb{E}(x)| \geq \varepsilon) \leq \alpha_x(n, \varepsilon)$ with α depending on the probabilistic distribution of X. We can generalize this to risk functions: $P(|\hat{r}(s) - r(s)| \geq \varepsilon) \leq \alpha_{\hat{r}(s)}(n, \varepsilon)$. The issue here is that the bound depends on a specific value of s. Accordingly, statistical learning theory derives bounds on $Max_{s \in S}|\hat{r}(s) - r(s)|$ (called "uniform bounds") thus getting $P\left(\max_{s \in S}|\hat{r}(s) - r(s)| \geq \varepsilon\right) \leq \alpha(n, \varepsilon)$; i.e. a bound that is not specific to one strategy. This deviation bounds both the generalization and the estimation error. Knowing α (i.e. studying the bound of this derivation) gives us an indication about the dynamics of both the estimation and the generalization error.

$$\text{Estimation error} \leq Max|\hat{r}(s) - r(s)| \leq \text{Some bound}$$

$$\text{Generalization error} \leq Max|\hat{r}(s) - r(s)| \leqslant \text{Some Bound}$$

Statistical learning theory derives the bounds with respect to different complexity measures. Notable examples are bounds with respect to $|H|$, the size of the hypothesis space (But this is not convenient as $|S| \longrightarrow \infty$ usually), the Vapnik–Chervonenkis dimension or even the Rademacher complexity (Mohri et al. 2012 [18]). Each bound has its own specificity (There are data dependent bounds, others are distribution dependent etc ...) but the general idea is that both the generalized and the estimation error are upper bounded by some measure of model's complexity.

Thus, from one hand the approximate error is minimized by reducing model's complexity, and from the other hand, the complexity limits how bad the estimation and the generalization errors can go.

[23] Starting from Markov's inequality, one can derive, under particular conditions, many different bounds such as Chebychev , exponential Markov (Chernoff) bounds etc...

Here, complexity is positively proportional to the bounds of the error. This result is very important as it establishes a theoretical "interpretation" of the estimation/approximation trade off [24],as well as the variance bias trade off .

These derived bounds constitute the theory that supports the dynamics of the approximation / estimation / bias / variance errors with respect to complexity: The models must be complex enough to get a low approximation error (or "bias" in regression terms) but not too complex that the complexity measure would increase the limiting upper bounds of the generalization and estimation error (or Variance). Notice here that the bounds constitute theoretical support for the Occam razor principal; i.e. the principal of parsimony that posits that the model should be as simple as possible.

The Variance-Bias trade off was not discussed in this paper, but its rationale and derivation method is similar to that of the estimation approximation error, yet , those are two different concepts. They are both derived by decomposition. The Variance bias decomposition is derived by breaking down the sample risk such that $\mathbb{E}[R(\hat{f}))] = \underbrace{\mathbb{E}_{xy}\left((y - \mathbb{E}_{y,x}(y))^2\right)}_{\text{Intrinsic Noise}} + \underbrace{\mathbb{E}_x\left[\mathbb{E}_D\left(\hat{f}_{(x)}\right) - \mathbb{E}_{y|x}(y)\right]^2}_{\text{Variance}} + \underbrace{\mathbb{E}_x\left[\mathbb{E}_D\left(\hat{f}_{(x)} - \mathbb{E}_D\left(\hat{f}_{(x)}\right)\right)^2\right]}_{\text{Bias}}$ (German et al. 1992). However, the variance bias decomposition is not equivalent to the Estimation/Approximation decomposition, further algebraic manipulations actually proves it (Refer to the variance bias decomposition section in appendix for a clear illustration of the difference between both decompositions).

This presented rationale derived from statistical learning theory provides the underlying theoretical background of the machine learning methods that I present.

[24] I purposely did not say "proof", but "interpretation", because upper bounds alone do not guarantee the trade off

4 L_1, L_2 and Elastic Net Regularizations

Given the underlying asset pricing theory explained above, linear models hold considerable relevance in the context of empirical asset pricing. I use them not only for theoretical reasons but also for practical applications. In fact, linear models are Taylor expansions of non linear parametric models. This is useful in a setting as complex as return forecasting: By acting as the most lenient approximation in a highly intricate setting, linear models emerge as optimal amidst unknown models. In the context of high dimensional factor modeling however, standard linear models are not adequate. Therefore, in this section, I introduce regularization methods tailored for linear models, allowing their application in high-dimensional settings. Regularization serves two main purposes: Preventing over fitting, and solving ill-posed problems. Both issues are fundamental in Returns forecasting given that we are searching for the model that generalizes best to unseen data using limited data [25] . I will tackle both approaches to regularization and propose three regularization methods Ridge, Lasso and Elastic Net to remedy the problem.

4.1 Regularization and Overfitting

As explained, finding the best predictive model is equivalent to finding the hypothesis space (or strategy) that generalizes best out of sample. Regularization introduces the idea of constraining the hypothesis space. Essentially, it defines a complexity measure $\Omega : F \longrightarrow [0, \infty)$ and restricts the hypothesis space to some level of complexity defined by $\Omega(F)$. Formally, For some hypothesis space F And a complexity measure $\Omega(F)$, we reduce the hypothesis place to a subset of hypothesis $F = \{f \in F \mid \Omega(F) \leqslant r\}$ For some level r of complexity. Complexity measures are defined as L_p Norms, such that

$$\begin{cases} l_0 \text{ complexity: } \Omega(f) = \text{ \# of Non zero coefficients} \\ l_1 \text{ 'Lasso' complexity: } \Omega(f) = \sum_{i=1}^{p} |w_i| \\ \text{Squared } l_2 \text{ 'Ridge' complexity: } \Omega(f) = \sum_{i=1}^{p} {w_i}^2 \end{cases}$$

Complexity, given a linear regression, is defined as the degrees of freedom of parameters. The more free to vary are the parameters, the more complex is the model and vice versa. Hence, constraining the hypothesis space by some complexity measure reduces the models complexity. Having defined the framework, regularization is defined as an Empirical Risk Minimization problem restricted by a subset of model's hypothesis space. Ivanov regularization (IR) Formalizes clearly this idea: For some complexity measure defined as an L_p norm and a tuning parameter $r > 0$ defining the complexity level. We define Ivanov regularization as : $\min_{f \in F} \left[\frac{1}{N} \sum_{i=1}^{N} l\left(y_i, f\left(x_i\right)\right) \right]$ such that $\Omega(F) \leq r$. By considering r as a hyper parameter, IR incorporates the models complexity to the minimization prob-

[25] Limited data will restrict our problem to an ill-posed one

lem: Not only empirical risk is minimized, but complexity is also adjusted so that model generalizes best on some validation set. A more common regularization is the Tikonov regularization (TR) : For some complexity $\Omega(F)$ defined by an L_p norm and a tuning parameter $\lambda \geq 0$, TR is defined as : $\text{Min}_{f \in F} \frac{1}{N} \sum_{i=1}^{N} l\left(y_i, f\left(x_i\right)\right) + \lambda\Omega(f)$. Tikhonov regularization is equivalent to Ivanov regularization for certain loss functions and for certain complexity measures. Importantly, equivalence holds for both Ridge and Lasso regressions(Oneto et al. [20]).IR gives a clear idea of what's happening under the hood, while, TR is an easier to solve, unconstrained, minimization problem.

In summary, Ridge and Lasso regressions represented by Tikhonov regularization, express the idea clearly formalized by Ivanov regularization: That is, finding the best empirical risk minimized model by cutting down the hypothesis space from $\mathcal{F}$ to $\mathcal{F}_r$; i.e. by reducing the complexity level. We now have a clear picture of how regularization controls overfitting:

Recall the approximation error $r\left(s^*\right) - r_0$ which is inversely proportional to changes in complexity; the estimation error $r(\hat{s}) - r\left(s^*\right)$ which, along the generalization error $\|\hat{r}(\hat{s}) - r(\hat{s})\|_p$ have positively proportional bounds with respect to complexity; by cutting down the hypothesis space and hence by controlling model's complexity, regularization "controls" overfitting by trading off generalization and estimation error for approximation error (Overfitting can thus never be completely eliminated) [26]

In forecasting returns and in predictive models in general, overfitting constitutes a fundamental issue; hence, regularization appears as a natural alternative to Simple ordinary least squares Factor modeling used traditionally.

4.2 Regularization and Ill-posedness

In addition to over fitting, ill-posedness is a customary issue in Factor modeling, i.e we almost always need to solve a problem where either the solution does not exist or the number of solutions is infinite. In fact, factor modelling is fundamentally an $Ax = b$ problem whereby A is generally not well determined. [27] Ill-posed systems are either due to over determined systems, those are typically the case in classical Factor modeling with P factors and N Returns such that P¡¡N; or due to under-determined systems: Those are typical in high dimensional machine learning applications whereby the number of features surpasses the number of observations; P¿¿N. In addition, Ill-posedness may also be the result of (Multi)-Collinearity or Near-(Multi)collinearity in the design matrix when two or more columns (or rows) vectors are linearly dependent (or nearly linearly dependent). Note: an

[26] Along this conclusion we also deduce from statistical learning theory that as the number of observations N increase we want less and less regularization. In fact, regularization increases approximation error and decreases estimation error; the speed at which approximation errors changes is independent of $N \Rightarrow O(1)$ while the estimation and generalization error bound depends on N (Assuming Rademacher complexity), we find that the bound $\leq O\left(\sqrt{\frac{\lg n}{n}}\right)$ Thus, one should regularize less as N increases.

[27] Note, in the context of factor modeling A represents the design Matrix - or 'the factor Matrix- , b the returns vector and X the coefficients.

underdetermined system implies multicollinearity in the design matrix [28] ; however, Multicollinearity does not strictly reduce to an underdetermined system it also may result in an overdetermined one [29]. Importantly: Multicollinearity implies ill-posedness of the system (See appendix for more information about multicollinearity evaluation measures).

Classical asset pricing is mainly concerned with the issue of over determined systems, this paper however deals with both over and under determined scenarios as high dimensional Factor models are explored.

Typically, from a classical linear algebra point of view, ill-posed problems are solved by finding the x that minimizes$||Ax - b||_2$. Laub (2005) [12] provides the general solution to this least squares problem as such: For $A \in \mathbb{R}^{N\times P}\&b \in \mathbb{R}^{N\times K}$, the solution to the least squares problem is:

$$X_{LS} = A^{+}b + \left(I - A^{+}A\right)Y \tag{3}$$

for some arbitrary vector $Y \in \mathbb{R}^{\mathbb{P}\times\mathbb{K}}$ and with A^{+} the Moore-Penrose Pseudo Inverse of A. Hence, we can always find a solution x_{LS} For any $Ax = b$ problem as a function of the pseudo inverse. The pseudo inverse is a generalization of the two-sided inverse [30]that applies on any Matrix whether singular or rectangular or Multicollinear etc... It is not exactly an inverse but its properties resemble that of an inverse [31] and is defined as $A^{+} = V\begin{pmatrix} S^{-1} & 0 \\ 0 & 0 \end{pmatrix}U^{\top}$ - which is obtained by exploiting the Singular Value decomposition of A [32] . By employing the pseudo inverse one can always find a unique (sometimes approximate) solution for ill-posed problems: In fact, using 3 we will either get a unique solution. Or infinitely many solutions and accordingly choose the minimum norm solution. For instance for some under determined system $Ax = b$ we get $X_{LS} = A^{+}b + (I - A^{+}A)Y$ infinitely many solutions, we can chose however a unique approximate solution: the minimum-norm solution by choosing $X_{LS} = A^{+}b$ such that $Y = 0$ [33] In this sense, by utilizing the Pseudo inverse, one can always get, an approximate unique solution no matter how ill-posed the system is: This is the standard

[28] If A $(N \times P)$ has $P >> N$, this implies multicollinearity by the rank nullity theorem. See Multicollinearity section in appendix for proof

[29] $\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ is an example of multicollinearity that reduces to an underdetermined system and $\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}x = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$ is an example of multicollinearity that reduces to an overdetermined system

[30] The two-sided inverse, is actually what we refer as the "inverse" in general. We define the 2-sided inverse of some matrix A as $G = A^{-1} : GA = AG = I$

[31] For some matrix A, those are, for instance, some of the main "inverse-like" properties of the Pseudo Inverse $G{=}A^{+}$:
$\begin{cases} AGA = A \\ GAG = G \\ (AG)^{T} = AG \quad \cdots\cdots \\ (GA)^{T} = GA \end{cases}$

[32] There are also closed forms for A^{+}.For example,for A having full row rank $A^{+} = A^{\top}\left(AA^{\top}\right)^{-1}$ and for A having full column rank $A^{\dagger} = \left(AA^{\top}\right)^{-1}A^{\top}$)

[33] See Least Squares Solutionappendix for graphical representation

approach to ill-posedness in linear algebra.

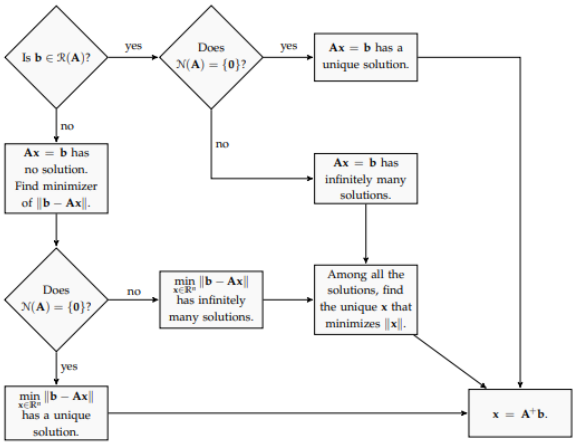


Figure 2: This Chart shows how using A^+ can always result in a unique (approximate) solution. This representation builds upon Strang's four fundamental subspaces of linear algebra to represent different reactions to ill-posedness.

Regularization as an alternative :

This "standard" method for solving ill-posed systems is ubiquitous [34] ; it however suffers from two major drawbacks. From the one hand, using the A^+ to compute the solution, does not directly address the issue that is causing ill-posedness (Multicollinearity for instance); but rather, solves the problem in a generic manner. Second, and most importantly, computing the Pseudo inverse can be computationally expensive for large matrices, as it requires computing the eigenvectors/eigenvalues of the matrix and the reciprocals of its singular values. For a large and sparse design matrix A, as it is in this paper, this may be practically very challenging. Hence, for interpretabilty and computational reasons, one ought to find alternatives to Pseudo-inverse; Regularization emerges here as a natural solution.

Instead of exploiting the singular value decomposition of A, Tikhonov regularization solves ill-posed problems numerically by controlling the norm of $||x||$ while minimizing $||Ax - b||_2$;and is formally defined as $\text{ArgMin}_x \, ||Ax - b||_2 + \lambda |||x||_p$. This method has also its limitations (which I discuss in the end of this section) but it is much less computationally expensive and more natural than the pseudo inverse solution. I will present three different expressions of the Tikhonov regularization and explain how I will use them in my paper:

First, the Ridge Regression (RR) is expressed as a Tikhonov regulatization with a squared l_2 normed penalty. Explicitly, Ridge is defined as: $\text{ArgMin}_x \, ||Ax - b||_2 + \lambda |||x||_2$, with λ a tuning parameter > 0

[34] Notably, many programming libraries, do incorporate this logic into their estimation algorithm for linear models. Hence, when fitting a linear model - with some ill-posedness resulting in under determined systems (for instance with a mutlicollinear matrix A , or using a design matrix with more features than observations...) we can still get a result. This solution is actually the approximate solution found by using the Moore Penrose pseudoinverse and picking the least norm solution.

,which translates in our linear regression framework into :

$$\hat{\beta}_{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

We have already seen in the previous section that - since Ivanov and Tikhonov regularizations are equivalent under ridge regression - we can express Ridge in a more "intuitive and interpretable form" :

$$\hat{\beta}_{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 \leq r$$

We can now clearly see how Ridge regularization essentially constrains the coefficients to some squared norm. Ridge regularization requires that the predictors are standardized in order to prevent an unfair shrinking between them. In fact, Ridge constrains the size of the coefficients using a squared l_2 norm: s.t. $\beta_1^2 + ... + \beta_p^2 < r$; if one (or more) of these variables is scaled differently than the others, this will be reflected in the magnitude of βs and induce unfair penalisation; we mitigate this problem by standardizing all factors. Consequently, the intercept is not penalized by ridge [35]. The estimation is a two steps procedure : After standardizing the predictors, one should first set the intercept as $\bar{y}$ then estimate the other coefficients by ridge penalization. This regularization method has a closed form [36] : $\hat{\beta}_R = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right)^{-1} \mathbf{X}^\top \mathbf{y}$. We can clearly see from the closed form solution how ridge mitigates ill posedness by adding to the Gramian matrix a diagonal matrix. For ill-posed systems due to multicollinearity, the Gramian matrix is singular, adding to it a diagonal element transforms it to an invertible matrix (Non-singular).

As explained Ridge solves ill-posedness, accordingly, one can prove algebraically, using the Singular value decomposition of X, that the Ridge regression is nothing more than some scaled version of the pseudo inverse solution which I introduced previously. Concretely, Setting aside algebraic manipulations [37] , ridge regression estimators can be written as $\hat{\beta}_{ridge} = \mathbf{V}\left(\mathbf{\Sigma}^2 + \lambda \mathbf{I}_n\right)^{-1} \mathbf{\Sigma} \mathbf{U}^\top \mathbf{Y}$; while, the "pseudoinverse" solution in the context of a linear regression is $\beta_{\text{pseudoinverse}} = X^+ Y = V\Sigma^{-1}U^T Y =$

[35] For some $\beta_0 + \sum \beta_j X_j$ regression model with standardized (more precisely centered) predictors we get $E[Y] = \beta_0$. Thus for estimation purposes, we do not penalize the intercept and we require it to be $\bar{Y}$

[36] The closed form solution of ridge regression is found by simply deriving with respect to β the objective function . This is feasible as the function is convex and involves a simple quadratic function

[37] Using Singular Value decomposition and simple algebraic and matrix manipulations; one can prove:

$\mathbf{V}\left(\Sigma^2\right)^{-1}\Sigma\mathbf{U}^\top\mathbf{Y}$ [38] . Comparing both formulae we clearly see that the ridge estimator is equivalent to a scaled pseudoinverse estimator by $\frac{\Sigma^2}{\Sigma^2+\lambda\mathbf{I}_n} \in [0,1]$. For $\lambda = 0$ both ridge and pseudoinverse coefficients are equivalent, and as the penalty term λ increases, the ridge coefficient converges to 0. In addition, expanding the last ridge estimator formula, $\hat{\beta}_{ridge} = \frac{\Sigma^2}{\Sigma^2+\lambda\mathbf{I}_n}\beta_{pseudoinv} = \left\{\frac{s_i^2}{s_i^2+\lambda}\right\}\beta_{pseudoinv}$ [39] one can infer the dynamics of the ridge regression:Ridge estimator shrinks as the squared singular values increase, and since the Since $\Sigma^2 = \left\{\frac{s_i^2}{s_i^2+\lambda}\right\}$ is the covariance matrix of the demeaned predictor X [40], and because its columns are indicative of the amount of variance within each principal component; Ridge regression is essentially, a regularization method that penalizes smoothly [41] low-variance Principal components directions.

The lasso regularization is a Tikhonov regularization method with a $l1$ normed penalization; which is defined as $\text{ArgMin}_x\, ||Ax-b||_2 + \lambda|||x||_1$. In the particular context of linear regression, lasso is formalized as : $\hat{\beta}_{\text{lasso}} = \underset{\beta}{\text{argmin}}\left\{\frac{1}{2}\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\right\}$. Or equivalently as an Ivanov regularization :

$$\hat{\beta}_{\text{lasso}} = \underset{\beta}{\text{argmin}}\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2$$
$$\text{subject to } \sum^{p}|\beta_j| \leq t.$$

The same rules of estimation apply on Lasso Regression. The estimation procedure is done in two steps : First we compute the intercept, then estimate the other coefficients. However, Lasso does not have a closed form solution, as the objective function is not differentiable. There are different numerical methods that solves this optimization problem, I propose, the accelerated proximal gradient descent method which I explain in details in the Numerical Methods chapter. Lasso Regression is widely used in asset pricing modeling primarily due to its inherent feature selection capability [42] making the model

$$\begin{aligned}
\hat{\beta}_{ridge} &= \left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p\right)^{-1}\mathbf{X}^\top\mathbf{Y} \\
&= \left(\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top + \lambda\mathbf{I}_p\right)^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{Y} \\
&= \left(\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top + \lambda\mathbf{I}_p\right)^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{Y} \\
&= \left(\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top + \lambda\mathbf{V}\mathbf{V}^\top\right)^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{Y} \\
&= \mathbf{V}\left(\mathbf{\Sigma}^2 + \lambda\mathbf{I}_n\right)^{-1}\mathbf{V}^\top\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{Y} \\
&= \mathbf{V}\left(\Sigma^2 + \lambda\mathbf{I}_n\right)^{-1}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{Y}
\end{aligned}$$

[38] We have discussed the pseudoinverse previously from a pure linear algebra point of view using Ax=b. In the context of linear algebra, nothing changes, but instead of writing A we write X,our factor matrix, and instead of x we write β the coefficients matrix and finally instead of b we write Y. Thus $x = A^+b$ is written as $\beta = X^+y$ (Refer to Figure R1 - for more information)

[39] For s_i^2 the singular values the "stretching" Sigma matrix

[40] Using the SVD of X ,X^TX/N, the covariance matrix is equal to $\mathbf{V}\mathbf{D}^2\mathbf{V}^T$ .Since, the covariance matrix is symmetric, then the V and D^2 matrices correspond to the covariance matrix's eigenvectors/values. This is an important result in Principal component analysis; whereby the eigenvector and eigenvalues of the covariance matrix of the demeaned X are indicatives of the PCs and their relevance; it is reviewed in the following chapters

[41] Penalty is smoothed by the tuning parameter λ

[42] Feature selection involves the elimination of certain features while retaining others

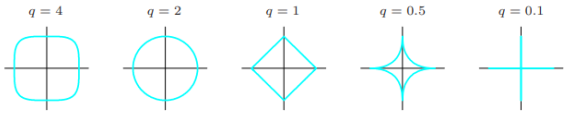


Figure 3: Illustration of different penalty contours $\sum_{j=1}^{p} |\beta_j|^q \leq 1$ for different values of q on a 2 dimensional parameter space. As q decreases the contour of the penalty promotes increasing sparsity in the coefficients.

more interpretable.

Specifically, Lasso Regression employs a least squares approach, effectively nullifying "irrelevant" coefficients[43]. One can interpret the sparsity of the result both algebraically and graphically. In fact, assuming a linear model with an l_1 penalization,the objective function of our problem is $\min y^T y - 2y^T x\hat{\beta} + \hat{\beta}x^T x\hat{\beta} + 2\lambda|\hat{\beta}|$. For $\hat{\beta} > 0$, the solution has a closed form $\hat{\beta} = \left(y^T x - \lambda\right) / \left(x^T x\right)$ which decreases as l_1 regularization tuning parameter λ increases until reaching $\hat{\beta} = 0$; and for $\hat{\beta} < 0$, the solution has also a closed form, $\hat{\beta} = (y^T x + \lambda)/(x^T x)$ which increases as the lasso regularization parameter increases, until $\hat{\beta} = 0$ is reached. We hence, ultimately expect sparsity when penalizing using lasso. This is contrary to the squared squared $l2$ normed ridge penalty which has the following closed form solution $\hat{\beta} = y^T x / \left(x^T x + \lambda\right)$ for both, $\beta <$ and > 0 ; the $\hat{\beta}$ here , however, does not decrease specifically to zero as regularization grows (Note: The coefficient does decrease, but not to zero as was the case for l_1 regularization). In addition, one can see that the sparsity of lasso coefficients arises from the distinctive shape of the l_1 penalty. This characteristic is a direct result of the contours of the l_1-normed penalty, where decreasing the degree of the l_p norm is associated with an increased expectation of sparsity.

By controlling the size of the coefficients, lasso regression does solve ill-posedness. However, it becomes unstable under perfect collinearity for two identical vectors in the design matrix. One can illustrate this result graphically in a two dimensional parameter space: Specifically, in the case of an ill-posed problem arising from multicollinearity, leading to an underdetermined system, the objective function is depicted as an "infinite" ellipsoid.

[43]Subtlety: In this context, "Irrelevance" does not imply "uninformative" or "statistically independent." Instead, it is defined within the framework of Lasso regression. Lasso regression may eliminate informative features if it perceives them as irrelevant to the least squares minimization objective.

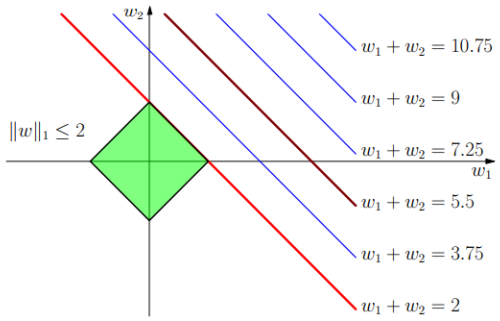


Figure 4: Mutlicollinearity might induce loss function's ellipsoid to degenerate wherein each point on the ellipsoid is now represented by a line

Parallel lines represent the loss function; which due to collinearity result in a "degenerate" ellipsoid (One can imagine a horizontally squished and infinitely elongated ellipsoid where each point is a now a line). Importantly this representation depicts instability under multicollinearity because the lasso penalty represented by a square , cannot solve for loss functions that are parallel to its contours (There are infinitely many solutions). This is coherent, since, for equivalent features, any variable selection method will arbitrarily select a matrix.

To resume, ridge regression shrinks the coefficients without performing feature selection while lasso regression results in variable selection, but may fail to provide stable solutions under particular cases of perfect multicollinearity. Hence, Elastic Net Regularization emerges as a middle ground solution between ridge and lasso. This method is a l_p normed regularization that combines both L_1 and L_2 penalties in an additive manner. Elastic Net regression is formally represented as: $\hat{\beta}_{\text{EN}} = \underset{\beta}{\text{argmin}}\{\frac{1}{2}\sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}\left(\rho\beta_j^2 + (1-\rho)\,|\beta_j|\right)$, with ρ a tuning parameter $\in [0,1]$. Intuitively, the dynamics of this method become apparent when conjecturing from the contour of the elastic net penalty in a two-dimensional parameter space representation (see Figure 5). These contours promote sparsity as they look like the l_1 squared penalty, yet, the slight curvature along their sides helps alleviate instability caused by multicollinearity, preventing the degenerate loss function from being parallel to its contours.

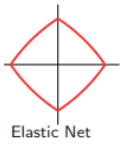


Figure 5: In a two dimensional coefficients space, the restricted Elastic Net penalty takes this form (Refer to Ivanov Regularization for an explanation of "restricted" penalties).

Thus from one side, by introducing absolute values, this penalization introduces sparsity and is a feature selection method; and from the other, the squared penalty assures the stability of the solution,

even under perfect multicollinearity with equal features. Like for lasso, and for the same reasons, elastic net regularization has no closed form solutions. I use Accelerated Proximal Gradient Descent to estimate the coefficients; I discuss the details of this method in the "Numerical Methods" chapter.

5 Dimension Reduction Methods

5.1 Principal Component Analysis

Principal components represent directions, in some vector space determined by some data. In Principal Component Analysis (PCA), we define the components to represent orthogonal - and hence uncorrelated - directions for which the data exhibits some level of variance that spans the data space. For instance, for some P-dimensional data; we can represent P directions [44] ordered by the amount of variance they capture. The first principal component corresponds to the direction of maximum variance, and each subsequent component captures orthogonal directions of decreasing variance.

Ultimately, principal component analysis is used as a dimension reduction technique by picking, among the PCs, the directions that exhibits most variance, and leaving out other directions.

Briefly, Principal Components estimation can be computed by following these three steps:

1. The initial step involves centering the design matrix. This is done for interpretational purposes and for simplicity, it is not necessary in practice [45].

2. Points are projected onto the first principal component, and the distance between the points and their projections is minimized to determine PC1.

3. Additional principal components are determined by selecting directions that are orthogonal to the initially found PC1. We iterate until the directions span the data space.

Mathematically, given some N-by-P design matrix, one can represent the principal components by some P-by-P dimensional unit vector $\vec{w}$ [46]; the points in the P-dimensional feature space are defined as $\overrightarrow{x_i}$; the projection of $\overrightarrow{x_i}$ on the PC is $(\vec{x}_i \cdot \vec{w})\,\vec{w}$ [47] and the distance between the point and its projection - also called the residual - is $\|\overrightarrow{x_i} - (\vec{w} \cdot \overrightarrow{x_i})\,\vec{w}\|^2$[48] . As mentioned, principal component analysis ultimately aims at minimizing the mean squared error of the residuals: $\text{MSE}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n} \|\vec{x}_i\|^2 - (\vec{w} \cdot \vec{x}_i)^2$;[49] Using matrix manipulations, one can prove that this mean squared residual

[44] That spans $\mathbb{R}^l$

[45] That is because centering X, allows us to interpret X^TX as the covariance matrix, which then permits the eigendecomposition of the X^TX matrix. I will discuss this later in the my analysis of PCA estimation

[46] $\vec{w}$ represent the directional vectors of the principal components. Its dimension is P-by-P as it represents P latent dimensions in some P dimensional feature space

[47] That is; we compute the dot product between $\overrightarrow{x_i}$ and $\overrightarrow{w}$, and since $\overrightarrow{w}$ is a unit vector, we get a scalar that represents the value on $\overrightarrow{w}$ which $\overrightarrow{x_i}$ gets projected to. To get the actual vector on which $\overrightarrow{x_i}$ gets projected, we multiply the inner product by the directional vector $\overrightarrow{w}$.

[48] The squared norm is used instead of the absolute value, because we want a differentiable residual

[49] That is because the residual can be reduced as such

$$\begin{aligned}
\|\overrightarrow{x_i} - (\vec{w} \cdot \overrightarrow{x_i})\,\vec{w}\|^2 &= (\overrightarrow{x_i} - (\vec{w} \cdot \overrightarrow{x_i})\,\vec{w}) \cdot (\overrightarrow{x_i} - (\vec{w} \cdot \overrightarrow{x_i})\,\vec{w}) \\
&= \overrightarrow{x_i} \cdot \overrightarrow{x_i} - \overrightarrow{x_i} \cdot (\vec{w} \cdot \overrightarrow{x_i})\,\vec{w} \\
&\quad - (\vec{w} \cdot \overrightarrow{x_i})\,\vec{w} \cdot \overrightarrow{x_i} + (\vec{w} \cdot \overrightarrow{x_i})\,\vec{w} \cdot (\vec{w} \cdot \overrightarrow{x_i})\,\vec{w} \\
&= \|\overrightarrow{x_i}\|^2 - 2\,(\vec{w} \cdot \overrightarrow{x_i})^2 + (\vec{w} \cdot \overrightarrow{x_i})^2\,\vec{w} \cdot \vec{w} \\
&= \overrightarrow{x_i} \cdot \overrightarrow{x_i} - (\vec{w} \cdot \overrightarrow{x_i})^2
\end{aligned}$$

minimization is equivalent to maximizing the variance of the projections $\widehat{\sigma^2}\left(\vec{w}\cdot\overrightarrow{x_i}\right)$ (i.e. the variance of the distance between the origin and the points' projection) ,which, in turn, is dependent on the covariance matrix $v = \frac{X^T X}{N}$ of the design matrix [50] [51] Principal Component Analysis can thus be mathematically summarized as performing:

$$\text{Argmax}_w w^T v w \quad \text{such that} \quad \mathbf{w}^T\mathbf{w} = 1$$

That is, we maximize the variance of the projections $w^T v w$ such that the direction vectors defining the principal components are unit vectors. This can be expressed as a Lagrangian problem : $L(\precapprox, \lambda) \equiv \precapprox^T \succapprox\precapprox - \lambda\left(\precapprox^T\precapprox - 1\right)$ Interestingly, we get that: $\mathbf{vw} = \lambda\mathbf{w}$: Principal components are thus the eigenvectors of the covariance matrix $v = \frac{X^T X}{N}$, with $\lambda \geq 0$ the corresponding eigenvalue matrix [52]. Since the covariance matrix is symmetric then its eigenvector matrix is orthogonal; thus w , the principal components' directions, are the eigenvectors of the covariance matrix that spans the whole p-dimensional space and λ corresponds to the magnitude of variance in the direction of each corresponding eigenvector. Hence, the (PC1) defined as the principal component along which the the data exhibits the highest variance level, is the w_j for which λ_j is the highest - the same logic follows for subsequent PCs.

Dimension reduction using Principal Component Analysis consists of choosing the most relevant direction among all the different directions. That is, for some P dimensional data space we would like to find a Q dimensional subspace - defined by a subset of orthogonal PCs - that summarizes best the data (More specifically , we would like to find the number of eigenvalues, Q) . There are different ways to achieve this: One can for example compute the $R^2 \equiv \frac{\sum_{i=1}^{q}\lambda_i}{\sum_{j=1}^{p}\lambda_j}$ metric that quantifies the ratio of variance explained by the subset of Principal components over the total variations in our model [53] ; graph a scree plot of R^2 with respect to λ and choose the lambda according to the shape of the plot [54]; however this method is not rigorous. Instead, one can also treat Q (or, equivalently, the number of

since $\vec{w}\cdot\vec{w} = \|\vec{w}\|^2 = 1$.

[50] $\text{MSE}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}\|\vec{x}_i\|^2 - (\vec{w}\cdot\vec{x}_i)^2 = \frac{1}{n}\left(\sum_{i=1}^{n}\|\vec{x}_i\|^2 - \sum_{i=1}^{n}\left(\vec{w}\cdot\overrightarrow{x_i}\right)^2\right)$ the first term is independent of w , it is thus not relevant to our Minimization problem , we end up with $-\frac{1}{n}\sum_{i=1}^{n}\left(\vec{w}\cdot\overrightarrow{x_i}\right)^2$; Minimizing a concave function is equivalent to maximizing a convex one, hence, we will instead maximize $\frac{1}{n}\sum_{i=1}^{n}\left(\vec{w}\cdot\overrightarrow{x_i}\right)^2$. Since $\mathbf{Var}[X] = E[X^2] - E[X]^2$; we decompose the MSE Problem: as such $\left(\frac{1}{n}\sum_{i=1}^{n}\overrightarrow{x_i}\cdot\vec{w}\right)^2 + \widehat{\sigma^2}\left(\vec{w}\cdot\overrightarrow{x_i}\right)$; and since the X is centered the first term cancels out and we end up by maximizing the variance of the projection: $\widehat{\sigma^2}\left(\vec{w}\cdot\overrightarrow{x_i}\right)$. Minimizing the residuals is thus equivalent to maximizing the variance of the projections. In turn, the variance of the projection can be expressed with respect to X's covariance matrix: $\widehat{\sigma^2}\left(\vec{w}\cdot\overrightarrow{x_i}\right) = \frac{1}{n}\sum_i\left(\overrightarrow{x_i}\cdot\vec{w}\right)^2 = \frac{1}{n}(\mathbf{xw})^T(\mathbf{xw}) = \frac{1}{n}\mathbf{w}^T\mathbf{x}^T\mathbf{xw} = \mathbf{w}^T\frac{\mathbf{x}^T\mathbf{x}}{n}\mathbf{w} = \mathbf{w}^T\mathbf{vw}$

[51] This equivalence between residual minimizing and variance maximizing can be also demonstrated by the Pythagorean theorem: Consider for simplicity finding PC1 only: since x_is are fixed and since the projections $(\vec{x}_i\cdot\vec{w})\,\vec{w}$ form a triangular rectangle, we apply the Pythagorean theorem: $\underbrace{\left[(\vec{x}_i\cdot\vec{w})\,\vec{w}\right]}_{\text{A}}^2 + \underbrace{\left[\overrightarrow{x_i} - \left(\vec{w}\cdot\overrightarrow{x_i}\right)\vec{w}\right]}_{\text{B}}^2 = \underbrace{\left[x_i^2\right]}_{\text{C}}$; Since A is fixed any increase in B corresponds to a decrease in C and vice versa.

[52] Since the elements of covariance matrices are always positive. Then the eigenvalues will be positive

[53] This metric is $\in [0, 1]$; The bigger it is the more is the sub-model representative

[54] One can choose the λ , on the "elbow" of the scree plot. That is the point above which the screen plot does not

eigenvalues) as a hyper parameter of the model and use cross validation for optimal selection (Detailed illustration is available in the empirical analysis chapter).

