Machine Learning For Finance - Elements

Robin D. Wiseman

January 9, 2022

Contents

Abstract

Notes in primer/review form on various topics for practical machine learning in finance.
Accompanying python code at: https://github.com/vorosbegy/finML/tree/main/src

Chapter 1

Introduction

The purpose of this document is to explore a range of topics related to practical machine learning in finance. The field is vast and rapidly growing, leveraging the best of financial probabilistic and statistical techniques that have been developed over many decades, alongside new machine learning and data science techniques originally developed exogenously to finance as well as directly with finance as the prime focus.

No single primer can hope to cover the full range of methods and potential applications. What follows highlights some relevant areas with theory, examples and accompanying code.

Chapter 1: considers the broad problem of classifier performance, and highlights a much under-appreciated development concerning a coherent measure for assessing performance

Chapter 2: considers the general topic of information and how it relates to establishing useful relationships between financial variables

Chapter 3: considers aspects of representations of data and inductive bias in machine learning models

Chapter 2

Classifiers

2.1 Assessing the performance of classifiers : H-Measure

Supervised binary classification problems arise in many domains of machine learning and finance is no exception. The canonical binary problem is to create a model that, given some set of features x, for each observation can accurately assign the observation to one of two classes, labelled 0 and 1. It is assumed that there is a set of observations and features on which the model can be trained, and for which the classes are known. At runtime, the model will then be presented with features for new observations and must assign each observation to a class.

The range of model types that can potentially be employed to discriminate between the classes is very broad and many possible models might perform better or worse for a given problem. The challenge then is to decide which model is better for the problem at hand and also, within a given class of model types find the best parameterisation.

In order to assess model performance, a very commonly used performance metric is the *Area Under the ROC Curve* (AUC). Unfortunately, despite being very widely used, *AUC is not a coherent metric for comparing models* (or even comparing different parameterisations of the same model).

[Hand2009] was first to highlight the problem and provides a very clear exposition as well as a carefully constructed alternative that resolves the incoherence: the semi-eponymous *H-Measure*. Despite having been created in 2009, the H-Measure is not widely known in the machine learning / data science community. The measure and the associated concepts concerning cost distributions deserve to be much more commonly understood, and should form part of the standard arsenal of tools for model development.

The following sections provide a brief summary of the incoherence of the AUC metric and the coherent alternative, the H-Measure, as well as illustrative examples.

A python implementation of the H-Measure and the examples discussed below can be found at:
https://github.com/vorosbegy/finML/tree/main/src/hmeasure

2.1.1 H-measure

The modelling framework for classifier assessment is as follows. It is assumed that there are two classes, labelled 0 and 1. Given a set of features, x, for an observation, the model under consideration produces a score $s(x)$: some, in general, unspecified monotonically increasing transformation of an estimate $\hat{p}(1|x)$ of the probability that the observation with features, x belongs to class 1.

For each class the scores have a different probability density: $f_k(s)$ for class $k = 0, 1$. If they didn't (if they had the same probability density), then, given a new score, there would be no way to tell which class it belonged to. The whole point of the classification model is to try to create distinct probability densities, $f_k(s)$, to enable partitioning the scores between classes as cleanly as possible. A perfect model is one that generates score probability densities that lie on knowably disjoint domains of the score. In general any given model will fail to be able to perfectly partition the score probabilitiy densities, so that there is some ambiguity about which class a given observation with score lying in the overlap region belongs to, and a selection rule must be applied which will generally get some classifications wrong. The corresponding cumulative distribution functions are denoted $F_k(s)$.

Defining class 0 to be *cases* and class 1 to be *non-cases* and assuming the models in question attempt to provide higher score probability density $f_0(s)$ at *lower* scores for class 0, and higher $f_1(s)$ at *higher* scores for

class 1, a classification rule is, for a given threshold, t, to assign a new observation to class 0 if $s(x) \leq t$ and to class 1 if $s(x) > t$.

The proportion of correctly classified *cases* is then given by $F_0(t)$ and commonly referred to as *sensitivity* (or *recall* or *true positive rate*). The proportion of correctly classified *non-cases* is given by $1 - F_1(t)$ and is commonly referred to as *specificity* (or *true negative rate*). The *ROC curve* is then defined as the plot of $F_0(t)$ on the vertical axis against $F_1(t)$ on the horizontal axis as t is varied.

A model that perfectly partitions $f_0(s)$ from $f_1(s)$, with all class 0 scores lying at lower values in the score domain than any of the class 1 scores, would have a ROC curve that on the $(F_1(t), F_0(t))$ plot shoots up from $(0, 0)$ to $(0, 1)$ and then runs across horizontally from $(0, 1)$ to $(1, 1)$ as t is varied from the low end of the s domain to the high end - i.e. all the class 0 observations are correctly classified before any class 1 non-cases are encountered.

The related measure that is then very widely used to assess the relative quality of different models on the sample population is the *Area Under the ROC Curve* which is defined as:

$$AUC := \int_{-\infty}^{\infty} F_0(s) f_1(s) ds \tag{2.1}$$

At the heart of the finding that AUC is an incoherent measure for comparing models is an examination of the relative cost of the model being wrong in classifying *cases* versus being wrong in classifying *non-cases*. [Hand2009] insightfully proceeds to consider what implicit misclassification cost distribution is assumed when using the AUC metric. The next few paragraphs briefly summarise the approach and finding.

Taking the cost of correct classification to be zero, and the cost of misclassifying a class k point to be $c_k \in [0, \infty], k = 0, 1$, at a classification threshold, t, the overall misclassification cost is:

$$c_0 \pi_0 (1 - F_0(t)) + c_1 \pi_1 F_1(t) \tag{2.2}$$

where π_0, π_1 are the class prior population probabilities (i.e. the probability that a randomly picked observation, with no additional information, is in class 0 or class 1 respectively).

The threshold that minimises the misclassification cost is defined as:

$$T(c_0, c_1) := \underset{t}{\operatorname{argmin}} \{c_0 \pi_0 (1 - F_0(t)) + c_1 \pi_1 F_1(t)\} \tag{2.3}$$

Making the convenient change of variables: $b := (c_0 + c_1)$ and $c := (1 + \frac{c_1}{c_0})^{-1}$, T can be written in terms of c alone:

$$T(c) = \underset{t}{\operatorname{argmin}} \{c \pi_0 (1 - F_0(t)) + (1 - c) \pi_1 F_1(t)\} \tag{2.4a}$$

$$Q(t; b, c) := \{c \pi_0 (1 - F_0(t)) + (1 - c) \pi_1 F_1(t)\} b \tag{2.4b}$$

with Q defined as the loss at arbitrary threshold t.

For differentiable score distributions, the minimising T satisfies:

$$c \pi_0 f_0(T) = (1 - c) \pi_1 f_1(T) \quad \text{or equivalently:} \tag{2.5a}$$

$$c = \frac{\pi_1 f_1(T)}{\pi_0 f_0(T) + \pi_1 f_1(T)} = \mathbb{P}(1|T) := P_1(T) \tag{2.5b}$$

where $\mathbb{P}(1|T)$ is the probability of an observation belonging to class 1 given the minimising score T and $P_1(T)$ is convenient notation, writing in functional form, with the inverse relationship $T = P_1^{-1}(c)$.

As an important next step in the exposition [Hand2009] highlights that b, c are exogenous values that depend on the underlying domain problem and which themselves (as for c_0, c_1) will generally have some related uncertainty and their own associated joint probability distribution, $\nu(b, c)$. [Hand2009] goes on to consider the overall expected minimum loss in terms of this distribution:

$$L := \int_0^1 \int_0^\infty Q(T(c); b, c) \nu(b, c) db dc \tag{2.6a}$$

$$= \int_{-\infty}^{\infty} \{c(T) \pi_0 (1 - F_0(T)) + (1 - c(T)) \pi_1 F_1(T)\} W(T) dT \tag{2.6b}$$

where $w(c) := \int b\nu(b,c)db$ followed by a change of variables $c \to T$ and $w(c) \to W(T)$. The functions $w(c)$ and $W(T)$ are seen to be domain-specific (subjective or otherwise) weighting functions of the cost (or equivalently cost-minimising threshold).

