Symphony in the Latent Space: Provably Integrating High-dimensional Techniques with Non-linear Machine Learning Models

Qiong Wu*[1], Jian Li[2], Zhenming Liu[1], Yanhua Li[3], Mihai Cucuringu[4]
[1]William & Mary
[2]Tsinghua University
[2]Worcester Polytechnic Institute
[4]University of Oxford and The Alan Turing Institute

Abstract

This paper revisits building machine learning algorithms that involve interactions between entities, such as those between financial assets in an actively managed portfolio, or interactions between users in a social network. Our goal is to forecast the future evolution of ensembles of multivariate time series in such applications (e.g., the future return of a financial asset or the future popularity of a Twitter account). Designing ML algorithms for such systems requires addressing the challenges of high-dimensional interactions and non-linearity. Existing approaches usually adopt an **ad-hoc** approach to integrating high-dimensional techniques into non-linear models and recent studies have shown these approaches have questionable efficacy in time-evolving interacting systems.

To this end, we propose a novel framework, which we dub as the ***additive influence model***. Under our modeling assumption, we show that it is possible to decouple the learning of high-dimensional interactions from the learning of non-linear feature interactions. To learn the high-dimensional interactions, we leverage kernel-based techniques, with provable guarantees, to embed the entities in a low-dimensional latent space. To learn the non-linear feature-response interactions, we generalize prominent machine learning techniques, including designing a new statistically sound non-parametric method and an ensemble learning algorithm optimized for vector regressions. Extensive experiments on two common applications demonstrate that our new algorithms deliver significantly stronger forecasting power compared to standard and recently proposed methods.

Introduction

We revisit the problem of building machine learning algorithms that involve interactions between entities, such as those between users and items in a recommendation system, or between financial assets in an actively managed portfolio, or between populations in different counties in a disease-spreading process. Our proposed forecasting model uses information available up to time t to predict $\mathbf{y}_{t+1,i}$, the future behavior of entity i at time $t+1$ (e.g., the future price of stock i at time $t+1$), for a total number of d entities (Laptev et al. 2017; Farhangi et al. 2022). Designing such models has proven remarkably difficult, as one needs to circumvent two main challenges that require often incompatible solutions.

1. Cross-entity interaction: high-dimensionality. In many ensembles of multivariate time series systems, it is often the case that the current state of one entity could potentially impact the future state of another. When considering the equity market as an example, Amazon's disclosure of its revenue change in cloud services could indicate that the revenues of other cloud providers (e.g., competitors) could also change.

The interaction is high-dimensional because the total possible number of interactions is usually much larger than the number of available observations. For example, in a portfolio of 3,000 stocks, the total number of potential links between pairs of stocks is $3,000 \times 3,000 \approx 10^7$, but we often have only 2,500 data points (e.g., 10 years of daily data), and thus capturing the cross-entity interactions becomes a very challenging problem.

2. Feature-response interactions: non-linearity. Linear models are usually insufficient to characterize the relationship between the response/label and the available information (features), thus techniques beyond simple linear regressions are heavily needed. For example, in a financial context, economic productivity is non-linear in temperature for most countries; similarly, electricity consumption is a nonlinear function of temperature, and modeling this relationship is crucial for pricing electricity derivative contracts. As shown in Fig. 1(a), the existing relevant learning models can be categorized into the following two groups.

1. Provable cross-entity models (CEM) for high-dimensionality. Cross-entity models solve a vector regression problem $\mathbf{y}_{t+1} = f(\mathbf{x}_t) + \xi_t$, to forecast the future behavior of all entities, where $\mathbf{y}_{t+1} \triangleq (y_{t+1,1}, \ldots, y_{t+1,d})$, and $\mathbf{x}_t$ denotes the features of all entities, constructed from their historical data. Since the features of one entity can be used to predict the future behavior of another, CEMs have stronger expressive and predictive power. CEMs are both computationally and statistically challenging because we need to solve the "high-dimensional" (overparametrized) problem and mathematically understand the root cause of the overfitting. Extensive research has been undertaken to design regularization techniques (Chen, Dong, and Chan 2013; Friedman, Hastie, and Tibshirani 2001; Wu et al. 2021) to address the issue, and most algorithms in this category are

*Currently working at AT&T Labs.

linear and have theoretical guarantees.

2. Practical univariate models (UM) for non-linearity. Univariate models fit a function $\mathbf{y}_{t+1,i} = f(\mathbf{x}_{t,i}) + \xi_{t,i}$ to forecast one entity's feature behavior by using features constructed from that entity's historical data. Univariate models primarily learn the feature-response interaction by using off-the-shelf ML techniques such as Deep learning (DL) (Abadi et al. 2016; Hochreiter and Schmidhuber 1997; Wu et al. 2019) or gradient boosted algorithms (Chen and Guestrin 2016; Ke et al. 2017; Dorogush, Ershov, and Gulin 2018). These practical models are effective in extracting non-linear signals but they often do not come with theoretical guarantees.

Existing integration techniques: ad-hoc methods It remains unclear how to integrate two seemingly incompatible modeling processes (i.e., UM and CEM) with different design philosophies. In Fig. 1 (b), we show that existing integration solutions predominately follow an **ad-hoc** approach, in part due to the belief that deep learning is the "holly-grail" for practical problems (Sejnowski 2018). For example, one often adds an ℓ_1- or ℓ_2-regularizer to a neural net's cost function, hoping such regularizers will also magically work in neural nets (Abadi et al. 2016; Paszke et al. 2017). However, the mathematical properties of a provable technique often break when combined into a neural net. Furthermore, latent embedding models have also been recently introduced (Wang et al. 2019; Feng et al. 2019; Chen et al. 2019). The central idea is to project the entities into points in a low-dimensional space so that similar entities (i.e., stocks in the above works) are closer to each other in this embedding. Because point interactions are more restrictive in the latent space, they have the potential to address the overfitting issues (Wang et al. 2019). However, these lines of work do not offer any theoretical guarantees and are often not robust in practice. Recent studies have demonstrated that the efficacy of such ad-hoc approaches is questionable in many interacting systems (Dacrema, Cremonesi, and Jannach 2019; Rendle, Zhang, and Koren 2019; Qiong et al. 2021).

Our approach & contributions We propose a general latent position model dubbed as the *additive influence model* to enable us to seamlessly orchestrate mathematically rigorous high-dimensional techniques with practically effective machine learning algorithms. In Fig. 1(c), we show that it is possible to decouple the learning of high-dim interactions between entities from the learning of the non-linear signals.