Principal Component Regression is simply a linear regression method applied on the Principal Components of the predictors. This method is used mitigate the problems of ill-posedness and overfitting already discussed ; by using a subset of principal components. Instead of using a penalized L_p regularization method, regularization is explicit and is done prior to fitting. This method solves overfitting, because, as for lp regularization methods, we reduce the complexity and hence control overfitting by choosing a subset of predictors. And ill-posedness, because, by applying PCA and selecting a subset of features, we mitigate some issues related to ill-posedness; Notably, linear dependence in the Design matrix is automatically corrected as the Principal components are uncorrelated. In addition,choosing a subset of predictors can solve underdetermined ; $P >> N$; design matrices.

Principal Component Regression first requires finding Q Principal Components using PCA. The subset of principal component spans a "latent" feature space. PCR consists of performing a linear regression using the new latent factors as covariates. Concretly, after identifying the orthogonal principal components (PCs), we perform a linear regression in a space defined by $PC_1, ..., PC_Q$. In this transformed space, each point from the original feature space is now represented by coordinates determined by their projections onto the PCs. The resulting Q-dimensional estimated parameter can then be projected back to the original P- dimensional space by multiplying it with w_Q[55] .This multiplication is intuitive when considering w_Q as a Factor Loadings, transforming latent factors to observed ones (This interpretation of w_Q as Factor Loadings is typical of Factor Analysis and Probabilistic Principal Component Analysis, however I will not delve into these concepts as they are beyond the intended scope of this study). In order to test/tune this regression, the testing data (i.e. the testing design matrix) is regressed on the already retrieved PCs.

Principal Component Analysis is widely used in high dimensional factor modeling; However, there exists a spectrum of efficient alternative dimension reduction techniques. The Probabilistic Principal Component Analysis (PPCA), for example, is a method derived from PCA that assumes a latent variable model with probabilistic assumptions on the latent variable. Factor analysis, for instance, is another ubiquitous method, that finds latent factors similar to PPCA, but assumes a non-isotropic Gaussian distribution for the covariance matrix. Fourier Analysis, for example, interprets the data as the sum of Fourier basis functions to compute latent factors. The Wavelet decomposition method too, is another notable feature selection method, where wavelets functions are exploited using topological methods to determine the latent components etc...

exhibit sharp variations

[55]Note, w_Q the principal components directional vector - or equivalently, the eigenvector matrix - is a matrix that defines Q dimensions in a P-dimensional space. It is thus a P-by-Q matrix, defined as a transformation from $R^Q \rightarrow R^P$. By multiplying β_Q by w_Q we are projecting the matrix to the original P dimensional space.

Principal Component Analysis, however, remains the most documented method in asset pricing feature selection applications, because it is interpretable, has applications in pure linear algebra, and importantly, because it provides an ordered list of Principal components: Unlike other dimension reduction techniques where the "relevance" of one latent dimension with respect to another is not explicitly indicated; PCA provides an ordered list of uncorrelated dimensions based on the level of variance in each PC direction, which facilitates dimension reduction.

One major drawback of Principal Component Analysis, is that while the resulting Principal Components (PCs) are uncorrelated, they are not guaranteed to be statistically independent.[56] Although the resulting PCs from PCA are linearly independent orthogonal directions, ensuring null correlation when feature points are projected onto them, this does not imply statistical independence. In fact, a null correlation is a second order degree of independence, while statistical independence can imply higher orders of dependencies. From an information theory point of view, for example, one can argue that, correlation "does not reflect the information distance" between two variables (Taleb, 2023) [27]. It's important to note that variables can be both statistically independent and uncorrelated when exhibiting only second-order dependence, but this is unlikely in the case of the features used in this dataset, given their intricate nature and their complexity. To address this limitation, I perform another dimension reduction method that specifically aims at finding higher orders of statistical independence between the latent factors.

5.2 Independent Component Analysis

Independent component analysis is a blind source separation method typically used in the field of signal processing in order to retrieve statistical independent sources of some set of signals (See appendix, for a graphical representation of ICA). I use Independent Component Analysis to retrieve statistically independent factors. Statistical independence is a strong measure of dependence[57]: It quantifies how much the occurrence (or non occurrence) of an event affects the occurrence (or non occurrence) of another. We define $X_1, ..., X_N$ random variables to be "statistically independent" when $P(x_1, \ldots, x_N) = \prod_{i=1}^{N} P(x_i)$.

The independent component analysis is built upon the following framework: Consider some observed multidimensional data represented by x (P-by-N) , to be some linear mixture of statistically independent sources s of dimension Q-by-N. Thus, $x = As$ with A a P-by-Q linear mixing matrix with both A and s unknowns. This method assumes that what we observe is an unknown linear transfor-

[56] Uncorrelated variables implies linear independence . While statistical independence, between variables X and Y for instance, implies that $f(\mathbf{X}, \mathbf{Y}) = f_X(\mathbf{X}) \cdot f_Y(\mathbf{Y})$: This means that none of the variables explains the other. Clearly, statistical independence is a much stronger assumption

[57] This is in contrast with the correlation metric. Which is considered a weak dependence measure from a statistical point of view

mation of unknown statistically independent latent factors; and the goal of ICA is to retrieve $\hat{s} = Wx$ with $W = A^{+}$, and hence obtain the statistically independent source factors defining the observed features. The issue with $x = As$ however, is that it is an $Ax = b$ problem with both X and A being unknown, and this is, a priori not solvable.

Independent Component Analysis, proposes a strategy to find W (and hence retrieve $\hat{s}$): First, Consider the Singular Value Decomposition of A: $A = U\Sigma V^{\top}$,which can be studied as a rotation-stretching-rotation transformation, and its "inverse" [58] expressed as $W = V\Sigma^{-1}U^{\top}$

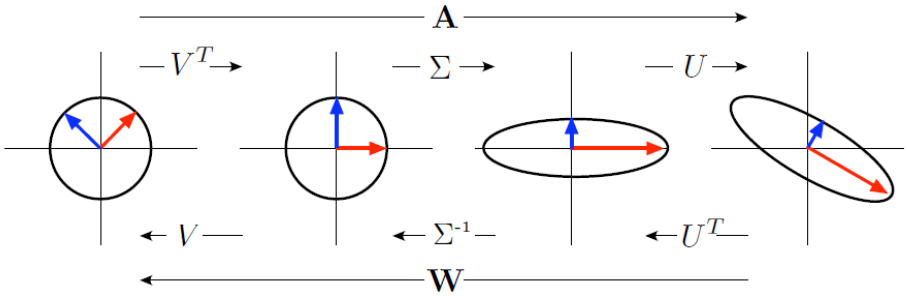


Figure 6: Graphical representation of the Singular Value Decomposition of the linear mixing matrix A, as well as its Pseudoinverse W. Hence, $x = As$ can be viewed as an SVD transformation from x to s (and vice versa)

Importantly this Singular Value Decomposition of the mixing matrix illustrates the idea that $x = As$ can be viewed as an SVD transformation from s to x (and vice versa). Independent Component Analysis's strategy unfolds in two stages: First, one needs to study the covariance of the observed data to find U and Σ. Independent Component Analysis assumes demeaned variables and introduces a crucial assumption: whitened covariance of sources, i.e. $E(s^T s) = I$.

By doing so, we have simplified our $Ax = b$ problem : In fact, we can now express the covariance of our observed variable $\mathbb{E}\left(xx^{\top}\right) = U\Sigma^2 U^{\top}$ independently of v and s [59] and since $\mathbb{E}(xx^T)$ is a symmetric matrix and it is always diagonalizable such that $E\left[xx^{\top}\right] = EDE$,with E and D corresponding to the eigenvectors and eigenvalues matrices. Thus, by imposing the whitening assumption we have now found both U and Σ such that $\hat{s} = Wx = V\Sigma^{-1}U^{\top}x$ gets reduced to $\hat{s} = VD^{-\frac{1}{2}}E^T x$ with now only V being unknown.

Independent Component Analysis's second stage is to exploit the statistical independence of sources in order to find V. In fact,V is a rotation matrix determined by some angle parameter θ (In a two dimensional space ,for instance, a rotation matrix V takes the form $\begin{pmatrix} \cos(\theta) & -\sin(\theta)) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$ where θ is the only defining variable). In the $\hat{s} = VD^{-\frac{1}{2}}E^T x$ problem, V represents the last rotation from a

[58] "Pseudo-inverse" is a technically better suited term since W is a double sided inverse if and only if it has Full Rank . I use the term "inverse" for simplicity

[59] $\mathbb{E}\left(xx^{\top}\right) = \mathbb{E}\left(Ass^{\top}A^{\top}\right) = \mathbb{E}\left(U\Sigma V ss^t V\Sigma U^{\top}\right) = U\Sigma V^{\top}\mathbb{E}\left(ss^{\top}\right)V\Sigma U^{\top}$, since $V^T = V^{-1}$ (Since it is an orthogonal matrix) and $\mathbb{E}(ss^T) = I$ then $\mathbb{E}\left(xx^{\top}\right) = U\Sigma^2 U^{\top}$

rotated-then-stretched source s to x (refer to figure). Hence, computing V can be viewed as finding the angle parameter of the last rotation that gives us statistically independent sources. ICA exploits the independence assumption to find the angle: Finding V can be formalized as finding θ such that some metric of independence is minimized. Information theory provides us with this metric: In fact, mutual information is an adequate metric here as it computes the information distance between two distributions; i.e. quantifies the amount of information one variable provides about another; and is thus a good proxy for statistical independence. However, since sources are usually more than two, "multi-information", a generalization of the mutual information, is better suited. Defined as $I(y) = \int p(y) \log_2 \frac{P(y)}{\prod_{i=1}^{N} p(y_i)} dy$, this metric is an ideal proxy to statistical independence. Now, one can find rotation matrix V and solve ICA by minimizing $I(\hat{s})$ where $\hat{s} = VD^{-\frac{1}{2}}E^T x$. That is, finding the θ such $I(\hat{s})$ is minimized. This minimization problem is not trivial, a reduced form of $I(\hat{s})$ is used instead [60]. Thus $V = \text{Argmin}_v \sum_i H\left[\left(VD^{-\frac{1}{2}}E^\top X\right)_i\right]$. Having found V, one can find W and s the statistically independent factors.

In this paper, observations x are the features (Those are P, N dimensional feature vectors) and sources s represent the reduced latent features (hence , s is a Q by N matrix). The linear transformation A, essentially transforms the sources which are points in the R^Q reduced latent feature space to signals (Those are the points we observe in the original feature space) in a new separate R^p space spanned by the known original features. A's columns thus represent the latent direction of the observations in the original feature space.

While principal components are orthogonal, the independent components resulting from an ICA are not (Unless sources dependence is limited to second order). This can be clearly illustrated in a 2 Dimensional ICA; where the signals are two dimensional features (Above 3 Dimensions this cannot be graphically illustrated). Below is a graphical representation of the difference between PCA and ICA, using my dataset, on two factors.

[60] The reduced form is obtained by noting that the multi-information metric can be expressed as $I(\mathbf{y}) = \sum_i H[y_i] - H[\mathbf{y}]$, thus $I(\hat{s}) = \sum_i H\left(\left(VD^{-\frac{1}{2}}E^T X\right)_i\right) - H\left(VD^{-\frac{1}{2}}E^T X\right)$. Given the following entropy property: for any continuous random variable X and transformation A the differential entropy $H(AX + b) = H(X) + log|A|$. Then $I(\hat{s}) = \sum_i H\left(VD^{-\frac{1}{2}}E^\top x\right)_i\Big) - \left(H\left(D^{-\frac{1}{2}}E^\top x\right) + \log_2|V|\right)$, since $\det(V) = 1$ (Property of a rotation matrix) then $\log_2|V| = 0$ and since $D^{-\frac{1}{2}}E^\top X$ is constant and independent of V we can neglect it. We end up with a simplified version of $I(\hat{s})$: $I(\hat{s})|_{\text{simple}} = \sum_i H\left(VD^{-1/2}E^\top X\right)_i\Big)$

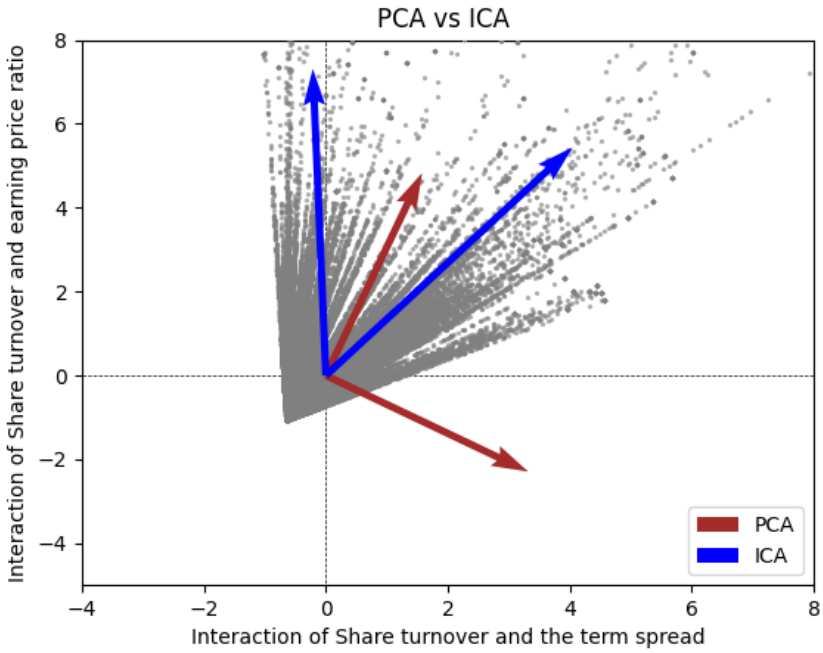
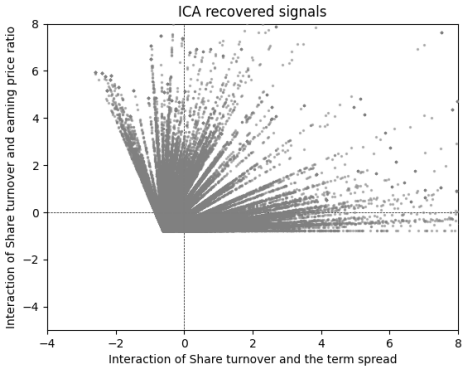
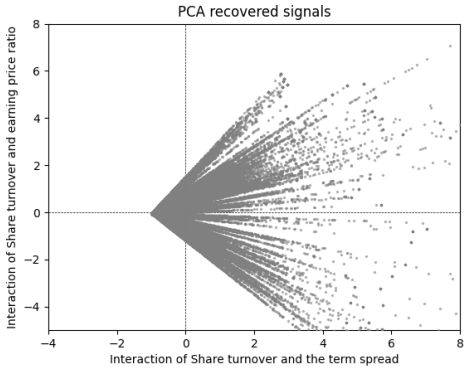


Figure 7: Graphical Representation of PCA vs ICA using a subset of Factors; notably Share turnover, Earnings to Price Ratio and the Term Spread on returns between 2001 and 2020. The Latent directions are different. In this example, Independent Components seem more relevant as they represent better the data.



((a)) Independent Component Analysis Recovered Signals



((b)) Principal Component Analysis Recovered Signals

Figure 8: Principal Components vs Independent Component Tested on a subset of the Dataset

This application of ICA on a subset of the data is indicative of the power and potential relevance of this method relative to the PCA. In fact, one can see that the PCA might not be particularly effective for orders of correlation above 2 as the above data seems to exhibit.

As for the Principal Component Regression, I use the Independent Components found in ICA as covariates of a linear regression. And as for PCR, the IC-based regression model is tuned on the validation set by changing the Number of Independent Components. However, the main drawback of ICA, is

that the independent components are not explicitly sorted with respect to their relevance , unlike PCA (This is, in fact, is the reason for which PCA is ubiquituous in Dimension reduction methods applications). ICA provides no indication of the relative relevance among subsets of independent components, and trying for different combinations is computationally infeasible [61]. I thus use the power data method proposed by Hendrikse et al. (2007)[10] . The rationale is as follow: The variance of the signals (i.e. the observed features) can be expressed with respect to the different independent component contributions. In fact, for some 1-Dimensional signal (P-by-1) we can express the variance of standardized signals as $Var(X) = E(X^2) = \sum_{i=1}^{P} \mathrm{E}\left[x_i^2\right] = \sum_{i=1}^{P} \mathrm{E}\left[(\mathrm{a}_i \cdot \mathrm{s})^2\right] = \sum_{j=1}^{Q} \left\{\mathrm{E}\left[s_j^2\right] \cdot \sum_{i=1}^{P} a_{i,j}^2\right\}$ for P the number of signals and Q the number of latent sources [62]. Hence, I compute for each $j \in \{1, Q\}$, $(E\left[s_j^2\right] \cdot \sum_{i=1}^{P} a_{i,j}^2)$ and choose the component j for which the contribution is the highest.

[61] For my dataset comprising 912 factors, testing all different of features combinations requires $\sum_{k=1}^{912} \begin{pmatrix} 912 \\ k \end{pmatrix}$ iterations. This is infeasible

[62] Note: In this paper, the signals are multidimensional. I use 1-Dimensional signals for simplicity

6 Robust Linear Estimation

I have, thus far, only presented the least squares function as the objective function. This approach may not be robust in the context of return forecasting. In fact, returns typically exhibit large tail behaviors, which translates to having outliers in the dataset. Outliers are observations that have the property to disrupt patterns. They are particularly damaging in least squares estimation methods as all observations are weighted equally when estimating the model. In addition, from a probabilistic point of view, ordinary least squares estimation is no more a relevant with outliers, as the Gauss Markov assumption of homoscedasticity is not respected, implying that the linear regression is no more the Best Linear Unbiased Estimator (BLUE).

One can remove them before estimating the model. This approach however assumes that outliers are erroneous, not representative samples of the underlying model; this is typically the case when there is a typo in the data. However, this is not the case with return outliers.

Alternatively, one can modify the model, opting for another model where outliers do not disrupt the pattern. I pursue this in subsequent chapters by introducing non-parametric smoothing methods, notably generalized additive models and Trees.

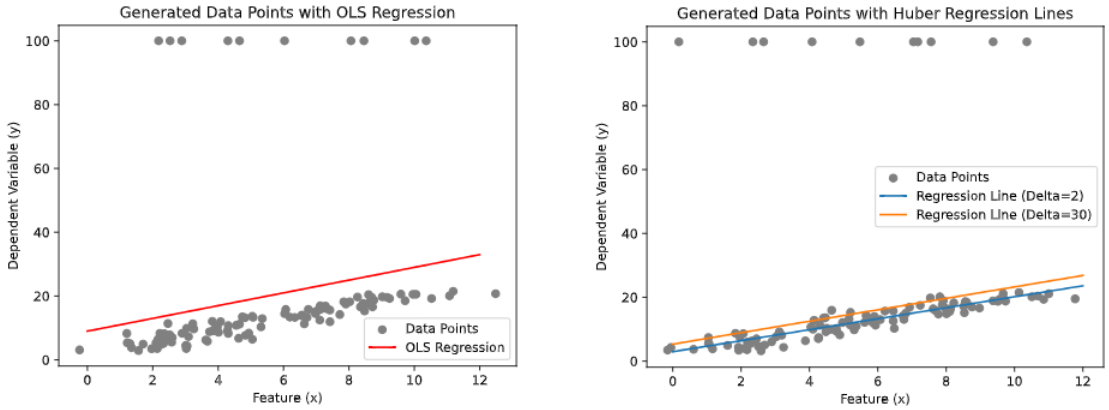
Another approach involves employing a different estimation method. In this paper, I choose to test the Huber loss function instead of a quadratic function to mitigate the issue of outliers. Huber loss is formally defined as follows:

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}a^2 & \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right) & \end{cases} = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta \\ \delta \cdot \left(|y - f(x)| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases}$$

with a the residual $y - f(x)$ and $\delta > 0$ the threshold tuning parameter. This loss function dynamically weights each observation in the estimation process: it computes the squared error of residuals when observations are not far way and instead computes the absolute value for outliers. Outliers are determined by the threshold parameter; which defines the critical level of error for which the observation would be considered an outlier. It is a tuning parameter set by cross validation.

Considering a well-defined model without outliers, introducing outliers would result in a higher loss on the observation when using squared loss. Huber loss address this issue by assigning less weight to outliers. In both Mean Squared Error (MSE) and Huber loss functions, the introduction of an outlier leads to an increased loss, necessitating adjustments to the parameters. However, Huber estimation ensures that the change in parameters is not as large compared to squared loss estimation. Huber's loss is particularly designed this way to be smooth at δ ; it is continuous at δ, $\lim_{a\to\infty} L_\delta(a) = \lim_{a\to-\infty} L_\delta(a) = L_\delta(\delta)$, and differentiable at δ $\lim_{h\to 0^+} \frac{L_\delta(\delta+h)-l_\delta(h)}{h} = \lim_{h\to 0^-} \frac{L_\delta(\delta+h)-l_\delta(h)}{h} = \delta$;

and thus easily optimizable [63]. In this paper I apply Huber, along quadratic loss,when evaluating linear models. Comparative results are reported in the Empirical Analysis Chapter.



((a)) Graphical Representation of outliers effect on estimation.

((b)) Graphical representation of Huber Loss estimation.

Figure 9: Huber Loss mitigates the effect of outliers on a 1-Dimensional subset of the dataset.

[63] This is why the loss is not simply defined as $L_\delta(a) = \begin{cases} \frac{1}{2}a^2 \\ |a| \end{cases}$. This loss function is not smooth at δ and thus is not the best candidate objective function for an optimization problem

7 Numerical Methods

I encounter in this paper different optimization problems that do not have closed forms: We have already seen that the lasso regression, the elastic net regression, the group lasso and the Huber loss function (and its combination with other penalization method) do not have closed form solution. In this paper, I propose to solve all of the following methods using an Accelerated Proximal Gradient Descent method.

Accelerated Proximal Gradient Descent method. In this paper, I use the accelerated proximal gradient descent algorithm in order to solve the various proposed regularized and non regularized convex loss functions. This algorithm combines two different generic convex optimization methods, namely the accelerated gradient descent and the proximal gradient descent methods.

The proximal gradient descent is a method to optimize convex non smooth functions. There are different other methods that also minimize non smooth functions, but what makes proximal method interesting is its speed. Hence, while the ubiquitous sub gradient method[64], for instance, converges at a rate of $o(\frac{1}{\epsilon^2})$, the accelerated gradient descent has a speed of $o(\frac{1}{\epsilon})$ (It converges as rapidly as the vanilla gradient descent). This optimization method relies principally on two pillars:

The first is the Moreau Proximal Point Algorithm (PPA) typically applied in the optimization of non-smooth functions. Formally, for some $\text{Min}_x f(x)$ problem, given a non-smooth function $f(.)$, the PPA algorithm is defined as such: $x^{t+1} = \text{prox}_{\gamma f}(x^t) = \text{Argmin}_y \gamma f(y) + \frac{1}{2}\|x^t - y\|_2^2$.[65] That is, the PPA defines some simple convex and differentiable function of y, $\gamma f(y) + \frac{1}{2}\|x^t - y\|_2^2$, which is tangent to $f(x)$ at $x^{(t)}$; such that its minimum is easily derivable. At each iteration, the new $x^{(t+1)}$ is the minimizer of the tangent function at $x^{(t)}$, and this repeats until some convergence criterion is met. The *prox* operator simply refers to this minimization subproblem defined by this algorithm.

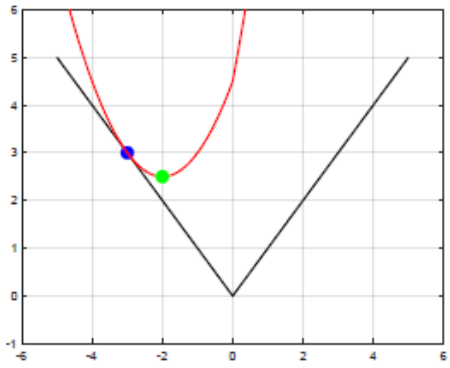


Figure 10: Illustration of Moreau Proximal Point Algorithm on some function $f(x) = |x|$. The proximal operator defines the red tangent simple smooth convex function and finds its minimum

[64]The subgradient method solves the problem of non differentiability of some objective function $f(.)$ by computing a subgradient of $f(.)$ at x^t, $g^t \in \partial f(x^t)$ and computing $x^{t+1} = x^t - \alpha^t g^{(t)}$ at each iteration t

[65]Here, the "y" is used to represent any variable.

Above, is an illustration of the Proximal Point Method: for $f(x) = |x|$, a non differential function. This is a representation of a single iteration of the algorithm where $x^{(t+1)}$ is the minimizer of a differentiable function defined by the proximal operator.

The second pillar of the proximal gradient descent method lies in the equivalence between the gradient descent algorithm, expressed as $x^{(t+1)} = x^{(t)} - \alpha \nabla f\left(x^{(t)}\right)$, and the minimization of the Taylor approximation of $f(.)$ around$x^{(t)}$ considering $\nabla^2 f(x) = \frac{1}{\alpha} I$.

In fact, this becomes clear when solving the *Argmin* of the Taylor expansion of $f(.)$ around $x^{(t)}$:

$$\underset{y}{\text{Argmin}} f\left(x^t\right) + \nabla f\left(x^{(t)}\right)^T \left(y - x^{(t)}\right) + \frac{1}{2\alpha} \left\| y - x^{(t)} \right\|_2^2$$

We find the minimum by equation the gradient of the objective function to zero $0 = \nabla f\left(x^{(t)} + \frac{1}{\alpha}\left(x^{(t+1)} - x^{(t)}\right)\right.$ $\implies x^{(t+1)} = x^{(t)} - \alpha \nabla f\left(x^{(t)}\right)$, which corresponds to the gradient descent method. Therefore, instead of employing the gradient descent algorithm, one can iteratively determine the minimum of the Taylor approximation of the objective function at a specific point until a convergence criterion is satisfied.

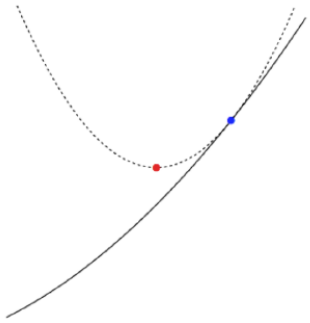


Figure 11: Graphical representation of the equivalence between Gradient descent and Taylor Approximation minimization. Instead of using Gradient Descent, one can minimize the Taylor approximation (Dotted curve) of the objected function (solid curve). This graph illustrates only one iteration.

To resume, the two pillars suggest: First, that non smooth functions can be optimized by using proximal operators at each iteration ; and Second that gradient descent on smooth differentiable functions is equivalent to minimizing the Taylor approximation of our function at each iteration.

Proximal Gradient Descent merges both methods: In fact,For some function $f(x) = g(x) + h(x)$ with $g(.)$ being convex and differentiable and $h(.)$ convex but non differentiable; Proximal gradient's approach involves iteratively minimizing "sub-problems" formed by the sum of the Taylor approximation of the convex and differentiable function $g(x)$ and the convex but non-differentiable function $h(x)$. Concretely; it consists of approaching the problem as if it were a gradient descent minimization

on some smooth function while keeping the non-smooth function untouched.

$$\begin{aligned} x^{(t+1)} &= \underset{y}{\operatorname{argmin}} \bar{g}_t(y) + h(y) \\ &= \underset{y}{\operatorname{argmin}} g(x^{(t)}) + \nabla g(x^{(t)})^T (y - x) + \frac{1}{2\alpha} \|y - x\|_2^2 + h(y) \\ &= \underset{y}{\operatorname{argmin}} \frac{1}{2\alpha} \|y - (x - \alpha \nabla g(x^{(t)})\|_2^2 + h(y) \end{aligned}$$

This is equivalent to the Proximal operator of $h(.)$ at $(x^{(t)} - \alpha \nabla g(x^{(t)}))$ with a proximal parameter α_{t}. Formally, the Proximal Gradient descent is defined as such:

First initialize $x^{(0)}$, $x^{(t)} = \operatorname{prox}_{h,\alpha_{(t-1)}} \left(x^{(t-1)} - \alpha_{t-1} \nabla g \left(x^{(t-1)} \right) \right), \quad t = 1, 2, 3, \ldots$, iterate until convergence

This hybrid optimization has a faster convergence rate than the standard proximal point method for non smooth functions (Tibshirani). Hence, whenever, the non-smooth objective function can be transformed into a composite optimization problem with smooth and non smooth components; it is preferable to use PGD than PPM for computational speed.

On the other hand, the accelerated proximal gradient descent, incorporates acceleration into the optimization. Introduced by Nesterov, the Accelerated Gradient Descent is a modification of the standard Gradient descent method designed to achieve a faster convergence rate. Gradient descent can exhibit very slow convergence depending on the shape of the objective function [66]; This is clearly the case when the convex objective function has a minimum in a "narrow valley" which causes the gradient descent to zigzag very slowly towards it [67]. In order to mitigate this problem, Nesterov incorporates memory into the Gradient descent method: For each iteration, the new direction incorporates the "momentum" of previous directions; this has the effect of tilting "degenerate" directions to "coherent" and "well behaved" ones. Formally, Nesterov is defined as such: For an unconstrained smooth and convex minimization problem: Initialize $y^{(0)}$, then compute $x^{(t)} = y^{(t-1)} - \alpha^{(t)} \nabla f \left(y^{(t-1)} \right)$ for $y^{(t)} = x^{(t)} + \frac{t-1}{t+2} \left(x^{(t)} - x^{(t-1)} \right)$ and iterate for $t = 1, 2, \ldots$ until convergence.[68] Ultimately, this method is faster than the Gradient descent method.

The Accelerated Proximal Gradient Descent Algorithm; simply incorporates the Nesterov Gradient descent to the Proximal Gradient descent. Formally, it is defined as such:

Initialize $x^{(0)}$ and $y^{(0)} = x^{(0)}$; then compute $x^{(t)} = \operatorname{prox}_{\alpha^{(t)} h} \left(y^{(t-1)} - \alpha^{(t)} \nabla g \left(y^{(t-1)} \right) \right)$ for $y^{(t)} = x^{(t)} + \frac{t-1}{t+2} \left(x^{(t)} - x^{(t-1)} \right)$; and iterate for $t = 1, 2, \ldots$ until convergence.

This optimization method is perfectly suited for our optimization problems as our objective func-

[66] And the set hyperparameters

[67] Or when a concave function has a maximum in a narrow space

[68] Geometrically, Nesterov Gradient descent simply extrapolates the previous direction by some "memorized" momentum (depending on previous trajectories); then follows the negative gradient

tions can be decomposed as smooth and non smooth function; and , importantly each of the non-smooth functions used in this paper has its own closed form proximal operator. This fact is important, as Proximal Gradient Descent computes the proximal operator of $h(.)$ at each iteration. (See: Appendix for Closed form $prox(.)$ of each of the non-smooth functions).

8 Generalized Additive models

A generalized linear model (GLM) is a flexible extension of the ordinary linear regression model and can represent a variety of distinct regression models. The configuration of a Generalized linear model is the following :

- A linear predictor $\eta(x) = \beta_0 + X\beta$. i.e. covariates are defined as a linear model, upon which is build the GLM.
- A random component is defined as following some distribution from the exponential distribution family [69]. For example, the ordinary linear regression model defines the random component $\epsilon \sim N(0, \sigma^2)$, and as a result $y|x \sim N(X\beta, \sigma^2)$
- A link function that links between the random $E[Y|X]$ and the covariates. The link function is a bijection that transforms $E[Y|X]$ to the linear predictor $\eta(x)$. For instance in a linear regression $\mu(x) = \beta_0 + X\beta$, the link function is the identity function.

In summary, the Generalized Linear Model is a way to express different regression models based on some linear predictor assuming some random component and given some link function

I will focus in this paper, on a more generalized version of the Generalized linear models, notably the Generalized additive model (GAMs) . Generalized additive models are defined as conditional expectation regressions linked to the sum of arbitrary smooth functions (one for each variable) by a link function. Formally, it is defined as:

$$g(\mathrm{E}(Y|X)) = \beta_0 + f_1(x_1) + f_2(x_2) + \cdots + f_p(x_p)$$

where $g(.)$ is the link function; $f_j(.)$ are unspecified smooth functions , and m is the number of factors. $f_j(.)$ are arbitrary functions that can change from one predictor to another, provided that it is smooth [70] ; and what makes this model special is that it is flexible: $f_j(.)$ can , for example, take the form of some fully-parametric functions (polynomial regressions, linear regressions etc...) , expansions of basis functions (Natural K-splines, Sigmoid basis expansions etc...) , or fully non-parametric smoothing functions (Nadaraya-Watson Kernel regressions, K-NN etc...) ... the list is expansive. Generalized additive models also assume $\mathbf{E}[Y] = \beta_0$ and $\mathbf{E}[f_j(X_j)] = 0$ in order to make the problem identifiable. Essentially, if we do not assume the following, we end up with "Concurvity" - The generalization of collinearity in an additive model framework- that is, there are infinitely many parameters that gives

[69] Do not confound with exponential distribution. An exponential distribution family is a set of probability distribution function expressed as $f_X(x \mid \theta) = h(x)\exp[\eta(\theta) \cdot T(x) - A(\theta)]$

[70] The smoothness of the functions refers to their continuity in their first and second order derivatives. Hence $f_j()$ can be represented by any C^2 function

us the same regression function. $g(.)$ the link function, is the same as the one defined for Generalized linear models; however, instead of linking to a linear predictor model it links to an additive model. In addition; because, under GAMs, $\eta(.)$ is no more a linear function, the estimation process changes[71]: GAMs use instead a "Back-fitting" algorithm to fit $f_j()$s . Hastie et al. define the back fitting algorithm as such: First Initialize: $\hat{\alpha} = \frac{1}{N}\sum_1^N y_i, \hat{f}_j \equiv 0, \forall i, j$. Then for: $j = 1, 2, \ldots, p, \ldots, 1, 2, \ldots, p, \ldots,$

$$\hat{f}_j \leftarrow \mathcal{S}_j \left[\left\{ y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k \left(x_{ik} \right) \right\}_1^N \right],$$
$$\hat{f}_j \leftarrow \hat{f}_j - \frac{1}{N} \sum_{i=1}^N \hat{f}_j \left(x_{ij} \right).$$

And iterate until convergence criterion is attained [72] (See Backfitting Algorithm section in Appendix for an explanation of the underlying logic of this algorithm) For S_j some smoothing operator which we choose according to the $f_j(.)$. In other words, the backfitting algorithm sequentially fits each factor while keeping others fixed. Updating a function involves applying the fitting method to a partial residual. For instance, if $f_1(.)$ and $f_2(.)$ are known, we can fit $f_3(.)$ by treating the partial residual as a response in some smooth regression on x_3. (Refer to appendix for detailed explanation)

We can clearly see that GLMs are special cases of GAMs, for $f_j(.)$ being linear in x, they only differ in their estimation, in their speed and in their biasedness [73]. I introduce both Generalized Additive Models and Generalized linear models, because in this paper I present a penalized form of GAM which, if penalized enough, may reduce to a GLM model. Moreover, I chose Generalized Additive models to represent non-linear models because it is the best suited to my problem; it is a good compromise between fully parametric non linear models and unstructured non-parametric smoothing methods .

In fact, on the one hand, even though fully-parametric models have been traditionally used in factor modeling and despite the fact that their estimation error converges quickly as the number of data increase[74]. The main problem is that it will always result in an approximation error if the underlying conditional expectation is not exactly matching the model.