A very particular choice of $W(T)$ is then considered:

$$W(T) = W_\gamma(T) := \pi_0 f_0(T) + \pi_1 f_1(T) \tag{2.7}$$

[Hand2009] demonstrates that under this specific choice for $W(T)$:

$$L|W_\gamma := 2\pi_0\pi_1(1 - AUC) \tag{2.8}$$

This is quite a dramatic result. A corollary is that:

$$w(c) = w_\gamma(c) := \pi_0 f_0(P_1^{-1}(c)) \left|\frac{dP_1^{-1}(c)}{dc}\right| + \pi_1 f_1(P_1^{-1}(c)) \left|\frac{dP_1^{-1}(c)}{dc}\right| \tag{2.9}$$

The reason this is dramatic is because it is saying that under AUC, the cost weight distribution varies from classifier model to classifier model. *The implicit AUC cost distribution is a function of f_0 and f_1 which are dependent on the model that is being evaluated.* Of course, this makes no sense: *the beliefs of likely values of c must be independent from the model that is being used*, but AUC is precisely not doing that - it is implicitly using different cost distributions for different models - rendering comparison of models using AUC useless.

As a coherent alternative, [Hand2009] builds on the properties of the expected minimum loss, and defines the coherent H-Measure metric as:

$$H := 1 - \frac{L}{L_{max}} = 1 - \frac{\int Q(T(c); b, c)u(c)dc}{\pi_0 \int_0^{\pi_1} cu(c)dc + \pi_1 \int_{\pi_1}^{1} (1-c)u(c)dc} \tag{2.10a}$$

This has the propery of being equal to zero when the model cannot discriminate at all between the two classes, and equal to one when it can discriminate perfectly. It also places the focus on the choice of the cost distribution $u(c)$: recalling $c := (1 + \frac{c_1}{c_0})^{-1}$. As [Hand2009] argues, even if the cost distribution is not known, it is better to be explicit about the choice that is being made, to enable like for like comparison between models on that basis. It also enables proposing a *standard* choice that can be used to enable comparison across different model studies (and in the absence of firm beliefs of the exact form of the cost distribution for the domain problem at hand). The Beta distribution is highlighted as a convenient distribution to work with (c lies in $[0, 1]$ with $c = 0$ when $c_1 \gg c_0$ and $c = 1$ when $c_0 \gg c_1$). The *standard* choice of $u(c) \sim Beta(c|2, 2)$ is proposed, but other choices of α, β in $Beta(c|\alpha, \beta)$ can be made, dependent on the domain beliefs, and giving a flexible range of distributional shapes.

The following section illustrates the performance of the H-Measure under different score distributions and cost distrbutions.

2.1.2 Illustrative Examples

The examples discussed in this section are generated using a python implementation of the H-Measure that can be found at:
https://github.com/vorosbegy/finML/tree/main/src/hmeasure
The section begins with a brief discription of the methodology used to generate the examples.

Methodology

To enable systematic analysis of the performance of the measure independent from any actual classification model, samples from score distributions $f_0(s), f_1(s)$ are generated using Beta distributions in each case:

$$f_0(s) := Beta(s|\alpha_0, \beta_0) \tag{2.11a}$$
$$f_1(s) := Beta(s|\alpha_1, \beta_1) \tag{2.11b}$$

It should be emphasised that the choice of Beta distributions to generate the per-class score samples is purely to work with a convenient and flexible form for purposes of exposition. It does not make any statement about expected score distributions from actual classification models. Neither is the choice of these Beta distributions

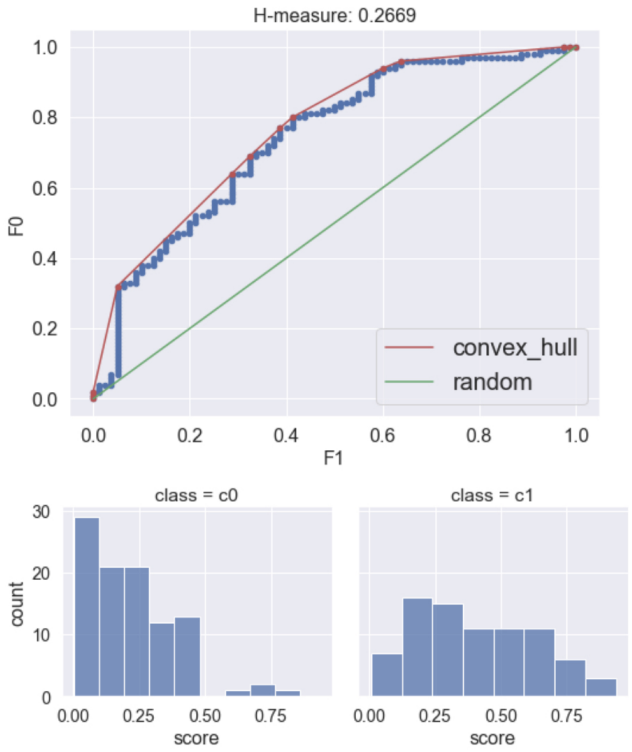

Figure 2.1: H-Measure for a classifier

for the score distributions related to the choice of distribution for the cost distribution, $u(c)$ (which also, unrelatedly, can be chosen to be a Beta distribution).

In each experiment:

(1) a large number of score samples is drawn for each class

(2) the ROC curve and convex hull of the ROC curve are generated from the samples: using the sample-implied cumulative distributions constructed from the sample points (not with knowledge of the actual sample distributions used to generate the data).

(3) a cost distribution u(c) is specified. In some cases this is the *standard* choice $c \sim Beta(c|2,2)$, but in other examples the cost distribution parameters α, β are varied to enable $Beta(c|\alpha,\beta)$ to penalise losses more strongly on class 0 or class 1.

(4) the associated H-Measure is created, and plots of the ROC curve plus input score distributions are provided and discussed.

The Figure 2.1 illustrates one example. In this case the example parameters were:
$(\alpha_0, \beta_0, \alpha_1, \beta_1) = (1.1, 4.0, 2.0, 2.0)$ for the f_0, f_1 samples. Note the score distribution plots at the bottom of the figure showing the class 0 scores more distributed to lower score values (the score values bounded on $[0,1]$ in this case), and the class 1 scores more distributed to higher values, although still with a fairly broad distribution with some significant overlap in the distribution domains. As a result the "model" performance, as quantified by the H-measure, is relatively poor.
In this example, relatively few samples were drawn for each class : 100 and 80 for classes 0, 1 respectively, giving a slight imbalance in population ($\pi_0 = \frac{5}{9}, \pi_1 = \frac{4}{9}$). Additionally, given the relatively smaller overall sample size, the discrete nature of the emprirical ROC curve is more apparent ([Hand2009] carefully introduces the *convex hull* over the ROC curve to handle the discrete nature of the empirical data-driven curve). Figure 2.1 additionally shows the *random* ROC curve (the green line) which pertains if the model is completely unable to discriminate between the two classes.

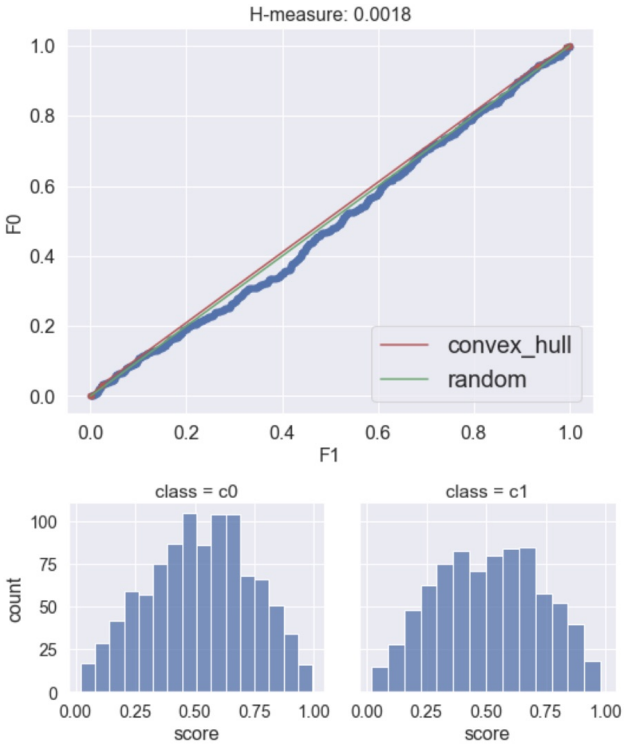

Figure 2.2: $(\alpha_0, \beta_0, \alpha_1, \beta_1) = (2.0, 2.0, 2.0, 2.0) \quad u(c) \sim Beta(c|2.0, 2.0) \quad N_0 = 1000, N_1 = 800$

Examples and Discussion

Case 1
Parameters:
$(\alpha_0, \beta_0, \alpha_1, \beta_1) = (2.0, 2.0, 2.0, 2.0)$
$u(c) \sim Beta(c|2.0, 2.0)$
$N_0 = 1000, N_1 = 800$
Observations:
In this example (Figure 2.2) the score distributions were deliberately chosen to be identical and symmetrically distributed around 0.5 in the $[0, 1]$ domain for the score. A small population imbalance was chosen, favouring class 0: $\pi_0 = \frac{5}{9}, \pi_1 = \frac{4}{9}$. It can be see that as a result of the model being unable to discriminate between the two classes (given they have the same score distribution), the model is essentially random, the ROC curve lies almost on the diagonal as expected, and the H measure approaches zero.