We assume each entity is associated with an embedded position $\mathbf{z}_i$ and at timestamp t, entity i is also associated with an unobserved signal $\mathbf{s}_{i,t} \in \mathbf{R}$ that is a function of $\mathbf{x}_{i,t}$. We assume the generative model $\mathbf{y}_{i,t} = \sum_{i \leq j} \kappa(\mathbf{z}_i, \mathbf{z}_j)\mathbf{s}_{j,t} + \epsilon_{i,t}$, where $\kappa(\mathbf{z}_i, \mathbf{z}_j)$ is a function that measures the interaction strength between $\mathbf{z}_i$ and $\mathbf{z}_j$, and can be any kernel function, such as a Gaussian kernel or simply an inner product, and $\epsilon_{i,t}$ denotes noise. Each entity could potentially influence $\mathbf{y}_{i,t}$. The influence of j on i depends on the "distance" or "similarity" between $\mathbf{z}_i$ and $\mathbf{z}_j$. On the other hand, we assume $\mathbf{s}_{j,t} = g(\mathbf{x}_{j,t})$ for some $g(\cdot)$, so that the model captures high-dimensional interactions via $\mathbf{z}_i$ and non-linearity via $g(\cdot)$.

Our proposed model allows for feature interactions through $g(\cdot)$, and addresses the overfitting problem arising from entity interactions because the distances (interaction strength) between entities are constrained by the latent Euclidean space: when both $(\mathbf{z}_i - \mathbf{z}_j)$ and $(\mathbf{z}_j - \mathbf{z}_k)$ are small, then $(\mathbf{z}_i - \mathbf{z}_k)$ is also small, and thus the degree of freedom for entity interactions becomes substantially smaller than $O(d^2)$.

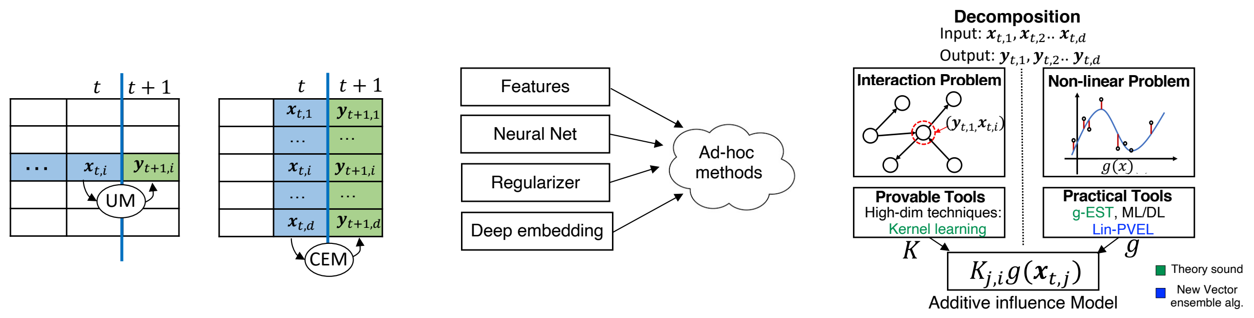
Our goal is to learn both the $\mathbf{z}_i$'s and $g(\cdot)$. We note that these two learning tasks can be *decoupled*: high-dimensional methods can be developed to *provably* estimate the $\mathbf{z}_i$'s *without the knowledge of* $g(\cdot)$, and when estimates of $\mathbf{z}_i$'s are given, an experiment-driven process can be used to learn $g(\cdot)$ by examining prominent machine learning methods such as neural nets and boosting. In other words, when we learn entity interactions, we do not need to be troubled by the overfitting problem escalated by fine-tuning $g(\cdot)$, and when we learn feature interactions, the generalization error will not be jeopardized by the curse of dimensionality from entity interactions.

- To learn the $\mathbf{z}_i$'s, we design a simple algorithm that uses low-rank approximation of $\mathbf{y}_t$'s covariance matrix to infer the closeness of the entities and develop a novel theoretical analysis based on recent techniques from high dimensionality and kernel learning (Belkin 2018; Tang et al. 2013; Wu et al. 2020a).
- To learn $g(\cdot)$, we generalize major machine learning techniques, including neural nets, non-parametric, and boosting methods, to the additive influence model when estimates of $\mathbf{z}_i$'s are known. We specifically develop a moment-based algorithm for non-parametric learning of $g(\cdot)$, and a computationally efficient boosting algorithm.
- Finally, we perform extensive experiments on a major equity market and social network datasets to confirm the efficacy of our modeling approaches and analysis.

Related work and comparison

Univariate machine learning models handle feature-response interactions and mostly rely on deep learning and GBRT (Goodfellow, Bengio, and Courville 2016; Wu et al. 2020b; Goodfellow, Bengio, and Courville 2016; Wüthrich, Permunetilleke et al. 1998; Chen and Guestrin 2016; Ke et al. 2017; Dorogush, Ershov, and Gulin 2018; Gong et al. 2017; Yang and Ding 2020; Ding et al. 2015; Zhang, Aggarwal, and Qi 2017; Feng, Polson, and Xu 2018; Han et al. 2018; Wu et al. 2020b; Chen, Pelger, and Zhu 2019; Kelly, Pruitt et al. 2019; Ke et al. 2019; Chen et al. 2019; Li et al. 2019; Wu et al. 2015). These models aim to optimize their empirical performance and limit theoretical investigations. Recent cross-entity models consider the high-dimensional interactions, where overfitting easily happens and theoretical justifications are essential to avoid spurious result in practice. Cross-entity models are mostly linear models (Bunea, She, and Wegkamp 2011; Koltchinskii, Lounici et al. 2011; Negahban and Wainwright 2011; Huang, Li, and Zhou 2019) that have theoretical guarantees, but they cannot effective for non-linear feature-response interactions. Efforts for building CEMs include (Tibshirani 1996; Candès and Wakin 2008; Tao and Series 2009; Hoerl and Kennard 1970; Tsigler and Bartlett 2020; Liu et al. 2019).

Ad-hoc approach for integration. Recent integrating solutions for high-dimensionality and nonlinearity challenges has



(a) UM and CEM. (b) Ad-hoc methods lead to overfitting. (c) Our proposed framework.

Figure 1: (a) UM for non-linearity and CEM for high-dimensionality. (b) Exsiting ad-hoc methods have questionable efficacy. (c) Our framework decouples the high-dimensional learning of entity interactions and non-linear learning of feature interactions.

been a frustrating endeavor, which we can call the ad-hoc approaches and many were shown to have questionable efficacy in interacting systems. *1. Deep learning + Lasso/Ridge* For example, one (Abadi et al. 2016; Paszke et al. 2017) often adds an l_1- or l_2-regularizer to a neural net's cost function, hoping these regularizers can also magically work in neural nets. *2. Deep embedding.* Recent studies have addressed high-dimensional entity interactions by using deep embedding, based on the idea that when entities are embedded in low-dim Euclidean space, they can interact in a quite restricted way, therefore preventing overfitting (Zhao et al. 2020; Shen et al. 2022; Xie, Girshick, and Farhadi 2016; Zhang, Aggarwal, and Qi 2017; Hu, Liu et al. 2018; Li et al. 2019; Wang et al. 2019). While this idea is effective for linear models (Abraham et al. 2015; Li et al. 2017), deep embedding-based solutions may have very high false positive rates, for instance, when forecasting the returns of financial assets (Qiong et al. 2021; Wang et al. 2019).