On the other hand, unstructured non-parametric smoothing methods (Those are regressions that impose no assumptions on the the shape of the regression function) [75] can asymptotically capture any true underlying conditional expectation function as their fitting approach is data-dependent, free

[71] We cannot use linear regression - as we did for GLM - for non-parametric $f_j(.)$s - It does not make sense

[72] One can either specify a tolerance level, or some fixed maximum number of iterations

[73] GLM converge faster while GAM are less bias

[74] For instance, mean squared error 's convergence of linear models is $MSE_{\text{linear}} = \underbrace{\sigma}_{\text{intrinsic error}} + \underbrace{a_{\text{linear}}}_{approximateerror} + \underbrace{O\left(n^{-1}\right)}_{estimationerror}$ (Shalizi,2021), this is derivable by using the Law of iterated expectations from MSE, essentially, there will always be some approximation error, even if we infinitely increase the sample size. These MSE convergence property is generalizable to any parametric model (Shalizi, 2021[23])

[75] They are generally defined as: $\mathbb{E}(Y|X) = \sum_{i=1}^n y_i w\left(x, x_i, h\right)$ with $w(.)$ some fully non-parametric function of some tuning parameter h. Kernel regressions or K-NN are notable instances of unstructured non-parametric models

of any model restrictions. However , the main issue with these methods is that their estimation error is dependent on p (the independent variables), and fitting these models may fall under the curse of high dimensionality: Intuitively, for some sample of observations, fitting the model just by looking at the data becomes increasingly difficult as the number of dimensions increase. Wassermann (2006)[30], derives the Mean squared error asymptotics of unstructured non parametric methods as :$MSE_{\text{nonpara}} - \underbrace{\sigma^2}_{intrinsic error} = \underbrace{O\left(n^{-4/(p+4)}\right)}_{rate of convergence of estimation error}$, as having no approximation error but with an estimation-error rate of convergence (to zero) dependent on the number of features i.e. this is a formalized representation of the curse for dimensionality for unstructured non parametric methods.

Generalized additive models emerge as a perfect compromise between fitting well the data and not falling in the high dimensionality curse trap: In fact, it is a structured non-parametric method that uses non parametric smoothing functions $f_j(X_j)$ on each of the predictors ; the regression is no more dependent on p parameters. Rather; what we have with GAMs is P non-parametric functions each dependent on a single parameter. We thus fit a non parametric function - that minimizes its specific approximation error - without suffering from a large estimation error due to high dimensionality - as GAMs smooths P times on 1 dimension. For instance, for a simple GAM on p features, with $f_1(X_1), ..., f_p(X_p)$ all being smoothing splines, Shalizi (2021) derives $MSE_{\text{additive}} - \sigma^2 = a_{\text{additive}} + O\left(n^{-4/5}\right)$[76]: i.e. there still is some approximation error $a_{additive}$ as the approximate error combining all dimensions together has not been tackled by GAM, however, approximation error is better than what we get for a linear model $a_{additive} \leq a_{linear}$ [77] and we do not fall into any dimensionality problem - the estimation error convergence solely depends on N not P (Note:$O(n^{-4/5})$ in the formula) Another, yet weaker, advantage of choosing GAMs is that they are interpretable models: By posing the problem as an additive one, one clearly see the parts constituting the overall model, and thus conjecture the dynamics of the model.

For these reasons I chose GAMs to modelize non-linearly the returns with respect to high dimensional factors. and choose to model the arbitrary functions with respect to second order splines with k knots (i.e $k+1$ intervals). The number of knots is a hyper parameter; which I tune using cross validation. Formally: Each $f_j(X_j)$ is modeled as $\theta_j p(x)$ with $p(x)$ a second order spline defined as $\beta_0 + \beta_1 x_j + \beta_2 x_j^2 + \beta_3 (x_j - k)_+^2$ with $(x_j - k)_+^2$ a truncated power basis defined as $(x - k_i)_+^2 = \begin{cases} (x-k_1)^2 & \text{if } x \in [k_1, k_2) \\ \cdots & \\ (x-k_K)^2 & \text{if } x > k_K \\ 0 & \text{otherwise} \end{cases}$ Second order splines are chosen in this framework simply

[76]This derivation requires Taylor approximating the MSE and using Oracle assumptions
[77]Since Additive models $\subset$ Linear models - this has been discussed in the statistical learning theory Chapter

because they represent standard flexible C^2 functions [78] .They are fitted by least squares regression, hence the smoothing operator S_j in the backfitting algorithm is the squared loss. Concretely, the Generalized additive model will look like this:

$$g(\mathrm{E}(Y|X)) = \theta_0 + \sum_{j=1}^{P} p\left(X_j\right)' \theta_j$$

with $g(.) = \mathbb{I}$ the Identity function , P the total number of factors, and I assume that $p(z_1), ..., p(z_j)$ are all second order splines with K knots (Notice that there are no feature index to the spline functions as all splines in this model are all the same for the different features; features differ in their coefficient θ_j). While generalized additive models mitigates the curse of high dimensionality, the model can become highly parameterized, particularly with an increased number of knots, as the number of parameters, k.(Order of the spline + 1), scales linearly, increasing by a constant factor of 3 for each additional knot. Given the increased parametrization, which complicates model interpretation, and considering my earlier discussion on regularization's role in enhancing generalization, I employ the Grouped Lasso Regularization method (Yuan, Lin 2005). Grouped lasso is, like Ridge and Lasso, a Tikhonov Regularization, on an l_2 normed (Non-squared) penalty. It thus follows the same rationale discussed in the regularization section. Grouped lasso has however two distinctive features: First, its penalty norm is l_2 normed thereby inducing sparsity; this is clearly shown in the Lasso Chapter [79]. Secondly, and importantly, it penalizes coefficients in batches rather than individually. In this paper, I utilize this regularization method to nullify all the k-splines associated with each feature if needed. Specifically, groups are formed by the k coefficients (β_k) of each basis function in each predictor. And formally, the smoothing operator in the back fitting algorithm is now defined as such: $\min_\beta \left(\left\| \mathbf{y} - \theta_0 - \sum_{j=1}^{P} p\left(X_j\right)' \theta_j \right\|_2^2 + \lambda \sum_{j=1}^{P} \sqrt{N^j} \left\| \beta_j \right\| \right)$, With $\beta_j = (\beta_1, ..., \beta_K)$ and N^j the number of elements the coefficients of the basis function of the K spline associated to each predictor. [80]

[78] It is common to choose cubic spline (As far as I know) as they are able to represent complex curvatures smoothly. Second order splines, are able to represent non linear functions too, however, they can be less efficient in representing complex curvatures (e.g. sharp wiggly behaviours) , they are nonetheless used in this paper for computational purposes. In fact, since I perform GAM on 920 features; cubic splines gives me $3680k$ parameters per smoothing function while the second order result in $2760k$.

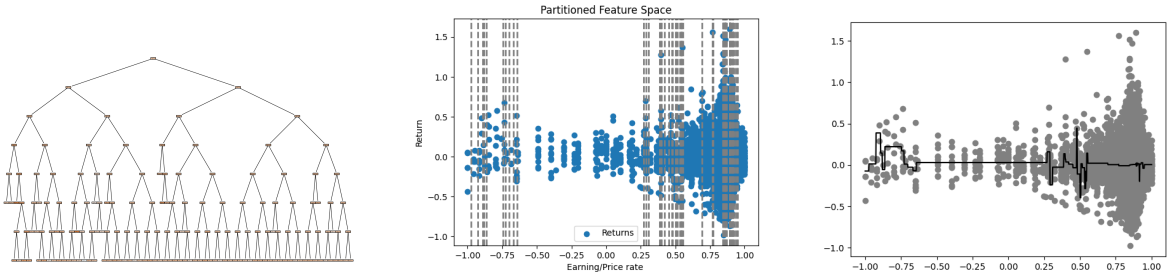
[79] Note on terminology: Even though ridge regression is commonly referred to as the "l_2" normed regularization; Ridge is effectively a "Squared l_2" normed regularization. The squared L_2 ridge penalty does not result in sparse regularization whereas the l_2, used in Group Lasso does induce sparsity.

[80] Notice that; like for the ridge and lasso regularization methods, the intercept is not penalized. In addition, standardization too is required in Grouped Lasso; for the same reasons discussed in the $l1$ and $l2$ regularization chapters

9 Regression Trees

9.1 Regression Trees

Regression trees are essentially recursive binary partitions on some feature space X resulting in a piece-wise prediction. That is, Trees are equivalent to recursively partitioning the feature space into two parts each time until obtaining different partitions, each corresponding to a constant prediction [81] .In a 2 dimensional feature space this equivalence between building a tree and partitioning a feature space can be clearly illustrated.



((a)) Regression Tree. ((b)) Partitioned Feature Space. ((c)) Constant Piecewise Regression.

Figure 12: Regression Trees can be interpreted in three ways: as trees, as a feature space partitioning, or as a constant piecewise regression function. I use my dataset to provide a visual representation using the Earnings to Price Ratio as the factor.

In fact, one can either, perform a recursive binary partitioning on some rectangular (Two dimensional) feature space: That is, I start with the first split , let's say in $X_1 = s_1$, then in each of the two defined regions, I split again. Thus in $X_1 > s_1$ I split at $X_1 = s_2$ and in $X_1 < s_1$ I split at $X_1 = s_3$ and I repeat the process recursively, until obtaining the desired partition. The resulting feature space is split in K regions, each corresponding to a constant prediction output. [82] Or equivalently, one can represent the same result in a tree: the leafs (i.e. the terminal nodes) represent the last partitioned regions in the feature space, the binary nodes represent the different binary splitting decisions, and the output of each leaf corresponds to the constant prediction output of each partitioned region.

Formally, we represent regression trees as such: For some design matrix X and a dependent variable Y; if we have K ;$R_1, ..., R_K$; partitioned regions with each a corresponding piece wise constant prediction c_K, one can define a regression tree model as such : $f(x) = \sum_{k=1}^{k} c_k I\left(x \in R_k\right)$. The model is thus characterized by two defining parameters: the partitioned regions and the associated constant prediction output.

Ideally, one would like to find the partitioning that minimizes the squared loss between the observed

[81] A recursive program is one in which a function, such as Binary Partition, relies on prior, simplified instances of itself.
[82] Notice, however that I still have not discussed the choice of the splitting parameters and the splitting point (I explain this below).

and the predicted outputs. Two facts emerge, first , knowing the partitioned regions R_k, our model prediction is the average of the dependent variables in the partition. That is, $Agmin_{c_k}(y_i - f(x_i))^2 \Rightarrow \hat{c}_k = \text{ave}(y_i \mid x_i \in R_k)$ [83]. Secondly, it is computationally unrealistic to find such a partition, as it involves evaluating the loss for all possible partitions,i.e. comparing resulting trees from every conceivable split and ordering.

Accordingly, I perform a greedy algorithm (Introduced by Breiman,1984 [2]) to solve this problem. Instead of considering all possible splits, the greedy algorithm, focuses only on a single partition: Explicitly, our problem reduces to finding the best partitioning feature X_j and point s that splits the region into two sub regions $R_1(j,s) = \{X \mid X_j \leq s\}$ and $R_2(j,s) = \{X \mid X_j > s\}$ - Using a squared loss function, we determine the parameters as such :

$$Argmin_{X_j,s}\left[\min_{c_1}\sum_{x_i \in R_1(j,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i \in R_2(j,s)}(y_i - c_2)^2\right]$$

Accordingly, $\hat{c}_1 = \text{ave}(y_i \mid x_i \in R_1(j,s))$ and $\hat{c}_2 = \text{ave}(y_i \mid x_i \in R_2(j,s))$.

Constraining the problem to a single split each time makes partition now feasible by computing the squared loss - also called the impurity function [84] - over all features for all split points [85], within a single level of the tree only. This algorithm facilitates the construction of a regression tree[86]; nevertheless, it exhibits certain limitations: By constraining parameter fitting to a single level only, we risk overlooking scenarios where a sub optimal splitting feature (or splitting point) at a particular location on the tree could, in fact, contribute to the creation of a more efficient tree overall. The use of a greedy algorithm neglects the possibility that a partition considered weak at a certain level might be advantageous for the entire tree. This issue is mitigated by the introduction of ensemble methods explained in the next section .

In the greedy algorithm, the tree size serves as a hyper parameter. Setting it too high may result in overfitting, while setting it too low can lead to underfitting. [87]. Consequently, it becomes essential

[83] In fact, assuming that the partition R_k is known; if $x \in R_k \rightarrow I(x \in R_k) = 1 \rightarrow f(x) = c_K \cdot 1$ Accordingly, in order to find the constant parameter in R_k one need to minimize the mean squared error of the observation and the constant prediction in the assumed partition region. $\underset{c_K}{\text{Argmin}}\mathbb{E}\left[(Y_K - c_K)^2\right] = \underset{c_k}{\text{Argmin}}\left[E(Y_K - c_k)\right]^2 + V(Y_K - c_k)$ $= \underset{c_K}{\text{Argmin}}\left[\mathbb{E}(Y_K) - c_K\right]^2 + V(c_K)$, we solve this optimization problem by equating the derivative with respect to c_k to zero (Since the objective function is a quadratic derivable convex function); thus, $\frac{\partial MSE(c_K)}{\partial c_K} = -2(\mathbb{E}(Y_K) - c_K) = 0$ solving this equation, we get $c_K = \mathbb{E}(Y_K)$. Hence, the optimal output for a squared loss objective function given the partition space is the average of the dependent variables in that partition.

[84] The impurity function is a generic function that quantifies the "purity" of the split. In regression trees, loss functions are used for this purpose, while in classification trees it takes different forms, notably Misclassification error , cross entropy or Gini index etc...

[85] There are different ways to choose which splitting points we should use when evaluating different splitting scenarios. I choose s values to be the different quantile values of the feature

[86] For N observations, Greedy Algorithm is $O(N^2)$ while the "Naive" infeasible partitioning is $O(2^N)$. Hence, for 100 observations , we need 10,000 operations in a Greedy Algorithm against 1.267×1030 for the naive splitting to decide on Xj and s

[87] This is due to the fact that the tree size controls complexity, impacting generalization, approximation, and estimation

to regularize the tree size. Various methods exist to achieve this goal. One approach involves defining a maximum number of observations in the leaves, or establishing a threshold beyond which the mean squared error at each iteration should not increase. However, I choose not to employ these methods due to their myopic behavior regarding the subsequent levels of the tree. Instead, I utilize a Tree pruning approach.

That is, I construct a large tree using the greedy algorithm, prune the tree at some nodes and obtain a more performing sub tree. Pruning is the act of cutting down the regression tree at a certain node. There are different ways of approaching this method; one can, for example, perform a "Reduced Error Pruning", which consists of assessing all nodes, and calculate the cost associated with collapsing each node then proceed to collapse the nodes that result in the smallest increase in mean squared error. Note: Pruning will always increase the overall mean squared error of the tree as the tree is a non parametric method and will surely overfit the data if grown large enough (The dynamics of non parametric models are discussed in the Generalized Additive Models chapter), this is why, we choose to prune for the smallest increase in the overall mean. The Reduced Error Method is a fast but naive method as there is no explicit penalty criterion for pruning other than overall mean squared error.

I thus adopt a more interpretable method: The "Cost-complexity pruning" method introduced by Breiman et al. (1984)[2], is a tree regularization method that first consists of building a large tree; by fixing very loose stopping criteria (For instance, choosing a a maximum number of observations per leaf to be 1, or choosing a large depth condition... The list of methods for building a large tree is exhaustive), then finding the subtree that has the minimum mean squared error given a penalty on its number of leafs (its terminal nodes).

Formally, I pick a subtree $T \subset T_0$; that minimizes the following Cost function

$$C_\alpha(T) = \sum_{k=1}^{|T|} L_k(T) + \alpha|T|$$

with $L_k(T) = \frac{1}{N_k}\left(y_i - \hat{c}_k\right)^2$ the mean squared error associated to the subtree; $\hat{C}_k = \frac{1}{N_k}\sum_{x_i \in R_k} y_i$, the constant prediction in Partition k; $|T|$ the total number of leafs of the tree and α the penalization tuning parameter. Cost complexity pruning involves penalizing the number of terminal nodes, and finding accordingly the best subtree. The rationale is as follows: As explained, since growing a tree invariably results in a decrease in the overall mean squared error (as trees ultimately overfit), we aim to select a subtree that mitigates overfitting. To achieve this, a comparison of various subtrees is necessary. Considering that a smaller tree inherently yields a higher mean squared error, the $|T|$ penalty introduces a size-related penalization to the mean squared error of each subtree. Introducing

errors. These dynamics have been discussed in the Statistical Leaning Theory chapter

this penalty term provides a dynamic method for comparing subtrees, where the cost of a smaller tree may be lower than that of a larger one, depending on the tree's size (or equivalently, its number of terminal nodes) and the penalty tuning. This method relies on cross validation: The penalization hyperparameter α is found by picking the most performing one on the validation set.

9.2 Ensemble methods

Ensemble methods are a collection of methods that consists of combining "weak" learning methods into a larger more efficient one. Their primary purpose is to reduce model variance by combining individual models, rendering them particularly interesting in the context of regression trees.

9.2.1 Random Forest

Random Forest (RF) is an ensemble method applied on regression trees. Before delving into random forests, it's essential to grasp another widely used ensemble method upon which RFs are based: notably "bagging" - also called "Bootstrap Aggregation". This method consists of building different models and averaging them to obtain a more robust model. The different models are built by sampling, M times with replacement from the dataset, resulting in M datasets upon which M models are built. By training the model on different datasets, bootstrap aggregation is widely assumed to effectively diminish the model's variance. It is thus a suitable method for high-variance and low-bias models such as trees, and it typically performs bad for high-bias models, like for example linear models. In fact, as explained in the Generalized additive model chapter , linear models exhibit high bias asymptotically ; $MSE_{\text{linear}} = \underbrace{\sigma}_{\text{intrinsic error}} + \underbrace{a_{\text{linear}}}_{approximate error} + \underbrace{O\left(n^{-1}\right)}_{estimation error}$; while non parametric methods are unbiased asymptotically, $MSE_{\text{nonpara}} = \underbrace{\sigma^2}_{intrinsic error} + \underbrace{O\left(n^{-4/(p+4)}\right)}_{rate of convergence of estimation error}$, making them ideal for bagging. In addition, another important and more obvious aspect of bagging is that it mitigates the effect of outliers in the data (Grandvalet , 2002[7]). By bootstrapping with replacement from the dataset, outliers are weighted less in the final estimation (Asymptotically). Finally, from a Bayesian point of view, Tibshirani et al. (1997[28]) interpret bootstrap aggregation by describing the distribution resulting from bagging as an "approximate non-informative Bayesian posterior". This result holds asymptotically. In other words, this means that $P(\theta|X = \text{data})$ is approximately obtained by iteratively bootstrapping $P(X = \text{data}|\theta)$, without the need of any prior information.

Random Forest, is an ensemble method applied on regression trees that relies on bootstrap aggregation and a variant of the greedy algorithm. Specifically, M bagged trees are constructed through iterative sampling (with replacement) of subsets from the original data points. At each iteration, a tree is constructed [88] using a greedy algorithm, wherein, at each split, only a subset of features is considered.

Hence, for P total features in the dataset, Random forest method consists of picking randomly $D < P$ features at each split. This feature selection method is motivated by the consideration that

[88] on the bootstrapped subset of data points

bootstrap aggregation alone may not be sufficient: In fact, from a probabilistic point of view, bagging results in M independent and identically distributed models with each a variance σ^2 and a mean μ, the average of these models has a variance

$$\text{Var}\left(\frac{1}{M}\sum_{i=1}^{n} x_i\right) = \rho\sigma^2 + \frac{1-\rho}{M}v^2 \tag{4}$$

(Refer to footnotes for a detailed explanation [89]). This result is a crucial theoretic element in Random Forest's defence:

The result suggests that by increasing the number of bagged trees M one can reduce the variance of the bagged model; however, there will always remain some variance due to the correlation between the bagged trees ρ ;i.e. the model will always exhibit variance asymptotically due ρ.

Random Forest addresses this by randomly selecting a subset of features at each split, aiming to reduce the correlation between the different bagged trees. Asymptotically, using RF results in a zero variance model. However, in practice , the second term in 4 does not cancel out, and hence reducing the correlation would also have some upward effect on the total variance through the second term. Thus, instead of looking for the set of bagged trees with zero (or negative) correlation, the focus should be on finding the optimal balance of correlation that minimizes the total variance of the model. Achieving this balance involves tuning the model's feature selection parameters. And in fact, the number of selected factors is a hyperparameter of random forest method determined through Cross-Validation.

9.2.2 Boosted Trees

I also use boosted trees in this paper. Boosting, is also an ensemble method, where, like for bagging, different weak learners are combined to form a unique model that performs better. Adaptive boosting was first introduced by Freund and Schapire (1997)[4]; Their algorithm aimed to construct a robust model by adaptively combining weak learners. While delving into the intricacies of the algorithm is beyond the paper's scope, grasping its rationale proves beneficial. AdaBoost.M1 iterates M times [90] ; at each iteration, training points are reweighted, and a new model is fitted on the reweighed sample. The new model is scaled, then added to the one fitted in the previous iteration. The algorithm's output is thus the scaled sum of these models. Observations are reweighed based on the associated errors;

[89] That is because $\text{Var}\left(\frac{1}{M}\sum_{i=1}^{M} x_i\right) = \frac{1}{M^2}\text{Var}\left(\sum_{i=1}^{M} x_i\right) = \frac{1}{M^2}\left[\mathbb{E}\left[\left(\sum_{i=1}^{M} x_i\right)^2\right] - \mathbb{E}\left[\sum_{i=1}^{M} x_i\right]^2\right]$; with $\mathbb{E}\left[\sum_{i=1}^{M} x_i\right] = \sum_{i=1}^{M}\mathbb{E}(x_i) = M\mu$; and $\mathbb{E}\left(\left(\sum_{i=1}^{M} x_i\right)^2\right) = \sum_{i,j=1}^{M}\mathbb{E}(x_i x_j) = M\mathbb{E}(x_i^2) + (M^2 - M)\,\mathbb{E}(x_i x_j)$; This can be further reduced by noting that the correlation coefficient of two random variables x_i and x_j ,$\rho_{ij} = \frac{\mathbb{E}((x_i-\mu_i)(x_j-\mu_j))}{\sigma_i\sigma_j}$; is defined as $\rho_{ij} = \frac{E((x_i-\mu)(x_j-\mu))}{\sigma^2}$ for bagged (and hence i.i.d) models x_i and x_j . This correlation formula implies that $E[x_i x_j] = \rho\sigma^2 + \mu^2$. Utilizing this formula, $M\mathbb{E}(x_i^2) + (M^2 - M)\,\mathbb{E}(x_i x_j)$ reduces to $M\sigma^2 + M^2\rho\sigma^2 + M^2\mu^2 - M\rho\sigma^2$. Accordingly, $\text{Var}\left(\frac{1}{M}\sum_{i=1}^{M} x_i\right) = \rho\sigma^2 + \frac{1-\rho}{M}\sigma^2$.

[90] M is a tuning parameter of the algorithm

misclassified observations receive higher weights (They are thus more relevant in the subsequent fit) and model scaling is determined by the training error of the new model on the reweighed training set; higher errors result in lower weights.

In summary, Adaboost.m1 is a greedy adaptive algorithm to construct additive models using simple basis functions, taking into account previous errors at each iteration. This is equivalent to building an additive model $f(\mathbf{x}) = \sum_{m=1}^{M} b_m(\mathbf{x})$, defined as the expansion of some basis function $b_m(x)$; using a forward stage-wise additive algorithm with an exponential loss function. [91]. In essence, boosting methods are all forward stage-wise algorithms as such.

Accordingly, boosted trees are the additive expansion of simple trees (which can be interpreted as basis functions) $f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$, with Θ_m representing the tree parameters (R and c) where $\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$ is iteratively evaluated by forward stage wise algorithm. Using a Squared loss function, the problem reduces to a simple regression tree fitting problem applied on the residual from fitting the previous model rather than the dependent variable [92]. In this paper, I apply a "penalized" version of the boosted model, by scaling each of the added models by a shrinking penalty. The logic remains the same, I simply multiply every added model (at each iteration) by a scalar $v \in (0, 1)$. Tibshirani et al. (2001) suggest that by adding the v term, one could, by analogy with the functional gradient descent method, view the shrinkage parameter as the learning rate (step size) of the gradient descent.

Why Trees I use trees (and related ensemble methods) in this paper for many reasons. First, Generalized additive models, even though, as explained previously, are a good compromise with respect to other non linear models; they may be far fetched in the context of return forecasting as they are global models whose predictive function is the same across all its domain - which suggests that the underlying function is defined as a single function. Furthermore, GAMs pose an additional challenge. Despite their enhanced interpretability due to their additive composition, they abstract the dynamics of the function. Understanding the dynamics from a generalized additive model is nearly impossible [93].

On the contrary, trees offer a distinct advantage. By recursively partitioning the space into different regions, the tree defines different predictive dynamics (functions) across different paths that are easily understandable. Moreover, it is noteworthy that the search for similarities in trees represents a more robust approach compared to the K-Nearest Neighbors method. While K-Nearest Neighbors focuses on interpolating and smoothing based solely on the similarity of features, neglecting the dependent

[91] The forward stage wise algorithm consists of first initializing $f_0(\mathbf{x}) \leftarrow 0$ and then For $m = 1, \ldots, M$ finding the best model $f_m \leftarrow \arg\min_\theta \sum_i L\left(f_{m-1}(\mathbf{x}) + b_m(\mathbf{x}, \hat{\theta}), y_i\right)$ for some loss function $L(.)$ and updating by adding to the previous model $f_m(\mathbf{x}) \leftarrow f_{m-1}(\mathbf{x}) + f_m(\mathbf{x})$

[92] In fact $\hat{\Theta}_m = Argmin(y - f_{m-1}(x) - T_m(x))^2 = Argmin(resid - T_m(x))^2$

[93] And it is worse for Non-parametric smoothing methods where there is no clear distinction of the dynamics between the different features

variable; trees take into account the similarity in both dependent and independent variables. Terminal nodes in trees can be regarded as neighborhoods in the feature space containing datapoints with similar responses; consequently, Shalizi (2021) characterizes trees as "adaptive nearest-neighbor methods." In addition, there are many practical reasons that make tree popular forecasting methods: for example, it is straightforward to see which variables are relevant, it helps making predictions when variables are missing (If one wants to predict but does not have all features used in the tree, he can simply skip the missing information , without loss of generality) , it does not assume true smooth underlying function as the piece wise constant prediction can approximately represent both smooth (approximately), and non smooth true underlying functions , and finally there is no need for calculations to make predictions, one can just look at the tree.

10 Predictive Evaluation Metrics

I focus in this paper on practical out of sample metrics to test for the predictive efficiency of the models. I use the R squared and the mean squared error estimation. In this section, I present the metrics used. First, the R-squared - also called the "coefficient of determination" is a goodness of fit metric that measures the proportion of variance kept by the predictors [94]. Concretely, for any smoothing method, the R-squared metric is nothing more than the fraction of variance that is not smoothed out. McFadden (1974)[16] defines the metric in its most general form: $R^2 = 1 - \frac{D_{res}}{D_{tot}} = \frac{D_{reg}}{D_{tot}}$ with $D_{\text{res}} = ||\hat{l}_s - \hat{l}_p||$ the residual deviance, $D_{reg} = ||\hat{l}_p - \hat{l}_0||$ the model deviance and $D_{\text{tot}} = ||\hat{l}_s - \hat{l}_0||$ the total deviance and $\hat{\ell}_s$, the maximum log likelihood of a saturated model[95], $\hat{l}_0$, the maximum log likelihood under a Null model (Intercept only), and $\hat{l}_p$, the maximum log likelihood under the model (with p-parameters) [96]. In essence, R-squared evaluates model performance relative to a simple null model (such as the average of dependent variables) or a saturated model. The metric approaches 1 when the model deviance closely aligns with the total deviance and tends towards zero as the explained deviance deviates further from the total deviance. McFadden's general framework is important because it incorporates the multitude of definitions for R^2. In the context of linear models alone, there are at least 5 formulations of R-squared(to the best of my knowledge); these definitions are model dependent, and are not generalizable to out of sample metrics [97] . In this paper I use the out of sample R^2 which can be thus understood as an out of sample application of McFadden's metric i.e a comparison of the out of sample performance of the model with respect to a some naive benchmark; and is typically defined as the $R^2_{OOS} = \frac{\text{MSE}_{\text{OOS}}}{\text{MSE}_{\text{bemchmark}}}$ with the benchmark being the training data. Contrary to the in-sample metric, the deviance in R^2_{OOS} is a reliable proxy for predictive error. And unlike in the in-sample metric, where the residual deviance was consistently smaller than the total deviance due to smoothing, this relationship no longer holds for OOS R-squared[98] .Consequently, OOS R-squared's

[94] I purposely chose the word "kept" and not "explained" to define $R2$ as it common to read that $R2$ is the amount of variance a regression "explains". This definition is misleading.

[95] A saturated model is one where each observation is parametrized

[96] For example, assuming the following probabilistic modeling: $Y \mid X \sim N(x\beta_p, \sigma_p)$ i.e. a Gauss Markov Linear model; we can compute the associated log likelihood $\hat{l}_p$; then using $\hat{l}_0$ the log likelihood of $Y \mid X \sim N\left(\beta_{\text{intercept}}, \sigma_o\right)$ and $\hat{\ell}_s$. the log likelihood of $Y \mid X \sim N(y_i, \sigma_s)$, We get $D_{\text{res}} = \frac{1}{\sigma^2} SS_{\text{res}}$. $D_{\text{reg}} = \frac{1}{\sigma^2} SS_{\text{reg}}$ $D_{\text{Tot}} = \frac{1}{\sigma^2} SS_{\text{tot}}$; this is the ubiquitous formula of R-squared in a linear setting.

[97] For some linear model; we get at least 5 formulations of the coefficient of determination. The R^2 coefficient can be expressed as the fraction between the sample variance of regressed observations and the sample variance of the dependent variables $R^2 \equiv \frac{s^2_{\hat{m}}}{s^2_Y}$, and since $c_{Y,\hat{m}} = c_{\hat{m}+e,\hat{m}} = s^2_{\hat{m}} + c_{e,\hat{m}} = s^2_{\hat{m}}$ The metric can also be equivalently represented by the ratio of the covariance of the dependent variable to the regressed observations, divided by the sample variance of the observations , $R^2 = \frac{c_{Y,\hat{m}}}{s^2_Y}$. Moreover, $s^2_{\hat{m}} = s^2_{\hat{\beta}_0 + \hat{\beta}_1 X} = s^2_{\hat{\beta}_1 X} = \hat{\beta}_1^2 s^2_X$, one can write the coefficient as $R^2 = \hat{\beta}_1^2 \frac{s^2_X}{s^2_Y}$. In addition, for some demeaned variable x and y, the regression coefficient is expressed as $\frac{c_{XY}}{s_X}$ we can hence write, $R^2 = \left(\frac{c_{XY}}{s_X s_Y}\right)^2$; lastly, decomposing the observed value $Y = \hat{Y} + \epsilon$ we get the following expression $R^2 = \frac{s^2_Y - \hat{\sigma}^2}{s^2_Y}$

[98] The residual deviance D_{res} in sample will always be smaller than the total deviance D_{tot}, because the smoothing model smoothes the observations, hence, $R^2 = 1 - \frac{D_{res}}{D_{tot}}$, will range between 0 and 1; this is no longer the case out-of-sample, where the D_{res}, which is dependent on new unseen data is independent on the smoothing and thus is no longer bounded by D_{tot}

values span from $-\infty$ to 1.

I report the R^2_{OOS} in this paper, but will not rely on it for model evaluation. Primarily because this ratio is dependent on the variance of the model's features. In fact, considering for example a linear model $\hat{\mu}(x) = \hat{\beta}X$ for which we know the true underlying linear relationship $\mu(x) = \beta X$ (Note: this is different from the true relationship - which might exhibit non linear properties), its $R^2 = \frac{Var(\hat{\mu}(x))}{Var(Y)}$ is reduced to $\frac{\beta^2 Var(X)}{\beta^2 Var(X)+\sigma_\epsilon}$ by simple algebraic manipulation, it is clear from this formula that, even after having found the true linear regression one can modify the R^2 simply by changing the variance: Thus, even for a perfect model specification, the R^2 might exhibit low values for low variance in X. [99]

I use instead a more intuitive and practical metric for predictive evaluation , the MSE_{OOS}. For every model used in this paper, I evaluate the mean squared error (After having tuned the hyper parameters) on some testing set. I report both the R^2 and the MSE metrics, but the ultimate model selection criterion is the out of sample mean squared error .

[99] Note: This critique holds for in and out of sample R-squared metrics.

11 Resampling Methods

What makes machine learning methods interesting is that they employ practical resampling methods to assess the robustness of models. This data-driven method is in contrast with classical Factor modeling which tends to rely more on goodness of fit measures (R squared, Adjusted R squared etc...) or statistical tests (t test , p values etc..) [100] . There are many different ways to resample the data: Ultimately, any method boils down to the determining at least a training set for fitting, and validation test for testing or tuning. A naive random resampling into two groups(training and testing sets) will do the job - but will not be efficient [101] as it is a highly variable method. On the flip side, the popular leave one out cross validation (LOOCV) sampling method erases this variance problem [102] , it is less bias than a naive simple split on the data, however, this method can be computationally expensive. Using this paper's dataset, for instance, one would need to fit the data approximately at least a thousand times under LOOCV . A middle ground solution is hence required; k fold cross-validation, for example, is a widely used method that serves as a good compromise between high variance and computational cost [103] . Given this trade-off logic between these different methods, and because resampling methods are ultimately dependent on the data set (its size and properties) ; one should use, a resampling method that is tailored for high dimensional Factor modeling using panel data. In fact, given the methodology chosen, Recursive Time-Ordered Cross Validation (TOCV) with a rolling training window emerges as the best compromise among all Resampling methods. Resampling is constructed as such: First , the data set is split in three categories of samples. The first set is the training set in which I fit my model. The second set is the validation set:In this sample, the evaluated model performance is used for hyper parameter tuning.And since this set is not an out of sample set as it is used for tuning; I introduce a third set : the testing set. This set is used for model's evaluation after training and tuning. I already explained how why a naive split CV or a LOOCV may not be efficient; and how a good compromise would be using K fold cross validation. However, given that the data contains time series measurements, a K-fold Cross validation is not adequate for at least two reasons: On the one hand, the K fold Cross validation exposes us to information leakage: By shuffling measurements across different times, one will end up with a predictive model that uses future information to forecast anterior phenomena . For example factors in 2016 will forecast 1990s returns - and we do not want this. On the second hand, the data might also hide underlying regime shifts: Shuffling measurements across time distorts the predictive logic contained in the model. Recursive Time-Ordered Cross validation

[100] Those are metrics that are dependent on statistical assumptions and hypotheses. I have already explained the shortcoming for using R squared here, the same logic applies to many statistical tests and GOF measures.

[101] this claim is relative - but applies in general - as it fails if the dataset is so small that testing is counterproductive

[102] LOOCV consists of iteratively fitting on all-but-one observations and testing on the left out observation and averaging the result over all iterations

[103] K- fold CV splits the data into K parts of approximately equal size and iteratively fits on k-1 sets and tests on the left out set. LOOCV is thus especially case of K-Fold CV for K = Number of observations

emerges as a tailored compromise for forecasting Returns using panel data.

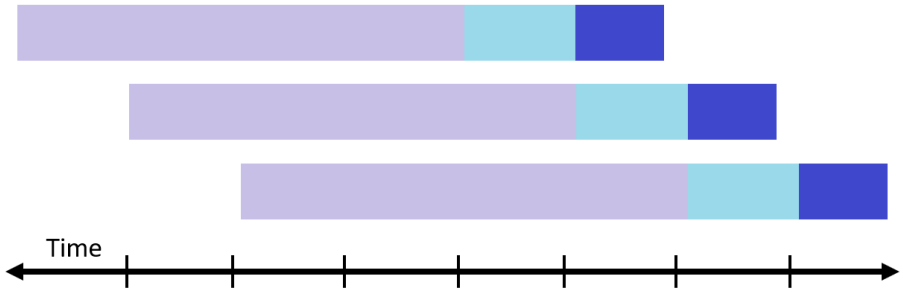


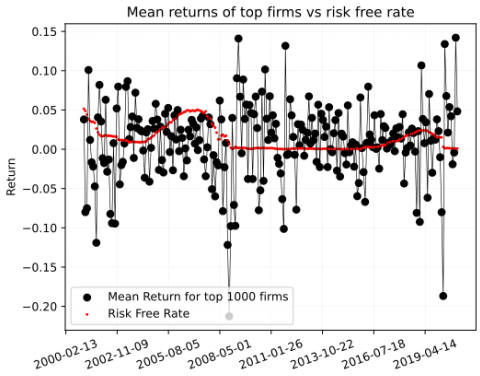
Figure 13: Graphical representation of the Time ordered Cross-Validation with rolling window resampling method

Accordingly, the TOCV splits the data into a training, validation and testing sets and a training window rolling forward in time. Ideally, the training window must be rolling and not expanding because it limits complications due to varying the size of the training set and it mitigates non-stationarity across time. In summary, the TOCV Resampling method is designed for my problem: This method is both computationally efficient (low computational cost) and non-varying, ensuring stability and preventing information leakage. However, due to computational constraints, I use in this paper a regular K-fold Cross validation resampling.[104]

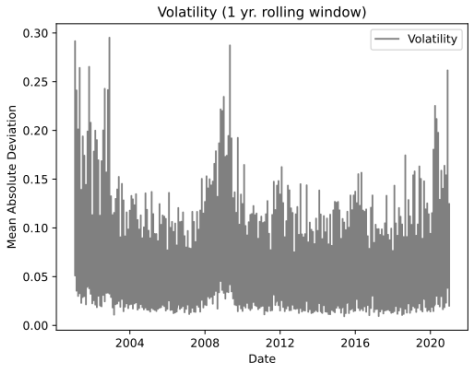
[104]The time ordered Cross Validation with a rolling window ultimately performs better, for the reasons explained above, however, the size of the dataset used given the available computational power, makes it suboptimal from a practical point of view. In fact, the dataset I use consists of 87,5 Million data points (96,000 observations by 912 features).