Case 2
Parameters:
$(\alpha_0, \beta_0, \alpha_1, \beta_1) = (2.0, 6.0, 6.0, 2.0)$
$u(c) \sim Beta(c|2.0, 2.0)$
$N_0 = 1000, N_1 = 800$
Observations:
In this example (Figure 2.3) the score distributions were chosen to be skewed with the class 0 distribution skewed towards lower scores and the class 1 distribution skewed towards higher scores. As a consequence it is possible to discriminate much more cleanly between the two classes, and the H-Measure reflects the much improved performance with a value around 0.83.

Case 3
Parameters:
$(\alpha_0, \beta_0, \alpha_1, \beta_1) = (2.0, 6.0, 6.0, 2.0)$
$u(c) \sim Beta(c|12.0, 2.0)$
$N_0 = 1000, N_1 = 800$
Observations:
In this example (Figure 2.4) the parameterization for the score distributions was identical to Case 2, but the cost

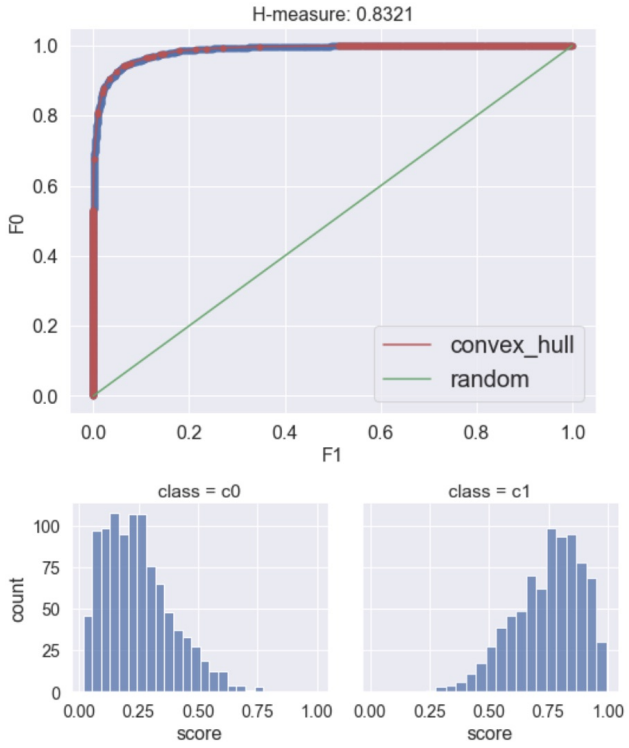

Figure 2.3: $(\alpha_0, \beta_0, \alpha_1, \beta_1) = (2.0, 6.0, 6.0, 2.0) \quad u(c) \sim Beta(c|2.0, 2.0) \quad N_0 = 1000, N_1 = 800$

distribution was skewed towards higher misclassification cost for class 1 than class 0. The H-Measure is seen to worsen relative to Case 2 (0.76 versus 0.83). While there is some variance in the H-Measure (or any sample-based measure for that matter) due to the finite population size, at $N_0 = 1000, N_1 = 800$ the sample variance is relatively smaller than the change in H-measure due to the change in cost distribution. This highlights that during model selection it is necessary to consider, for the given problem domain, what a reasonable range of cost distributions should be, and to consider the relative performance of the candidate models across that range of cost distributions.

2.1.3 Conclusion

From the discussion above and examples, it is clear that a proper assessment of misclassification cost will be important in model selection and hyperparameter tuning of many classification models. AUC should not be used to assess the relative merits of different classification models as it implicitly assumes different classification costs distributions for each model - which clearly makes no sense (the cost of being wrong cannot depend in reality on the tool that is being used to make the classification). The H-Measure is a superior alternative, making explicit the choice of classification cost distribution and enabling models to be assessed in light of the explicit choice.

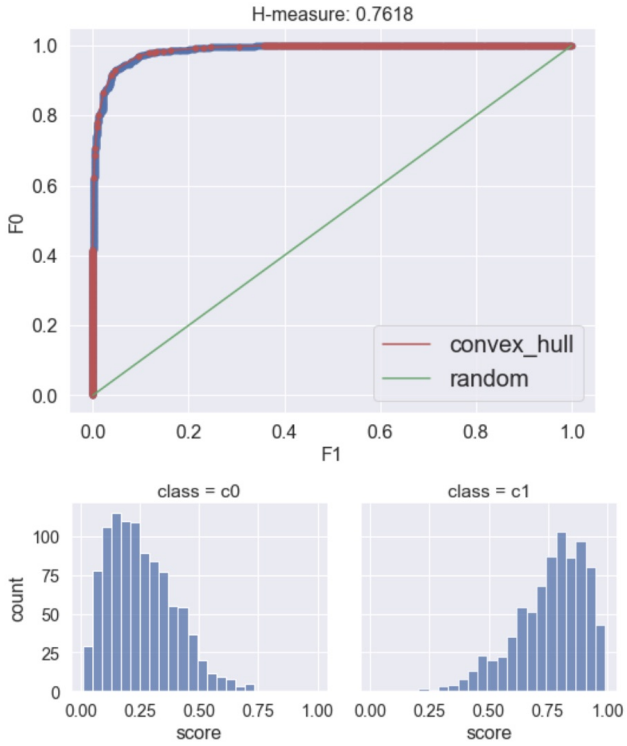

Figure 2.4: $(\alpha_0, \beta_0, \alpha_1, \beta_1) = (2.0, 6.0, 6.0, 2.0) \quad u(c) \sim Beta(c|12.0, 2.0) \quad N_0 = 1000, N_1 = 800$

Chapter 3

Information

Extracting information from data lies at the heart of many problems in Finance, as in other fields of data science and machine learning. The notion of *information* requires some careful thought in order to appropriately quantify what is meant by the term, and how it can be used in practice.

[Caticha2007] provides an elegant discussion of information, and begins with the notions that: *Information is whatever constrains rational beliefs*; and *Information is that which induces a change from one state of belief to another*. The development then continues by proposing three axioms and a related principle in order to decide *how* beliefs should be updated when new information arrives:

Principle of Minimum Updating: *Beliefs should be updated only to the extent required by the new information*
Axiom 1: Locality *Local information has local effects*
Axiom 1: (special case) *When there is no new information, there is no reason to change one's mind*
Axiom 2: Coodinate Invariance *The system of coodinates carries no information*
Axiom 3: Consistency for independent subsystems *When a system is composed of subsystems that are* ***known*** *to be independent it should not matter whether the inference procedure treats them separately or jointly*

From these minimal and reasonable axioms alone, [Caticha2007] proceeds to derive the unique functional form for the relative entropy $S[p,q]$ and the associated *ME method* as a method for updating from a prior probability distribution $q(x)$ to a new probability distribution $p(x)$ when presented with new information in the form of a constraint that specifies the allowed posteriors. The ME method involves maximising the relative entropy, $S[p,q]$:

$$S[p,q] := -\int p(x)\log\left(\frac{p(x)}{q(x)}\right)dx \tag{3.1}$$

subject to the available constraints. Entropy maximisation is therefore seen to be at the heart of how rational beliefs should be updated when presented with new information.

In finance, many situations arise where new information is made available and should be assessed in a rational manner. Examples include:
(1) deciding whether a new asset will be economically useful e.g. to generate a better return/risk ratio in combination with an existing pool of assets
(2) deciding which sources of data/signals will improve the performance of an existing model

[Samo2018] presents a compelling case for the use of *Differential Mutual Information Time* as a measure of the amount of time required to see one additional bit of mutual/shared information between the returns of a new asset under consideration and the returns of an existing reference asset pool. The financial problem considered, is to try to quantify how much diversification the returns of a new asset represented by the discrete time series $\{y_t\}$ can bring to an existing portfolio with returns represented by the discreted time series $\{\mathbf{x}_t\}$.