Remark: *(i) Modeling framework.* Our framework proposes a key algorithmic insight that the latent position estimation should be decoupled from the learning link function $g(\cdot)$. We develop the first algorithm that can provably estimate the entity's latent positions and provide theoretical guarantees. Our novel analysis leverages a diverse set of tools from kernel learning, non-parametric methods, and random walks. *(ii) Comparison to deep embedding.* While embedding can be learned by deep learning (Hu, Liu et al. 2018; Wang et al. 2019), it usually does not provide any theoretical guarantee, whereas our framework makes stricter assumptions (e.g., how embedding and features should interact) and delivers a quality guarantee. Deep embedding also requires every component including the function $g(\cdot)$ in the architecture to be represented by a neural net to run SGD, whereas we allow $g(\cdot)$ to be learned by a wide range of algorithms such as boosting or non-parametric techniques.

Problem definition

Notations. For a matrix A, $\mathcal{P}_r(A)$ denotes its rank-r approximation obtained by keeping the top r singular values and the corresponding singular vectors. $\sigma_i(A)$ (resp. $\lambda_i(A)$) is the i-th singular value (resp. eigenvalue) of A. We use Python/MATLAB notation when we refer to a specific row or column. For example, $A_{1,:}$ is the first row of A, and $A_{:,1}$ is the first column. $\|A\|_F$ and $\|A\|_2$ denote the Frobenius and spectral norms, respectively, of A. In general, we use boldface upper case (e.g., $\mathbf{X}$) to denote data matrices and boldface lower case (e.g., $\mathbf{x}$) to denote one sample. $\mathbf{x}_{t,i}$, which refers to the features associated with stock i at time t, can be one or multi-dimensional. Let $(\mathbf{x}_{t,i})_j$ be the j-th coordinate (feature) of $\mathbf{x}_{t,i}$. An event occurring with high probability (whp) means that it happens with probability $\geq 1 - n^{-10}$, where 10 is an arbitrarily chosen large constant and is not optimized. A bivariate function is a Gaussian kernel if $\kappa(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/\sigma^2)$, an inverse multi-quadratic (IMQ) kernel if $\kappa(\mathbf{x}, \mathbf{x}') = (c^2 + \|\mathbf{x} - \mathbf{x}'\|^2)^{-\alpha}$ ($\alpha > 0$), and an inner product kernel if $\kappa(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$. A function $g(\cdot)$ is Lipschitz-continuous if $|g(\mathbf{x}_1) - g(\mathbf{x}_2)| \leq c\|\mathbf{x}_1 - \mathbf{x}_2\|$ for a constant c. A distribution $\mathcal{D}$ with bounded domain and probability density function $f_{\mathcal{D}}$ is near-uniform if $\frac{\sup f_{\mathcal{D}}(\mathbf{x})}{\inf f_{\mathcal{D}}(\mathbf{x})} = O(1)$.

The forecasting problem. We operate in a time-dependent setting, where each timestamp t can be construed as the t^{th} round. An interacting system consisting of d entities (e.g., denoting stocks in the equity market or user accounts in a network), that are updated at each round, for a total number of T rounds. Let $\mathbf{y}_{t,i} \in \mathbf{R}$ denote the next-period forecast of entity i at the t-th round, and $\mathbf{y}_t = (\mathbf{y}_{t,1}, \ldots, \mathbf{y}_{t,d}) \in \mathbf{R}^d$. Our goal is to forecast $\mathbf{y}_t$ based on all information available up to (but excluding) round t.

Model Assumptions. Under the additive influence model, a generic model takes the form

$$\mathbf{y}_{t,i} = \sum_{j \leq d} \kappa(\mathbf{z}_i, \mathbf{z}_j) g(\mathbf{x}_{t,j}) + \xi_{t,i}, \tag{1}$$

and our goal is to learn $g(\cdot)$ and $\mathbf{z}_i$'s with a total number of n observations. Let $K \in \mathbf{R}^{d \times d}$ such that $K_{i,j} = \kappa(\mathbf{z}_i, \mathbf{z}_j)$. Here, we assume that • *(A.1)* the vector representations $\mathbf{z}_i$'s of the stocks and features $\mathbf{x}_{t,i}$ are i.i.d. samples from (two different) near-uniform distributions on bounded supports, • *(A.2)* $\mathbf{x}_{t,i} \in [-1, 1]$ and $\mathbb{E}[g(\mathbf{x}_{t,i})] = 0$, • *(A.3)* $g(\cdot)$ is Lipschitz-continuous, and • *(A.4)* $\xi_{t,i}$'s are zero-mean i.i.d. Gaussian random variables with standard deviation σ_ξ.

We remark that (A.1) is standard in the literature (Abraham et al. 2015; Sussman, Tang, and Priebe 2013; Tang et al. 2013; Li et al. 2017; Rastelli, Friel, and Raftery 2016). Assuming (A.2) simplifies the calculation and is without loss of generality, and (A.4) can also be relaxed to settings in which the $\xi_{t,i}$ variables are sub-Gaussian. See App. A for a more detailed discussion of the assumptions.

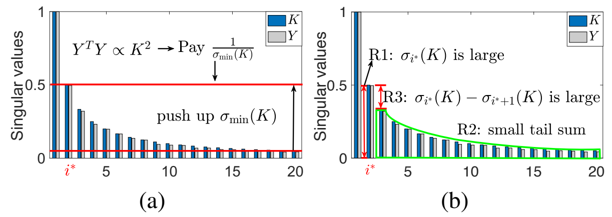


Figure 2: (a) We use the square root of $\mathcal{P}_{i^*}(\mathbf{Y}^{\mathrm{T}}\mathbf{Y})$ to approximate K so that we pay a factor of $1/\sigma_{i^*}(K)$, instead of $1/\sigma_{\min}(K)$. (b) Three key requirements for i^*: • (R1) $\sigma_{i^*}(K)$ is large, • (R2) $\mathcal{P}_{i^*}(K^2)$ is close to K^2, and • (R3) $\sigma_{i^*}(K) - \sigma_{i^*+1}(K)$ is large.

Our algorithms

This section introduces our algorithmic pipeline in full detail. Sec. 4 describes an algorithm for learning the embedding without knowing $g(\cdot)$. Sec. 4 explains the estimation of $g(\cdot)$ using machine learning techniques. Due to the space limit, detailed proofs of all the Props are deferred to App. B.