12 Empirical Analysis

To construct high-dimensional factor models employing the specified machine learning methods, an extensive financial dataset sourced from diverse channels is utilized. My dataset is comprised of monthly holding period returns of firms from years 2001 to 2022 and a list of firms characteristics as well as a set of macroeconomic indicators. The returns data is sourced from CRSP database; firms characteristic data is made public by D. Xiu [9], and macroeconomic indicators published by Welsh and Goyal (2008) [31]. The characteristics dataset encompasses 94 distinct firm attributes, regularly updated on a monthly, quarterly, or annual basis. These selected features are thoroughly documented and represent recurrent factors widely employed in asset pricing academia (refer to the factors section in appendix for a detailed explanation). Additionally, the dataset incorporates 73 dummy variables, each referencing the industry sector of the corresponding stock, constructed using the standard industrial code. A comprehensive explanation of processing and cleaning procedures is provided in the code section in appendix, where I detail the merging of all three datasets and the construction of interactions. After prepossessing the dataset, I end up with nearly a million observations, I choose however to work with a smaller subset of observations for computational purposes; I pick the biggest 1,000 firms every month, for 240 periods, from 2001 to 2020. The total number of different firms in the panel are 3,362 (as firms' size change with time). The dependent variables of my models are monthly returns and the independent variables are a set of 94 robust firm characteristics (Details in appendix), eight fundamental macro economic features as well as their interaction. I end up with a panel regression, with 912 features for 240,000 observations across time and stocks. In order to use Cross validation, I divide my dataset into a training , a validation and a testing set. As explained before, the training set is used for fitting, the validation set, for parameter tuning and the testing set for evaluating the model's performance.

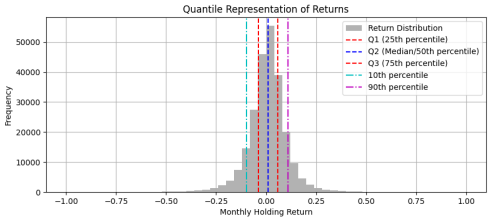


((a)) Cross sectional mean return against the treasury bill rate; across 240 time periods

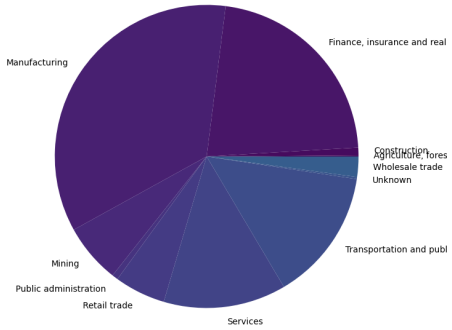


((b)) Volatility of the panel data evaluated by mean absolute deviation across a 1-year rolling window

Figure 14: Empirical characteristics of the dependent variable $r_{i,(t+1)}$



((a)) Empirical Distribution of the dataset and its quantiles



((b)) Industries represented in the dataset

Figure 15: General Overview of the Dataset

In this empirical analysis, I primarily rely on two metrics to determine the predictive accuracy of the tested models: The out of sample R-squared and out-of-sample Mean Squared Error. The benchmarks models I use are factor models documented by Lewellen (2015) (See Methodology chapter for more details on the factors). Specifically, those are three different Ordinary Least Squares Linear Models estimated on 3, 7 and 15 "robust" firm characteristic factors. The empirical results I get from these models using my dataset do not correspond to the two documented results by Gu et al. (2020) [105]. They nonetheless do not perform badly out of sample.

[105]Gu et al. (2020) document the out of sample performance of Lewellen's firm characteristics factors. They test the features on two different response variables: stock returns and S&P 500 returns over time; and report two different results accordingly.

	Documented R^2_{OOS}	Other documented R^2_{OOS}	R^2_{OOS}	MSE_{OOS}
3-factor model	.16	-.22	.0068	0.0086258
7-factor model	.18	-0.24	-0.0147	0.0087155
15-factor model	.19	.68	-0.0052	0.0089151

Table 1: Comparison between, found predictive performance metrics vs documented measures

Results are represented in the table above; the R-squared levels that I obtain are all very close to zero; but they do not match the two other documented metrics. This difference in R-squared is anticipated due to variations in the datasets employed. However, the out of sample mean squared error levels I get are reasonably low, indicating an acceptable out of sample performance. In this empirical analysis, I primarily use these three models' out of sample performance as benchmarks for model selection.

I perform a naive OLS regression on the whole panel in order to illustrate the failure of Ordinary Least Squares methods in high dimensional factor modeling. As expected, the model does not generalize, the resulting out of sample mean squared error is very high [106] and the out of sample R-squared (Which can take negative values as explained in the evaluation metrics chapter) takes extremely low values.

Naive OLS	In-sample	Out-of-Sample
R^2	0.1992	-142.7661
MSE	0.009	1.236

Table 2: Naive Ordinary Least Squares on the whole panel

Lasso, Ridge and Elastic Net regressions are then evaluated. Parameter tuning for these methods are however limited to sets of 10 to 15 values for computational purposes , consequently, my estimations may overlook more optimal results. I use the Accelerated Proximal Gradient Descent method, employing both, the quadratic and the Huber loss as my objective functions and report the result for each method.

Lasso	MSE_{OOS}	R^2_{OOS}
Quadratic Loss	0.00858677	0.99509
Huber Loss	0.00864447	0.36282

Table 3: Lasso Regression

Elastic Net	MSE_{OOS}	R^2_{OOS}
Quadratic Loss	0.00854828	1.08035
Huber Loss	0.00853352	1.45110

Table 4: Elastic Net Regression

[106] Refer to the distribution and the Quantile illustrations of the dependent variable to get an idea of the orders of magnitude. A mean squared error of 1,236 is far beyond the range of returns.

Ridge	MSE_{OOS}	R^2_{OOS}
Quadratic Loss	0.012	-48.36
Huber Loss	0.00876974	-5.62371

Table 5: Ridge Regression

Empirical results, are as anticipated: The three L_p regularization methods perform much better than the Naive OLS. The out-of-sample metrics are drastically improved by the introduction of regularization. In addition, since the Elastic Net Regularization contains both the L_1 and L_2 methods, it is theoretically expected to outperform, or at least match the performance of Ridge and Lasso Regularization. This fact is verified empirically: Elastic Net has the smallest out of sample mean squared error followed by Lasso then Ridge. Moreover, Huber Loss outperforms the squared error function for both Ridge and Elastic Net Regressions. For lasso, even though the quadratic loss performs better, the restricted range of hyperparameters used (Notably the limited set of threshold parameters for Huber loss) prevents us from definitively asserting that Huber is sub optimal.

Principal Component Regression is then evaluated both using Huber loss and the quadratic loss as objective functions. Tuning parameters, specifically the number of principal components and Huber threshold parameters, are restricted to predefined sets of 15 values.

	MSE_{OOS}	R^2_{OOS}
Huber Loss	0.00859008	0.67
Quadratic Loss	0.00846502	0.24441

Table 6: Principal Component Regression

First of all, Principal Component Regression greatly outperforms the naive Ordinary Least Squares: This is evident, as principal component regression, being a regularization method, inherently outperforms standard OLS in high-dimensional settings. In addition, Empirical findings are slightly more satisfying than L_p normed penalizations. These congruent results are not surprising as Principal Component Analysis is also a regularization method. The results also suggest that the dimension reduction to uncorrelated dimensions (i.e. PCs) is reasonable, indicating the substantial strength of second-order dependence between factors.

The Independent Component Analysis is subsequently tested on the dataset. I perform ICA, choose Independent Components using the power data method (The number of independent components is a tuning parameter of the model), then I regress linearly the projected observations on the chosen independent components using both Huber and quadratic loss functions. Regressing using independent components results in a performance comparable to l_p normed regularizations, and is lower but not far from Principal Component Regression's performance. Importantly, what this result imply is that new logics of dimension reduction other than the traditional uncorrelated Factors methods (PCA, PPCA,

	MSE_{OOS}	R^2_{OOS}
Huber Loss	0.00860797	0.85049
Quadratic Loss	0.00865182	0.34541

Table 7: Independent Components based Regression

	MSE_{OOS}	R^2_{OOS}
Quadratic Loss	0.00868181	0.00000

Table 8: Generalized Additive Model

FA methods for dimension reduction) are also viable in the context of factor modeling. In addition, the model is computationally intensive, and its tuning is limited to few hyperparameters. One can hence expect better results given more computational power. [107]

Generalized Additive Models are also evaluated using both Huber and quadratic loss functions. I use second order splines as the smooth functions and apply Group Lasso Penalty on the GAM.

Empirical results on GAM with Group Lasso suggest that, while not significantly outperforming other models, GAM shows slight improvement compared to benchmark low-dimensional models. Its out-of-sample performance is slightly inferior to Lasso and ICA, yet superior to ridge. It's worth noting that empirical findings might not fully capture optimal outcomes due to limited exploration of hyperparameters. In summary, GAM's empirical performance highlights its relevance, although results don't offer definitive guarantees.

Finally, Regression trees' performance is assessed. First, Regression trees with cost complexity pruning are tested.

Regression Tree with Cost Complexity Pruning	MSE_{OOS}	R^2_{OOS}
	0.04377	-143.1835427

Table 9: Standard Regression Tree with Pruning

This model performs the worst among all others and its out of sample performance is far from benchmark models' performance. Hence, Regression Trees with Cost Complexity Pruning largely fails to outperform traditional factor models. I have explained in the Regression Tree Chapter that trees do overfit asymptotically in-sample. This theoretical guarantee has not even been observed empirically due to practical computational constraints [108] Nonetheless, Compared to a Naive Ordinary Least Squares, Trees perform better.

Boosting is then applied on Regression Trees. The set of hyperparameters needed for tuning is the learning rate, the maximum depth of the sub tree, and the number of estimators (That is, the number

[107] Another practical limitation of this model is that the unmixing process results in very small values (Given that my dataset is comprised of numbers between 0 and 1), which is not easily manipulable.

[108] As the number of features used (912) is large, the tree is sensitive to very small Cost Complexity Penalties. These, small magnitudes could not be reached computationally due to round off error for small values of penalization. Adding to this issue the inherent Cost Complexity of trees; Regression trees with pruning is very costly computationally.

of submodels).

Boosted Regression Tree	MSE_{OOS}	R^2_{OOS}
	0.01294	1.2535847

Table 10: Gradient Boosted Regression Tree

Even though Boosted Trees do perform better than the Regression Tree with cost complexity pruning,their out-of-sample performance falls significantly short when compared to other regularization methods. Furthermore, they do not compete with the benchmark factor models. The main issue with Gradient Boosted Trees in the context of high dimensional factor modeling is computational: In fact, Trees are computationally expensive methods, and tuning three parameters makes them practically infeasible for the large panel data I am working on. In this empirical test, the best model has a learning rate of 0.01, a maximum depth of 1, and 500 submodels; testing for another depth measure, for instance, would further require 500 new model estimations. Hence, Because the number of sets of hyperparameters used is small in my estimation, I cannot be conclusive about the bad performance of boosting.

Random Forest are finally evaluated. The hyperparameters of this model are the maximum depth of the subtrees, the maximum number of selected features at each split and the number of estimators.

Random Forest	MSE_{OOS}	R^2_{OOS}
	0.00810284	1.56944

Table 11: Random Forest

Random Forests exhibit the highest out of sample performance among all other models. This result is not surprising given the nature of the problem and the theoretical guarantees associated with random forests. In fact, building a high dimensional panel model is highly complex and Random Forests are ideal to smooth out non linear complex relationships. Here too, the tuning process is far from being optimal; like for the gradient boosted trees, I expect even better results for more available computational power.

13 Conclusion

In this paper, I review the underlying logic behind factor models derived from the assumptions of the law of one price and the absence of arbitrage opportunities. Factor models primarily involve establishing a connection between the stochastic discount factor and empirical data. While there exists a long standing reliance on economically motivated methods to link the SDF to data (ICAPM and APT), their weak predictive power has prompted a shift in academia towards a more pragmatic factor modeling tradition. Within this context, high-dimensional factor modeling and machine learning techniques come into play.

I explain in details , with rigorous mathematical arguments, why Machine learning methods are important in the context of High dimensional modeling .

Building upon the Euler equation, I construct high-dimensional models using machine learning techniques. Machine learning both regularizes the Euler equation and adds non linearity to it. The empirical analysis involves constructing high-dimensional models using extensive panel data for return forecasting. The dependent variable is the returns on stocks for various firms across different time periods, while the factors encompass a comprehensive list of firm characteristics, macro indicators, and their interactions.

Ridge , Lasso , Elastic Net regressions , Principal Component Analysis, Independent Component Analysis , Generalized additive Models, Regression Trees with pruning, Random Forests , Boosted Trees are all explained in details. For each of the following methods, I outline their mathematical properties, their limitation, and their corresponding estimation method - The accelerated proximal gradient descent method, a common estimation approach for all linear models, constitutes a separate chapter and is explained thoroughly .

Empirical performances of High dimensional models using Machine Learning Methods are compared to standard robust low dimensional factor models. Empirical results are all in accordance with the initial premise: Machine Learning Methods are necessary for high dimensional Factor modeling. In fact, all of the methods tested in this paper perform much better than a naive OLS on the high dimensional dataset. Results are also in accordance with the theoretical properties of each model .

In ascending order, the most performing models are: Random forests, Principal Component regression , Elastic net (and Lasso), Regression using Independent Component Analysis and finally Generalized additive models. These models all exhibit performance on par with or surpass the benchmark low-dimensional models.

This paper uniquely contributes by putting Machine Learning methods into perspective with the theoretical underpinnings of factor models. It distinguishes itself through a rigorous and comprehensive presentation of the mathematical foundations underlying each employed method. Furthermore, it

conducts a practical out-of-sample empirical analysis and introduces an innovative dimension reduction technique—Independent Component Analysis (ICA), a seemingly effective dimension reduction method which is rarely documented in Academic papers[109].

Having clarified the objectives and the strength of my paper, it is important to note that this paper does not propose these High Dimensional Factor Models as practical tools for investing. As explained in the introduction, predicting returns using my panel model is far fetched from a purely practical perspective, and is only relevant when compared to other factor models, or to economically motivated models. That being said, the tools presented in this paper can be undoubtedly beneficial in more specific contexts i.e. cross-sectional, time series or more constrained panel forecasts.

[109] As far as I know, there has never been any documented independent Component-based Factor Model based on power data for IC ordering in Empirical Asset Pricing Academia.

14 Technical Appendix

14.1 Multicollinearity Evaluation

Even though linear dependence between a pair of vectors is easily identified by computing their correlation, identifying the linear dependence or near-linear dependence of **multiple** vectors is not straightforward. We use metrics like the variance inflation factor (VIF) and the conditional number to quantify multicollinearity.

14.1.1 The Variance Inflation Factor

The VIF quantifies multicollinearity in a linear regression by computing the ratio of the variance of the regression coefficient var $\left(\widehat{\beta}_i\right) = \sigma^2 \left(X^T X\right)^{-1}_{i,i}$ (Assuming a Gauss Markov model with demeaned predictors X) over the variation of the regression coefficient assuming the predictors are uncorrelated var $\left(\widehat{\beta}_i\right) = \sigma^2 \left(n s^2_{X_i}\right)^{-1}$ (See how, considering predictors are uncorrelated, the covariance matrix of X is now assumed to be the diagonal sample variance of X .We multiply by n, as our goal is to represent the diagonalized version of $X^T X$ The average VIF for all predictors gives us a measure of multicollinearity; one can see from the mentioned formula that the $\overline{VIF}$ corresponding to no multicollinearity is 1 and it is widely assumed - but not proved (Shalizi , O'Brien 2007)- that a VIF¿10 indicates high multicollinearity .

14.1.2 The Conditional Number

Conditional number is another metric for measuring multicollinearity. Defined as $K(A) = \|A\| \cdot \left\|A^{-1}\right\|$ with $\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$ and $\left\|A^{-1}\right\| = \min_{x \neq 0} \frac{\|Ax\|}{\|x\|}$. It measures the ratio of the biggest relative stretching of X by the transformation over its biggest relative shrinking. Intuitively,K(A) can be seen as the amount of distortion of some unit sphere by A and hence is a good measure of ill-conditionality (and.(un)stability) of the matrix: A high K(A) indicates a high distortion and vice versa. In addition, Like for the VIF there are no proven benchmarks above which there is serious Multicollinearity .

14.2 Backfitting Algorithm for GAMs

To understand the logic underlying the back fitting algorithm: I first minimize the squared loss between the observed values and my Generalized additive model:

$$\min E\left[Y - \left(\alpha + \sum_{j=1}^{p} f_j\left(X_j\right)\right)\right]^2 \quad (A1)$$

And by the theory of projections :

$f_K(X_K) = E\left[Y - \left(\alpha + \sum_{j\neq K}^{p} f_j(X_j)\right) \mid X_K\right]$, for all $K = 1, ..., p (A2)$

That is, if the Generalized Additive model is valid on all predictors, the conditional (on X_K) expected value of the partial residual (the difference between Y and all-expect-the k-th feature) is equal to the smoothing function applied on the k-th predictor. This is intuitive : In a well defined additive model, one should expect that the smoothing function at each feature K is fit by fitting the partial residual (The Y after we have gotten rid of the effects of all the j $\neq$ k smoothers) on X_K.

We can simply represent $(A2)$ in the following matrix form:

$$\begin{pmatrix} I & P_1 & \cdots & P_1 \\ P_2 & I & \cdots & P_2 \\ \vdots & & \ddots & \vdots \\ P_p & \cdots & P_p & I \end{pmatrix} \begin{pmatrix} f_1(X_1) \\ f_2(X_2) \\ \vdots \\ f_p(X_p) \end{pmatrix} = \begin{pmatrix} P_1 Y \\ P_2 Y \\ \vdots \\ P_p Y \end{pmatrix}$$, where each $P_i(\cdot) = E(\cdot \mid X_i)$

Since, we are finding $f_j()$s using smoothing methods [110], We can write:

$$\begin{pmatrix} I & S_1 & \cdots & S_1 \\ S_2 & I & \cdots & S_2 \\ \vdots & & \ddots & \vdots \\ S_p & \cdots & S_p & I \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_p \end{pmatrix} = \begin{pmatrix} S_1 Y \\ S_2 Y \\ \vdots \\ S_p Y \end{pmatrix}$$, where S_i is a smoothing matrix that estimates $P_i Y = E(Y|X_i)$

Solving with the Gauss–Seidel iterative method this $Ax = b$ problem, we get the Backfitting Algorithm:

$\hat{f}_j^{(\ell)} \leftarrow \text{Smooth}\left[\left\{y_i - \hat{\alpha} - \sum_{k\neq j} \hat{f}_k(x_{ik})\right\}_1^N\right]$, at each iteration (l)

14.3 The Variance-Bias Decomposition

Detailed illustration of how Variance-Bias Decomposition is different than the Estimation-Approximation error:

The following illustration is sourced from "Bias/Variance is not the same as Approximation/Estimation " paper (2023)

$$\underbrace{\mathbb{E}_{\boldsymbol{x}}\left[\ell\left(\boldsymbol{y}^*, \dot{f}_\phi(\boldsymbol{x})\right)\right]}_{\text{bias}} = \underbrace{R(f^*) - R(\boldsymbol{y}^*)}_{\text{approximation error}} + \underbrace{R\left(\dot{f}_\phi\right) - R(f^*)}_{\text{estimation bias}}$$

$$\underbrace{\mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_D\left[\ell\left(\hat{f}_\phi(\boldsymbol{x}), \hat{f}(\boldsymbol{x})\right)\right]\right]}_{\text{variance}} = \underbrace{\mathbb{E}_D\left[R(\hat{f}) - R\left(\hat{f}_{\text{erm}}\right)\right]}_{\text{optimisation error}} + \underbrace{\mathbb{E}_D\left[R\left(\hat{f}_{\text{erm}}\right) - R\left(\hat{f}_\phi\right)\right]}_{\text{estimation variance}}$$

Bias is not equivalent to approximation error, and variance is not the same as estimation error.

[110] Note: Smoothing method encompass a myriad of models. Smoothing does not mean spline smoothing. But rather refers to statistical regression methods that involves interpolation/extrapolation method : Shalizi (2021) defines smoothing to be any model of the form $\widehat{\mu}(x) = \sum_{i=1}^{n} y_i w(x, x_i, b)$

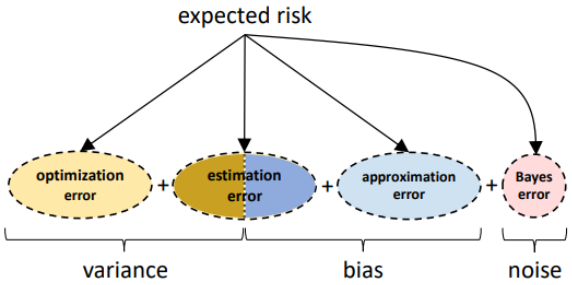


Figure 16: The estimation - approximation bias is not equivalent to the variance - bias error, even though they share some common rationale

14.4 Least-Norm solution

Graphical Representation of the Least-Norm Solution in Under determined systems

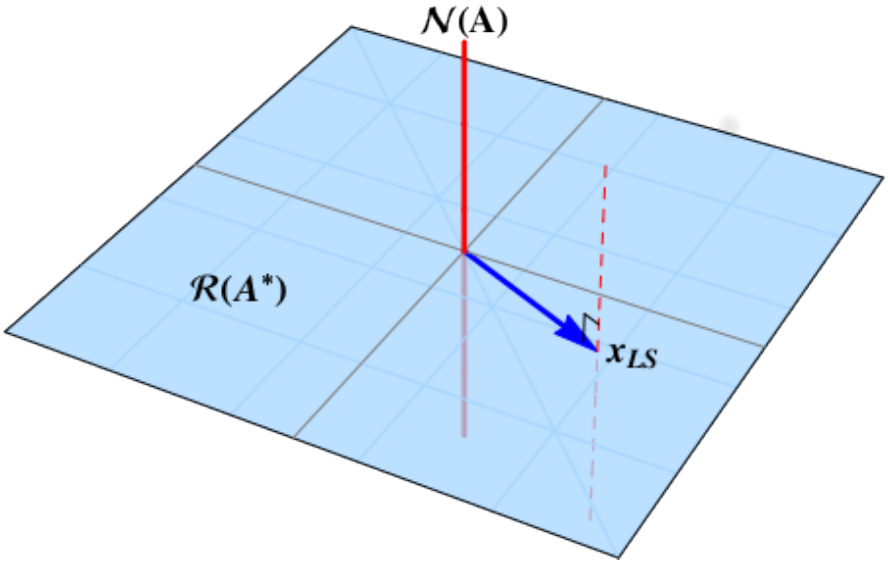


Figure 17: Enter Caption

14.5 Proximal Operators

Closed forms proximal operators:

Objective functions can be split into smooth and non smooth function. In our optimization problems; our Non-smooth functions are Ridge, Lasso, Elastic Net and Group Lasso. Their Proximal operator Closed Form solutions are:

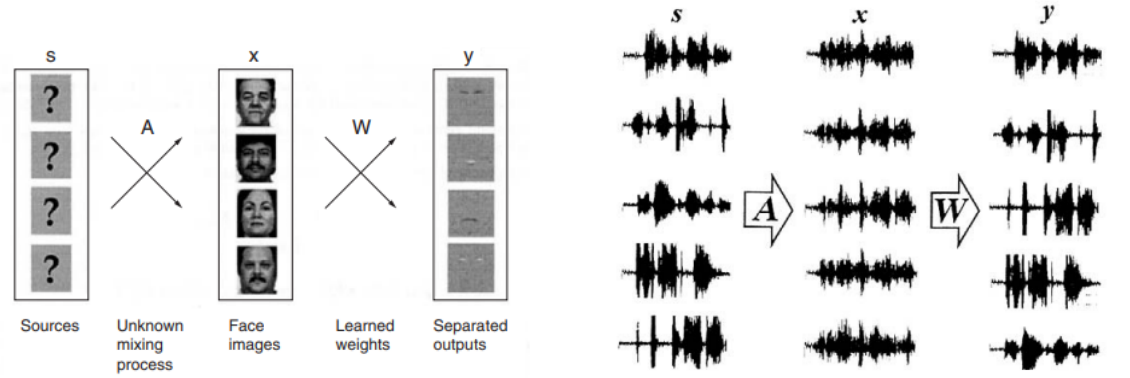
$$\text{prox}_{\gamma\phi}(\theta) = \begin{cases} \frac{\theta}{1+\lambda\gamma}, & \text{Ridge;} \\ \lambda S(\theta, \lambda\gamma), & \text{Lasso;} \\ \frac{1}{1+\lambda\gamma\rho} S(\theta, (1-\rho)\lambda\gamma), & \text{Elastic Net;} \\ \left(\widetilde{S}\left(\theta_1, \lambda\gamma\right)^\top, \widetilde{S}\left(\theta_2, \lambda\gamma\right)^\top, \ldots, \widetilde{S}\left(\theta_P, \lambda\gamma\right)^\top\right)^\top, & \text{Group Lasso.} \end{cases},$$

where $S(x,\mu)$ and $\widetilde{S}(x,\mu)$ are defined by:

$$(S(x,\mu))_i = \begin{cases} x_i - \mu, & \text{if } x_i > 0 \text{ and } \mu < |x_i|\,; \\ x_i + \mu, & \text{if } x_i < 0 \text{ and } \mu < |x_i|\,;, \\ 0, & \text{if } \mu \geq |x_i|\,. \end{cases} \qquad (\widetilde{S}(x,\mu))_i = \begin{cases} x_i - \mu \frac{x_i}{\|x_i\|}, & \text{if } \|x_i\| > \mu \\ 0, & \text{if } \|x_i\| \leq \mu. \end{cases}$$

14.6 Independent Component Analysis Visualized

Typical applications of independent Component Analysis are in signal processing and in computer vision. Here is a schematic representation of ICA's rationale - Stone (2004)[25] :



((a)) Independent Component Analysis for Computer Vision

((b)) Independent Component Analysis for Signal Processing

Figure 18: Independent Component Analysis Visualized

In my paper, I use independent component analysis on factor models, offering a distinctive approach with respect to Empirical Asset Pricing Academia as the utilization of ICA in factor models is scarcely documented in existing literature.

14.7 List of Factors

The list of Independent Variables used is extensive. They can be grouped into two different categories: Macro economic indicators and firm specific features.

This list of firms characteristics is made public by D. Xiu (2020) on his website, resenting an updated version of Zhang et al. (2017) [8]already documented list of firms characteristics. Those are a list of 94 Characteristics widely used and tested Factors in Empirical Asset Pricing Academia.

No.	Acronym	Firm characteristic	Paper's author(s)	Year, Journal	Data Source	Frequency
1	absacc	Absolute accruals	Bandyopadhyay, Huang & Wirjanto	2010, WP	Compustat	Annual
2	acc	Working capital accruals	Sloan	1996, TAR	Compustat	Annual
3	aeavol	Abnormal earnings announcement volume	Lerman, Livnat & Mendenhall	2007, WP	Compustat+CRSP	Quarterly
4	age	# years since first Compustat coverage	Jiang, Lee & Zhang	2005, RAS	Compustat	Annual
5	agr	Asset growth	Cooper, Gulen & Schill	2008, JF	Compustat	Annual
6	baspread	Bid-ask spread	Amihud & Mendelson	1989, JF	CRSP	Monthly
7	beta	Beta	Fama & MacBeth	1973, JPE	CRSP	Monthly
8	betasq	Beta squared	Fama & MacBeth	1973, JPE	CRSP	Monthly
9	bm	Book-to-market	Rosenberg, Reid & Lanstein	1985, JPM	Compustat+CRSP	Annual
10	bm_ia	Industry-adjusted book to market	Asness, Porter & Stevens	2000, WP	Compustat+CRSP	Annual
11	cash	Cash holdings	Palazzo	2012, JFE	Compustat	Quarterly
12	cashdebt	Cash flow to debt	Ou & Penman	1989, JAE	Compustat	Annual
13	cashpr	Cash productivity	Chandrashekar & Rao	2009, WP	Compustat	Annual
14	cfp	Cash flow to price ratio	Desai, Rajgopal & Venkatachalam	2004, TAR	Compustat	Annual
15	cfp_ia	Industry-adjusted cash flow to price ratio	Asness, Porter & Stevens	2000, WP	Compustat	Annual
16	chatoia	Industry-adjusted change in asset turnover	Soliman	2008, TAR	Compustat	Annual
17	chcsho	Change in shares outstanding	Pontiff & Woodgate	2008, JF	Compustat	Annual
18	chempia	Industry-adjusted change in employees	Asness, Porter & Stevens	1994, WP	Compustat	Annual
19	chinv	Change in inventory	Thomas & Zhang	2002, RAS	Compustat	Annual
20	chmom	Change in 6-month momentum	Gettleman & Marks	2006, WP	CRSP	Monthly
21	chpmia	Industry-adjusted change in profit margin	Soliman	2008, TAR	Compustat	Annual
22	chtx	Change in tax expense	Thomas & Zhang	2011, JAR	Compustat	Quarterly
23	cinvest	Corporate investment	Titman, Wei & Xie	2004, JFQA	Compustat	Quarterly
24	convind	Convertible debt indicator	Valta	2016, JFQA	Compustat	Annual
25	currat	Current ratio	Ou & Penman	1989, JAE	Compustat	Annual
26	depr	Depreciation / PP&E	Holthausen & Larcker	1992, JAE	Compustat	Annual
27	divi	Dividend initiation	Michaely, Thaler & Womack	1995, JF	Compustat	Annual
28	divo	Dividend omission	Michaely, Thaler & Womack	1995, JF	Compustat	Annual
29	dolvol	Dollar trading volume	Chordia, Subrahmanyam & Anshuman	2001, JFE	CRSP	Monthly
30	dy	Dividend to price	Litzenberger & Ramaswamy	1982, JF	Compustat	Annual
31	ear	Earnings announcement return	Kishore, Brandt, Santa-Clara & Venkatachalam	2008, WP	Compustat+CRSP	Quarterly

No.	Acronym	Firm characteristic	Paper's author(s)	Year, Journal	Data Source	Frequency
32	egr	Growth in common shareholder equity	Richardson, Sloan, Soliman & Tuna	2005, JAE	Compustat	Annual
33	ep	Earnings to price	Basu	1977, JF	Compustat	Annual
34	gma	Gross profitability	Novy-Marx	2013, JFE	Compustat	Annual
35	grCAPX	Growth in capital expenditures	Anderson & Garcia-Feijoo	2006, JF	Compustat	Annual
36	grltnoa	Growth in long term net operating assets	Fairfield, Whisenant & Yohn	2003, TAR	Compustat	Annual
37	herf	Industry sales concentration	Hou & Robinson	2006, JF	Compustat	Annual
38	hire	Employee growth rate	Bazdresch, Belo & Lin	2014, JPE	Compustat	Annual
39	idiovol	Idiosyncratic return volatility	Ali, Hwang & Trombley	2003, JFE	CRSP	Monthly
40	ill	Illiquidity	Amihud	2002, JFM	CRSP	Monthly
41	indmom	Industry momentum	Moskowitz & Grinblatt	1999, JF	CRSP	Monthly
42	invest	Capital expenditures and inventory	Chen & Zhang	2010, JF	Compustat	Annual
43	lev	Leverage	Bhandari	1988, JF	Compustat	Annual
44	lgr	Growth in long-term debt	Richardson, Sloan, Soliman & Tuna	2005, JAE	Compustat	Annual
45	maxret	Maximum daily return	Bali, Cakici & Whitelaw	2011, JFE	CRSP	Monthly
46	mom12m	12-month momentum	Jegadeesh	1990, JF	CRSP	Monthly
47	mom1m	1-month momentum	Jegadeesh & Titman	1993, JF	CRSP	Monthly
48	mom36m	36-month momentum	Jegadeesh & Titman	1993, JF	CRSP	Monthly
49	mom6m	6-month momentum	Jegadeesh & Titman	1993, JF	CRSP	Monthly
50	ms	Financial statement score	Mohanram	2005, RAS	Compustat	Quarterly
51	mvel1	Size	Banz	1981, JFE	CRSP	Monthly
52	mve_ia	Industry-adjusted size	Asness, Porter & Stevens	2000, WP	Compustat	Annual
53	nincr	Number of earnings increases	Barth, Elliott & Finn	1999, JAR	Compustat	Quarterly
54	operprof	Operating profitability	Fama & French	2015, JFE	Compustat	Annual
55	orgcap	Organizational capital	Eisfeldt & Papanikolaou	2013, JF	Compustat	Annual
56	pchcapx_ia	Industry adjusted % change in capital expenditures	Abarbanell & Bushee	1998, TAR	Compustat	Annual
57	pchcurrat	% change in current ratio	Ou & Penman	1989, JAE	Compustat	Annual
58	pchdepr	% change in depreciation	Holthausen & Larcker	1992, JAE	Compustat	Annual
59	pchgm_pchsale	% change in gross margin - % change in sales	Abarbanell & Bushee	1998, TAR	Compustat	Annual
60	pchquick	% change in quick ratio	Ou & Penman	1989, JAE	Compustat	Annual
61	pchsale_pchinvt	% change in sales - % change in inventory	Abarbanell & Bushee	1998, TAR	Compustat	Annual
62	pchsale_pchrect	% change in sales - % change in A/R	Abarbanell & Bushee	1998, TAR	Compustat	Annual

No.	Acronym	Firm characteristic	Paper's author(s)	Year, Journal	Data Source	Frequency
63	pchsale_pchxsga	% change in sales - % change in SG&A	Abarbanell & Bushee	1998, TAR	Compustat	Annual
64	pchsaleinv	% change sales-to-inventory	Ou & Penman	1989, JAE	Compustat	Annual
65	pctacc	Percent accruals	Hafzalla, Lundholm & Van Winkle	2011, TAR	Compustat	Annual
66	pricedelay	Price delay	Hou & Moskowitz	2005, RFS	CRSP	Monthly
67	ps	Financial statements score	Piotroski	2000, JAR	Compustat	Annual
68	quick	Quick ratio	Ou & Penman	1989, JAE	Compustat	Annual
69	rd	R&D increase	Eberhart, Maxwell & Siddique	2004, JF	Compustat	Annual
70	rd_mve	R&D to market capitalization	Guo, Lev & Shi	2006, JBFA	Compustat	Annual
71	rd_sale	R&D to sales	Guo, Lev & Shi	2006, JBFA	Compustat	Annual
72	realestate	Real estate holdings	Tuzel	2010, RFS	Compustat	Annual
73	retvol	Return volatility	Ang, Hodrick, Xing & Zhang	2006, JF	CRSP	Monthly
74	roaq	Return on assets	Balakrishnan, Bartov & Faurel	2010, JAE	Compustat	Quarterly
75	roavol	Earnings volatility	Francis, LaFond, Olsson & Schipper	2004, TAR	Compustat	Quarterly
76	roeq	Return on equity	Hou, Xue & Zhang	2015, RFS	Compustat	Quarterly
77	roic	Return on invested capital	Brown & Rowe	2007, WP	Compustat	Annual
78	rsup	Revenue surprise	Kama	2009, JBFA	Compustat	Quarterly
79	salecash	Sales to cash	Ou & Penman	1989, JAE	Compustat	Annual
80	saleinv	Sales to inventory	Ou & Penman	1989, JAE	Compustat	Annual
81	salerec	Sales to receivables	Ou & Penman	1989, JAE	Compustat	Annual
82	secured	Secured debt	Valta	2016, JFQA	Compustat	Annual
83	securedind	Secured debt indicator	Valta	2016, JFQA	Compustat	Annual
84	sgr	Sales growth	Lakonishok, Shleifer & Vishny	1994, JF	Compustat	Annual
85	sin	Sin stocks	Hong & Kacperczyk	2009, JFE	Compustat	Annual
86	sp	Sales to price	Barbee, Mukherji, & Raines	1996, FAJ	Compustat	Annual
87	std_dolvol	Volatility of liquidity (dollar trading volume)	Chordia, Subrahmanyam & Anshuman	2001, JFE	CRSP	Monthly
88	std_turn	Volatility of liquidity (share turnover)	Chordia, Subrahmanyam, &Anshuman	2001, JFE	CRSP	Monthly
89	stdacc	Accrual volatility	Bandyopadhyay, Huang & Wirjanto	2010, WP	Compustat	Quarterly
90	stdcf	Cash flow volatility	Huang	2009, JEF	Compustat	Quarterly
91	tang	Debt capacity/firm tangibility	Almeida & Campello	2007, RFS	Compustat	Annual
92	tb	Tax income to book income	Lev & Nissim	2004, TAR	Compustat	Annual
93	turn	Share turnover	Datar, Naik & Radcliffe	1998, JFM	CRSP	Monthly
94	zerotrade	Zero trading days	Liu	2006, JFE	CRSP	Monthly

Macro economic indicators are made public by Goyal on his website. By manipulating some of the proposed variables (See Code Processing Section) I reduce the list to eight macro economic factors:

Dividents/Price Ratio
Earnings/Price Ratio
Book/Market Ratio
Net Equity Expansion
Treasury Bill Rate
Term Spread
Default Spread
Stocks Variance

Table 12: List of Macroeconomic Factors

Finally, Dummy Factors represent firms' corresponding Industry. They are derived from the Standard Industrial Code (SIC) provided by D. Xiu. The first two digits of the SIC are indicative of the sector. Below is the list of sectors and their corresponding code.