The exposition is (as might be expected given the introduction above highlighting the intrinsic relationship between information arrival, belief update and entropy) rooted in the use of entropy-related measures. Of several practical implementation methods presented in [Samo2018] for calculating the entropy rate of discrete stochastic stationary time series, an entropy maximisation procedure is one approach. The following section briefly summarises that approach and illustrates the method with some examples. A python implementation

of Differential Mutual Information Time and the examples discussed below can be found at:
https://github.com/vorosbegy/finML/tree/main/src/dmutinft

3.1 Assessing the *Usefulness* of Assets: $\mathbb{D}(A;P)$-measure

3.1.1 $\mathbb{D}(A;P)$-measure

There are various quantities that a new financial asset should possess in order to be *useful* when traded in combination with an existing portfolio of financial assets. [Samo2018] begins by discussing these properties in general terms, before focussing on diversification: the ability for a new asset, when traded in suitable proportion to an existing pool of assets, to reduce the risk of the combined pool for the same level of expected return. The work then carefully establishes and develops a general quantitative measure of diversification. The next few paragraphs summarise the approach and findings.

For a random variable, X, with probability density function, $p(x)$, a natural measure of the information carried by the random variable is its associated entropy:

$$h(X) := -\int p(x)\log_2(p(x))dx \tag{3.2}$$

(expressing the entropy in bits rather than nats). The related quantity, *conditional entropy* measures the amount of information carried by the random variable Y, that is not already available in X:

$$h(Y|X) := h(X,Y) - h(X) = -\int p(x,y)\log_2(p(x,y))dxdy - \int p(x)\log_2(p(x))dx \tag{3.3}$$

[Samo2018] then introduces the *entropy rate* and *conditional entropy rate* respectively as:

$$h(\{\mathbf{x}_t\}) := \lim_{T\to\infty}\frac{1}{T}h(\mathbf{x}_1,\ldots,\mathbf{x}_T) \tag{3.4a}$$

$$h(\{y_t\}|\{\mathbf{x}_t\}) = h(\{y_t\},\{\mathbf{x}_t\}) - h(\{\mathbf{x}_t\}) \tag{3.4b}$$

The financial context under consideration is to try to quantify how much diversification the returns of a new asset represented by the discrete time series $\{y_t\}$ can bring to an existing portfolio with returns represented by the discreted time series $\{\mathbf{x}_t\}$). The quantity $h(\{y_t\}|\{\mathbf{x}_t\})$ is seen to be a relative measure of the amount of information per unit of time contained in $\{y_t\}$ that is not already contained in $\{\mathbf{x}_t\}$. In terms of the entropy rate and conditional entropy rate, the *differential mutual information rate* and its inverse, the *differential mutual information time* are then motivated and introduced:

$$I(\{y_t\};\{\mathbf{x}_t\}) := h(\{y_t\}) - h(\{y_t\}|\{\mathbf{x}_t\}) \tag{3.5a}$$

$$= h(\{y_t\}) + h(\{\mathbf{x}_t\}) - h(\{y_t\},\{\mathbf{x}_t\}) \tag{3.5b}$$

Definition: *Let P be a reference pool of assets and factors, whose time series of returns and factor values are denoted $\{\boldsymbol{x}_t\}$. Let A be an asset not in P, whose time series of returns are denoted $\{y_t\}$. Assuming entropy rates of $\{y_t\}$ and $\{\boldsymbol{x}_t\}$ exist$^{(*)}$, the measure of incremental diversification the asset A adds to the reference pool P is defined to be the differential mutual information time between $\{\boldsymbol{x}_t\}$ and $\{y_t\}$:*

$$\mathbb{D}(A;P) := \frac{1}{I(\{y_t\};\{\mathbf{x}_t\})} \tag{3.6}$$

The differential mutual information rate, $I(\{y_t\};\{\mathbf{x}_t\})$, is always positive and is invariant to any smooth change of variables. It takes larger positive values when $\{y_t\}$ and $\{\mathbf{x}_t\}$ have higher similarity. As its inverse, $\mathbb{D}(A;P)$ by contrast is a measure of the amount of time required to see one additional bit of mutual/shared information between the returns of the new asset $\{y_t\}$ and the existing pool, $\{\mathbf{x}_t\}$. If $I(\{y_t\};\{\mathbf{x}_t\})$ is higher ($\{y_t\}$ and $\{\mathbf{x}_t\}$ are more similar) then $\mathbb{D}(A;P)$ is lower, and conversely, if $\{y_t\}$ and $\{\mathbf{x}_t\}$ are less similar ($\{y_t\}$ provides more diversification), then $\mathbb{D}(A;P)$ is higher.

The technical point $^{(*)}$ highlighted above relates to the existence of the entropy rates. In general, entropy rates are not guaranteed to exist. However, they are guaranteed to exist in the case of stationary stochastic proceses and consequently, a careful consideration of stationarity should be made before proceeding to use

metrics or results based on entropy rates.

A successful implementation of $\mathbb{D}(A;P)$ can be decomposed into calculation of the entropy rates for y_t, $\{\mathbf{x}_t\}$, and their joint entropy rate. Three possible practical procedures are described for calculating the entropy rate of a stationary discrete time stochastic process, including maximum entropy estimation, for which the following theorem is used:

Theorem: *Let $\{\boldsymbol{z}_t\}$ be a stationary $\mathbb{R}^n$-valued discrete-time stochastic process. Among all stationary processes whose (matrix-valued) autocovariance functions coincide with that of $\{\boldsymbol{z}_t\}$ from lag $h = 0$ to lag $h = p$, the mean-zero Gaussian Vector Autoregressive process of order p ($VAR(p)$) has the highest entropy rate and:*

$$h(\{\mathbf{z}_t\}) = \frac{n}{2}\log_2(2\pi e) + \frac{1}{2}\Big[\frac{det(\Sigma_p)}{det(\Sigma_{p-1})}\Big], \tag{3.7}$$

where Σ_p is the block-matrix such that

$$\Sigma_p[i,j] := \mathbb{C}ov(\{\mathbf{z}_{t+i}\},\{\mathbf{z}_{t+j}\}) \tag{3.8a}$$
$$:= C(i-j) \tag{3.8b}$$

and where the sample autocovariance function of the sample path $(\hat{\boldsymbol{z}}_1,\ldots,\hat{\boldsymbol{z}}_T)$ is defined as

$$\hat{C}(h) := \bar{\mathbf{z}}\begin{cases}\frac{1}{T}\Sigma_{t=1+h}^{T}(\hat{\mathbf{z}}_t-\bar{\mathbf{z}})(\hat{\mathbf{z}}_{t-h}-\bar{\mathbf{z}})^\dagger, & if h \geq 0\\ \hat{C}(-h)^\dagger & if h \leq 0\end{cases} \tag{3.9}$$

and $\dagger$ denotes the transpose.

The subsequent section illustrates the $\mathbb{D}(A;P)$ measure using an implementation based on the above maximum entropy estimation.

3.1.2 Illustrative Examples

The examples discussed in this section are generated using a python implementation of Differential Mutual Information Time which can be found at:
https://github.com/vorosbegy/finML/tree/main/src/dmutinft
The section begins with a brief description of the methodology used to generate the examples.

Methodology

In order to illustrate the performance of $\mathbb{D}(A;P)$, sample paths of length $N = 1000$ time steps are created from draws of a multi-variate normal distribution: $\tilde{y}_t \in \mathbb{R}^{N\times 1}$, $\tilde{\mathbf{z}}_t \in \mathbb{R}^{N\times 3}$ where $(\tilde{y}_t,\tilde{\mathbf{z}}_t) \sim N(\tilde{y}_t,\tilde{\mathbf{z}}_t|0,\Sigma_{zy})$ and Σ_{zy} is chosen to be equal to one on the diagonal, ρ_{zy} on the off-diagonal elements between z and y, and ρ_{zz} on the off-diagonal elements between components of $\mathbf{z}$. Care is taken to ensure the resulting correlation matrix is still positive definite for the choices of ρ_{yz},ρ_{zz} considered. The increments $\tilde{y}_t$ and $\tilde{\mathbf{z}}_t$ are iid across time. Each sample path is then cumulatively summed in the time dimension, before fracdiff with order 0.8 is applied to achieve stationarity ([LDP2018]). An additional *non-linear* case is also separately considered, in which the increments $\tilde{y}_t$ are transformed $\tilde{y}_t \to \tilde{y}_t^3$ before the cumulative sum and fracdiff is applied. During estimation of the sample autocorrelation matrix $\hat{C}(h)$ no additional *denoising* was performed ([LDP2020]). In each case considered, 10 separate runs were performed, to enable plotting the variation in $\mathbb{D}(A;P)$ for each choice of time series parameterisation.