Learning vector representation provably

This section presents a provable algorithm to estimate the kernel matrix K and the embedding $\mathbf{z}_i$'s. Our algorithm does not require knowledge of $g(\cdot)$, thus providing a conceptually new approach to construct CEMs: high-dimensional learning of entity interactions can be decoupled from using ML techniques to fit the features. Because learning entity interactions could be a major source of causing overfitting, disentangling it from the downstream task of learning $g(\cdot)$ enables us to leverage the function-fitting power of ML techniques without the cost of amplifying generalization errors.

We next walk through our design intuition and start by introducing additional notation. Let $\mathbf{Y} \in \mathbf{R}^{n\times d}$ be such that $\mathbf{Y}_{t,i} = \mathbf{y}_{t,i}$ ($\mathbf{Y}$ is a matrix and $\mathbf{y}$ a random variable), $\mathbf{S} \in \mathbf{R}^{n\times d}$ with $\mathbf{S}_{t,i} = \mathbf{s}_{t,i} \triangleq g(\mathbf{x}_{t,i})$, and $E \in \mathbf{R}^{n\times d}$ with $E_{t,i} = \xi_{t,i}$. Recall that $K \in \mathbf{R}^{d\times d}$ s.t. $K_{i,j} = \kappa(\mathbf{z}_i, \mathbf{z}_j)$, and $\mathcal{P}_r(A)$ denotes A's rank-r approximation obtained by keeping the top r singular values and vectors. Finally, for any PSD matrix A with SVD $A = U\Sigma U^{\mathrm{T}}$, let $\sqrt{A} \triangleq U\Sigma^{\frac{1}{2}}U^{\mathrm{T}}$.

Eq. (1) can be re-written as $\mathbf{Y} = \mathbf{S}K + E$, in which we need to infer K using only $\mathbf{Y}$. We first observe that while none of the entries in $\mathbf{S}$ are known, the $\mathbf{S}_{t,i}$'s are i.i.d. random variables (because the $\mathbf{x}_{t,i}$'s are i.i.d.); therefore, our problem resembles a *dictionary learning* problem, in which K can be viewed as the dictionary to be learned, and $\mathbf{S}$ is the measurement matrix (see e.g., (Arora, Bhaskara et al. 2014)). However, in our case, K is neither low-rank nor sparse, and we cannot use standard dictionary learning techniques.

First, we observe that, if infinitely many samples were available, then $\mathbf{Y}^{\mathrm{T}}\mathbf{Y}/n$ approaches to K^2. Hence, intuitively we could use $\sqrt{\mathbf{Y}^{\mathrm{T}}\mathbf{Y}/n}$ to approximate $\sqrt{K^2} = K$. However, the existing standard matrix square root result has the notorious "$1/\sigma_{\min}$-blowup" problem, i.e., it gives us only $\|\sqrt{\mathbf{Y}^{\mathrm{T}}\mathbf{Y}/n} - KW\|_F \propto 1/\sigma_{\min}(K)$ (W a unitary matrix), where typically $\sigma_{\min}(K)$ is extremely small, thus rendering the bound too loose to be useful (Bhojanapalli, Kyrillidis, and Sanghavi 2016).

To tackle the problem, our algorithm uses $\sqrt{\mathcal{P}_{i^*}(\mathbf{Y}^{\mathrm{T}}\mathbf{Y})/n}$ to approximate K for a carefully chosen i^* so that we pay a factor of $\sigma_{i^*}(K)$, instead of $\sigma_{\min}(K)$, to substantially tighten the error. See Alg. 2 in App. B and Fig. 2(a). To implement this idea, we need to show that there always exists an i^* such that • *(R1):* $\sigma_{i^*}(K)$ is sufficiently large, • *(R2):* $\mathcal{P}_{i^*}(K^2)$ is close to K^2, and • *(R3):* the spectral gap $\sigma_{i^*}(K) - \sigma_{i^*+1}(K)$ is sufficiently large so that we can use the Davis-Kahan theorem to prove that $\mathcal{P}_{i^*}(K^2) \propto \mathcal{P}_{i^*}(Y^{\mathrm{T}}Y)$ (Stewart 1990). See also Fig. 2(b).

These three requirements may not always be met simultaneously. For example, when $\sigma_i(K^2) \propto \frac{1}{i}$, the gap is insufficient and the tail diverges (R2 and R3 are violated). Therefore, we integrate the following two results. • *(i)* The eigenvalues decay fast. This stems from two classical results from the *kernel learning* literature. First, when $\kappa(\cdot,\cdot)$ is sufficiently smooth (such as the Gaussian, IMQ, or inner product kernels), the eigenvalues of the kernel operator $\mathcal{K}$ associated with $\kappa(\cdot,\cdot)$ decay exponentially (e.g., $\lambda_i(\mathcal{K}) \leq \exp(-Ci^{\frac{1}{r}})$ for Gaussian kernels (Belkin 2018)). Second, it holds true that $\sum_{i\geq 1} |\lambda_i(\mathcal{K}) - \lambda_i(K/d)|_F^2 \propto \frac{1}{n}$, a convergence result under the PAC setting (Tang et al. 2013). Therefore, $\lambda_i(K)$ also approximately decays exponentially. • *(ii)* Combinatorial analysis between gaps and tails. We then leverage a recent analysis (Wu et al. 2020a) showing that when $\lambda_i(K)$ decays fast, it is always possible to find an i^* such that $\lambda_{i^*}(\mathcal{K}) - \lambda_{i^*+1}(\mathcal{K})$ is sufficiently large (R1 & R3 are satisfied) and $\sum_{j\geq i^*} \lambda_j^2(\mathcal{K}) = o(1)$ (R2 is satisfied). Putting all these together leads to the following statement.

Proposition 4.1. *Consider the additive influence model. Let $\kappa(\mathbf{z}_i, \mathbf{z}_j)$ be a Gaussian, inverse multi-quadratic (IMQ) or inner product kernel. Let $n \geq d$ be the number of observations and $\epsilon = \frac{c_0 \log^3 d}{\sqrt{d}}$. Assume that the noise level $\sigma_\xi = O(\sqrt{d})$. Let δ be a tunable parameter (also appeared in Alg. 2 in App. B) such that $\delta^3 = \omega(\epsilon^2)$. There exists an efficient algorithm that outputs $\hat{K}$ such that $\frac{1}{d^2}\|\hat{K} - K\|_F^2 = O(\frac{\epsilon^2}{\delta^3} + \delta^{\frac{4}{5}})(= \tilde{O}(d^{-\Theta(1)}))$.*

We remark that *(i)* the algorithm does not need to know the exact form of κ, so long as it is one of Gaussian, IMQ, or inner product kernels, *(ii)* once K is estimated, an Isomap-flavored algorithm may be used to estimate $\mathbf{z}_i$'s (Li et al. 2017), and *(iii)* knowing $\hat{K}$ (without reconstructing $\mathbf{z}_i$'s) is sufficient for the downstream $g(\cdot)$-learners.