SIC first digits	Industry
01-09	Agriculture, Forestry, Fishing
10-14	Mining
15-17	Construction
20-39	Manufacturing
40-49	Transportation and Public Utilities
50-51	Wholesale Trade
52-59	Retail Trade
60-67	Finance, Insurance, Real Estate
70-89	Services
91-99	Public Administration

Table 13: List of Industries Corresponding to the dummy variables

14.8 Code

The empirical findings are implemented in Python. The complete code is presented in a notebook format, organized into three main sections: Data Processing, Data Analysis, and Statistical Modeling. I have printed only important outputs and commented secondary but significant code segments.

Code:

Dataset: Feature dataset is public on Dacheng Xiu's website. The file's name is 'datashare.csv'(file found in directory). This file is a 3.6 GB file and does not open on excel. I use Pandas library to preprocess the data. The dependent variables (1-month holding period returns) are extracted from CRSP database. I extract monthly holding period returns from from 12-1925 to 12-2022 , as I could not access any more recent return. (NB : I am using crsp_a_stock package on WRDS). Macro dataset is public on Amit Goyal's webiste. File name is 'PredictorData2022.csv' file is in directory.

I process and clean the data, then provide a summary of the dataset by analysising its empirics. I finally apply statistical methods on the data.

I comment out sections of code when their output occupies excessive screen space.

Data Analysis

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pickle
import sys
import time
```

[ ]:
```
file_return = '/content/drive/MyDrive/charbel/returns_only.csv'
file_feature = '/content/drive/MyDrive/charbel/datashare.csv'

data_return= pd.read_csv(file_return)
data_feature =  pd.read_csv(file_feature)
```

[ ]:
```
#print("List of variables \n", data_feature.columns.tolist()  ) # list of all␣
 ↪vars in this file  - factors file
#print("\n head \n",data_feature.head())  # first few rows
print("\n Count Null \n" ,data_feature.isnull().sum())  # Shows the number of␣
 ↪null values in each column
```

```
 Count Null
 permno             0
DATE                0
mvel1            3070
beta           400564
betasq         400564
                 ...
retvol           3479
std_dolvol     316358
std_turn       305807
zerotrade      309813
sic2           300359
```

```
Length: 97, dtype: int64
```

Those correspond to the (97-DATE-permno-sic2) 94 characteristics described in the table in appendix . Permno is each stock's identifier in CRSP's database & and sic2 are Standard Industrial Classification (SIC) codes. RET refers to the Holding Period Return Now for the returns file

[ ]:
```
print("List of variables \n", data_return.columns.tolist()  ) # list of all vars␣
 ↪in this file  - factors file
print("\n Count Null \n" ,data_return.isnull().sum())  # Shows the number of␣
 ↪null values in each column
```

```
List of variables
 ['PERMNO', 'date', 'COMNAM', 'PERMCO', 'PRC', 'RET']

 Count Null
 PERMNO         0
date            0
COMNAM      35001
PERMCO          0
PRC        150689
RET         85298
dtype: int64
```

I change variable names in order to merge both dataframes later and for clarity.

[ ]:
```
data_return = data_return.rename(columns={'PERMNO': 'permno'})
data_return = data_return.rename(columns={'date': 'DATE'})
```

[ ]:
```
print(len(data_return), len(data_feature))
```

```
4927531 4117300
```

0.0.1 Merging returns and their associated features

[ ]:
```
merged_data = pd.merge(data_feature, data_return, how='inner', on=['permno',␣
 ↪'DATE'])
```

[ ]:
```
print("\n Count Null \n" ,merged_data.isnull().sum())  # Shows the number of␣
 ↪null values in each column
```

```
 Count Null
 permno         0
DATE            0
mvel1        3070
beta       400564
betasq     400564
            ...
sic2       300359
COMNAM          0
```

```
PERMCO         0
PRC        20481
RET        20468
Length: 101, dtype: int64
```

I have merged both (inner merge wrt pernmo and date) in the "merged_data" dataframe

Use different dates

[ ]:
```
#start_date, end_date = 19570101, 20161231
start_date, end_date = 20010101, 20201231
```

[ ]:
```
merged_data =␣
↪merged_data[(merged_data['DATE']>=start_date)&(merged_data['DATE']<=end_date)].
↪reset_index(drop=True)
merged_data['DATE'] = pd.to_datetime(merged_data['DATE'],format='%Y%m%d')+pd.
↪offsets.MonthEnd(0) #change time format
```

[ ]:
```
#print("List of variables for merged and constrained \n", merged_data.columns.
↪tolist()  ) # list of all vars in this file  - factors file
#print("\n head \n",merged_data.head())  # first few rows
print("\n Count Null \n" ,merged_data.isnull().sum())  # Shows the number of␣
↪null values in each column
print("number of observations" , len(merged_data))
```

```
 Count Null
 permno         0
DATE            0
mvel1         255
beta        98296
betasq      98296
            ...
sic2        22491
COMNAM          0
PERMCO          0
PRC          7810
RET          7799
Length: 101, dtype: int64
number of observations 1487672
```

I end up with 1,487,672 observations

I define a list of firms' characteristics as defined in paper. I exclude company name , database IDs (permno & permnco) , date, returns and industry code (sic) to get the list of characteristics

[ ]:
```
characteristics = list(set(merged_data.columns).
↪difference({'DATE','permno','COMNAM','sic2','PERMCO','PRC','RET'}))
```

```
[ ]: print(str(characteristics) + " are the firms' characteristics. \nThere are " +␣
      ↪str(len(characteristics)) + " characteristics.")
```

```
['stdcf', 'divo', 'pchgm_pchsale', 'chpmia', 'orgcap', 'pchsaleinv', 'maxret',
'beta', 'roavol', 'cfp', 'grcapx', 'cfp_ia', 'tang', 'pchcapx_ia', 'ep',
'invest', 'chinv', 'securedind', 'agr', 'age', 'cashdebt', 'divi', 'lgr',
'pchsale_pchinvt', 'chatoia', 'chcsho', 'egr', 'roic', 'pricedelay', 'aeavol',
'pchquick', 'operprof', 'chtx', 'dolvol', 'rd_mve', 'pchdepr',
'pchsale_pchxsga', 'mom12m', 'mve_ia', 'std_turn', 'herf', 'chempia', 'sp',
'gma', 'grltnoa', 'currat', 'rsup', 'ms', 'absacc', 'bm', 'dy', 'stdacc',
'turn', 'tb', 'secured', 'cinvest', 'cashpr', 'mvel1', 'ill', 'chmom', 'sgr',
'mom6m', 'pchcurrat', 'idiovol', 'convind', 'roeq', 'cash', 'mom1m', 'lev',
'salerec', 'depr', 'sin', 'saleinv', 'ps', 'nincr', 'quick', 'zerotrade',
'pctacc', 'ear', 'hire', 'salecash', 'indmom', 'acc', 'realestate', 'rd_sale',
'pchsale_pchrect', 'rd', 'retvol', 'baspread', 'roaq', 'std_dolvol', 'bm_ia',
'mom36m', 'betasq'] are the firms' characteristics.
There are 94 characteristics.
```

0.1 MISSING DATA

I drop observations for which I do not have the monthly holding period return and Replace the missing features with median of characteristic's value for each stock

```
[ ]: merged_data.isnull().sum()
```

```
[ ]: permno        0
     DATE          0
     mvel1       255
     beta      98296
     betasq    98296
               ...
     sic2      22491
     COMNAM        0
     PERMCO        0
     PRC        7810
     RET        7799
     Length: 101, dtype: int64
```

There are missing returns in the data

```
[ ]: #merged_data[merged_data['RET'].isnull()].tail(20)[['permno', 'DATE']] #last 20␣
      ↪missing returns
     #we see that the N/A observations are not specific to some particular time frame
     #merged_data[merged_data['RET'].isnull()].head(20)[['permno', 'DATE']] #First 20␣
      ↪missing returns
```

Drop obsevations for which returns are not available Now : the merged_data has become "merged_data_updt",after dropping the missing return

[ ]:
```
merged_data_updt = merged_data.dropna(subset=['RET']).reset_index(drop=True)
```

[ ]:
```
print(merged_data_updt.isnull().sum())
```

```
permno        0
DATE          0
mvel1       244
beta      98118
betasq    98118
          ...
sic2      21717
COMNAM        0
PERMCO        0
PRC          11
RET           0
Length: 101, dtype: int64
```

More cleaning * In the returns columns (RET) there are also some defective values (characters instead of numerical values) * I spot them, then update the merged_data_updt dataframe accordingly

[ ]:
```
non_convertible_indices = merged_data_updt.loc[pd.
 ↪to_numeric(merged_data_updt['RET'], errors='coerce').isna()].index
non_convertible_elements = merged_data_updt.loc[pd.
 ↪to_numeric(merged_data_updt['RET'], errors='coerce').isna(), 'RET']
print(non_convertible_indices)
print(non_convertible_elements.head(3))
```

```
Int64Index([   8499,  415015,  639821,  699752,  795249,  926856, 1339620,
            1377438, 1383112, 1432729, 1444702, 1454346],
           dtype='int64')
8499      B
415015    B
639821    B
Name: RET, dtype: object
```

[ ]:
```
print("the number of defective returns", len(non_convertible_indices))
```

```
the number of defective returns 12
```

[ ]:
```
merged_data_updt = merged_data_updt.drop(index=non_convertible_indices).
 ↪reset_index(drop=True)
```

[ ]:
```
print("the number of observations", len(merged_data_updt))
```

```
the number of observations 1479861
```

Sort out top and bottom firms. I create two datasets : One for the 10000 biggest firms and another for the 10000 smallest firms (cross-sectionally) merged_data_top : For each

date, I sort the top 1000 firms (according to their size : mvel1) descendingly. merged_data_bottom : same but top 1000 smallest firms .

[ ]:
```
merged_data_top = merged_data_updt.sort_values('mvel1',ascending=False).
 ↪groupby('DATE').head(1000).reset_index(drop=True)
merged_data_bottom = merged_data_updt.sort_values('mvel1',ascending=False).
 ↪groupby('DATE').tail(1000).reset_index(drop=True)
```

[ ]:
```
print(merged_data_top.isnull().sum())
#print(merged_data_bottom.isnull().sum())
#print(len(merged_data_bottom) , len(merged_data_top))
#rint(len(merged_data_bottom.columns))
```

```
permno       0
DATE         0
mvel1        0
beta      5453
betasq    5453
          ...
sic2      1540
COMNAM       0
PERMCO       0
PRC          0
RET          0
Length: 101, dtype: int64
```

I end up with ~1,5 Mln observations for the whole panel and 240,000 observations for top and bottom panels (That is because for each of the 240 time frames, I pick the 1000 biggest and smallest firms)

I now want to replace the missing values by the cross sectional median value at each month. For this, I first group by date then for each characteristic (i), I use the function defined by lambda x:x.fillna(x.median()). lambda is a generic way to define a function (w/out passing by def()) to transform each characteristic to an updated one whereby N/A values are filled by the median in a given date group.

[ ]:
```
for i in characteristics:
     merged_data_updt[i] = merged_data_updt.groupby('DATE')[i].transform(lambda␣
 ↪x: x.fillna(x.median()))
```

[ ]:
```
columns_with_missing_values = merged_data_updt.columns[merged_data_updt.isnull().
 ↪any()]
print(columns_with_missing_values)
```

```
Index(['sic2'], dtype='object')
```

[ ]:
```
temp_merged_updt_2001 = merged_data_updt    # create a temp var for comparison -␣
 ↪later in the code (nxt 2 lines)
```

For the 2001-2020 time windows, this is sufficient, but, it might be the case that all values of characteristics in a certain group might be null,replacing NA by the median

would be innefective. To be on the safe side, I replace the remaining N/A by 0.

Because the previous step is dependent on the initial time frame, then, it may or may not be needed. I verify this here:

[ ]:
```
for i in characteristics:
     merged_data_updt[i] = merged_data_updt[i].fillna(0)
```

[ ]:
```
# Checking
are_equal = temp_merged_updt_2001.equals(merged_data_updt)
print("Are the DataFrames equal?", are_equal,". if true, we're good to go.␣
↪nothing has changed ( NOTE: This result changes from an initial time window to␣
↪another)")
```

```
Are the DataFrames equal? True . if true, we're good to go. nothing has changed
( NOTE: This result changes from an initial time window to another)
```

I do the same for bottom and top dataframes Now add to the name of top and bottom dataframes : #### "_updt"

[ ]:
```
def fill_missing(data, characteristics):
    for i in characteristics:
         data[i] = data.groupby('DATE')[i].transform(lambda x: x.fillna(x.
↪median()))
    for i in characteristics:
         data[i] = data[i].fillna(0)
    return data
```

[ ]:
```
merged_data_top_updt = fill_missing(merged_data_top, characteristics)
merged_data_bottom_updt = fill_missing(merged_data_bottom, characteristics)
```

[ ]:
```
print("missing data in top:",merged_data_top_updt.columns[merged_data_top_updt.
↪isnull().any()]) #checking for missing vars
print("missing data in bottom",merged_data_bottom_updt.
↪columns[merged_data_bottom_updt.isnull().any()] )
```

```
missing data in top: Index(['sic2'], dtype='object')
missing data in bottom Index(['sic2'], dtype='object')
```

No relevant missing variable. We are good to go. I check if there are still 1000 stocks per month.

[ ]:
```
#[(merged_data_top_updt.groupby('DATE')['permno'].nunique() != 1000).any()]
#[(merged_data_bottom_updt.groupby('DATE')['permno'].nunique() != 1000).any()] #␣
↪yes, we are good to go
```

Now that I got rid of observations with missing dependent variables ; and solved the problem of missing value. I create dummy vars for the SIC variable. SIC are the standard industial classification codes: The 2 first digits defining it represent an industry section. I want to represent them as dummy variables

I use get_dummies() from pandas library. and define a function that first on gets dummies on for the sic2 column and fills missing values by 619619 in order to drop them later; Now this function - in its standard version creates a col for each diff variable and gives it the same name as the variable - this function also permits adding a prefix (written prefix_), I add sic; then at the end drop the NA sic

I will also drop irrelevant variables in the process

[ ]:
```
def create_sic_dummies(df):
    sic_dummies = pd.get_dummies(df['sic2'].fillna("619619").
↪astype(int),prefix='sic').drop('sic_619619',axis=1)
    df_dummy = pd.concat([df,sic_dummies],axis=1) # merge
    df_dummy.drop(['PERMCO','PRC','sic2'],inplace=True,axis=1) #drop sic2 , we␣
↪do not need it anymore. And  drop irrelevant columns. Keep Permno, Date and␣
↪Company name
    return df_dummy
```

[ ]:
```
merged_data_updt_dummy = create_sic_dummies(merged_data_updt)
```

[ ]:
```
#print("List of variables W/ DUMMY \n", merged_data_updt_dummy.columns.tolist() ␣
↪)
#print("\n head \n",merged_data_updt_dummy.head())  # first few rows
#print("\n tail \n",merged_data_updt_dummy.tail())  # last few rows
#print(merged_data_updt_dummy.isnull().any().sum()) #should print false
print( "\nthe number of dummy variables" ,len((merged_data_updt_dummy).columns)␣
↪- len((merged_data_updt).columns) + 3 )
#print(len((merged_data_updt_dummy).columns))
```

```
the number of dummy variables 73
```

The merged data is now called: merged_data_updt_dummy Given the chosen time period I have 73 dummies representing the different industry sectors

No more missing values in firms' characteristics, and observations with n/a returns are dropped

We're done with merging returns to features and cleaning the merged data

To resume : We are using (Given the 2001-2020 time frame) : 1,479,873 - (defective returns) observations from the beginning of 2001 to the end of 2020. for 14,614 firms and 94 characteristics 73 sector dummies and the dataframe contains 171 columns (Note: In the cleaning process I got rid of unuseful cols

I do the same thing with top and bottom dataframes

[ ]:
```
merged_data_top_updt_dummy = create_sic_dummies(merged_data_top_updt)
merged_data_bottom_updt_dummy = create_sic_dummies(merged_data_bottom_updt)
```

[ ]:
```
#[(merged_data_top_updt_dummy.groupby('DATE')['permno'].nunique() != 1000).any()]
#[(merged_data_bottom_updt.groupby('DATE')['permno'].nunique() != 1000).any()]
```

Note we get for top 1000 :

```
[ ]: #print(merged_data_top_updt_dummy.head())
#print(merged_data_bottom_updt_dummy.head())
#print(len((merged_data_top_updt_dummy.columns)))
print( "\nthe number of dummy variables for top dataframe is "␣
 ↪,len((merged_data_top_updt_dummy).columns) - len((merged_data_top_updt).
 ↪columns) + 3 )
print( "\nthe number of dummy variables for bottom dataframe is "␣
 ↪,len((merged_data_bottom_updt_dummy).columns) - len((merged_data_top_updt).
 ↪columns) + 3 )
#print("\n\n\n\n\n\n\n\n")
#print(merged_data_top_updt_dummy[merged_data_top_updt_dummy['DATE']<(np.
 ↪datetime64('2009-01-31'))].tail())
```

```
the number of dummy variables for top dataframe is  66

the number of dummy variables for bottom dataframe is  71
```

0.1.1 Note

I spot an error in the permnos

```
[ ]: #print(merged_data_updt_dummy.columns.tolist())
print("number of different company names" , merged_data_updt_dummy['COMNAM'].
 ↪nunique() , "Number of different permnos", merged_data_updt_dummy['permno'].
 ↪nunique() )
```

```
number of different company names 17851 Number of different permnos 15614
```

This suggests that the permnos refer to more than one company name. I solve this issue

```
[ ]: # Determining the issue
#print(merged_data_updt_dummy.head())
result_df = merged_data_updt_dummy.groupby('permno')['COMNAM'].
 ↪nunique(dropna=True).reset_index(name='unique_names_count')
result_df = result_df[result_df['unique_names_count'] > 1]
print(result_df[['permno', 'unique_names_count']])
```

```
       permno  unique_names_count
0       10001                   3
1       10002                   2
2       10012                   2
7       10028                   3
10      10037                   2
...       ...                 ...
15597   93416                   2
15606   93429                   2
15608   93431                   2
```

```
15610   93433                   2
15613   93436                   2

[3086 rows x 2 columns]
```

Permnos are bad identifiers as they refer to more than 1 company name sometimes. For example for 10001 we can see the list of company names

[ ]:
```
merged_data_updt_dummy.loc[merged_data_updt_dummy['permno'] == 10001 , 'COMNAM'].
 ↪unique()  #for example here are 3 COMNAM for permno 10001
#merged_data_updt_dummy.loc[merged_data_updt_dummy['permno'] == 10016 ,␣
 ↪'COMNAM'].unique()  # Just checking, As expected , I got just 1 permno for␣
 ↪10016
```

[ ]:
```
array(['ENERGY WEST INC', 'ENERGY INC', 'GAS NATURAL INC'], dtype=object)
```

[ ]:
```
#checking if there are in the same time period, two permnos .
has_duplicates = merged_data_updt_dummy.groupby('DATE')['permno'].
 ↪transform(lambda x: x.duplicated(keep=False))
print(has_duplicates.any()) #False. I am good to go .
```

```
False
```

I want for every company name an id, then delete the permno and replace it by the ids

[ ]:
```
print(merged_data_updt_dummy['COMNAM'].isnull().sum())  #checking if there are␣
 ↪missing column names; should get 0
merged_data_updt_dummy['ID'] = merged_data_updt_dummy.groupby('COMNAM').ngroup()␣
 ↪# create id for each new name
```

```
0
```

[ ]:
```
#uncomment to check
#print(merged_data_updt_dummy.head())
#print(merged_data_updt_dummy[merged_data_updt_dummy['ID'] == 347]['COMNAM'].
 ↪head(1))  #CHANGE ID number to see the observations (and corresponding permno)
#print("The number of firms/IDs is", merged_data_updt_dummy["ID"].nunique())
```

I can now drop the permno. And use the ID. I will then rename the "ID" to "permno". To keep the code simple. ##### merged_data_mod

[ ]:
```
merged_data_mod = merged_data_updt_dummy.drop('permno',axis=1)
```

Change 'ID' to 'permno'

[ ]:
```
#print(merged_data_mod.head())
merged_data_mod = merged_data_mod.rename(columns={'ID':'permno'})
```

[ ]:
```
#print(merged_data_mod.head(3))
```

I am good to go . I now have identifiers for each firm . Before proceeding: I also replace permnos for top and bottom firms. To do this:

- I first re-organize the top panel. Sort them by date; then by size whithin each date : "_org"
- Then, I check if the ID issue is present (should be if data is correct), and solve it.

[ ]:
```
#print(merged_data_top_updt_dummy.head())
merged_top_org = merged_data_top_updt_dummy.sort_values(by=['DATE', 'mvel1'],␣
↪ascending=[True, True]).reset_index(drop=True)
merged_bottom_org = merged_data_bottom_updt_dummy.sort_values(by=['DATE',␣
↪'mvel1'], ascending=[True, True]).reset_index(drop=True)
```

[ ]:
```
#print(merged_top_org.head(3))
#print(merged_bottom_org.head())
#check for id problem
print("For the top panel number of different company names" ,␣
↪merged_top_org['COMNAM'].nunique() , "Number of different permnos",␣
↪merged_top_org['permno'].nunique() )
print("\nNow for the bottom panel number of different company names" ,␣
↪merged_bottom_org['COMNAM'].nunique() , "Number of different permnos",␣
↪merged_bottom_org['permno'].nunique() )
```

I have 3,362 firms for the top panel and 6,997 for the bottom pannel. These numbers are coherent. I am picking from 2001 to 2020 :

- top 1000 firms. Many of them are the same at different periods. And some of them (fewer) are not repeated.
- and Bottom 1000 firms. Here the number of repeated firms is more volatile as lower size firms are less stable

However: The issue of ID is present in the "top" and "bottom" panels data . let's solve it

[ ]:
```
#checking for top panel ( uncomment )

#result_df = merged_top_org.groupby('permno')['COMNAM'].nunique(dropna=True).
↪reset_index(name='unique_names_count')
#result_df = result_df[result_df['unique_names_count'] > 1]
#print(result_df[['permno', 'unique_names_count']].tail(10))
#data_return.loc[data_return['permno'] == 92602  , 'COMNAM'].iloc[0]
#unique_names = data_return.loc[data_return['permno'] == 93002 , 'COMNAM'].
↪unique()
#print(unique_names)

# end of checking
#adding ID

merged_top_org['ID'] = merged_top_org.groupby('COMNAM').ngroup() # create id for␣
↪each new name
merged_bottom_org['ID'] = merged_bottom_org.groupby('COMNAM').ngroup()
```

```
#verifying if ID are well defined
#print(merged_top_org)
#print(merged_top_org[merged_top_org['ID'] == 3].head(10))  #CHANGE ID number to see the observations (and corresponding permno)
#print("The number of firms/IDs is", merged_top_org["ID"].nunique())
merged_top_mod = merged_top_org.drop('permno',axis=1)     #DROP PERMNO
merged_bottom_mod = merged_bottom_org.drop('permno',axis=1)        #DROP PERMNO
```

added "_mod" to top and bottom

```
merged_top_mod= merged_top_mod.rename(columns={'ID':'permno'})
merged_bottom_mod = merged_bottom_mod.rename(columns={'ID':'permno'})
```

Lastly, I get rid of the Company name column in each of the three dataframes. The ID (now called permno) is sufficient

```
merged_top_mod= merged_top_mod.drop('COMNAM', axis=1)
merged_bottom_mod = merged_bottom_mod.drop('COMNAM',axis =1 )
merged_data_mod = merged_data_mod.drop('COMNAM',axis =1 )
```

Now I have well defined dataframes

- merged_top_mod
- merged_bottom_mod
- merged_data_mod

```
#Remove comment to see them
#print(merged_top_mod.columns.tolist())
#print(merged_bottom_mod )
#print(merged_data_mod)
```

Adding Macro variables

Those are macroeconomics monthly indicators; Time range beginning 1871 - end 2022. I find analyze the content then clean it

This data is provided and thus Provided by Goyal. This data was used in Goyal and Welch's paper "Performance of Equity Premium Prediction" (2008)

The index variable represent the S&P500 index return , the E12 and the D12 are the 12-month moving sums of the earnings and dividend of the S&P500 ,I instead use the earnings price ratio (ep) and dividend price ratio (dp) which are computed as the ratio between the 12month div/earnings on the S&P and the index return. b/m - which is redefined as bm , is the book to market for Dow Jones industrial average. The tms is the term spread defined as the diference between the Long Term Yield (lty) and Treasury Bills (tbl) And the Default Yield Spread (dfy) is the difference between BAA and AAA-rated corporate bond

Ultimately, I will use these macro metrics only :Dividend-price ratio(dp), earnings-price ratio (ep), book-to-market ratio (bm), net equity expansion (ntis), Treasury-bill rate (tbl), term spread (tms), default spread (dfy), and stock variance (svar).

```
[ ]: data_macro = pd.read_csv('/content/drive/MyDrive/charbel/PredictorData2022.xlsx- Monthly.csv')
```

```
[ ]: print("OVERVIEW OF THE IMPORTED DATAFRAME.  List of variables \n", data_macro.columns.tolist()  )
#print("number of observations",len(data_macro))
#print("\n head \n",data_macro.head())  # first few rows
#print("\n tail \n",data_macro.tail())  # last few rows
#print("\n Summary \n",data_macro.info())  # summary
#print("\n col names \n",data_macro.columns)  #  column names
#print("\n nrow names \n" ,data_macro.index)    # index (row labels)
#print("\n data type \n",data_macro.dtypes)  # data types of each column
#print("\n Count Null \n" ,data_macro.isnull().sum())  #  number of null values in each column
```

I constraint the dataframe to some time range

```
[ ]: data_macro = data_macro[(data_macro['yyyymm']>=start_date//100)&(data_macro['yyyymm']<=end_date//100)].reset_index(drop=True)
```

```
[ ]: #print("\n head \n",data_macro.head())  # first few rows
print(data_macro.columns[data_macro.isnull().any()].tolist())
#print(len(data_macro.columns))
```

```
['csp']
```

```
[ ]: #create a copy of the DataFrame
data_macro_mod = data_macro.copy()   # because last line is causing memory issues
data_macro_mod['Index'] = data_macro_mod['Index'].str.replace(',', '').astype('float64')
data_macro_mod['ep'] = data_macro_mod['E12'] / data_macro_mod['Index']
data_macro_mod['dp'] = data_macro_mod['D12'] / data_macro_mod['Index']
data_macro_mod.rename({'b/m': 'bm'}, axis=1, inplace=True)
data_macro_mod['tms'] = data_macro_mod['lty'] - data_macro_mod['tbl']
data_macro_mod['dfy'] = data_macro_mod['BAA'] - data_macro_mod['AAA']
macro_indicators = ['dp', 'ep', 'bm', 'ntis', 'tbl', 'tms', 'dfy', 'svar']
data_macro_mod = data_macro_mod[['yyyymm'] + macro_indicators] #remove irrelevant vars
data_macro_mod['yyyymm'] = pd.to_datetime(data_macro_mod['yyyymm'],format='%Y%m') + pd.offsets.MonthEnd(0)
```

```
[ ]: #print("List of variables \n", data_macro_mod.columns.tolist()  )
#print("number of observations",len(data_macro_mod))
#print("\n head \n",data_macro_mod.head())  # first few rows
#print(data_macro_mod.columns[data_macro_mod.isnull().any()].tolist()) # No N/A as csp gets dropped
print("data_macro_mod contains", len(data_macro_mod), "observations and",len(data_macro_mod.columns)-1, "macro indicators")
```

```
data_macro_mod contains 240 observations and 8 macro indicators
```

Interacting macro economics indicators with the data I want to merge macro indicators with the data. I first build a table of macro indicator indexed by dates defined by the merged_data dataframe then I create an interaction function because I have three datasets to interact: data; merged_data_top I and merged_data_bottom (i.e. the top and bottom 1000 firms at each time period)

[ ]:
```
def interact_macro_data(merged_data_updt_dummy, data_macro, characteristics,␣
↪macro_indicators, minmax=True):
    data = merged_data_updt_dummy.copy()
    merged_and_macro = pd.
↪merge(data[['DATE']],data_macro,left_on='DATE',right_on='yyyymm',how='left')
    data = data.reset_index(drop=True)
    merged_and_macro = merged_and_macro.reset_index(drop=True)
    for i in characteristics:
        for j in macro_indicators:
            data[i+'*'+j] = data[i]*merged_and_macro[j]
    features = sorted(list(set(data.columns).difference({'DATE', 'RET',␣
↪'permno'})))
    if minmax:
        X = MinMaxScaler((-1,1)).fit_transform(data[features])
        X = pd.DataFrame(X, columns=features)
        X = pd.concat([data[['DATE', 'RET', 'permno']], X], axis=1)
    else:
        X = data
    y = data['RET']
    return X, y
```

SPLITING INTO TRAINING TESTING AND VALIDATION SETS I transform dates to np in order to manipulate them later

[ ]:
```
start_date_validation = np.datetime64('2009-01-31')
start_date_testing = np.datetime64('2015-01-31')
```

[ ]:
```
def splitting_sets(data):
    X_training, y_training =␣
↪interact_macro_data(data[data['DATE']<start_date_validation],data_macro_mod[data_macro_mod['y
↪macro_indicators,minmax=True)
    X_validation, y_validation =␣
↪interact_macro_data(data[(data['DATE']<start_date_testing)&(data['DATE']>=start_date_validati
↪minmax=True)
    X_testing, y_testing =␣
↪interact_macro_data(data[data['DATE']>=start_date_testing],data_macro_mod[data_macro_mod['yyy
↪macro_indicators,minmax=True)
    return X_training, X_validation, X_testing, y_training, y_validation,␣
↪y_testing
```

The final form is organized as such:

- For each of the 3 dataframes: top bottom and whole panel
- create validation testing and training dependent and independent matrices
- create reference matrices for each design matrix containing the date, the ID and the return associated to each set of feature - This will be used for checking purposes
- In the reference dataframes and for each dependent variable representing the returns, transform the returns from Objects to Float ; for manipulation purposes

[ ]:
```
def final_form(data):

  X_trn, X_vld, X_tst, y_trn, y_vld, y_tst = splitting_sets(data)

  X_trn_ref = X_trn
  X_vld_ref = X_vld
  X_tst_ref = X_tst

  X_trn_ref['RET'] = X_trn['RET'].astype(float)
  X_vld_ref['RET'] = X_vld['RET'].astype(float)
  X_tst_ref['RET'] = X_tst['RET'].astype(float)

  X_trn = X_trn.drop(['DATE', 'RET', 'permno'],axis=1)
  X_vld = X_vld.drop(['DATE', 'RET', 'permno'],axis=1)
  X_tst = X_tst.drop(['DATE', 'RET', 'permno'],axis=1)

  y_trn = y_trn.astype(float)
  y_vld = y_vld.astype(float)
  y_tst = y_tst.astype(float)

  return X_trn, X_vld, X_tst, X_trn_ref , X_vld_ref , X_tst_ref ,␣
↪y_trn,y_vld,y_tst
```

[ ]:
```
X_trn_bot, X_vld_bot, X_tst_bot, X_trn_ref_bot , X_vld_ref_bot , X_tst_ref_bot,␣
↪y_trn_bot,y_vld_bot, y_tst_bot = final_form(merged_bottom_mod)

X_trn, X_vld, X_tst, X_trn_ref , X_vld_ref , X_tst_ref, y_trn,y_vld, y_tst =␣
↪final_form(merged_top_mod)
```

Saving in drive

[ ]:
```
import pickle
```

[ ]:
```
'''
import pandas as pd

# List of DataFrames
dfs_bot = [X_trn_bot, X_vld_bot, X_tst_bot, X_trn_ref_bot, X_vld_ref_bot,␣
↪X_tst_ref_bot, y_trn_bot, y_vld_bot, y_tst_bot]
```

```
dfs = [X_trn, X_vld, X_tst, X_trn_ref, X_vld_ref, X_tst_ref, y_trn, y_vld, y_tst]