Examples and Discussion

Case1
Parameters:
$N = 1000$, $\rho_{zz} = 0.6$, $\rho_{yz} = [0,\ldots,0.8]$
Observations:
In this example (Figure 3.1) as ρ_{yz} is varied across the x-axis from 0 to 0.8, the relationship between y and z becomes stronger. The $\mathbb{D}(y;z)$ measure reflects the increasing strength of relationship, decreasing monotonically as ρ_{yz} is increased. It can also be seen, however, that there is increasing sample variance in the estimation

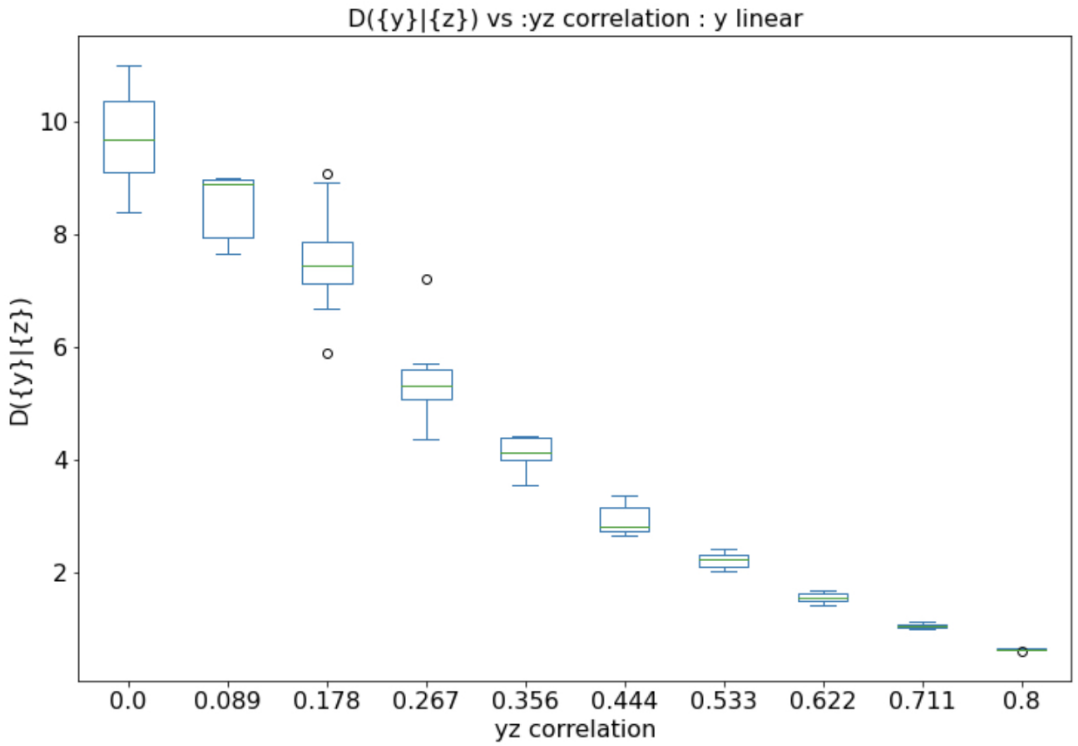

Figure 3.1: $\rho_{zz} = 0.6$ in pre-fracdiff increments

of $\mathbb{D}(y; z)$ as ρ_{yz} is lowered. Given that in general only a single sample path will be observable for real world financial observations, it is possible, for any given sample path, that the true relative ordering of strength of relationship of two assets y and y' with a given pool of assets z might be reversed for assets that have a similar strength i.e. $\tilde{\mathbb{D}}(y; z) < \tilde{\mathbb{D}}(y'; z)$ versus $\mathbb{D}(y; z) > \mathbb{D}(y'; z)$ (where the tilde indicates the sample estimate here versus the true underlying relationship - if it were knowable). That said, the $\mathbb{D}(y; z)$ measure clearly provides a good characterisation of strength of relationship across a broad range.

Case2

Parameters:

$N = 1000$, $\rho_{zz} = 0.6$, $\rho_{yz} = [0, \ldots, 0.8]$, $\tilde{y} \to \tilde{y}^3$ before cumulation and fracdiff.

Observations:

In this example (Figure 3.2) the experimental setup was identical to Case1, save for the transformation of the sampled $\tilde{y}$ increments before performing the cumulative sum and fracdiff step in the time dimension. It can be seen that $\mathbb{D}(y; z)$ once again provides a good measure of the relative strength of relationship across a broad range. The variance at lower ρ_{yz} correlation is observed to be somewhat broader than for the linear Case1, but once again the measure performs well in general across the range.

3.1.3 Conclusion

In finance, it is regularly the case that it is important to be able to quickly assess the strength of relationships between financial variables. Such variables are often the returns of financial assets, and the preceding discussion was motivated by considering portfolio diversification. However, there is a broader class of problems, such as attempting to build a predictive model for one financial variable in terms of other financial (or other economic or real-world) variables where an assessment of the strength of relationship between potential explanatory variables and the target variable is of strong interest. The $\mathbb{D}(y; z)$ measure can be seen to be a principled measure, well-grounded in information theory that enables a quantitative assessment of variable relationship strength in advance of any specific model building procedure.

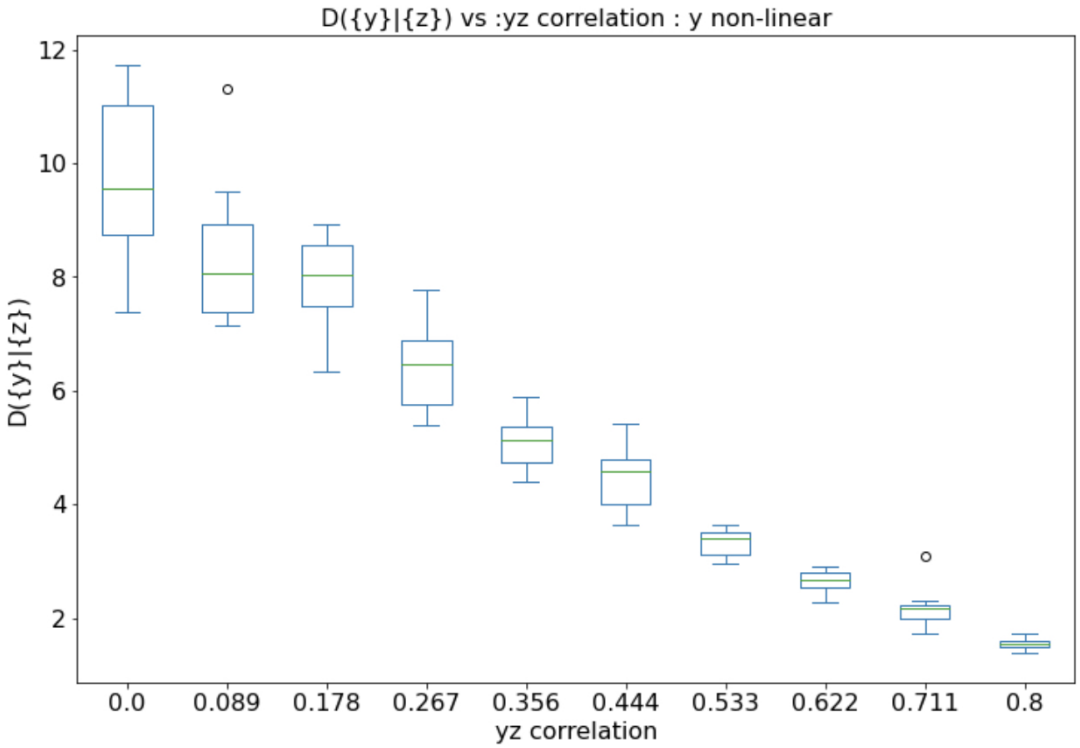

Figure 3.2: $\rho_{zz} = 0.6$, $\tilde{y} \to \tilde{y}^3$ pre-fracdiff transformation

Chapter 4

Representations

In the last decade, the broad use of deep neural networks (DNN) has become commonplace in many fields of human endeavour. In comparison to more traditional statistical learning methods, the increasing use of DNN was accompanied by a move towards *letting the data speak for itself*: meaning, amongst other things, less focus on manual expert human construction of possible features (transformations of potential explanatory variables) for a given problem, and more focus on letting the model (network) discover for itself what combinations, or *representations*, of the data are most useful to it in order to minimize/maximize some target objective function for the problem at hand. New *Universal Approximation Theorems* ([Lu2017], [Gabrielsson2020]) extended earlier work from past decades on the extent to which combinations of layers of neural networks with prescribed width and/or depth could approximate arbitrary target functions.