Learning $g(\cdot)$

Here, we explain how prominent machine learning techniques, including neural nets (deep learning), non-parametric methods, and boosting, can be used to learn $g(\cdot)$. These techniques make different functional form assumptions of $g(\cdot)$, and possess different "iconic" properties: deep learning assumes that $g(\cdot)$ can be represented by a possibly sophisticated neural net and uses stochastic gradient descent to train the model; non-parametric methods learn a Lipschitz-continuous $g(\cdot)$ with statistical guarantees; boosting consolidates forecasts produced from computationally efficient weak learners.

Our setting has a different cost structure: in univariate models, $g(\mathbf{x}_{t,j})$ controls only one response $\hat{\mathbf{y}}_{t,j}$, but here, $g(\mathbf{x}_{t,j})$ impacts all responses $\hat{\mathbf{y}}_{t,i}$, $i \in [d]$, as $\hat{\mathbf{y}}_{t,i} = \sum_j K_{i,j} g(\mathbf{x}_{t,j})$. We generalize ML techniques under the new cost functions, while retaining the iconic properties of each technique.

Algorithm 1 nparam-gEST:

```
    Input $\mathbf{X}$, $\mathbf{Y}$, $\hat{K}$;
    Output $\mu_1$ (estimating other $\mu_i$'s is similar)
 1: procedure NPARAM-GEST($\hat{K}, \mathbf{X}, \mathbf{Y}$)
 2:     for all $t \leftarrow 1$ to $n$ do
 3:         $q_t = \mathrm{Rand}(d)$
 4:         $L_{(t,q_t),j} = \textsc{Map-Regress}(q_t, \hat{K}, \mathbf{X}_{t,:})$
 5:     return $\mu_1 \leftarrow \textsc{FlipSign}\ (q_t, \{\mathbf{y}_t, L_{(t,q_t),j}\}_{t \le n})$
 6: procedure MAP-REGRESS($q_t, \hat{K}, \mathbf{x}_t$)
 7:     Let $L_{(t,q_t),j} = 0$
 8:     for all $k \leftarrow 1$ to $d$ do
 9:         $L_{(t,q_t),j} += \hat{K}_{q_t,k}$ with $j$ s.t. $\mathbf{x}_{t,k} \in \Omega_j$.
10:     return $L_{(t,q_t),j}$
11: procedure FLIPSIGN($q_t, \{\mathbf{y}_t, L_{(t,q_t),j}\}_{t \le n}$)
12:     for all $t \leftarrow 1$ to $n$ do
13:         $\hat{\Pi}_1^{(q_t)}(t) \triangleq L_{(t,q_t),1} - \frac{1}{\ell-1}\left(\sum_{j \ne 1} L_{(t,q_t),j}\right)$
14:         $\tilde{b}_{t,q_t} = \begin{cases} 1 & \text{if } \hat{\Pi}_1^{(q_t)}(t) \ge \frac{c}{\log d}\sqrt{\frac{d}{\ell}} \\ -1 & \text{if } \hat{\Pi}_1^{(q_t)}(t) < -\frac{c}{\log d}\sqrt{\frac{d}{\ell}} \\ 0 & \text{otherwise} \end{cases}$
15:     return $\mu_1 = \frac{\sum_{t \le n} \tilde{b}_{t,q_t} \mathbf{y}_{t,q_t}}{\sum_{t \le n} \tilde{b}_{t,q_t} \hat{\Pi}_1^{(q_t)}(t)}$
```

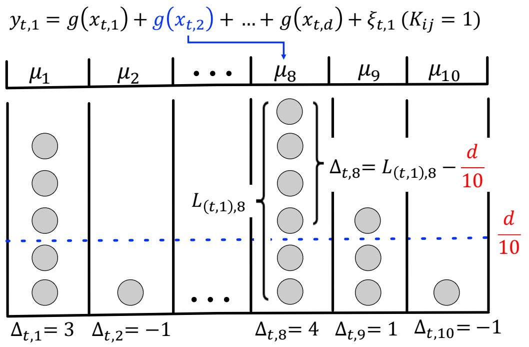


Figure 3: A toy example of nparam-gEST when $K_{i,j} = 1$ for all i and j and $\Omega = [-1, 1]$ and is uniformly partitioned into 10 pieces. Sampling a $g(\mathbf{x}_{t,i})$ corresponds to randomly placing a ball into a total number of 10 bins. For example, $\mathbf{x}_{t,2}$ falls into the 8-th interval so μ_8 is used to approximate $g(\mathbf{x}_{t,2})$, which may be viewed as a new ball of type μ_8 (or in 8-th bin) is created. The mean load for each bin is $d/\ell = d/10$. We calculate $\sum_{i \le d} g(\mathbf{x}_{t,i})$ by counting the balls in each bin: $\mathbf{y}_{t,1} = 5 \times \mu_1 + 1 \times \mu_2 + \dots + 6 \times \mu_8 + 3 \times \mu_9 + 1 \times \mu_{10} + \xi_{t,1}$.

Technique 1. Learn $g(\cdot)$ using neural nets. When an estimate $\hat{K}$ is given, the training cost is $\sum_{t,i}(\mathbf{y}_{t,i} - \sum_{j \in [d]} \hat{K}_{i,j} g(\mathbf{x}_{t,j}))^2$, in which case one can employ stochastic gradient descent when $g(\cdot)$ is a neural net.

Technique 2. Learn $g(\cdot)$ using non-parametric methods. When the response is univariate, e.g., $\mathbf{y}_{t,i} = g(\mathbf{x}_{t,i}) + \xi_{t,i}$, we can use a neighbor-based approach to estimate $g(\mathbf{x})$ for a new $\mathbf{x}$: we identify one (or multiple) $\mathbf{x}_{t,i}$'s in the training set that are close to the new $\mathbf{x}$, and output $\mathbf{y}_{t,i}$ (or their averages, when multiple $\mathbf{x}_{t,i}$ are chosen), using $g(\mathbf{x}) \approx g(\mathbf{x}_{t,i})$, whenever $\mathbf{x}$ is close to $\mathbf{x}_{t,i}$.

Here, we do not directly observe the values of individual $g(\mathbf{x}_{t,i})$'s. Instead, each response is a linear combination of multiple $g(\cdot)$'s evaluated at different points, e.g., $\mathbf{y}_{t,1} = K_{i,1} \cdot g(\mathbf{x}_{t,1}) + \cdots + K_{i,d} \cdot g(\mathbf{x}_{t,d}) + \xi_{t,i}$. We show that finding neighbors reduces to solving a linear system. Furthermore, we design a moment-based algorithm, namely "nparam-gEST", which estimates $g(\cdot)$ with provable guarantees, as summarized in the following result.