# Base path
base_path = "/content/drive/MyDrive/charbel/"

# Save DataFrames
for df, name in zip(dfs_bot + dfs, ['X_trn_bot', 'X_vld_bot', 'X_tst_bot',␣
↪'X_trn_ref_bot', 'X_vld_ref_bot', 'X_tst_ref_bot', 'y_trn_bot', 'y_vld_bot',␣
↪'y_tst_bot',
                                    'X_trn', 'X_vld', 'X_tst', 'X_trn_ref',␣
↪'X_vld_ref', 'X_tst_ref', 'y_trn', 'y_vld', 'y_tst']):
    path = base_path + f"{name}.pkl"
    df.to_pickle(path)
'''
```

The problem here is that since we found the top 1000 firms at each date Some firms may be present in a given month (t) but not in t+1 . Thus r_i,t+1 for z_i,t is not feasible

I WILL WORK,with ri,t and x i,t

NOW

- X_trn , X_vld , X_tst , y_trn , y_tst , y_vld have corresponding indeces add _bot for bottom panel data
- Use X_trn_ref , X_vld_ref , X_tst_ref to check on date/ret/permno add _bot for bottom panel

Empirics: What is the dataset about?

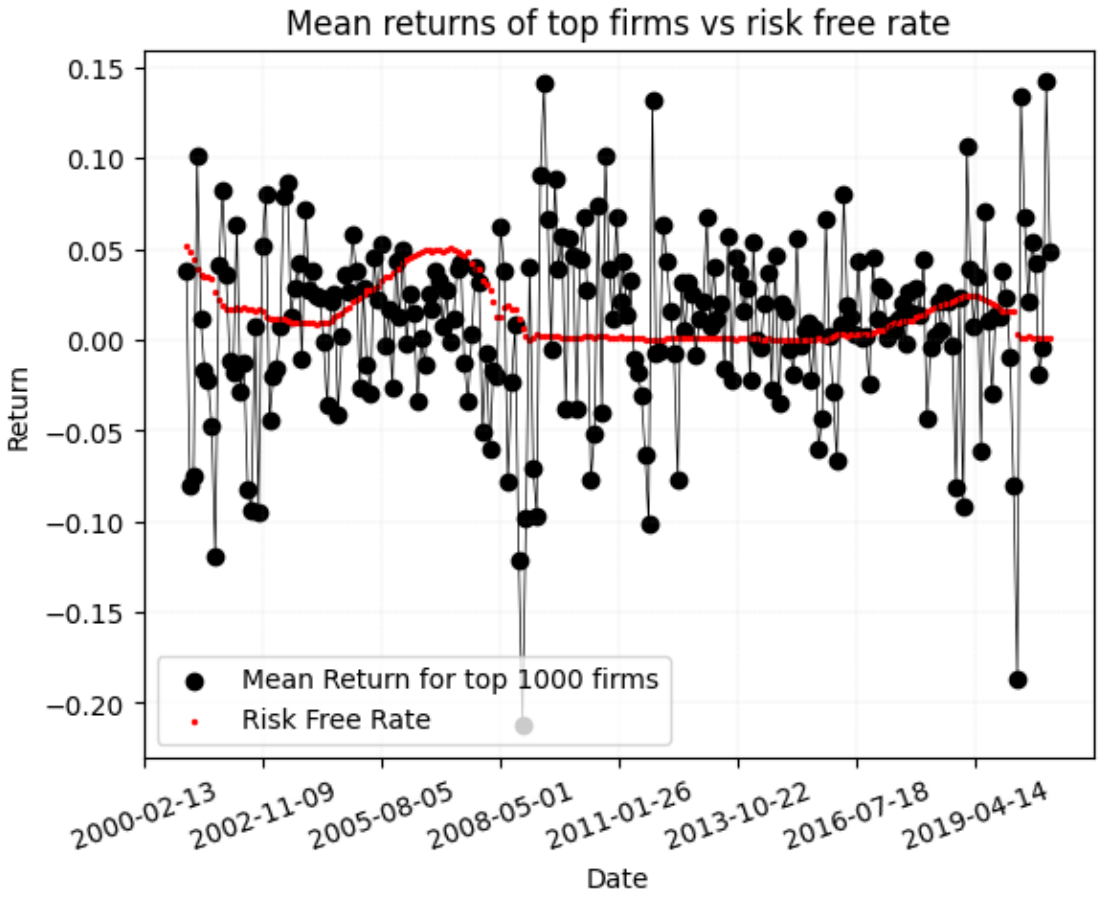
[ ]:
```
merged_top_mod['RET'] =merged_top_mod['RET'].astype(float)  # NEED TO CONVERT␣
↪RET TO FLOAT64
```

[ ]:
```
mean_ret = merged_top_mod.groupby("DATE")['RET'].mean()
#print(mean_ret.head(3))
```

[ ]:
```
#Now for the tbl ( risk free rate)
rf = data_macro_mod['tbl']
```

[ ]:
```
df = pd.DataFrame({'index': mean_ret.index, 'values': mean_ret.values})
df1 = pd.DataFrame({'index': mean_ret.index, 'values': rf})

plt.plot(df['index'], df['values'],  linestyle='-', marker='', color='black',␣
↪linewidth=0.5)
plt.scatter(df['index'], df['values'], label='Mean Return for top 1000 firms',␣
↪color='black')


plt.scatter(df['index'],df1['values'], label='Risk Free Rate', color = 'red',␣
↪s=2)
```

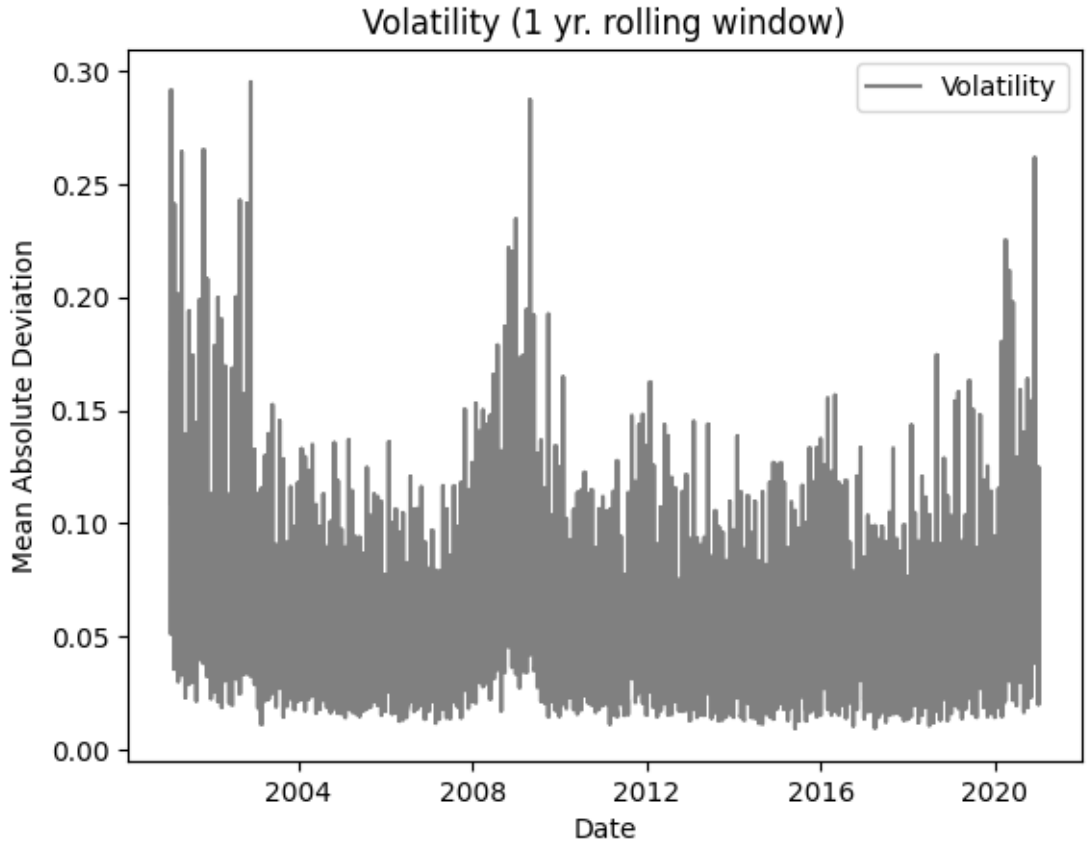
```
plt.gca().xaxis.set_major_locator(plt.MaxNLocator(integer=True))
plt.xticks(rotation=20)
plt.title('Mean returns of top firms vs risk free rate')
plt.xlabel('Date')
plt.ylabel('Return')
plt.legend()

plt.grid(True, linestyle=':', linewidth=0.2)
plt.savefig('plot.svg', format='svg')


plt.show()
```



[ ]:
```
rolling_window_size = 12
def mad_calculation(data):
    return np.abs(data - data.mean()).mean()
```

```
mad_result = merged_top_mod['RET'].rolling(window=rolling_window_size).
 ↪apply(mad_calculation, raw=True)
```

[ ]:
```
mad_result = merged_top_mod['RET'].rolling(window=rolling_window_size).
 ↪apply(mad_calculation, raw=True)
plt.plot(merged_top_mod['DATE'], mad_result, label="Volatility", color='grey')
plt.xlabel('Date')
plt.ylabel('Mean Absolute Deviation')
plt.title('Volatility (1 yr. rolling window)')
plt.legend()

# Save the plot as an SVG file
plt.savefig('mad_plot.svg')

# Show the plot
plt.show()
```



[ ]:
```
#sector diagram
#print(merged_top_mod.head())

result = merged_top_mod.groupby('permno').apply(lambda group: group.
 ↪filter(like='sic_').eq(1).idxmax(axis=1))
result_df = result.reset_index(level=0)
result_df.columns = ['permno', 'sic']
result_df = result_df.drop_duplicates(subset='permno', keep='first')

# Print or use the result DataFrame as needed
print(result_df)

#result.reset_index(drop=True)


#result.columns = ['id', 'sic_associated']


#print(result)
#print(len(result))
```

```
        permno     sic
232087       0   sic_1
234022       1   sic_1
209067       2  sic_82
2004         3  sic_63
144055       4  sic_35
...        ...     ...
213368    3357  sic_60
147026    3358  sic_28
224075    3359   sic_1
212003    3360   sic_1
218011    3361  sic_73

[3362 rows x 2 columns]
```

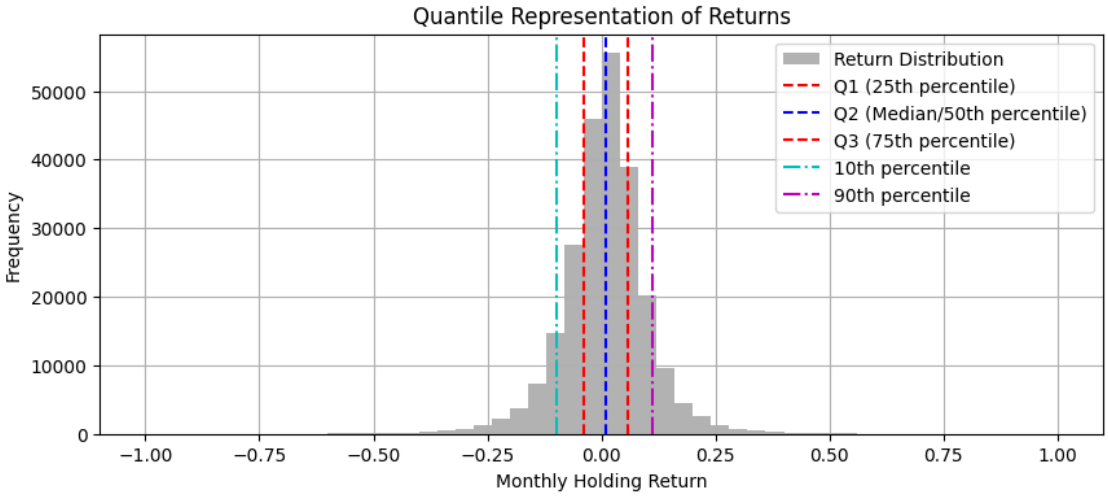
Quantile Analysis

[ ]:
```
returns_data = merged_top_mod['RET']
quant_val = 1/8*(np.arange(1,8))
quantiles = returns_data.quantile(quant_val)
for i in quant_val:
  if i==1/8:
    Qi_interval = (float('-inf'), quantiles[i])
  elif i ==8/8:
    Qi_interval = ( quantiles[i], float('inf') )
  else:
    Qi_interval = (quantiles[i-1/8], quantiles[i])
```

```
  print(Qi_interval)
```

```
(-inf, -0.083271625)
(-0.083271625, -0.0390485)
(-0.0390485, -0.011415375)
(-0.011415375, 0.0099945)
(0.0099945, 0.031626375)
(0.031626375, 0.05726025)
(0.05726025, 0.09729550000000001)
```

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
data = returns_data.tolist()
# Assuming 'data' is the correct variable name
q1 = np.percentile(data, 25)
q2 = np.percentile(data, 50)
q3 = np.percentile(data, 75)
p10 = np.percentile(data, 10)
p90 = np.percentile(data, 90)

plt.figure(figsize=(10, 4))
plt.hist(data, bins=50, alpha=0.6, color='grey', label="Return Distribution",␣
 ↪range=[-1, 1])
plt.axvline(x=q1, color='r', linestyle='--', label="Q1 (25th percentile)")
plt.axvline(x=q2, color='b', linestyle='--', label="Q2 (Median/50th percentile)")
plt.axvline(x=q3, color='r', linestyle='--', label="Q3 (75th percentile)")
plt.axvline(x=p10, color='c', linestyle='-.', label="10th percentile")
plt.axvline(x=p90, color='m', linestyle='-.', label="90th percentile")

plt.title("Quantile Representation of Returns")
plt.xlabel("Monthly Holding Return")
plt.ylabel("Frequency")
plt.legend()
plt.grid(True)

# Save the plot as a PNG file
plt.savefig('quantile_representation.png', format='png')
plt.show()
```



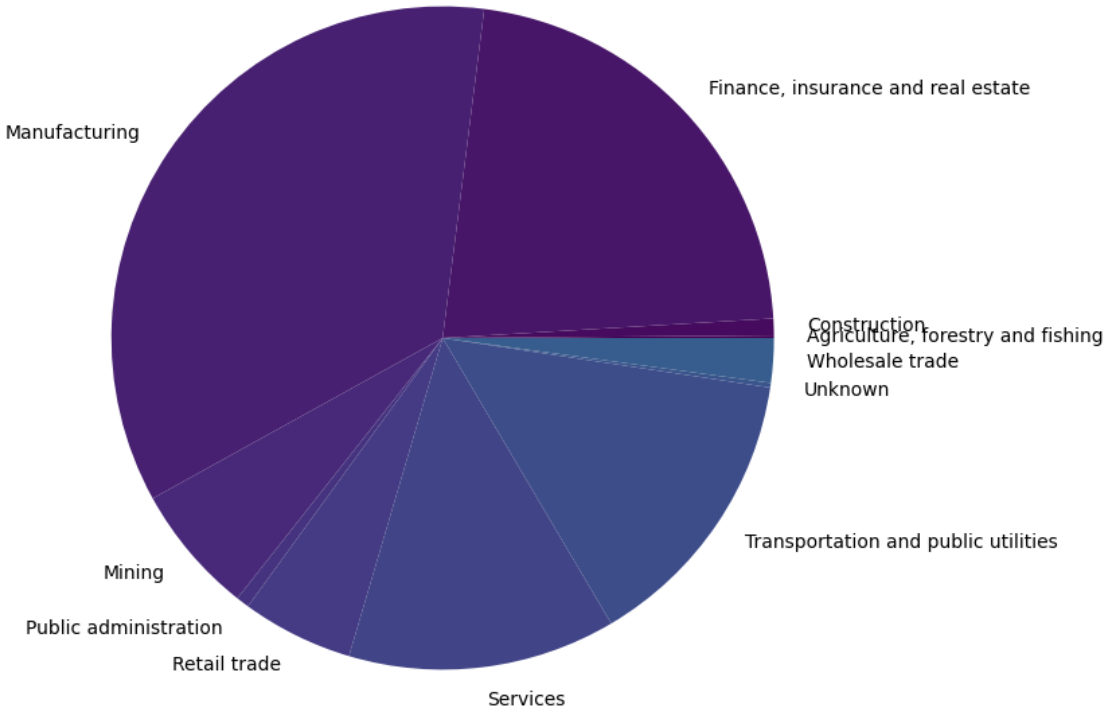
Industries Represented

[ ]:
```
industry_mapping = {
    range(1,9): 'Agriculture, forestry and fishing',
    range(10,14): 'Mining',
    range(15,19): 'Construction',
    range(20, 40): 'Manufacturing',
    range(40, 50): 'Transportation and public utilities',
    range(50, 52): 'Wholesale trade',
    range(52, 60): 'Retail trade',
    range(60, 68): 'Finance, insurance and real estate',
    range(70, 90): 'Services',
    range(91, 100): 'Public administration'
}
```

[ ]:
```
proportions_df = pd.DataFrame(list(proportions.items()), columns=['sic',␣
↪'proportion'])

def map_sic_to_group(sic_value):
    for sic_range, industry_group in industry_mapping.items():
        if int(sic_value.split('_')[1]) in sic_range:
            return industry_group
    return 'Unknown'

proportions_df['industry_group'] = proportions_df['sic'].apply(map_sic_to_group)
final_ind = proportions_df.groupby('industry_group').sum()

plt.figure(figsize=(8, 8))
num_parts = 70
```

```
colors = plt.cm.viridis(np.linspace(0, 2, num_parts))

plt.pie(final_ind['proportion'], labels=final_ind.index, colors=colors)

# Save the plot as a PNG file
plt.savefig('pie_chart.png')

# Show the plot
plt.show()
```

```
<ipython-input-165-3720d48fe442>:10: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  final_ind = proportions_df.groupby('industry_group').sum()
```



Those selected permnos will be used for the tailor made performance evaluation metric

- I first select the permnos that are always repeated int he dataset
- There are 199 stocks available

[ ]:
```
grouped_df = merged_top_mod.groupby('permno')['DATE'].agg(list).reset_index()
selected_permnos = []
for index, row in grouped_df.iterrows():
    if row['DATE'] == unique_dates_df['unique_dates'].tolist():
        selected_permnos.append(row['permno'])
```

[ ]:
```
print("Number of available stocks for building a dynamic strategy "␣
 ↪,len(selected_permnos))
#print(selected_permnos)
#print(grouped_df.iloc[44,:].tolist())   # checking if code worked
```

```
Number of available stocks for building a dynamic strategy  199
```

0.1.2 Statistical Modeling

Linear modeling on top panel. I first regress using a standard linear regression on the top panel and on the benchmark models The first R squared function is best suited for the model , But I had to use another R squared function , as the Most suited R squared is not convienient for more complex models : With Trees, GAM and even With elasticNet (using Huber) - it becomes computationally expensive) in order to be consistent, I use two types of out of sample R squared.

In the following 15 boxes I run a linear regression on benchmark low dimensional factor models and on the whole panel.

[ ]:
```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import ParameterGrid
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import ParameterGrid
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import HuberRegressor
```

[ ]:
```
def R_oos(actual, predicted):
    actual, predicted = np.array(actual), np.array(predicted)
    actual_mean = np.mean(actual)
    return 1 - (np.dot((actual-predicted),(actual-predicted)))/np.
 ↪dot((actual-actual_mean),(actual-actual_mean))
```

[ ]:
```
def evaluate(actual, predicted, insample):
    if insample == True:
        print('*'*15+'In-Sample Metrics'+'*'*15)
        print(f'The in-sample R2 is {r2_score(actual,predicted)*100:.7f}%')
        print(f'The in-sample MSE is {mean_squared_error(actual,predicted):.7f}')
    else:
        print('*'*15+'Out-of-Sample Metrics'+'*'*15)
        print(f'The out-of-sample R2 is {R_oos(actual,predicted)*100:.7f}%')
        print(f'The out-of-sample MSE is {mean_squared_error(actual,predicted):.
 ↪7f}')
```

Naive OLS on the feature set

[ ]:
```
from sklearn.linear_model import LinearRegression

OLS = LinearRegression().fit(X_trn,y_trn)
evaluate(y_trn, OLS.predict(X_trn), insample=True)
evaluate(y_tst, OLS.predict(X_tst),insample = False)
```

```
***************In-Sample Metrics***************
The in-sample R2 is 19.9166341%
The in-sample MSE is 0.0092240
***************Out-of-Sample Metrics***************
The out-of-sample R2 is -14276.6131380%
The out-of-sample MSE is 1.2355870
```

Low dimensional Linear Model using OLS I use well documented factors.Typically considered robust ols_3

[ ]:
```
from sklearn.linear_model import LinearRegression

# OLS with preselected size, bm, and momentum covariates
features_3 = ['mvel1','bm','mom1m','mom6m','mom12m','mom36m']
OLS_3 = LinearRegression().fit(X_trn[features_3],y_trn)
evaluate(y_trn, OLS_3.predict(X_trn[features_3]), insample=True)
evaluate(y_tst, OLS_3.predict(X_tst[features_3]),insample=False)
```

```
***************In-Sample Metrics***************
The in-sample R2 is 0.4359409%
The in-sample MSE is 0.0114678
***************Out-of-Sample Metrics***************
The out-of-sample R2 is -1.1550289%
The out-of-sample MSE is 0.0086937
```

Naive linear model, estimated with huber loss function

[ ]:
```
from sklearn.linear_model import HuberRegressor

epsilon = np.max(((y_trn-OLS.predict(X_trn)).quantile(.999),1))
OLS_H = HuberRegressor(epsilon=epsilon).fit(X_trn,y_trn)
evaluate(y_trn, OLS_H.predict(X_trn), insample=True)
evaluate(y_tst, OLS_H.predict(X_tst),insample= False)
```

```
***************In-Sample Metrics***************
The in-sample R2 is 13.7138623%
The in-sample MSE is 0.0099385
***************Out-of-Sample Metrics***************
The out-of-sample R2 is -36.9084712%
The out-of-sample MSE is 0.0117665

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_huber.py:342:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

linear model with selected features estimated using huber loss

[ ]:
```
from sklearn.linear_model import HuberRegressor

# OLS by Huber robust objective function
# with preselected size, bm, and momentum covariates
epsilon = np.max(((y_trn-OLS_3.predict(X_trn[features_3])).quantile(.999),1))
features_3 = ['mvel1','bm','mom1m','mom6m','mom12m','mom36m']
OLS_H_3 = HuberRegressor(epsilon=epsilon).fit(X_trn[features_3],y_trn)

evaluate(y_trn, OLS_H_3.predict(X_trn[features_3]), insample=True)
evaluate(y_tst, OLS_H_3.predict(X_tst[features_3]),insample=False)
```

```
***************In-Sample Metrics***************
The in-sample R2 is 0.2131522%
The in-sample MSE is 0.0114935
***************Out-of-Sample Metrics***************
The out-of-sample R2 is -0.4378110%
The out-of-sample MSE is 0.0086321
```

[ ]:
```
from sklearn.linear_model import LinearRegression

features_3 = ['mvel1','bm','mom12m']
OLS_3 = LinearRegression().fit(X_trn[features_3],y_trn)

evaluate(y_tst, OLS_3.predict(X_tst[features_3]),insample=False)
```

```
***************Out-of-Sample Metrics***************
The out-of-sample R2 is -0.3647499%
The out-of-sample MSE is 0.0086258
```

[ ]:
```
features_7 = ['mvel1','bm','mom12m','acc','roaq','agr','egr']
OLS_7 = LinearRegression().fit(X_trn[features_7],y_trn)

evaluate(y_tst, OLS_7.predict(X_tst[features_7]),insample=False)
```

```
***************Out-of-Sample Metrics***************
The out-of-sample R2 is -1.4084132%
The out-of-sample MSE is 0.0087155
```

[ ]:
```
features_15 =␣
 ↪['mvel1','bm','mom12m','acc','roaq','agr','egr','dy','beta','retvol','turn','lev','sp','mom36
OLS_15 = LinearRegression().fit(X_trn[features_15],y_trn)
```

```
evaluate(y_tst, OLS_15.predict(X_tst[features_15]),insample=False)
```

```
**********Out-of-Sample Metrics**********
The out-of-sample R2 is -3.7306822%
The out-of-sample MSE is 0.0089151
```

[ ]:
```
def R_oos_other(actual, predicted):
    actual, predicted = np.array(actual), np.array(predicted).flatten()
    predicted = np.where(predicted<0,0,predicted)
    return 1 - (np.dot((actual-predicted),(actual-predicted)))/(np.
 ↪dot(actual,actual))
```

[ ]:
```
def eval_other(actual, predicted, insample):
    if insample == True:
        print('*'*15+'In-Sample Metrics'+'*'*15)
        print(f'The in-sample R2 is {r2_score(actual,predicted)*100:.2f}%')
        print(f'The in-sample MSE is {mean_squared_error(actual,predicted):.3f}')
    else:
        print('*'*15+'Out-of-Sample Metrics'+'*'*15)
        print(f'The out-of-sample R2 is {R_oos_other(actual,predicted)*100:.
 ↪2f}%')
        print(f'The out-of-sample MSE is {mean_squared_error(actual,predicted):.
 ↪7f}')
```

[ ]:
```
from sklearn.linear_model import LinearRegression
features_3 = ['mvel1','bm','mom12m']
OLS_3 = LinearRegression().fit(X_trn[features_3],y_trn)

eval_other(y_tst, OLS_3.predict(X_tst[features_3]),insample=False)
```

```
**********Out-of-Sample Metrics**********
The out-of-sample R2 is 0.68%
The out-of-sample MSE is 0.0086258
```

[ ]:
```
features_7 = ['mvel1','bm','mom12m','acc','roaq','agr','egr']
OLS_7 = LinearRegression().fit(X_trn[features_7],y_trn)

eval_other(y_tst, OLS_7.predict(X_tst[features_7]),insample=False)
```

```
**********Out-of-Sample Metrics**********
The out-of-sample R2 is 0.41%
The out-of-sample MSE is 0.0087155
```

[ ]:
```
features_15 =␣
 ↪['mvel1','bm','mom12m','acc','roaq','agr','egr','dy','beta','retvol','turn','lev','sp','mom36
OLS_15 = LinearRegression().fit(X_trn[features_15],y_trn)

eval_other(y_tst, OLS_15.predict(X_tst[features_15]),insample=False)
```

```
**********Out-of-Sample Metrics**********
The out-of-sample R2 is -0.52%
The out-of-sample MSE is 0.0089151
```

This is another Notebook. There are many output cells that I have to delete as they take too much screen space.Retreive the full notebook on github.com/charbelkhazen

[ ]:
```
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import ParameterGrid
import time
from sklearn.linear_model import LinearRegression, HuberRegressor
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

Ideally I would like to use the R-squared function used previously (i.e. the commented function below). However, this function is computationally expensive, as it can consume more than 50gb RAM if used in PCR.

[ ]:
```
def R_oos(actual, predicted):
    actual, predicted = np.array(actual), np.array(predicted).flatten()
    predicted = np.where(predicted<0,0,predicted)
    return 1 - (np.dot((actual-predicted),(actual-predicted)))/(np.
 ↪dot(actual,actual))

def val_fun(model, params: dict, X_trn, y_trn, X_vld, y_vld, illustration=True,␣
 ↪sleep=0):
    best_mse_oos = None
    lst_params = list(ParameterGrid(params))
    for param in lst_params:
        if best_mse_oos == None:
            mod = model().set_params(**param).fit(X_trn, y_trn)
            best_mod = mod
            y_pred = mod.predict(X_vld)
            best_ros = R_oos(y_vld, y_pred)
            best_mse_oos = mean_squared_error(y_vld,y_pred)
            best_param = param
            if illustration:
                print(f'Model with params: {param} finished.')
                print(f'with out-of-sample MSE on validation set: {best_mse_oos:.
 ↪5f}')
                print(f'with out-of-sample R-squared on validation set:␣
 ↪{best_ros*100:.7f}%')
                print('*'*60)
        else:
            time.sleep(sleep)
            mod = model().set_params(**param).fit(X_trn, y_trn)
            y_pred = mod.predict(X_vld)
            ros = R_oos(y_vld, y_pred)
            mse_oos = mean_squared_error(y_vld,y_pred)
            if illustration:
                print(f'Model with params: {param} finished.')
```

```
                print(f'with out-of-sample MSE on validation set: {mse_oos:.5f}')
                print(f'with out-of-sample R-squared on validation set: {ros*100:
↪.7f}%')
                print('*'*60)
            if mse_oos < best_mse_oos:
                best_mse_oos = mse_oos
                best_mod = mod
                best_param = param
    if illustration:
        print('\n'+'#'*60)
        print('Tuning process finished!!!')
        print(f'The best setting is: {best_param}')
        print(f'with MSE OOS {best_mse_oos:.5f} on validation set.')
        print('#'*60)
    return best_mod
```

[ ]:
```
# Evaluation Output
def evaluate(actual, predicted, insample=False):
    if insample:
        print('*'*15+'In-Sample Metrics'+'*'*15)
        print(f'The in-sample R2 is {r2_score(actual,predicted)*100:.2f}%')
        print(f'The in-sample MSE is {mean_squared_error(actual,predicted):.3f}')
    else:
        print('*'*15+'Out-of-Sample Metrics'+'*'*15)
        print(f'The out-of-sample R2 is {R_oos(actual,predicted)*100:.5f}%')
        print(f'The out-of-sample MSE is {mean_squared_error(actual,predicted):.
↪8f}')
```

[ ]:
```
class PCRegressor:

    def __init__(self,n_PCs=1,loss='mse'):
        self.n_PCs = n_PCs
        if loss not in ['huber','mse']:
            raise AttributeError(
            f"The loss should be either 'huber' or 'mse', but {loss} is given"
            )
        else:
            self.loss = loss

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self

    def fit(self,X,y):
        X = np.array(X)
        N,K = X.shape
```

```
        y = np.array(y_trn).reshape((N,1))
        self.mu = np.mean(X,axis=0).reshape((1,K))
        self.sigma = np.std(X,axis=0).reshape((1,K))
        self.sigma = np.where(self.sigma==0,1,self.sigma)
        X = (X-self.mu)/self.sigma  #standardize
        pca = PCA()   #call the sklearn class
        X = pca.fit_transform(X)[:,:self.n_PCs]  #Projected x ( Q-dimensional)
        self.pc_coef = pca.components_.T[:,:self.n_PCs]   #P by Q factor loading
        if self.loss == 'mse':
            self.model = LinearRegression().fit(X,y)    #fit the low dimensional␣
↪data
        else:
            self.model = HuberRegressor().fit(X,y)
        return self

    def predict(self,X):
        X = np.array(X)
        X = (X-self.mu)/self.sigma
        X = X @ self.pc_coef
        return self.model.predict(X)
```

[ ]:
```
params = {'n_PCs':[1,3,5,7,10,50],'loss':['mse','huber']}
PCR =␣
↪val_fun(PCRegressor,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
evaluate(y_trn, PCR.predict(X_trn), insample=True)
evaluate(y_tst, PCR.predict(X_tst))
```

[ ]:
```
params = {'n_PCs':[100,300,500],'loss':['mse','huber']}
PCR =␣
↪val_fun(PCRegressor,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
```

[ ]:
```
evaluate(y_tst, PCR.predict(X_tst))
```

```
***************Out-of-Sample Metrics***************
The out-of-sample R2 is 0.14980%
The out-of-sample MSE is 0.01285622
```

[ ]:
```
params = {'n_PCs':[50],'loss':['mse']}
PCR =␣
↪val_fun(PCRegressor,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
evaluate(y_tst, PCR.predict(X_tst))
params = {'n_PCs':[50],'loss':['huber']}
PCR =␣
↪val_fun(PCRegressor,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
evaluate(y_tst, PCR.predict(X_tst))
```

OVERALL THE BEST TUNING PARAMETER IS NPC = 50

INDEPENDENT COMPONENT ANALYSIS I USE FAST ICA TO REDUCE THE FEA-

TURE SPACE UNFORTUNATELY; THE RELEVANCE CRITERION OF PC SELECTION IS NOT EXPLICIT IN FAST ICA (It might be random), I use the POWER DATA METHOD

Independent Component Analysis

[ ]:
```
from sklearn.decomposition import FastICA


class MyNewICA:

    def __init__(self, n_PCs=1, loss='mse'):
        self.n_PCs = n_PCs
        if loss not in ['huber', 'mse']:
            raise AttributeError(
                f"The loss should be either 'huber' or 'mse', but {loss} is given"
            )
        else:
            self.loss = loss
        self.model = None

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self

    def fit(self, X, y):
        X = np.array(X)
        N, K = X.shape
        y = np.array(y).reshape((N, 1))
        self.mu = np.mean(X, axis=0).reshape((1, K))
        self.sigma = np.std(X, axis=0).reshape((1, K))
        self.sigma = np.where(self.sigma == 0, 1, self.sigma)
        X = (X - self.mu) / self.sigma
        ica = FastICA(whiten='unit-variance',random_state=42) #Fix some random number So that the directions remain the same ( otherwise you will get the same span but different directions)
        s_ica = ica.fit(X).transform(X)
        ica_coef = ica.mixing_
        self.original_coef = ica_coef
        self.original_s = s_ica
        # POWER DATA FOR SOURCE ORDERING
        sources_trans = s_ica.T
        squared_source_matrix = np.square(sources_trans)
        squared_A = np.square(ica_coef)
        sum_squared_elements = np.sum(squared_A, axis=0)
        result_matrix = sum_squared_elements.reshape(1, -1)
        qd_matrix =   result_matrix.T * squared_source_matrix
```

```
        mean_per_row = np.mean(qd_matrix, axis=1, keepdims=True)
        selected_indices = np.argsort(mean_per_row.flatten())[::-1][:self.n_PCs]
        #GOT A LIST OF SELECTED ORDERS
        self.selected_indices = selected_indices
        self.new_ica_coef = ica_coef[:, selected_indices]
        self.selected_sources = s_ica[:,selected_indices]
        if self.loss == 'mse':
            self.model = LinearRegression().fit( self.selected_sources , y)
        else:
            self.model = HuberRegressor(alpha=0.025).fit( self.selected_sources,␣
↪y)

        return self

    def predict(self, X):
        X = np.array(X)
        X = (X - self.mu) / self.sigma
        projections = np.dot(X, self.new_ica_coef)
        self.projections = projections
        self.predicted_y = np.dot(projections, self.model.coef_.T) + self.model.
↪intercept_
        return self.predicted_y
```

[ ]:
```
params = {'n_PCs':[10,50,100,200],'loss':['mse','huber']}
MyNewICA1 = val_fun(MyNewICA, params=params, X_trn=X_trn, y_trn=y_trn,␣
 ↪X_vld=X_vld, y_vld=y_vld, sleep=3)
evaluate(y_tst, MyNewICA1.predict(X_tst))
```

NOW; I estimate l1, l2, and EN (Each using MSE and Huber as the loss function) using **Accelerated Proximal Gradient Descent**

I use APGDpy libray for Accelerated Proximal Gradient descent method as coding an accelerated proximal gradient descent method was resulting in degenerate results. The library Provides an apg_solve() function.

- Solving APGD requires defining a gradient function
- and a Proximal operator corresponding to the penalization used
- I define a gradient function for Huber and for the quadratic loss
- and a proximal operator for l1 l2 and EN penalties (Their corresponding mathematical formulae are found in appendix (Numerical Optimization section)

I fix the huber treshold for computational purposes

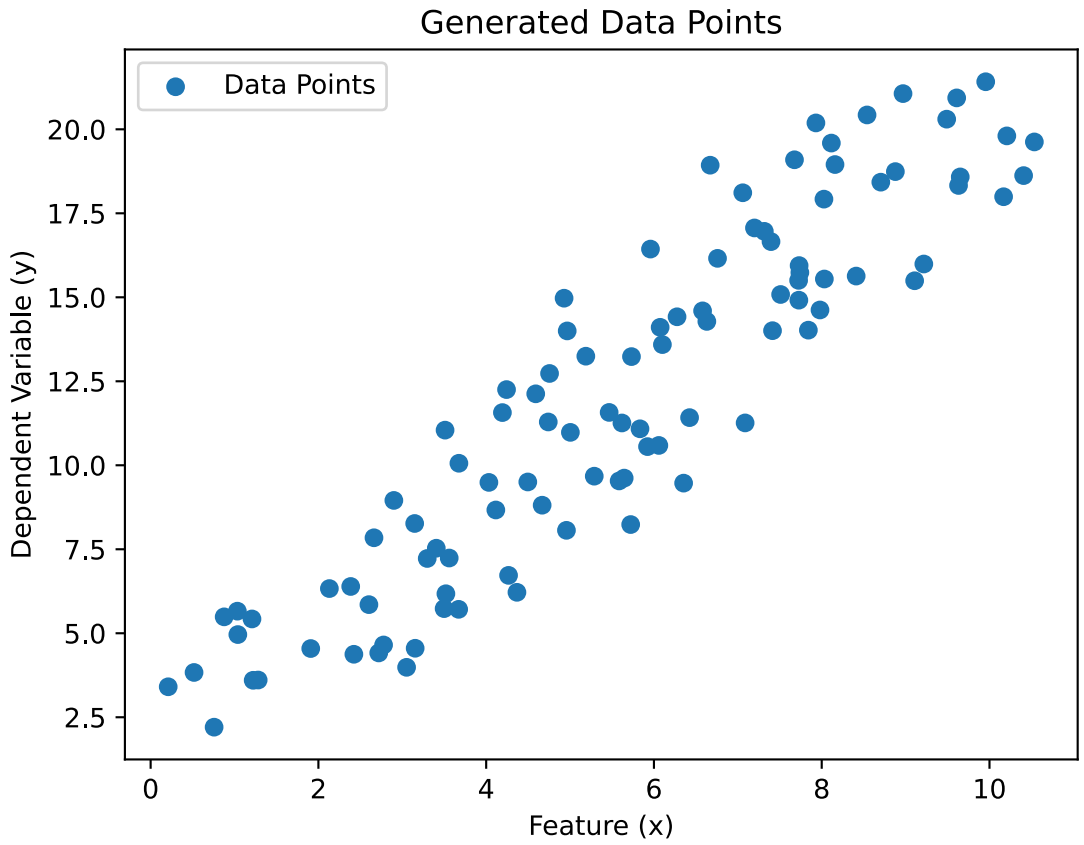
This is a benchmark example of a Static Proximal Gradient descent algorithm on Lasso using Squared loss . “Static” because the Gradient descent and regularization tuning parameters are fixed - see mu and t . Note: The value of the Gradient descent parameter (step parameter) “t” is not explicitely defined. The function solve.apgd() does the tuning automatically.