That said, there is also a parallel and recent body of work that focusses on *inductive bias*: meaning, in this context, endowing models with a specific structure to represent the data that, while not as prescriptive and manual as the earlier decades of human feature creation, still forces some reasonable (and useful) constraints on the form that representations of the data can take. The reason that this is of interest is because, while DNN are, provably, universal function approximators, the theoretical result of universal function approximation says nothing in practice about *how easy* it will be for the model to learn the target function when presented with finite training data.

Early examples of inductive bias in feature representations are founds in the construction of Convolutional Neural Networks (CNN), where the "convolution" layers in the structure act to approximately enable the model to learn translational and rotational invariance when presented with data and a problem that is invariant to translation/rotation (e.g. image classification - where the translation/rotation of the image should not change the conclusion the model reaches on what the image represents).

A further good example (admittedly not in the field of finance) of endowing models with inductive bias is recent work to use the theory of Lie Groups to capture the dynamic symmetry structure of the Hamiltonian of a physical system when constructing neural networks. [Hutchinson2021] demonstrate that by appropriately encapsulating the dynamic symmetry properties of the system in the layer construction, the model is able to learn and predict the particle dynamics of the problem with orders of magnitude error reduction relative to models that are unaware of the symmetry properties, as well as being able to learn from smaller training data sets. An interesting topic for future research is: *what is the Lie Group invariance equivalent for financial dynamics - what inductive biases are most relevant to express the dynamic "symmetries" in finance?*. Given the coodinate invariance properties of differential mutual information rate as discussed in the earlier chapter, it is perhaps an interesting avenue to explore.

Another body of work in recent years focusses on how to represent non-Euclidean data sets such as graphs within DNN. Graph data sets are commonly found in many problem domains, and finance is no exception. For example, learning investor-asset preferences can be formed as a problem of link prediction in a bivariate graph where the nodes are in two flavours (investors or assets) and the links between the nodes express a measure of preference of one for the other. Learning functional causal graphs is another example currently subject to substantial research by many groups ([Xia2021], [Kalainathan2018]).

The following sections discuss one aspect of graph representation learning, illustrating some spectral graph techniques in the context of creating layers for (deep) learning models with inductive bias that captures the

spectral structure of the graph of the underlying problem. The python code for the examples below can be found at:
https://github.com/vorosbegy/finML/tree/main/src/representations

4.1 Leveraging Non-Euclidean Data: Spectral Graph Representations

Spectral graph techniques have long been used to extract informative structural information from graphs. [Luxburg2007], [Spielman2019] provide excellent coverage of many of the important properties that general graphs structures are endowed with. Central to many results is the concept of the *graph Laplacian*. Of many recent works that exploit the graph Laplacian, [Zheng2018] is a good example. The following sections introduce the associated theory, and provide some illustrative examples.

4.1.1 Spectral Graph Representations: Graph Laplacian

For an undirected graph $G = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$ and edge weights $w_{ij} \geq 0$ the weighted *adjacency matrix* of the graph is the matrix $W = (w_{ij})$ with $i, j \in 1, \ldots, n$. The *degree* of a vertex $v_i \in V$ is defined as:

$$d_i := \sum_{j=1}^{n} w_{ij} \tag{4.1}$$

and the *degree matrix* D is defined as the diagonal matrix with $d_1, \ldots, d_n$ on the diagonal. A subset $A \subset V$ of a graph is *connected* if any two vertices in A can be joined by a path such that all intermediate points also lie in A. A subset is called a *connected component* if ti is connected and if there are no connections between vertices in A and $\bar{A}$ where $\bar{A}$ is the the set of vertices in V that are not in A. In terms of these quantities the *symmetric normalised Graph Laplacian* is defined as:

$$L := \mathbf{1} - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \tag{4.2}$$

The Graph Laplacian is a highly structured mathematical object whose properties encapsulate many deep features of the underlying structure of the associated graph. It is symmetric positive semi-definite with non-negative real eigenvalues $0 \leq \lambda_1 \leq \lambda_2 \cdots \leq \lambda_n$, and the multiplicity k of the eigenvalue 0 of L equals the number of connected components $A_1, A_2, \ldots A_k$ in the graph.

Spectral clustering is a technique that leverages the structural properties of the Graph Laplacian in order to cluster points on the graph that are most strongly weighted. [Luxburg2007] provides an excellent review. There are a few variants of the method, but working with the symmetric normalised Graph Laplacian, [Ng2002] introduce:
(i) compute the first k eigenvectors $u_1, \ldots, u_k$ of L
(ii) let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \ldots, u_k$ as columns
(iii) form the matrix $T \in \mathbb{R}^{n \times k}$ from U by row normalising $t_{ij} := u_{ij} / (\sum_k u_{ik}^2)^{1/2}$
(iv) for $i = 1, \ldots, n$ let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i^{th} row of T
(v) cluster the points $(y_i), i = 1, \ldots, n$ with the k-means algorithm into clusters $C_1, \ldots, C_k$

It is worth highlighting the projection into the truncated, normalised row vectors of the eigenvector matrix of L. The method discussed further below, although not directly employing this spectral clustering algorithm, involves training a model on linear combinations of these vectors y_i, enabling the model to *learn* in the spectral space of the underlying graph problem.

Adjacency matrices in finance include, for example, matrices generated from information theoretic measures of strength of association between asset returns, for example $w_{ij} := \mathbb{D}(r_i, r_j)$ (refering to the earlier chapter on $\mathbb{D}$).

A particular form of adjacency matrix that arises in many problems, is where the adjacency matrix is additionally *bipartite*. This means in practice that the vertices of the graph come in two *flavours*. In a financial context, these might represent *investor* nodes and *asset* nodes. Investors are not (in this analysis) directly connected to other investors, and assets are not directly connected to assets, but investors and assets may be

connected (have non-zero w_{ij} between their respective connecting vertex edges) with the weight representative of some measure of strength of interest (either explicit or implicit). The adjacency matrix then takes the form:

$$A = \begin{bmatrix} 0 & R \\ R^T & 0 \end{bmatrix} \tag{4.3}$$

with $R \in \mathbb{R}^{N_I \times N_A}$ where N_I is the number of unique investors, and N_A the number of unique assets under consideration.

4.1.2 Spectral Convolution: Gradient Descent Methods

[Zheng2018] considers a related bipartite problem in the context of a deep learning model, where a *spectral convolution layer* is introduced. In summary, the layer acts to take input feature vectors for the problem, transform them into the spectral space of the related Graph Laplacian, *mix/scale* the contribution of the projected vectors by trainable weights in the model applied to the eigenvalues of the Graph Laplacian and then project back into the space of feature vectors. An activation function on the transformed vectors is performed, and the layer returns its result:

$$\begin{bmatrix} x^I_{new} \\ x^A_{new} \end{bmatrix} = \sigma\Big(U g_\theta(\Lambda) U^T \begin{bmatrix} x^I \\ x^A \end{bmatrix} \Big) \tag{4.4}$$

where $\Lambda = \{\lambda_0, \ldots, \lambda_{N_I+N_A-1}\}$ are the eigenvalues of the Graph Laplacian, x^I, x^A are input feature vectors for the model for investors and assets respectively (in the terminology used here), and
$g_\theta(\Lambda) := diag([\theta_0\lambda_0, \ldots \theta_{N_I+N_A-1}\lambda_{N_I+N_A-1}])$
is the scaling/mixing matrix with θ_i learnable model parameters and which have the effect of increasing/decreasing the strength of contribution of different spectral components for the problem. The function σ is an activation function (e.g. elu, relu, ...) in the usual way. Multiple layers may then be combined sequentially, for example (and potentially in combination with other connections). For computational efficiency purposes an expansion of the term $U g_\theta(\Lambda) U^T$ is made, but the form of the transformation ("convolution") leveraging the spectral structure of the Graph Laplacian of the problem and endowing the model with an inductive bias related to the graph structure is the key point.

In the investor-asset case, it is of interest to learn implicit relationships that indicate a strength of interest/association of a given investor with a given asset and in particular to learn *relative* associations. In order to train the model to learn those associations, the *Bayesian Personalised Ranking* (BPR) objective function is employed ([Rendle2009]). In brief, this takes the implicit vectors learned by the model, x^I_i, x^A_j, x^A_k and for each investor and a pair of assets, generates a measure of relative interest of the investor i in the assets j, k. The adjacency matrix for the problem R is taken to represent (to the extent that is has been observed) the strength of association, and the measure seeks to learn implicit vectors, x, that maximise stronger association over weaker association for each triplet using:

$$\sigma\big(x^I_i.x^A_j - x^I_i.x^A_k\big) := \sigma(x_{ijk}) := \frac{1}{1+e^{-x_{ijk}}} \tag{4.5}$$

and associated obective function:

$$-\sum_{(i,j,k)} \ln(\sigma(x_{ijk})) + \lambda_\Theta \|\Theta\|^2 \tag{4.6}$$

where the sum is taken over ij and ik pairs where the association is known to be stronger for j than k, λ_Θ is a regularisation strength, and Θ are the learnable parameters of the problem. The objective function can be minimised by gradient descent.