Proposition 4.2. *Consider the problem of learning an additive influence model with the same setup/parameters as in Prop. 4.1. Assume that $\mathbf{x}_{t,i} \in \mathbf{R}^{O(1)}$. Let ℓ be a tunable parameter. There exists an efficient algorithm to compute $\hat{g}(\cdot)$, based on $\hat{K}$ such that $\sup_{\mathbf{x}} |\hat{g}(\mathbf{x}) - g(\mathbf{x})| \le (\log^6 n)\left(\sqrt{\gamma} + \sqrt{\frac{\ell}{n}} + \frac{1}{\ell}\right) = \tilde{O}(d^{-c})$ for suitable parameters, where $\gamma \triangleq \frac{\epsilon^2}{\delta^3} + \delta^{\frac{4}{5}}$.*

Our algorithm (Alg. 1) consists of the following 3 steps:
Step 1. Approximation of $g(\cdot)$. Partition $\Omega = [-1, 1]^k$ into subsets $\{\Omega_j\}_{j \le \ell}$, and use piece-wise constant function to approximate $g(\cdot)$, i.e., $\tilde{g}(\mathbf{x}_{t,i})$ takes the same value for all $\mathbf{x}_{t,i}$ in the same Ω_j. We partition $\{\Omega_j\}_{j \le \ell}$ in a way such that $\Pr[\mathbf{x}_{t,i} \in \Omega_j]$ are the same for all j.
Step 2. Reduction to linear regression. Each observation can be construed as a linear combination of μ_j's ($j \in [\ell]$), where $\mu_j = \mathbb{E}[g(\mathbf{x}_{t,i}) \mid \mathbf{x}_{t,i} \in \Omega_j]$. For example, $\mathbf{y}_{t,1} = \sum_{i \le d} K_{1,i}\mu_{j_i} + \xi_{t,1} + o(1)$, where $\mathbf{x}_{t,i} \in \Omega_{j_i}$, and in general, we have

$$\mathbf{y}_{t,i} = \sum_{j \le \ell} L_{(t,i),j}\mu_j + \xi_{t,i} + o(1), \tag{2}$$

where $L_{(t,i),j} = \sum_{m \in \mathcal{L}_{t,j}} K_{i,m}$ and $\mathcal{L}_{t,j} = \{m : \mathbf{x}_{t,m} \in \Omega_j\}$.

Therefore, our learning problem reduces to a linear regression problem, in which the $L_{(t,i),j}$'s are features and the $\{\mu_j\}_{j \le \ell}$ are coefficients to be learned.

Step 3. Moment-based estimation. An MSE-based estimator is consistent but finding its confidence interval (error bound) requires knowing the spectrum of the features' covariance matrix, which is remarkably difficult in our setting. Therefore, we propose a moment-based algorithm with provable performance (FLIPSIGN in Alg. 1).

We illustrate each steps above through a toy example, in which we assume $K_{i,j} = 1$ for all i and j so the model simplifies to $\mathbf{y}_{t,1} = \sum_{j \le d} g(\mathbf{x}_{t,j}) + \xi_{t,1}$. See Fig. 3 for additional details.

Steps 1 & 2. First, we view the generation of samples as a balls-and-bins process so that the $g(\cdot)$-estimation problem reduces to a regression problem (Steps 1 & 2). Specifically, we generate $(\mathbf{y}_{t,1}, \{\mathbf{x}_{t,i}\}_{i \le d})$ as first sequentially sampling $\{\mathbf{x}_{t,i}\}_{i \le d}$ and computing the corresponding $g(\mathbf{x}_{t,i})$, then summing each term up together with $\xi_{t,1}$ to produce $\mathbf{y}_{t,1}$. When an $\mathbf{x}_{t,i}$ is sampled, it falls into one of Ω_i's with uniform probability. Let j_i be the bin that $\mathbf{x}_{t,i}$ falls into. Then $g(\mathbf{x}_{t,i})$ is approximated by μ_{j_i} according to Step 1. Thus, we may view a ball of "type μ_{j_i}" (or in j_i-th bin) is created. For example, in Fig. 3, $\mathbf{x}_{t,2}$ falls into the 8-th interval so a ball is added in the 8-th bin. After all $\mathbf{x}_{t,i}$'s are sampled, compute $\mathbf{y}_{t,1}$ by counting the numbers of balls in different bins. Recalling that the load of j-th bin is $L_{(t,1),j}$, we have $\mathbf{y}_{t,1} \approx \sum_{j \leq d} L_{(t,1),j} \cdot \mu_j + \xi_{t,1}$. Let $\Delta_{t,j} = L_{(t,1),j} - d/\ell$ and using that $\mathbb{E}[L_{(t,1),j}] = d/\ell$ and $\sum_{j \leq d} \mu_j = 0$, we have

$$\mathbf{y}_{t,1} = \Delta_{t,1}\mu_1 + \cdots + \Delta_{t,\ell}\mu_\ell + \xi_{t,1}. \tag{3}$$

Eq. (3) is a standard (univariate) regression: for each t, we know $\mathbf{y}_{t,1}$, and know all $\Delta_{t,j}$'s because all $\mathbf{x}_{t,j}$'s are observed so the number of balls in each bin can be calculated. We need to estimate the unknown μ_j's. Note that $\mathbb{E}[\Delta_{t,j}] = 0$.

Steps 3. We solve the regression (Step 3). Our algorithm "tweaks" the observations so that the features associated with μ_1 are always positive: let $b_{t,1} = 1$ if $\Delta_{t,1} > 0$ and -1 otherwise. Multiply $b_{t,1}$ to both sides of Eq. (3) for each t,

$$b_{t,1}\mathbf{y}_{t,1} = |\Delta_{t,1}|\mu_1 + \cdots + b_{t,1} \cdot \Delta_{t,\ell} \cdot \mu_\ell + b_{t,1}\xi_{t,1}. \tag{4}$$

We sum up the LHS and RHS of (4) and obtain

$$\sum_{t \leq n} b_{t,1}\mathbf{y}_{t,1} = \big(\sum_{t \leq n} |\Delta_{t,1}|\big)\mu_1 + \cdots + \big(\sum_{t \leq n} b_{t,1} \cdot \Delta_{t,\ell}\big)\mu_\ell + \sum_{t \leq n} b_{t,1}\xi_{t,1}. \tag{5}$$

Next, we have $\sum_{t \leq n} |\Delta_{t,1}| = \Theta(n)$ whp. Also, we can see that $b_{t,1}$ and $\Delta_{t,j}$ are "roughly" independent for $j \neq 1$ (careful analysis will make it rigorous). Therefore, for any $j \neq 1$, $\mathbb{E}[b_{t,1} \cdot \Delta_{t,j}] = 0$, and thus $\sum_{t \leq n} b_{t,1} \cdot \Delta_{t,j} = O(\sqrt{n})$ whp. Now (5) becomes $\sum_{t \leq n} b_{t,1} \cdot \mathbf{y}_{t,1} = \big(\sum_t |\Delta_{t,1}|\big)\mu_1 + O(\ell \cdot \sqrt{n})$. Thus our estimator is $\hat{\mu}_1 \triangleq \frac{\sum_t b_{t,1} \cdot \mathbf{y}_{t,1}}{\left(\sum_t |\Delta_{t,1}|\right)} = \mu_1 + \frac{O(\ell \cdot \sqrt{n})}{\Theta(n)} = \mu_1 + O\big(\frac{\ell}{\sqrt{n}}\big)$. Here, the covariance analysis for the $\Delta_{t,j}$'s is circumvented because $\Delta_{t,j}$'s interactions are compressed into the term $O\big(\frac{\ell}{\sqrt{n}}\big)$. We remark that the above analysis contains some crude steps and can be tightened up, as we have done in App. C.