Testing Accelerated Proximal Gradient descent on simulated data exhibiting linear relationship see plot (3 boxes futher)

[ ]:
```
import apgpy
```

[ ]:
```
import apgpy as apg
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

n = 2
m = 100
A = np.column_stack((np.ones(m), np.linspace(1, 10, m) + np.random.randn(m)))
b = 2 * np.linspace(1, 10, m) + 1 + 0.5 * np.random.randn(m)
```

[ ]:
```
import matplotlib.pyplot as plt

# Plotting the points formed by A and b
plt.scatter(A[:, 1], b, label='Data Points')
plt.xlabel('Feature (x)')
plt.ylabel('Dependent Variable (y)')
plt.title('Generated Data Points')
plt.legend()
plt.show()
```



[ ]:
```
coef1_values = np.array([])  # Initialize array for coefficient 1
coef2_values = np.array([])  # Initialize array for coefficient 2

def quad_grad(y):
  return np.dot(A.T, (np.dot(A, y) - b))

def soft_thresh(y, t):
  return np.sign(y) * np.maximum(abs(y) - t * mu, 0)

mu_values = np.logspace(2, 0, num=100)
for mu in mu_values:
  x = apg.solve(quad_grad, soft_thresh, np.zeros(n), use_restart=True,␣
↪eps=1e-12, quiet=True)
  coef1_values = np.append(coef1_values, x[0])
  coef2_values = np.append(coef2_values, x[1])
  print(f"For mu = {mu}, the coefficients are {x}")
```

SEE HOW COEFFICIENT GETS SPARSE WHEN WE INCREASE LAMBDA.

- I apply this Accelerated Proximal Gradient Descent Algorithm on my data
- This takes 30 minutes

[ ]:
```
np_ess_X = np.array(X_trn)
```

[ ]:
```
#num_rows_to_display = 3
#subset_array = np_ess_X[:num_rows_to_display, :]
#print(subset_array)
np_ess_y = np.array(y_trn)
```

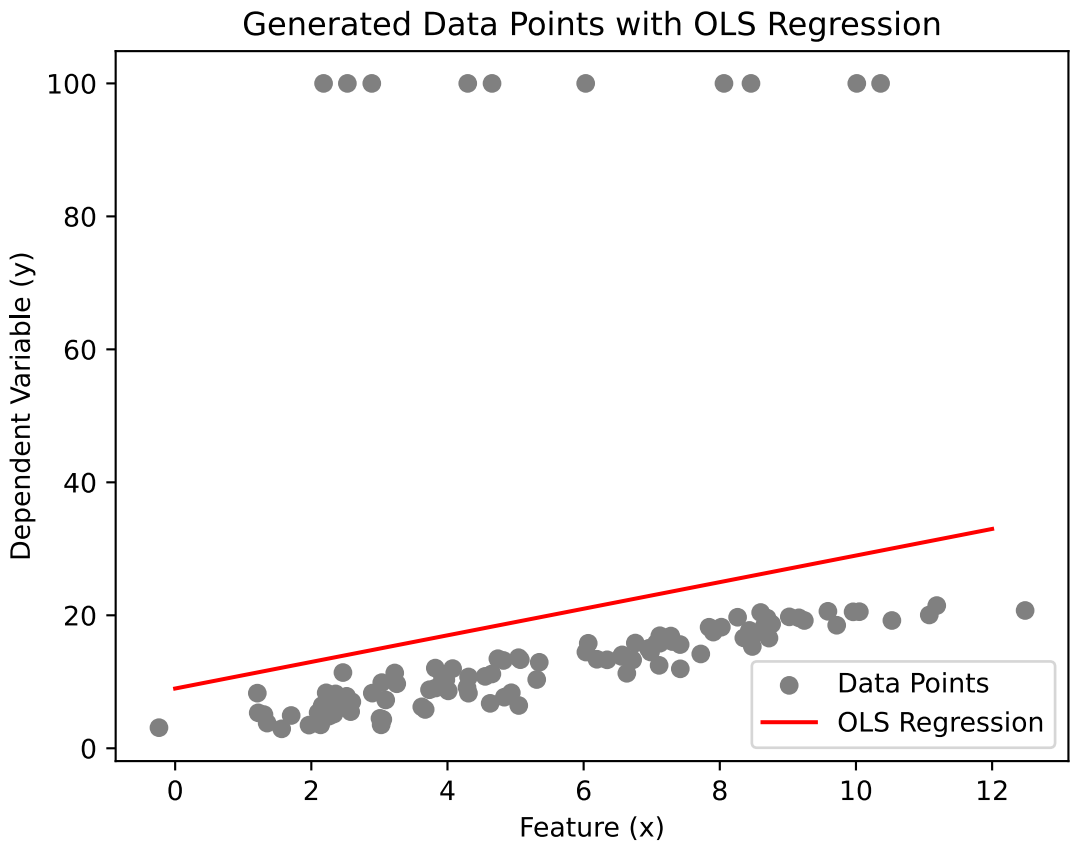
[ ]:
```
n = np_ess_X.shape[1]
```

[ ]:
```
for mu_value in range(20, 0, -1):
    mu = mu_value

    def quad_grad(y):
        return np.dot(np_ess_X.T, (np.dot(np_ess_X, y) - np_ess_y))

    def soft_thresh(y, t):
        return np.sign(y) * np.maximum(abs(y) - t * mu, 0)

    x = apg.solve(quad_grad, soft_thresh, np.zeros(n), use_restart=True,␣
↪eps=1e-12, quiet=True)


    print(f"For mu = {mu}, the coefficients are {x}")
```

Static accelerated Proximal Gradient descent algorithm for huber loss (With lp normed pernalties).

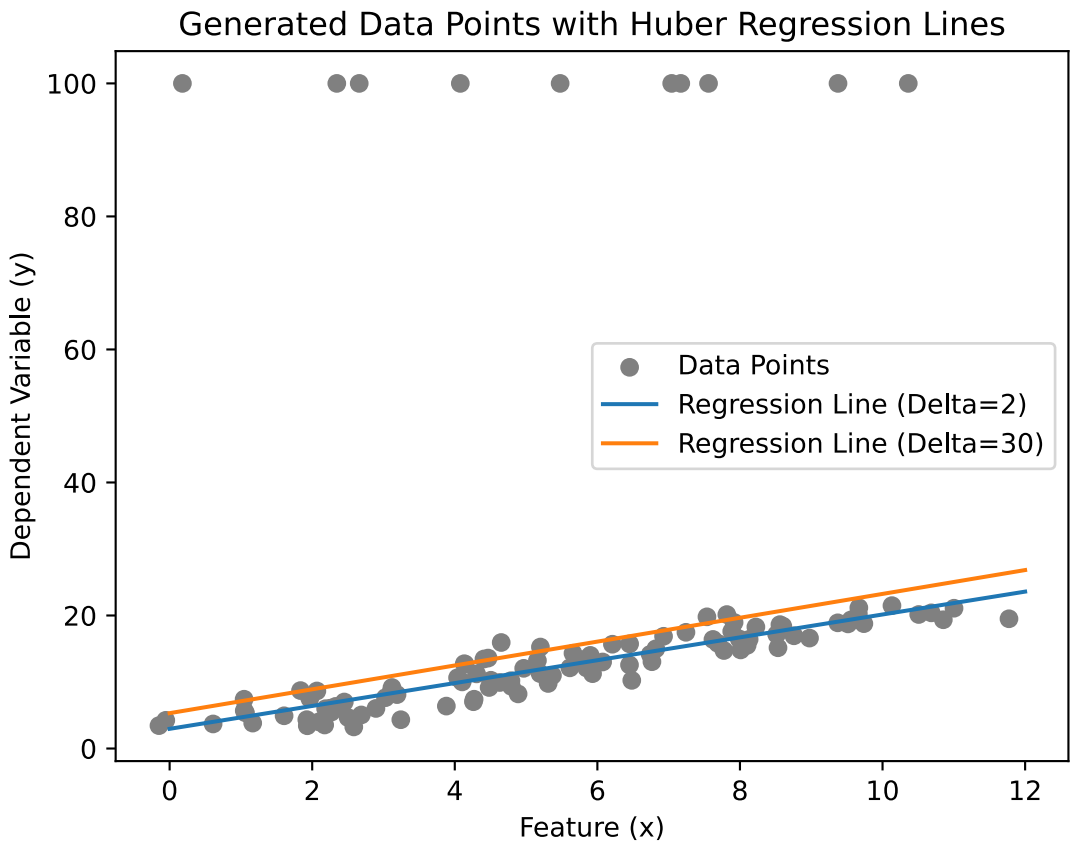
I simulate a LM model with outliers and then 1. Fit by OLS 2. Fit Huber loss (using APGD)

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

n = 2
m = 100

# Generating data
A = np.column_stack((np.ones(m), np.linspace(1, 10, m) + np.random.randn(m)))
b = 2 * np.linspace(1, 10, m) + 1 + 0.5 * np.random.randn(m)

# Introducing outliers
outliers_A = np.column_stack((np.ones(10), np.linspace(1, 10, 10) + np.random.
 ↪randn(10)))
outliers_b = 100 * np.ones(10)
A = np.vstack([A, outliers_A])
b = np.concatenate([b, outliers_b])

# Computing OLS regression
ols_model = LinearRegression().fit(A, b)
ols_intercept, ols_slope = ols_model.intercept_, ols_model.coef_[1]

# Plotting the points formed by A and b
plt.scatter(A[:, 1], b, label='Data Points', color="grey")

# Plotting OLS regression line
x_vals = np.linspace(0, 12, 100)
y_ols = ols_intercept + ols_slope * x_vals
plt.plot(x_vals, y_ols, label='OLS Regression', color='red')

plt.xlabel('Feature (x)')
plt.ylabel('Dependent Variable (y)')
plt.title('Generated Data Points with OLS Regression')
plt.legend()
plt.show()
```



[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import apgpy as apg
n = 2
m = 100

A = np.column_stack((np.ones(m), np.linspace(1, 10, m) + np.random.randn(m)))
b = 2 * np.linspace(1, 10, m) + 1 + 0.5 * np.random.randn(m)


outliers_A = np.column_stack((np.ones(10), np.linspace(1, 10, 10) + np.random.
 ↪randn(10)))
outliers_b = 100 * np.ones(10)
A = np.vstack([A, outliers_A])
b = np.concatenate([b, outliers_b])


plt.scatter(A[:, 1], b, label='Data Points', color="grey")
```

```
coef1_values = np.array([])
coef2_values = np.array([])
delta_values = [2, 30]

for delta in delta_values:
    mu = 0  #i.e. MSE ; NO Lnormed Penalizarion

    def huber_loss_gradient(y):
        residual = np.dot(A, y) - b
        huber_grad = np.dot(A.T, np.where(np.abs(residual) <= delta, residual,␣
↪delta * np.sign(residual)))
        return huber_grad

    def soft_thresh(y, t):
        return np.sign(y) * np.maximum(abs(y) - t * mu, 0)

    x = apg.solve(huber_loss_gradient, soft_thresh, np.zeros(n),␣
↪use_restart=True, eps=1e-12, quiet=True)
    coef1_values = np.append(coef1_values, x[0])
    coef2_values = np.append(coef2_values, x[1])


    y_reg = x[0] + x[1] * np.linspace(0, 12, 100)
    plt.plot(np.linspace(0, 12, 100), y_reg, label=f'Regression Line␣
↪(Delta={delta})')

plt.xlabel('Feature (x)')
plt.ylabel('Dependent Variable (y)')
plt.title('Generated Data Points with Huber Regression Lines')
plt.legend()
plt.show()
```

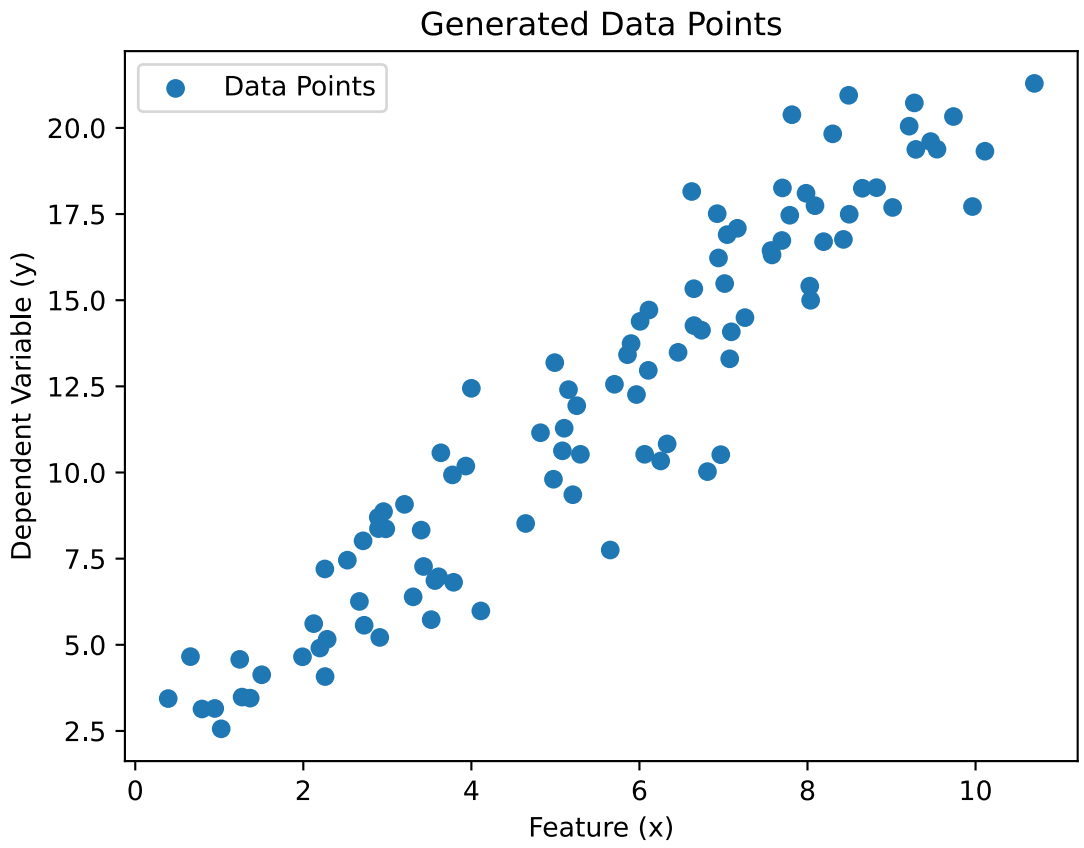


We see how huber penalty mitigates the effect of outliers. And we see how the huber loss minimization can be done by using the accelerated proximal gradient descent method. (By changing the mu values , you can test for lasso + huber)

Static Accelerated Proximal Gradient descent for Elastic Net penalization.

[ ]:
```
n = 2
m = 100
A = np.column_stack((np.ones(m), np.linspace(1, 10, m) + np.random.randn(m)))
b = 2 * np.linspace(1, 10, m) + 1 + 0.5 * np.random.randn(m)
```

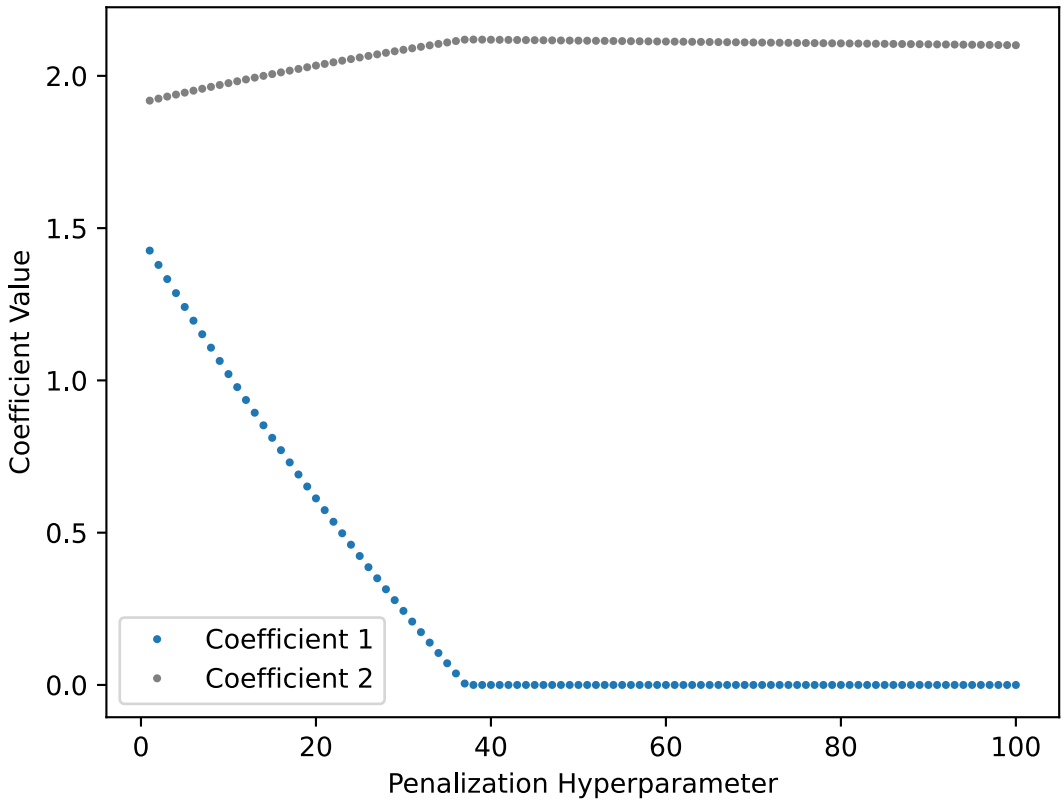
[ ]:
```
# Plotting the points formed by A and b
plt.scatter(A[:, 1], b, label='Data Points')
plt.xlabel('Feature (x)')
plt.ylabel('Dependent Variable (y)')
plt.title('Generated Data Points')
plt.legend()
plt.show()
```



[ ]:
```
coef1_values_en = np.array([])  # Initialize array for coefficient 1
coef2_values_en = np.array([])  # Initialize array for coefficient 2

rho = 0.1
for mu_value in range(100, 0, -1):
    mu = mu_value

    def quad_grad(y):
        return np.dot(A.T, (np.dot(A, y) - b))

    def elastic_net_proximal_operator(y, t):
      soft_thresh_term = np.sign(y) * np.maximum(np.abs(y) - t * mu * (1 - rho),␣
↪0)
      return soft_thresh_term / (1 + t * rho * mu)


    x_en = apg.solve(quad_grad, elastic_net_proximal_operator, np.zeros(n),␣
↪use_restart=True, eps=1e-12, quiet=True)
    coef1_values_en = np.append(coef1_values_en, x_en[0])
```

```
    coef2_values_en = np.append(coef2_values_en, x_en[1])

    print(f"For mu = {mu}, the coefficients are {x_en}")
```

[ ]:
```
mu_values = np.arange(100, 0, -1)
plt.plot(mu_values, coef1_values_en , 'o', label='Coefficient 1',markersize = 2 )
plt.plot(mu_values, coef2_values_en , 'o', label='Coefficient 2', markersize =␣
↪2, color = "grey")
plt.xlabel('Penalization Hyperparameter')
plt.ylabel('Coefficient Value')
plt.legend()
plt.show()
```



Elastic Net Induces sparsity A low Rho and large lambda (here called mu) will encourage sparsity as they increase penalty on L1 **See EN penalty formula**

I apply Accelerated Proximal Gradient Descent to find:

1. Lasso with squared loss
2. Lasso with Huber loss
3. Elastic Net with squared loss
4. Elastic Net with huber loss

[ ]:
```
class LassoMse:

    def __init__(self, mu=1):
      self.mu = mu

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self


    def fit(self, X, y):
      X = np.array(X)
      n = X.shape[1]
      y = np.array(y)
      def quad_grad(beta):
        return np.dot(X.T, (np.dot(X, beta) - y))

      def soft_thresh(beta, t):
        return np.sign(beta) * np.maximum(abs(beta) - t * self.mu, 0)

      coef_lasso_mse = apg.solve(quad_grad, soft_thresh, np.zeros(n),␣
↪use_restart=True, eps=1e-12, quiet=True)
      self.coef = coef_lasso_mse
      return self


    def predict(self, X):
        return np.dot(X, self.coef)
```

I define a set of 10 log scaled range from e^-3 to 20 lambda values. I cannot rely on a bigger set. This set alone takes approx 20 min without validation.

[ ]:
```
params = {'mu': np.logspace(np.log10(0.001), np.log10(20), 10).tolist()}
lasso_mse =␣
↪val_fun(LassoMse,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
```

This suggests that a higher penalization tuning rate might be better I repeat with another higher range

[ ]:
```
params = {'mu': np.logspace(np.log10(20), np.log10(900),10).tolist()}
lasso_mse =␣
↪val_fun(LassoMse,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
```

I evaluate

[ ]:
```
params = {'mu': [386.2441821090113]}
LassoMse =␣
↪val_fun(LassoMse,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
```

```
evaluate(y_tst, LassoMse.predict(X_tst))
```

[ ]:
```
class LassoHuber:

    def __init__(self, mu=1, delta = 1):
      self.mu = mu
      self.delta = delta

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self


    def fit(self, X, y):
      X = np.array(X)
      n = X.shape[1]
      y = np.array(y)

      def huber_loss_gradient(beta):
        residual = np.dot(X, beta) - y
        huber_grad = np.dot(X.T, np.where(np.abs(residual) <= self.delta,␣
↪residual, self.delta * np.sign(residual)))
        return huber_grad

      def soft_thresh(beta, t):
        return np.sign(beta) * np.maximum(abs(beta) - t * self.mu, 0)


      coef_lasso_huber = apg.solve(huber_loss_gradient, soft_thresh, np.
↪zeros(n), use_restart=True, eps=1e-12, quiet=True)
      self.coef = coef_lasso_huber
      return self


    def predict(self, X):
        return np.dot(X, self.coef)
```

[ ]:
```
params = {'mu': np.linspace(250,390,10).tolist(),'delta':[0.025]}
lassoHuber =␣
↪val_fun(LassoHuber,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
evaluate(y_tst, lassoHuber.predict(X_tst))
```

Now; **Elastic Net using MSE loss function**

[ ]:
```
class ElasticNetMse:
```

```
    def __init__(self, mu=1, rho = 0.1):    #birmania
      self.mu = mu
      self.rho = rho

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self


    def fit(self, X, y):
      X = np.array(X)
      n = X.shape[1]
      y = np.array(y)

      def quad_grad(beta):
        return np.dot(X.T, (np.dot(X, beta) - y))

      def elastic_net_proximal_operator(beta, t):
        soft_thresh_term = np.sign(beta) * np.maximum(np.abs(beta) - t * self.mu␣
↪* (1 - self.rho), 0)
        return soft_thresh_term / (1 + t * self.rho * self.mu)

      coef_en_mse = apg.solve(quad_grad, elastic_net_proximal_operator, np.
↪zeros(n), use_restart=True, eps=1e-12, quiet=True)
      self.coef = coef_en_mse

      return self



    def predict(self, X):
        return np.dot(X, self.coef)
```

Low rho and Large mu will encourage sparisty ; I thus chose the following sets for mu and rho

[ ]:
```
params = {'mu': np.logspace(np.log10(50), np.log10(900), 7).tolist(),'rho':[0.1]}
```

[ ]:
```
ElasticNetMse =␣
↪val_fun(ElasticNetMse,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
evaluate(y_tst, ElasticNetMse.predict(X_tst))
```

[ ]:
```
class ElasticNetHuber:

    def __init__(self, mu=1, rho = 0.1,delta =0.02 ):    #birmania
      self.mu = mu
      self.delta = delta
```

```
        self.rho = rho

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self


    def fit(self, X, y):
      X = np.array(X)
      n = X.shape[1]
      y = np.array(y)

      def huber_loss_gradient(beta):
        residual = np.dot(X, beta) - y
        huber_grad = np.dot(X.T, np.where(np.abs(residual) <= self.delta,␣
↪residual, self.delta * np.sign(residual)))
        return huber_grad

      def elastic_net_proximal_operator(beta, t):
        soft_thresh_term = np.sign(beta) * np.maximum(np.abs(beta) - t * self.mu␣
↪* (1 - self.rho), 0)
        return soft_thresh_term / (1 + t * self.rho * self.mu)

      coef_en_huber = apg.solve(huber_loss_gradient,␣
↪elastic_net_proximal_operator, np.zeros(n), use_restart=True, eps=1e-12,␣
↪quiet=True)
      self.coef = coef_en_huber

      return self



    def predict(self, X):
        return np.dot(X, self.coef)
```

[ ]:
```
params = {'mu': np.logspace(np.log10(20), np.log10(500), 10).tolist(),'delta':np.
↪array([0.02, 0.03, 0.05, 0.1]),'rho':[0.1]}
ElasticNetHuber=␣
↪val_fun(ElasticNetHuber,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3
↪  # #need to run
```

[ ]:
```
params = {'mu': np.linspace(60, 800, 5).tolist(),'delta':[0.025],'rho':[0.1]}
ElasticNetHuber=␣
↪val_fun(ElasticNetHuber,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3
evaluate(y_tst, ElasticNetHuber.predict(X_tst))
```

TREES

- I show how trees are equivalent to partitionning the feature space and to fitting a piecewise constant function
- This is done for illustrative purposes only

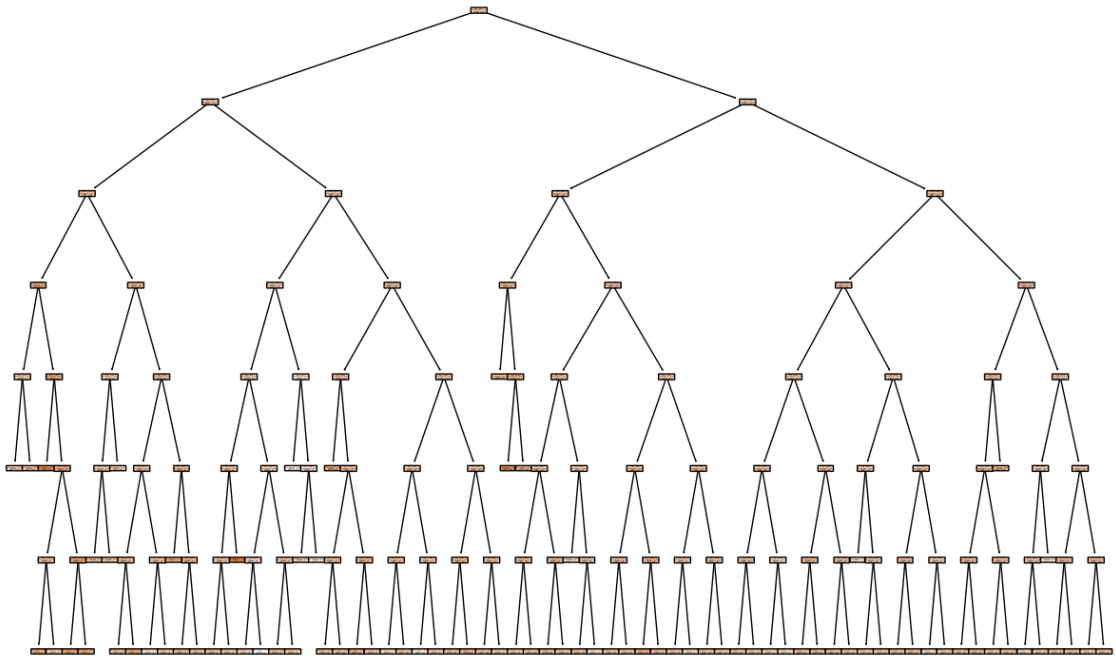
[ ]:
```
X_train_macro = X_trn[['ep']]
X_test_macro = X_vld[['ep']]
```

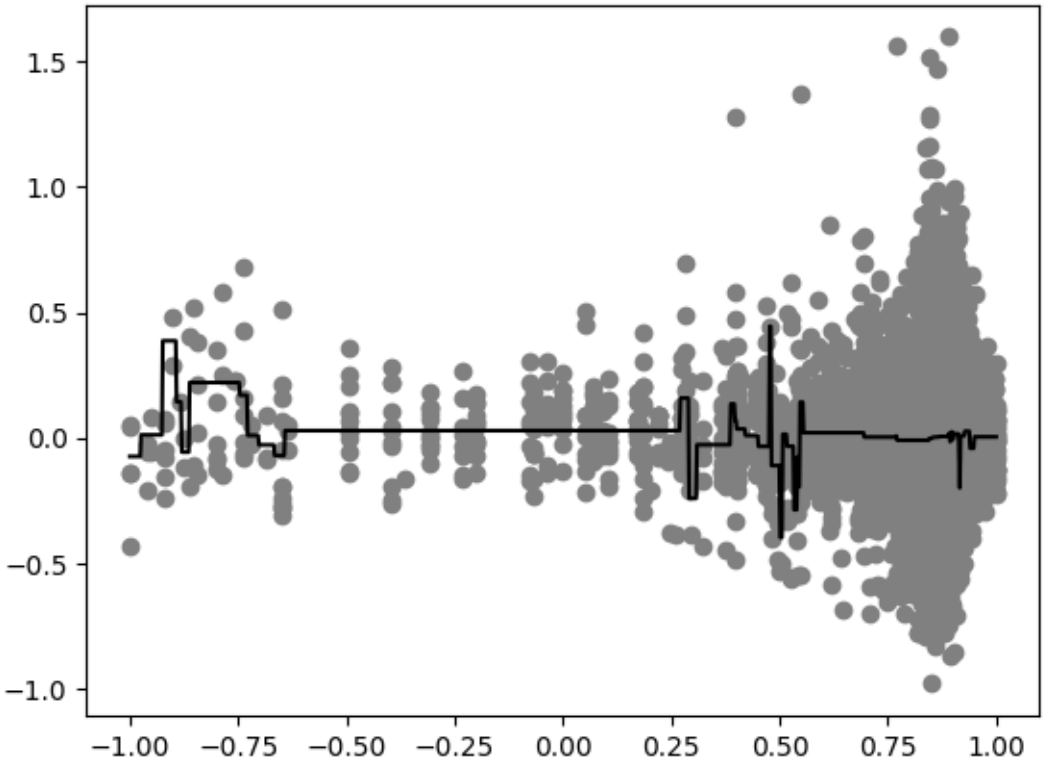
[ ]:
```
combined_df = pd.concat([X_train_macro, X_test_macro], ignore_index=True)
combined_y =  pd.concat([y_trn, y_vld], ignore_index=True)
```

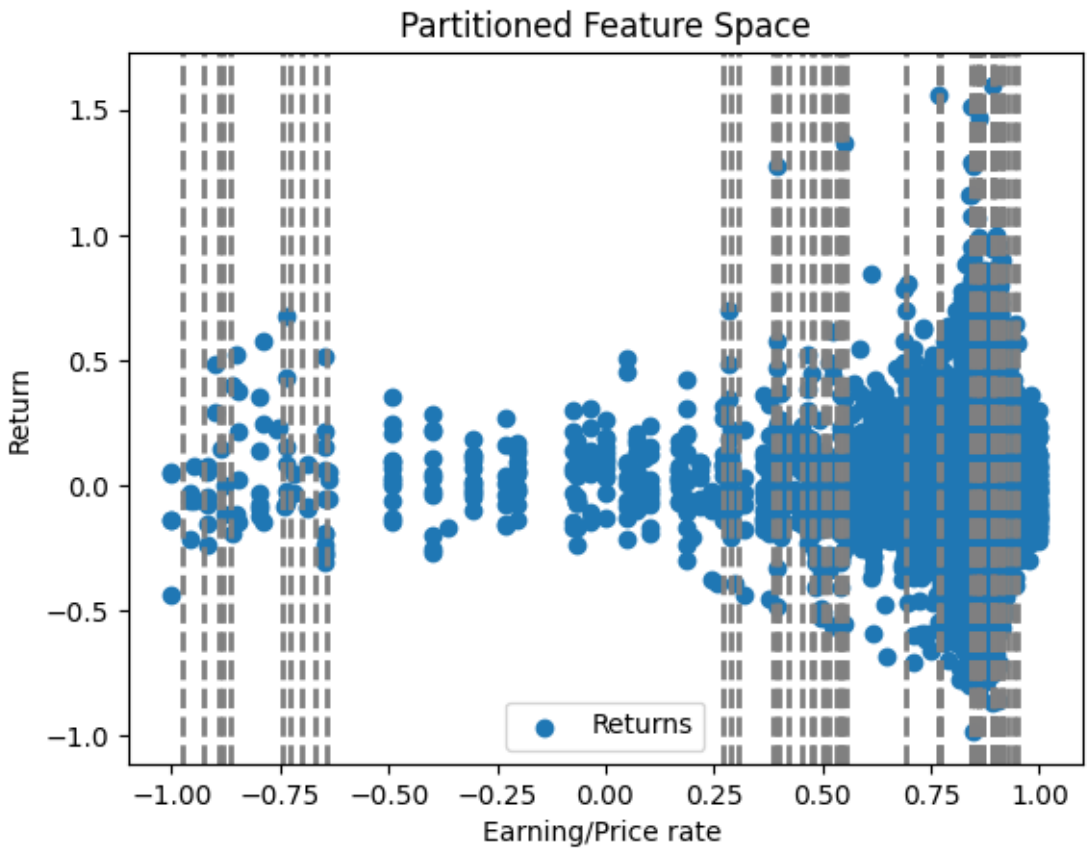
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import train_test_split


reg_tree = DecisionTreeRegressor( max_depth=7)
reg_tree.fit(X_train_macro, y_trn)

plt.figure(figsize=(15, 10))

# Plot the data points
plt.scatter(combined_df, combined_y, label='Returns')



plot_tree(reg_tree, filled=True, rounded=True, feature_names=X_train_macro.
 ↪columns)

plt.savefig('regression_tree_plot.svg', format='svg', bbox_inches='tight')

# Plot the piecewise constant function represented by the regression tree
plt.figure()
X_range = np.linspace(combined_df.min(), combined_df.max(), 1000).reshape(-1, 1)
y_pred = reg_tree.predict(X_range)

plt.scatter(combined_df, combined_y, label='Data Points', color='grey')
plt.plot(X_range, y_pred, color='black', label='Piecewise Constant Function')

# Save the plot as an SVG file
plt.savefig('piecewise_constant_function_plot.svg', format='svg',␣
 ↪bbox_inches='tight')

# Plot the partitioned feature space
plt.figure()
plt.scatter(combined_df, combined_y, label='Returns')
```

```
# Plot the partitioned feature space
for split_value in reg_tree.tree_.threshold[reg_tree.tree_.threshold != -2]:
    plt.axvline(x=split_value, color='gray', linestyle='--', linewidth=2)

plt.title('Partitioned Feature Space')
plt.xlabel('Earning/Price rate ')
plt.ylabel('Return')
plt.legend()

# Save the plot as an SVG file
plt.savefig('partitioned_feature_space_plot.svg', format='svg',␣
 ↪bbox_inches='tight')

# Show the plots
plt.show()

# Evaluate the model
mse = np.mean((reg_tree.predict(X_test_macro) - y_vld) ** 2)
print(f"Mean Squared Error on test data: {mse}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but DecisionTreeRegressor was fitted with feature
names
  warnings.warn(
```




1.5
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00




```
Mean Squared Error on test data: 0.008963609722164017
```

Now let's build a Regression **Tree with cost complexity pruning regularization**

Regression tree with pruning

[ ]:
```
from sklearn.datasets import make_regression
from sklearn.tree import DecisionTreeRegressor, export_text, plot_tree
import matplotlib.pyplot as plt
```

[ ]:
```
class RegressionTree:
    def __init__(self, ccp_alpha=1e-06):
        self.ccp_alpha = ccp_alpha
        self.reg_tree = DecisionTreeRegressor(ccp_alpha=self.ccp_alpha)

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self

    def fit(self, X, y):
```

```
        X = np.array(X)
        y = np.array(y)
        self.reg_tree.fit(X, y)
        return self

    def predict(self, X):
        return self.reg_tree.predict(X)
```

[ ]:
```
params = {'ccp_alpha': np.logspace(np.log10(1e-06), np.log10(1e-01), 5).tolist()}
RegressionTree=␣
↪val_fun(RegressionTree,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
params = {'ccp_alpha': [1e-10,1e-8]}
RegressionTree=␣
↪val_fun(RegressionTree,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
```

Bagging and boosting

Starting with boosting

[ ]:
```
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