In the next section illustrative examples are presented for a range of bipartite problems where the implicit vectors x^I, x^A are learned directly and form the parameters learned by the model, leveraging the projection into spectral space, but without activation. Extension to the case where the vectors are themselves learned combinations of input feature vectors is straightforward, but it is worth highlighting the properties of the simpler case before moving to a more complex model.

Python code for the examples is provided at:
https://github.com/vorosbegy/finML/tree/main/src/representations
using the Tensorflow2 framework to implement the model and perform gradient descent optimization of the objective function.

4.1.3 Illustrative Examples

Methodology

In order to generate illustrative bipartite graphs with controllable spectral properties, cluster periodicities were defined as a sequence of increasing integers $C := [n_1, n_2, n_3, \ldots, n_m]$ where $n_1 \leq n_2 \leq \cdots \leq n_m$ and $n_i \in \mathbb{Z}^+$. For a given cluster periodicity, the bipartite upper right block matrix was then defined as:

$$R^C_{pq} := \sum_{n_k \in C} \frac{1}{n_k} \mathbf{1}_{\{(p+q) \bmod n_k = 0\}} \tag{4.7}$$

using the indicator function $\mathbf{1}_{True} = 1$, $\mathbf{1}_{False} = 0$ and with $p \in [0, 1, \ldots, N_I]$, $q \in [0, 1, \ldots, N_A]$ so that $R^C \in \mathbb{R}^{N_I \times N_A}$. This creates a *coloured checkerboard* of varying spatial frequencies, dependent on C. For example $C = [2]$ is presented in the lower half of (Figure 4.1) and $C = [2, 3, 5, 7]$ in (Figure 4.2). The bipartite adjacency matrix was then constructed from R^C in the usual way:

$$A = \begin{bmatrix} 0 & R^C \\ (R^C)^T & 0 \end{bmatrix} \tag{4.8}$$

To enable observing the eigenvalue spectrum of the Graph Laplacian clearly, in the figures the number of investors and assets was taken to be $N_I = 20, N_A = 20$, giving a total of 40 eigenvalues. In each case model training procedure was:
(1) input learnable vectors $x^I \in \mathbb{R}^{N_I \times k}, x^A_j \in \mathbb{R}^{N_A \times k}$ with k chosen in advance of training
(2) form the vectors $t_{ij} := u_{ij}/(\sum_k u^2_{ik})^{1/2}$ and matrix $T \in \mathbb{R}^{(N_I+N_A)\times k}$ per the spectral clustering procedure discussed earlier. The matrix T is fixed, and does not vary during training.
(3) project the vectors x into $T.X \in \mathbb{R}^{(N_I+N_A)\times(N_I+N_A)}$
(4) use the *projected* vectors as input to the BPR objective function $\sigma\big(x^I_i.x^A_j - x^I_i.x^A_k\big)$
(5) calculate the BPR objective, compute gradients with respect to the original learnable vectors at step (1), update and iterate
The value of k was chosen using a heuristic that reflects the size of the eigengap: the distance between consecutive eigenvalues. The heuristic seeks to retain eigenvalues that are small relative to the other eigenvalues, based on results from a perturbation theory perspective on the spectral graph problem ([Luxburg2007]). The examples in the next section illustrate the consequence of different choices of k.

In this way, the method is seen to be enabling the model to minimise the BPR objective by learning vectors in the *cluster* space but with a representation that is small in dimension (k) relative to the size of $N_I + N_A$. The learned vectors are weights that dictate how to combine the classic spectral clustering vectors. The following section discusses a number of examples.

Examples and Discussion

Case1
Parameters:
C=[3]
Observations:
The (Figure: 4.3) illustrates a simple case with cluster periodicity = [3]. It can be seen that there are, as expected three eigenvalues equal to zero, which reflect the partitioning of the investor and asset universe into three separate groups. The dimension k was chosen to be $k = 3$ using the eigengap heuristic. The lower portion of the figure shows the associated matrix, R (the first 20 entries in each case). The periodicity of three is clearly apparent (every third column, or every third row is identical). (Figure: 4.4) shows the reconstructed R matrix from the x^I, x^A vectors learned by the model (following the spectral projection procedure above during BPR minimisation). Clearly the learned vectors capture the spatial frequency of the input adjacency matrix. Additionally, (Figure: 4.5) then plots the learned x^I vectors in their $k = 3$ dimensional space. The clustering of the learned vectors is quite apparent: a small amount of jitter just enables distinguishing the points which essentially lie on top of each others in the three separate and clearly distinguished clusters.

Case2
Parameters:
C=[3], noise $\sim U[0, 0.3]$ added to R
Observations:
The (Figure: 4.6) extends the simple case of cluster periodicity = [3] by adding some noise to the matrix

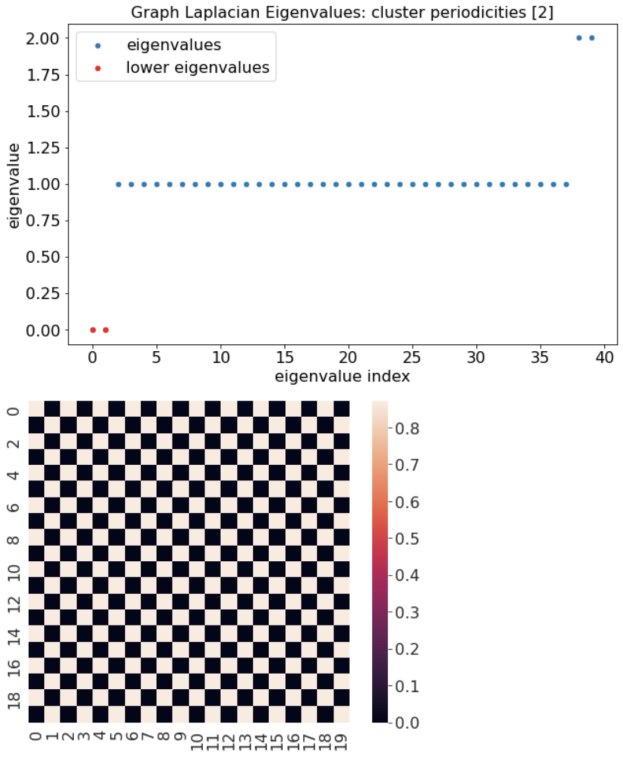

Figure 4.1: Graph Laplacian Eigenvalues: C=[2]

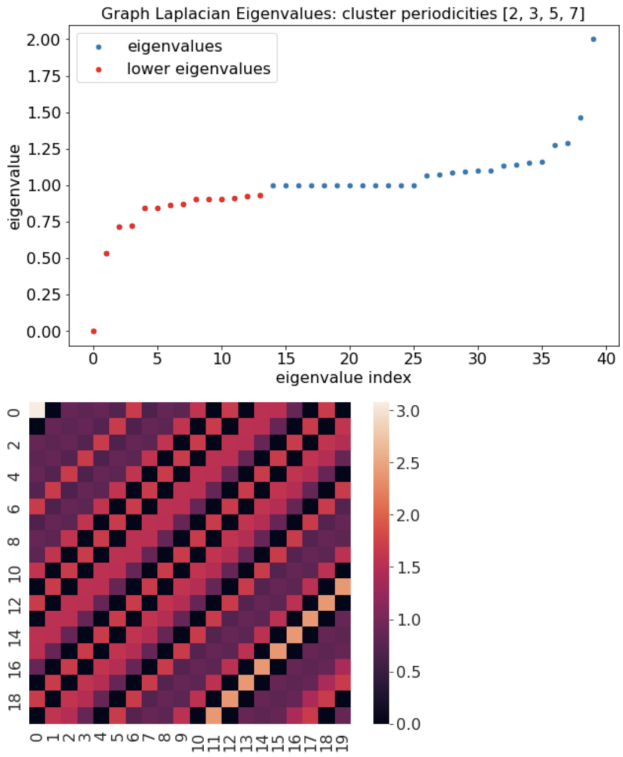

Figure 4.2: Graph Laplacian Eigenvalues: C=[2, 3, 5, 7]