Technique 3. Learn $g(\cdot)$ using boosting. In the univariate setting, we have $\mathbf{y}_{t,i} = \sum_{m \leq b} g_m(\mathbf{x}_{t,i}) + \xi_{t,i}$, in which each $g_m(\mathbf{x}_{t,i})$ is a weak learner. Standard boosting algorithms, such as (Quinlan 1986; Chen and Guestrin 2016), assume that each $g_m(\cdot)$ is represented by a regression tree and constructed sequentially. A greedy strategy is used to build a new tree, e.g., iteratively splitting a node in a tree by choosing a variable that optimizes prediction improvement. In our setting, $\mathbf{y}_{t,i}$ depends on evaluating $g_m(\cdot)$ at d different locations $\mathbf{x}_{t,1}, \ldots, \mathbf{x}_{t,d}$, so the splitting procedure either is d (e.g. 3000 for equity market) times slower in a standard implementation, or requires excessive engineering tweak of existing systems.

Here, we propose a simple and effective weak learner based on the intuition of the tree structure. Let

$$(\mathbf{x}_t)_i = \big((\mathbf{x}_{t,1})_i, (\mathbf{x}_{t,2})_i, \ldots, (\mathbf{x}_{t,d})_i\big) \in \mathbf{R}^d,$$
$$(\mathbf{x}_t)_{i,j} = \big((\mathbf{x}_{t,1})_i \cdot (\mathbf{x}_{t,1})_j, \ldots, (\mathbf{x}_{t,d})_i \cdot (\mathbf{x}_{t,d})_j\big) \in \mathbf{R}^d,$$

and $(\mathbf{x}_t)_{i,j,k}$ can be defined in a similar manner. We observe that regression trees used in GBRT models for equity return are usually shallow and can be *linearized*: we may unfold a tree into disjunctive normal form (DNF) (Abasi, Bshouty, and Mazzawi 2014), and approximate the DNF by a sum of multiple interaction terms, e.g., $I((\mathbf{x}_{t,i})_1 > 0) \cdot I((\mathbf{x}_{t,i})_2 > 0)$ can be approximated by $(\mathbf{x}_{t,i})_1 \cdot (\mathbf{x}_{t,i})_2$.

Our algorithm, namely LIN-PVEL (linear projected vector ensemble learner), consists of weak learners in linear forms. Each linear learner consists of a subset of features and their interactions. The number of features included and the depth of their interactions are hyper-parameters corresponding to the depth of the decision tree. For example, if the first three features are included in the learner, we need to fit $\mathbf{y}_{t,i}$ against

$$\sum_{j \in [d]} \underbrace{\hat{K}_{i,j}}_{\text{given}} \cdot \Big[\underbrace{\beta_1(\mathbf{x}_{t,j})_1 + \ldots}_{\text{linear terms}} \underbrace{+\beta_4(\mathbf{x}_{t,j})_{1,2} + \cdots + \beta_7(\mathbf{x}_{t,j})_{1,2,3}}_{\text{interaction terms}}\Big], \tag{6}$$

by MSE. Conceptually, although we use linearized models to approximate the trees, the "target" trees are unavailable (for the computational efficiency reasons above). We need a new procedure to select features for each learner. Our intuition is that, if an interaction term could have predictive power, each feature involved in the interaction should also have predictive power. Our procedure is simply to select a fixed number of i's with the largest $\mathrm{corr}((\mathbf{y}_{\mathrm{Res}})_t, \hat{K}(\mathbf{x}_t)_i)$, where $(\mathbf{y}_{\mathrm{Res}})_t$ is the residual error.

Using feature interactions to approximate DNF ($I((\mathbf{x}_{t,i})_1 > 0) \cdot I((\mathbf{x}_{t,i})_2 > 0) \approx (\mathbf{x}_{t,i})_1 \cdot (\mathbf{x}_{t,i})_2$) may not always be accurate, however, in our setting, linear interaction models often outperform decision trees or DNFs. We believe this occurs because interaction terms are continuous (whereas DNFs are discrete functions), and thus they are more suitable to model smooth changes.

Evaluation

We evaluate our algorithms on two real-world data sets: an equity market to predict stock returns, and a social network data set to predict user popularity, respectively. Additional details and experiments for the equity market and Twitter data sets are in APP. G. We remark that this is a *theoretical* paper; examining the performance on more data sets and baselines is a promising direction for future work.

Models under our framework. We estimate K and $g(\cdot)$ separately. To estimate K, we use both the algorithm discussed in Sec. 4 and other refinements discussed in App. B. To estimate $g(\cdot)$, we use SGD-based algorithms (MLP and LSTM), nparam-gEST, and LIN-PVEL.

Baselines. Our baselines include the commonly used models and domain specific models. *(i) The UMs* include linear, MLP, LSTM, GBRT, and SFM (Zhang, Aggarwal, and Qi 2017). We also implement a "poor man's version" of both LIN-PVEL and nparam-gEST for UM, which assumes that influences from other entities are 0; *(ii) The CEMs* include a standard linear VAR (Negahban and Wainwright 2011), ARRR (Wu et al. 2020a). *(iii) Ad-hoc integration* AlphaStock (Wang et al. 2019), and HAN (Hu, Liu et al. 2018) for the equity data set; Node2Vec (Grover and Leskovec 2016) for the Twitter data set.