[ ]:
```
class BoostedMse:

    def __init__(self, learning_rate=0.1, n_estimators=50, max_depth=1):
        self.learning_rate = learning_rate
        self.n_estimators = n_estimators
        self.max_depth = max_depth


    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self

    def fit(self, X, y):
        self.model = GradientBoostingRegressor(learning_rate=self.learning_rate,␣
↪n_estimators=self.n_estimators, max_depth=self.max_depth)
        self.model.fit(X, y)
        return self

    def predict(self, X):
        return self.model.predict(X)
```

[ ]:
```
params = {'ccp_alpha': [0.1,0.2,0.3], 'n_estimators' : [ 50 , 100 , 200] ,␣
↪'max_depth' :[ 1] }
BoostedMse=␣
↪val_fun(BoostedMse,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
```

There is a typo here ccp_alpha is actually the learning rate; I will not fix it as the subsequent code takes 3 hours to run

more - Boosted

[ ]:
```
params = {'ccp_alpha': [0.01], 'n_estimators' : [ 500 , 1000 ] , 'max_depth' :
↪[1,2] }
BoostedMse=␣
↪val_fun(BoostedMse,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
evaluate(y_tst, BoostedMse.predict(X_tst))
```

```
Model with params: {'ccp_alpha': 0.01, 'max_depth': 1, 'n_estimators': 500}
finished.
with out-of-sample MSE on validation set: 0.01294
with out-of-sample R-squared on validation set: 1.2535847%
************************************************************
```

Boosting using GradientBoostingRegressor is too expensive computationally for the parameters below

I will use xgboost instead

[ ]:
```
from xgboost import XGBRegressor

params = {
    'n_estimators': [500,1000],
    'max_depth': [7],
    'random_state': [12308],
    'learning_rate': [.01]
}
XGB =␣
↪val_fun(XGBRegressor,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld)
params = {
    'n_estimators': [1000],
    'max_depth': [10],
    'random_state': [12308],
    'learning_rate': [.1]
}
XGB =␣
↪val_fun(XGBRegressor,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld)
```

Random Forest

[ ]:
```
from sklearn.ensemble import RandomForestRegressor
```

[ ]:
```
class RandomForest:

    def __init__(self,   n_estimators=50 , max_depth=2 , max_features = 20 ):

        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.max_features= max_features

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self

    def fit(self, X, y):
        self.model = RandomForestRegressor(n_estimators=self.n_estimators,␣
 ↪max_depth=self.max_depth)
        return self

    def predict(self, X):
        self.max_depth, max_features=self.max_features)
        self.model.fit(X, y)
        return self.model.predict(X)

params = {'n_estimators': [300,400], 'max_depth' : [ 5 , 6 , 7] , 'max_features'␣
 ↪:[30,50,100] }
```

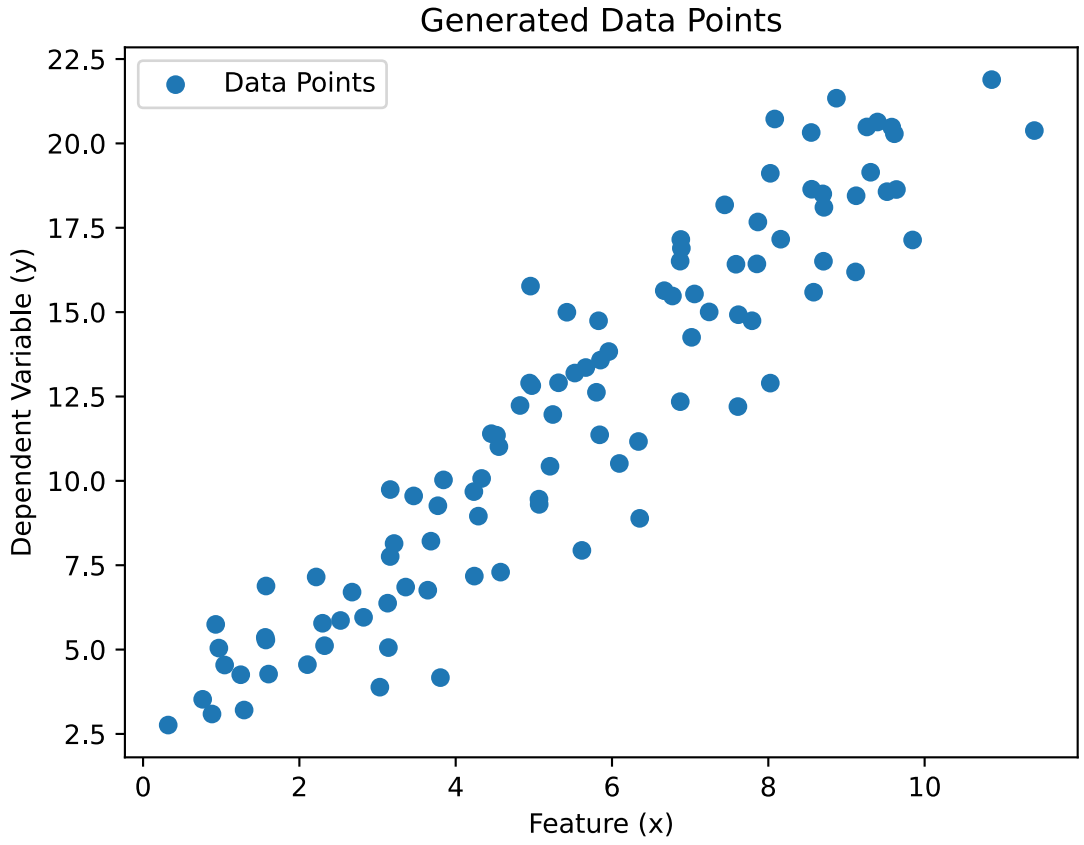
[ ]:
```
RandomForest=␣
 ↪val_fun(RandomForest,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
evaluate(y_tst, RandomForest.predict(X_tst))
```

RIDGE

- I use Accelerated Proximal Gradient descent - the proximal operator of ridge penalty in defined in appendix
- See Ridge's regularization path using simulated data
- I then construct apply ridge on the dataset

[ ]:
```
import apgpy as apg
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

n = 2
m = 100
A = np.column_stack((np.ones(m), np.linspace(1, 10, m) + np.random.randn(m)))
b = 2 * np.linspace(1, 10, m) + 1 + 0.5 * np.random.randn(m)
import matplotlib.pyplot as plt
```

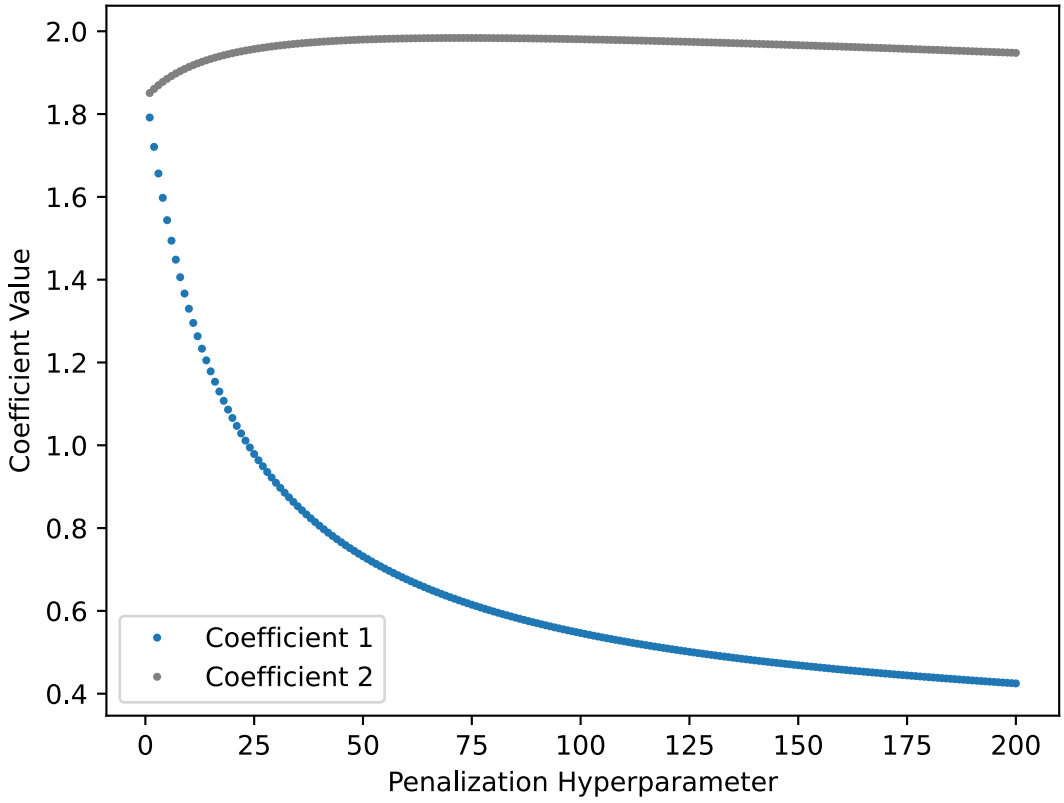
[ ]:
```
# Plotting the points formed by A and b
plt.scatter(A[:, 1], b, label='Data Points')
plt.xlabel('Feature (x)')
plt.ylabel('Dependent Variable (y)')
plt.title('Generated Data Points')
plt.legend()
plt.show()
coef1_values_rid = np.array([])  # Initialize array for coefficient 1
coef2_values_rid = np.array([])  # Initialize array for coefficient 2
```



[ ]:
```
mu_values = np.arange(200, 0, -1)

def quad_grad(y):
  return np.dot(A.T, (np.dot(A, y) - b))

def prox_ridge(y, t):
  return  y / (1 + t  * mu)

for mu in mu_values:
```

```
  x = apg.solve(quad_grad, prox_ridge, np.zeros(n), use_restart=True, eps=1e-12,␣
↪quiet=True)
  coef1_values_rid = np.append(coef1_values_rid, x[0])
  coef2_values_rid = np.append(coef2_values_rid, x[1])
```

[ ]:
```
plt.plot(mu_values, coef1_values_rid , 'o', label='Coefficient 1',markersize = 2␣
↪)
plt.plot(mu_values, coef2_values_rid , 'o', label='Coefficient 2', markersize =␣
↪2, color = "grey")
plt.xlabel('Penalization Hyperparameter')
plt.ylabel('Coefficient Value')
plt.legend()
plt.show()
```



Dynamics of Ridge Penalization *EVEN FOR LARGE PENALIZATION (LAMBDA = 200) , WE DO NOT GET SPARSITY.*

Applying Ridge penalty on dataset

[ ]:
```
class RidgeMse :
```

```
    def __init__(self, mu = 0.1 ):
        self.mu = mu

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self

    def fit(self, X, y):
      X = np.array(X)
      n = X.shape[1]
      y = np.array(y)

      def quad_grad(beta):
        return np.dot(X.T, (np.dot(X, beta) - y))

      def prox_ridge(beta, t):
        return  beta /(1 + t  * self.mu)

      coef_ridge_mse = apg.solve(quad_grad, prox_ridge, np.zeros(n),␣
↪use_restart=True, eps=1e-12, quiet=True)
      self.coef = coef_ridge_mse

      return self

    def predict(self, X):
        return np.dot(X, self.coef)
```

[ ]:
```
params = {'mu': np.linspace(0.1,100,7).tolist()}
RidgeMse =␣
 ↪val_fun(RidgeMse,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
evaluate(y_tst, RidgeMse.predict(X_tst))
```

Ridge with huber loss

[ ]:
```
class RidgeHuber :

    def __init__(self, mu = 0.1 ,delta = 0.02):
        self.mu = mu
        self.delta = delta

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self

    def fit(self, X, y):
```

```
      X = np.array(X)
      n = X.shape[1]
      y = np.array(y)

      def huber_loss_gradient(beta):
        residual = np.dot(X, beta) - y
        huber_grad = np.dot(X.T, np.where(np.abs(residual) <= self.delta,␣
↪residual, self.delta * np.sign(residual)))
        return huber_grad

      def prox_ridge(beta, t):
        return  beta /(1 + t  * self.mu)

      coef_ridge_mse = apg.solve(huber_loss_gradient, prox_ridge, np.zeros(n),␣
↪use_restart=True, eps=1e-12, quiet=True)
      self.coef = coef_ridge_mse

      return self

    def predict(self, X):
        return np.dot(X, self.coef)
```

[ ]:
```
params = {'mu': np.linspace(0.1,100,7).tolist(),'delta':np.array([0.02, 0.03, 0.
↪05, 0.1]) }
RidgeHuber =␣
↪val_fun(RidgeHuber,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
```

I evaluate on new tuning parameters. The choice of the new parameters was done after evaluating the previous fit. (Note I mistakenly deleted the previous evaluation results)

[ ]:
```
params = {'mu': (np.logspace(np.log10(100), np.log10(800), 4)).tolist(), 'delta':
↪ [0.025]}
RidgeHuber =␣
↪val_fun(RidgeHuber,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld,sleep=3)
```

[ ]:
```
 evaluate(y_tst, RidgeHuber.predict(X_tst))
```

```
***************Out-of-Sample Metrics***************
The out-of-sample R2 is -5.62371%
The out-of-sample MSE is 0.00876974
```

Generalized additive model using second order splines

I first install the group lasso package then regresss using GLM with group lasso using 2nd order splines

[ ]:
```
from group_lasso import GroupLasso

def flatten(l):
```

```
    return [item for sublist in l for item in sublist]

def SplineTransform(data,knots=3):
    spline_data = pd.DataFrame(np.ones((data.shape[0],1)),index=data.
 ↪index,columns=['const'])
    for i in data.columns:
        i_dat = data.loc[:,i]
        i_sqr = i_dat**2
        i_cut, bins = pd.cut(i_dat, 3, right=True, ordered=True, retbins=True)
        i_dum = pd.get_dummies(i_cut)
        for j in np.arange(knots):
            i_dum.iloc[:,j] = i_dum.iloc[:,j]*((i_dat-bins[j])**2)
        i_dum.columns = [f"{i}_{k}" for k in np.arange(1,knots+1)]
        spline_data = pd.concat((spline_data,i_dat,i_dum),axis=1)
    return spline_data

class GLMRegression:

    def __init__(self,knots=3,lmd=1e-4,l1_reg=1e-4,random_state=12308):
        self.knots = knots
        self.lmd = lmd
        self.random_state = random_state
        self.l1_reg = l1_reg

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self

    def fit(self,X,y):
        groups = [0]+flatten([list(np.repeat(i,self.knots+1))[:] for i in np.
 ↪arange(1,X.shape[1]+1)])
        X = SplineTransform(X)
        self.mod = GroupLasso(
            groups=groups,group_reg=self.lmd,l1_reg=self.l1_reg,
            fit_intercept=False,random_state=self.random_state
        )
        self.mod = self.mod.fit(X,y)
        return self

    def predict(self,X):
        X = SplineTransform(X)
        return self.mod.predict(X)
```

[ ]:
```
params = { 'knots':[3], 'lmd':[1e-4,1e-1], 'l1_reg':[1e-4,0]}
GLM =␣
 ↪val_fun(GLMRegression,params=params,X_trn=X_trn,y_trn=y_trn,X_vld=X_vld,y_vld=y_vld)
```

```
evaluate(y_tst, GLM.predict(X_tst))
```

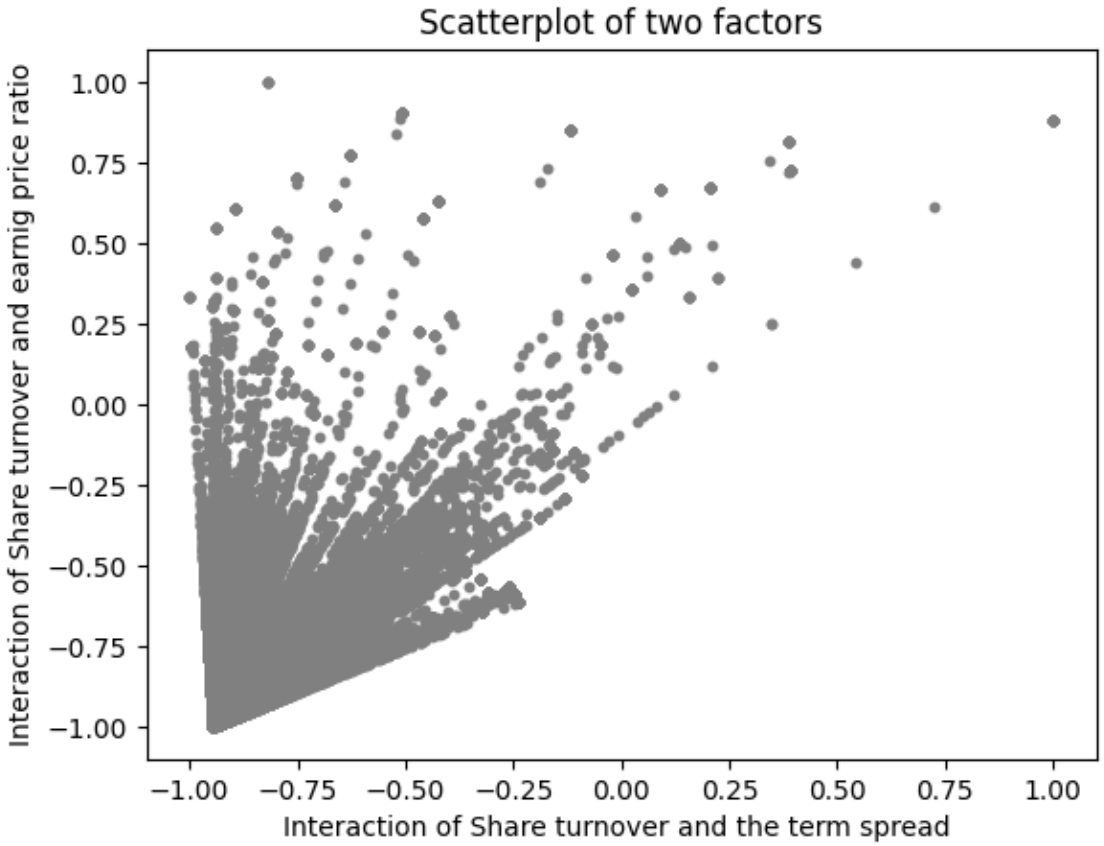
[ ]:
```
class GLMRegressionHuber:

    def __init__(self, knots=3, lmd=1e-4, l1_reg=1e-4, random_state=12308,␣
 ↪huber_delta=1.0):
        self.knots = knots
        self.lmd = lmd
        self.random_state = random_state
        self.l1_reg = l1_reg
        self.huber_delta = huber_delta

    def set_params(self, **params):
        for param in params.keys():
            setattr(self, param, params[param])
        return self

    def fit(self, X, y):
        groups = [0] + flatten([list(np.repeat(i, self.knots + 1))[:] for i in␣
 ↪np.arange(1, X.shape[1] + 1)])
        X_transformed = SplineTransform(X)

        # Introduce Huber loss
        huber_loss = lambda x: 0.5 * x**2 if np.abs(x) <= self.huber_delta else␣
 ↪self.huber_delta * (np.abs(x) - 0.5 * self.huber_delta)

        self.mod = GroupLasso(
            groups=groups, group_reg=self.lmd, l1_reg=self.l1_reg,
            fit_intercept=False, random_state=self.random_state, loss=huber_loss
        )
        self.mod = self.mod.fit(X_transformed, y)
        return self

    def predict(self, X):
        X_transformed = SplineTransform(X)
        return self.mod.predict(X_transformed)
```

[ ]:
```
params = {'knots': [3], 'lmd': [1e-4, 1e-1], 'l1_reg': [1e-4, 0], 'huber_delta':␣
 ↪[1.0, 0.5]}
GLM_H = val_fun(GLMRegression, params=params, X_trn=X_trn, y_trn=y_trn,␣
 ↪X_vld=X_vld, y_vld=y_vld)
evaluate(y_tst, GLM_H.predict(X_tst))
```

CODE FOR PCA VS ICA on dataset(PLOTS)

[ ]:
```
plt.scatter(X_trn['turn*tms'], X_trn['turn*ep'], color='grey', s=10)
plt.xlabel('Interaction of Share turnover and the term spread ')
```

```
plt.ylabel('Interaction of Share turnover and earnig price ratio')
plt.title('Scatterplot of two factors')
plt.show()
```



[ ]:
```
X1 = np.array(X_trn['turn*tms'])
X2 = np.array(X_trn['turn*ep'])
X_design= np.column_stack((X1, X2))
```

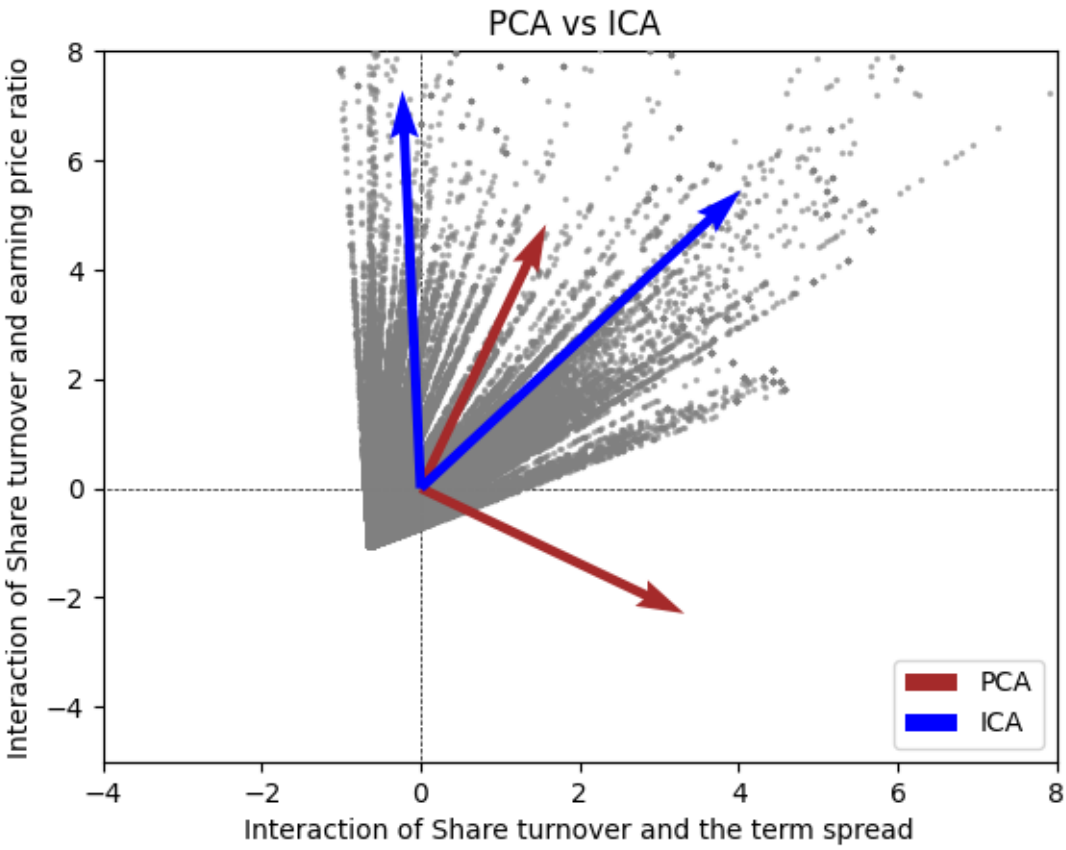
[ ]:
```
X = (X_design - np.mean(X_design))/np.std(X_design)
```

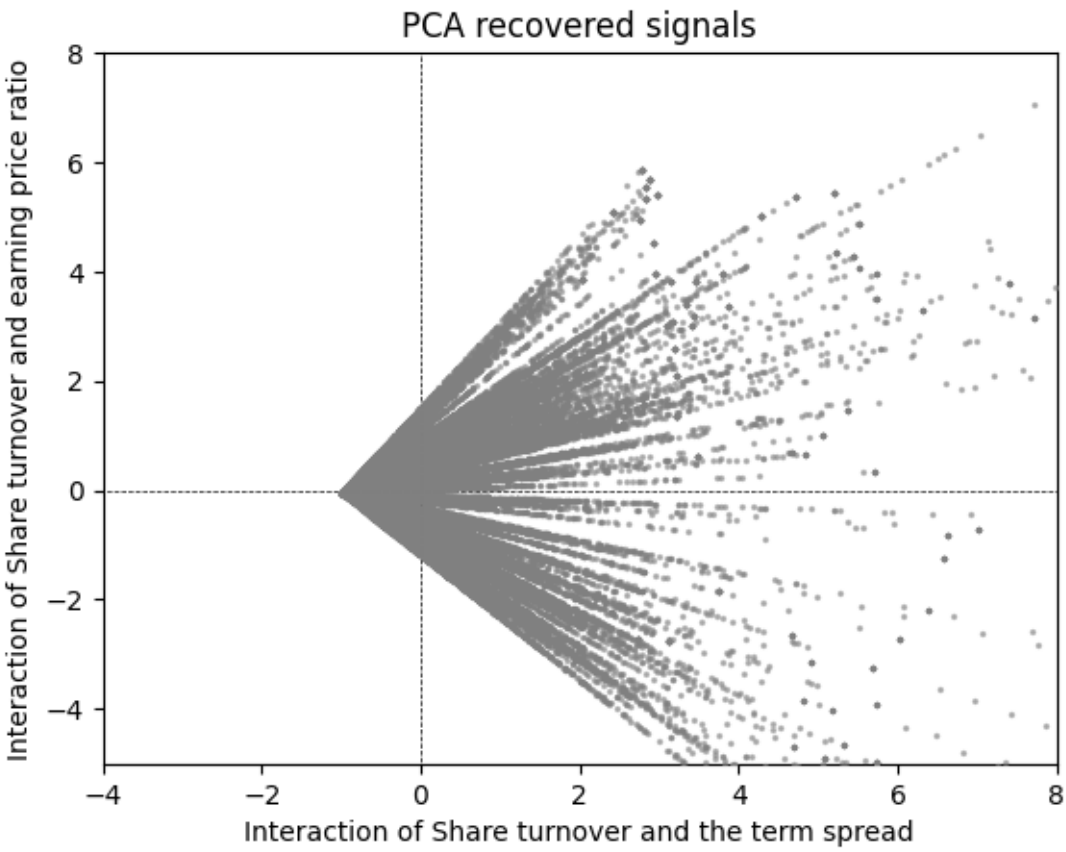
[ ]:
```
pca = PCA()
S_pca_ = pca.fit(X).transform(X)

ica = FastICA(whiten="unit-variance")
S_ica_ = ica.fit(X).transform(X)  # Estimate the sources
```

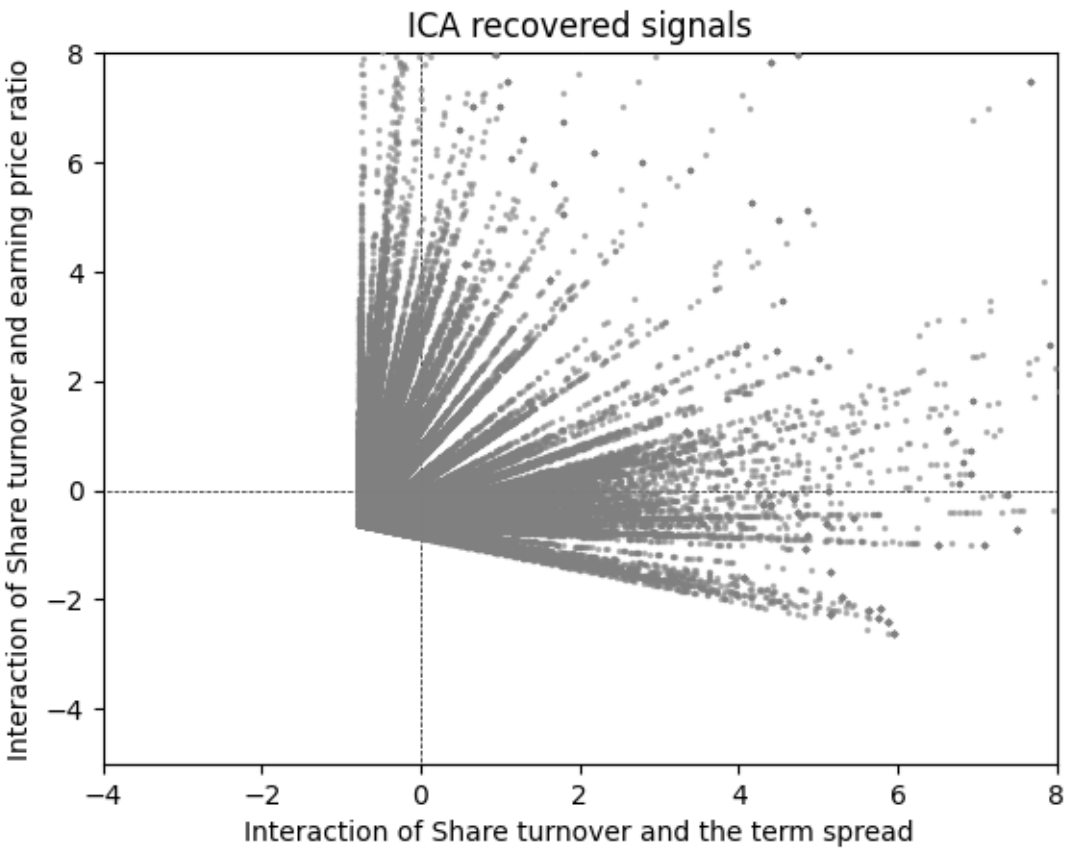
[ ]:
```
import matplotlib.pyplot as plt
```

[ ]:
```
def plot_samples(S, axis_list=None):
    plt.scatter(
        S[:, 0], S[:, 1], s=2, marker="o", zorder=10, color="grey", alpha=0.5
    )
    if axis_list is not None:
        for axis, color, label in axis_list:
            axis /= axis.std()
            x_axis, y_axis = axis
            plt.quiver(
                (0, 0),
                (0, 0),
                x_axis,
                y_axis,
                zorder=11,
                width=0.01,
                scale=6,
                color=color,
                label=label,
            )

    plt.hlines(0, -10, 10, linestyles='dashed', colors='black', linewidth=0.5)
    plt.vlines(0, -10, 10, linestyles='dashed', colors='black', linewidth=0.5)
    plt.xlim(-4, 8)
    plt.ylim(-5, 8)
    plt.xlabel('Interaction of Share turnover and the term spread')
    plt.ylabel('Interaction of Share turnover and earning price ratio')


axis_list = [(pca.components_.T, "brown", "PCA"), (ica.mixing_, "blue", "ICA")]

# Plot 1: Observations
plt.figure()
plot_samples(X, axis_list=axis_list)
legend = plt.legend(loc="lower right")
legend.set_zorder(100)
plt.title("PCA vs ICA ")
plt.savefig('observations_plot.png')  # Save the plot as a PNG file

# Plot 2: PCA recovered signals
plt.figure()
plot_samples(S_pca_ / np.std(S_pca_, axis=0))
plt.title("PCA recovered signals")
plt.savefig('pca_recovered_plot.png')  # Save the plot as a PNG file

# Plot 3: ICA recovered signals
plt.figure()
plot_samples(S_ica_ / np.std(S_ica_))
```

```
plt.title("ICA recovered signals")
plt.savefig('ica_recovered_plot.png')  # Save the plot as a PNG file

plt.show()
```




PCA recovered signals
Interaction of Share turnover and earning price ratio
Interaction of Share turnover and the term spread
8
6
4
2
0
−2
−4
−4
−2
0
2
4
6
8


ICA recovered signals
Interaction of Share turnover and earning price ratio
Interaction of Share turnover and the term spread
8
6
4
2
0
−2
−4
−4
−2
0
2
4
6
8

References

[1] Peter Bossaerts and Pierre Hillion. Implementing Statistical Criteria to Select Return Forecasting Models: What Do We Learn? *The Review of Financial Studies*, 12(2):405–428, 06 2015.

[2] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984.

[3] J. Cochrane. *Asset Pricing: Revised Edition*. Princeton University Press, 2009.

[4] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[5] Joachim Freyberger, Andreas Neuhierl, and Michael Weber. Dissecting Characteristics Nonparametrically. *The Review of Financial Studies*, 33(5):2326–2377, 04 2020.

[6] Amit Goyal and Ivo Welch. Predicting the equity premium with dividend ratios. *Management Science*, 49(5):639–654, 2003.

[7] Yves Grandvalet. Bagging equalizes influence. *Mach. Learn.*, 55(3):251–270, jun 2004.

[8] Jeremiah Green, John R. M. Hand, and X. Frank Zhang. The Characteristics that Provide Independent Information about Average U.S. Monthly Stock Returns. *The Review of Financial Studies*, 30(12):4389–4436, 03 2017.

[9] Shihao Gu, Bryan Kelly, and Dacheng Xiu. Empirical Asset Pricing via Machine Learning. *The Review of Financial Studies*, 33(5):2223–2273, 02 2020.

[10] A.J. Hendrikse and Lieuwe Jan Spreeuwers. Component ordering in independent component analysis based on data power. In Raymond N.J. Veldhuis, R.N.J. Veldhuis, and H.S. Cronie, editors, *Proceedings of the 28th Symposium on Information Theory in the Benelux*, number LNCS4549, pages 211–218, Netherlands, June 2007. Werkgemeenschap voor Informatie- en Communicatietheorie (WIC). 28th Symposium on Information Theory in the Benelux 2007 ; Conference date: 24-05-2007 Through 25-05-2007.

[11] Bryan T. Kelly and Dacheng Xiu. Financial Machine Learning. NBER Working Papers 31502, National Bureau of Economic Research, Inc, July 2023.

[12] A.J. Laub. *Matrix Analysis for Scientists and Engineers*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2005.

[13] Stephen F. LeRoy and Jan Werner. *Principles of Financial Economics*. Cambridge University Press, 2 edition, 2014.

[14] Jonathan Lewellen. The cross-section of expected stock returns. *Critical Finance Review*, 4, 01 2011.

[15] Sydney C. Ludvigson and Serena Ng. The empirical risk–return relation: A factor analysis approach. *Journal of Financial Economics*, 83(1):171–222, 2007.

[16] D. McFadden. Conditional logit analysis of qualitative choice behavior. Technical Report 105-142, Frontiers in Econometric, New York, 1974.

[17] Rajnish Mehra and Edward C. Prescott. The equity premium: A puzzle. *Journal of Monetary Economics*, 15(2):145–161, 1985.

[18] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2012.

[19] Stefan Nagel. *Machine Learning in Asset Pricing*, volume 8. Princeton University Press, 2021.

[20] L. Oneto, S. Ridella, and D. Anguita. Tikhonov, ivanov and morozov regularization for support vector machine learning. *Machine Learning*, 103(1):103–136, 2015.

[21] David Rapach and Guofu Zhou. Chapter 6 - forecasting stock returns. In Graham Elliott and Allan Timmermann, editors, *Handbook of Economic Forecasting*, volume 2 of *Handbook of Economic Forecasting*, pages 328–383. Elsevier, 2013.

[22] Richard Roll. A critique of the asset pricing theory's tests part i: On past and potential testability of the theory. *Journal of Financial Economics*, 4(2):129–176, 1977.

[23] Cosma Rohilla Shalizi. Advanced data analysis from an elementary point of view. 2012.

[24] James H. Stock and Mark W. Watson. Forecasting using principal components from a large number of predictors. *Journal of the American Statistical Association*, 97(460):1167–1179, 2002.

[25] James V. Stone. Independent component analysis: an introduction. *Trends in Cognitive Sciences*, 6(2):59–64, 2002.

[26] Nassim Nicholas Taleb. Statistical consequences of fat tails: Real world preasymptotics, epistemology, and applications, 2022.

[27] Nassim Nicholas Taleb, Pierre Zalloua, Khaled Elbassioni, Andreas Henschel, and Daniel E. Platt. Informational rescaling of pca maps with application to genetic distance, 2023.

[28] ROBERT TIBSHIRANI. The lasso method for variable selection in the cox model. *Statistics in Medicine*, 16(4):385–395, 1997.

[29] V. Vapnik. *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer New York, 1999.

[30] Larry Wasserman. *All of Nonparametric Statistics (Springer Texts in Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[31] Ivo Welch and Amit Goyal. A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies*, 21(4):1455–1508, 2008.