Predicting equity returns. We use 10 years of equity data from an emerging market to evaluate our algorithms and focus on predicting the next 5-day returns, for which the last three years are out-of-sample. The test period is substantially

	Universe 800				*Full universe*				Backtesting	
Models	corr	w_corr	t-stat	w_t-stat	corr	w_corr	t-stat	w_t-stat	PnL	Sharpe
Ours: LIN-PVEL	**0.0764**	**0.0936**	**6.7939**	**6.3362**	**0.0944**	**0.1009**	**8.2607**	**6.4435**	**0.5261**	**10.97**
Ours: nparam-gEST	**0.0446**	**0.0320**	**3.2961**	**1.5753**	**0.0618**	**0.0553**	**5.7327**	**3.5212**	**0.3386**	**7.59**
Ours: MLP	**0.0550**	**0.0567**	**6.4782**	**5.0172**	**0.0738**	**0.0692**	**9.2034**	**6.4151**	**0.4202**	**9.43**
Ours: LSTM	**0.0286**	**0.0347**	**3.4517**	**3.0261**	**0.0473**	**0.0491**	**6.3615**	**4.2385**	**0.2487**	**7.10**
UM: poor man Lin-PVEL	0.0674	0.0866	6.0947	5.7312	0.0827	0.0884	7.4297	5.6659	0.4565	9.76
UM: poor man nparam-gEST	0.0432	0.0309	3.1505	1.4912	0.0584	0.0509	5.0098	3.0844	0.3070	6.59
UM: MLP	0.0507	0.5050	6.0234	4.4966	0.0606	0.0467	8.2857	4.4555	0.2782	6.38
UM: LSTM	0.0178	0.0200	2.2136	1.8077	0.0352	0.0297	4.0602	2.3619	0.175	4.33
UM: Linear models	0.0106	0.0192	1.6471	2.3030	0.0290	0.0251	4.4711	2.6010	0.1888	4.79
UM: GBRT	0.0516	0.0591	7.5739	5.6310	0.0673	0.0747	9.3379	7.8931	0.3858	4.45
UM: SFM	0.0027	0.0032	0.4688	0.4050	0.0147	0.0051	1.2683	0.3892	0.0169	0.54
Existing CEM: VR	0.0156	0.0159	2.4997	1.7046	0.0041	-0.0025	0.8847	-0.3021	0.0430	1.20
Existing CEM: ARRR	0.0314	0.0382	2.5336	2.4213	0.0222	0.0273	1.8557	1.8968	0.1674	3.24
Ad-hoc integration: AlphaStock	0.0085	0.0063	2.1045	1.2516	0.0027	0.0032	0.4688	0.4050	0.0045	0.10
Ad-hoc integration: HAN	0.0105	0.0081	1.7992	1.0017	0.0080	0.0050	1.5716	0.7340	0.0570	2.02
Consolidated: All Ours	**0.0775**	**0.0950**	**6.8687**	**6.4108**	**0.0958**	**0.1025**	**8.5703**	**6.6487**	**0.5346**	**11.30**

Table 1: Summary of results for equity raw return forecasts. LIN-PVEL is the gradient boosting method with the linear learner. Boldface denotes the best performance in each group. Backtesting results pertain to the *Full universe*.

longer than those employed in recent works (Zhang, Aggarwal, and Qi 2017; Hu, Liu et al. 2018; Li et al. 2019), adding to the robustness of our results. We constructed 337 standard technical factors to serve as a feature database for all models. We consider two universes: (i) *Universe 800* can be construed as an equivalence to the S&P 500 in the US, and consists of 800 stocks, and (ii) *Full universe* consists of all stocks except for the very illiquid ones. Visualizations are shown in App. G.

We next describe our evaluation metrics and argue why they are more suitable and different from those employed in standard ML problems (see App. G) • *(i) Correlation vs MSE.* While the MSE is a standard metric for regression problems, correlations are better-suited metrics for equity data sets (Zhou and Jain 2014). • *(ii) Significance testing.* The use of t-statistics estimators (Newey and West 1986) can account for the serial and cross-sectional correlations (App. G) • *(iii) Stock capacity/liquidity considerations.* Predicting illiquid stocks is less valuable compared to predicting liquid ones because they cannot be used to build large portfolios. We use a standard approach to weight correlations (w_corr) and t-statistics by a function of historical *notional (dollar) traded volume* to reflect the capacity of the signals.

Results. See Table 1 for the results and the simulated Profit & Loss (PnL). The experiments confirm that • *(i)* Models under our framework consistently outperform prior works. In addition, our LIN-PVEL model has the best performance; • *(ii)* By using a simple consolidation algorithm, the aggregated signal outperforms all individual ones. Our new models pick up signals that are orthogonal to existing ones because we rely on a new mechanism to use stock and feature interactions.

Predicting user popularity in social networks. We use a Twitter data set to build models for predicting a user's *next 1-day* popularity, defined as the sum of retweets, quotes, and replies received by the user. We collected 15 months of Twitter data streams related to US politics. In total, there are 804 million tweets and 19 million distinct users. User u has one interaction if and only if he or she is retweeted/replied/quoted by another user v. Due to the massive scale, we extract the subset of 2000 users with the most interactions, for evaluation purposes. For each user, we compute his/her daily popularity for 5 days prior to day t as the features.

Results. We report the MSE and correlation for both *in-sample* and *out-of-sample* in Table 2. We observe the consistent results with equity return experiments: *(i)* Methods under our framework achieve better performance in out-of-sample MSE and correlation, with LIN-PVEL attaining the overall best performance. *(ii)* Our methods yield the best generalization error by having a much smaller gap between training and test metrics.

Models	MSE (in)	MSE (out)	Corr (in)	Corr (out)
Ours: Lin-PVEL	0.472	**0.520**	0.733	**0.712**
Ours: nparam-gEST	0.492	**0.559**	0.688	**0.658**
Ours: MLP	0.486	**0.547**	0.716	**0.692**
Ours: LSTM	0.484	**0.541**	0.724	**0.703**
UM: Poor man Lin-PVEL	0.488	0.552	0.710	0.684
UM: Poor man nparam-gEST	0.544	0.584	0.634	0.605
UM: Poor man MLP	0.506	0.562	0.703	0.673
UM: Poor man LSTM	0.496	0.559	0.710	0.679
UM: Linear models	0.616	0.663	0.618	0.592
UM: Random forest	0.611	0.659	0.623	0.587
UM: Xgboost	0.530	0.571	0.671	0.647
CEM: VR	0.540	0.729	0.649	0.408
CEM: ARRR	0.564	0.652	0.610	0.573
Ad-hoc: Node2Vec	0.537	0.690	0.693	0.468
Consolidated: All Ours	**0.459**	**0.502**	**0.767**	**0.742**

Table 2: Overall in-sample and out-of-sample performance on the Twitter data set. Boldface denotes the best performance in each group.

Conclusion

This paper revisits the problem of building machine learning algorithms that involve interactions between entities. We propose an *additive influence framework* that enables us to decouple the learning of the entity-interactions from the learning of feature-response interactions. Our upstream entity interaction learner has provable performance guarantees, whereas our downstream $g(\cdot)$-learners can leverage a wide set of effective ML techniques. All these methods under our framework are proven to be superior to the existing baselines.

Acknowledgement

We thank anonymous reviewers for helpful comments and suggestions. Jian Li was supported in part by the National Natural Science Foundation of China Grant 62161146004, Turing AI Institute of Nanjing and Xi'an Institute for Interdisciplinary Information Core Technology. Yanhua Li was supported in part by NSF grants IIS-1942680 (CAREER), CNS-1952085, CMMI- 1831140, and DGE-2021871. Zhenming Liu and Qiong Wu were supported by NSF grants NSF-2008557, NSF-1835821, and NSF-1755769.