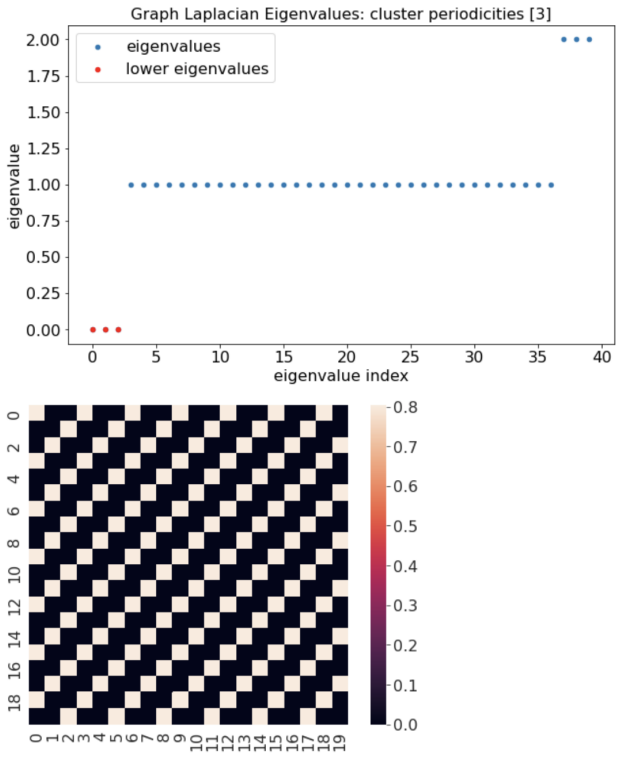

Figure 4.3: Graph Laplacian Eigenvalues: C=[3]

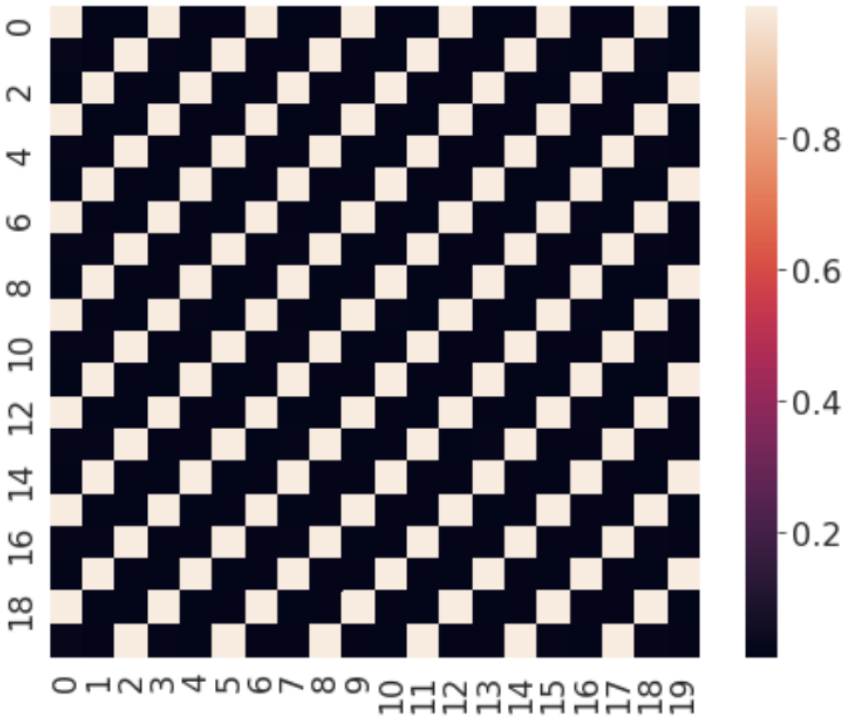

Figure 4.4: R reconstruction post training: Cluster Periodicity C=[3]

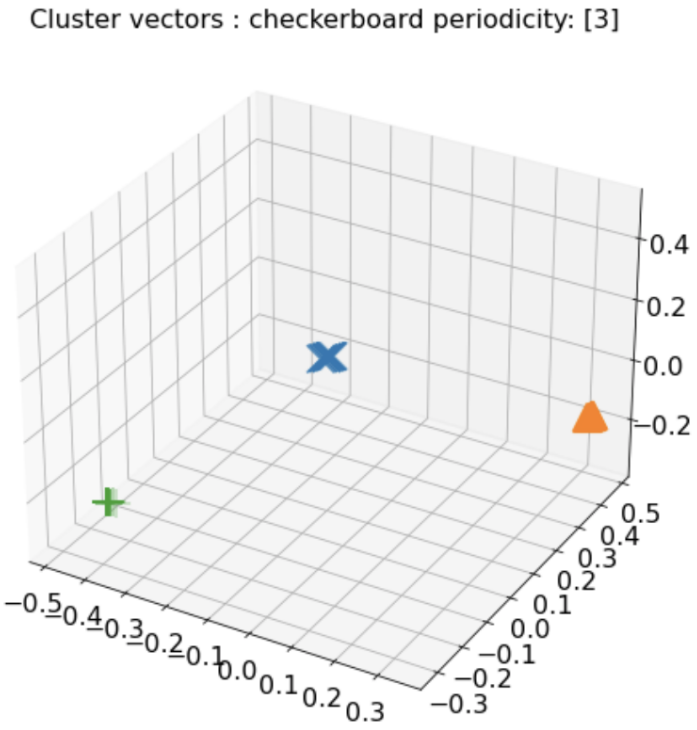

Figure 4.5: *Investor* cluster reconstruction post training: C=[3]

R before constructing the Graph Laplacian. Noise was added randomly and uncorrelated separately to each element of R as a sample from $U[0, 0.3]$. It can be seen that two of the three lowest eigenvalues are non longer zero. In this example, more of the eigenvalues were included in the spectral construction: the eigengap was deliberately not respected and all eigenvalues highlighted in red were included. (Figure: 4.6) shows the less clear spatial frequency of R (although it is still apparent). The results of the model post training however are stark: (Figure: 4.7) and (Figure: 4.8) respectively show that the model struggled to retrieve the R matrix, and the learned x^I vectors are now poorly spatially resolved across the three known clusters (as explicitly designed in the input using C=[3]).

(Figure: 4.9), (Figure: 4.10) and (Figure: 4.11) further extend the case to consider the adhering to spectral gap heuristic, and retaining only the first three spectral components. It can be seen then that, despite the noise, the model is able to learn good representative vectors that are much better spatially resolved, and which enable reconstructing the spatial frequencies of the input R matrix to a much higher degree.

4.1.4 Conclusion

The topic of learning suitable representations for data is broad and important. Endowing learning models with inductive biases that reflect the structure (in many cases *symmetries*) of the problem is a growing and active area of research, building on many developments in recent decades. Enabling models to be sufficiently expressive, but constraining them such they reflect the structure of the problem is seen to produce more accurate models that required smaller observational data sets on which to train.

The examples provided in this chapter focussed on spectral graph representations in particular as one powerful tool to representing data that naturally exists in graph-form. Although the examples were toy in nature, it is important to fully digest the consequence of different representation embedding choices before moving to more complex models.

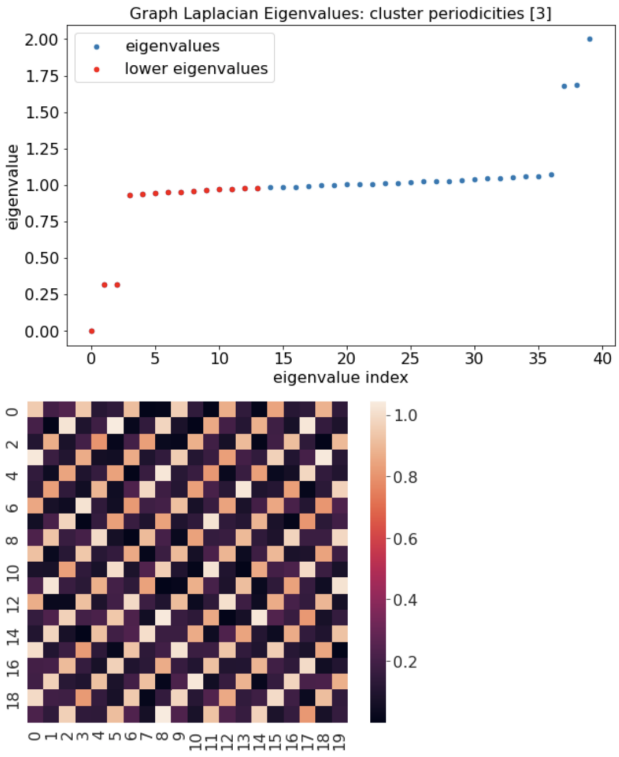

Figure 4.6: Graph Laplacian Eigenvalues: C=[3] with R noise $\sim U[0, 0.3]$

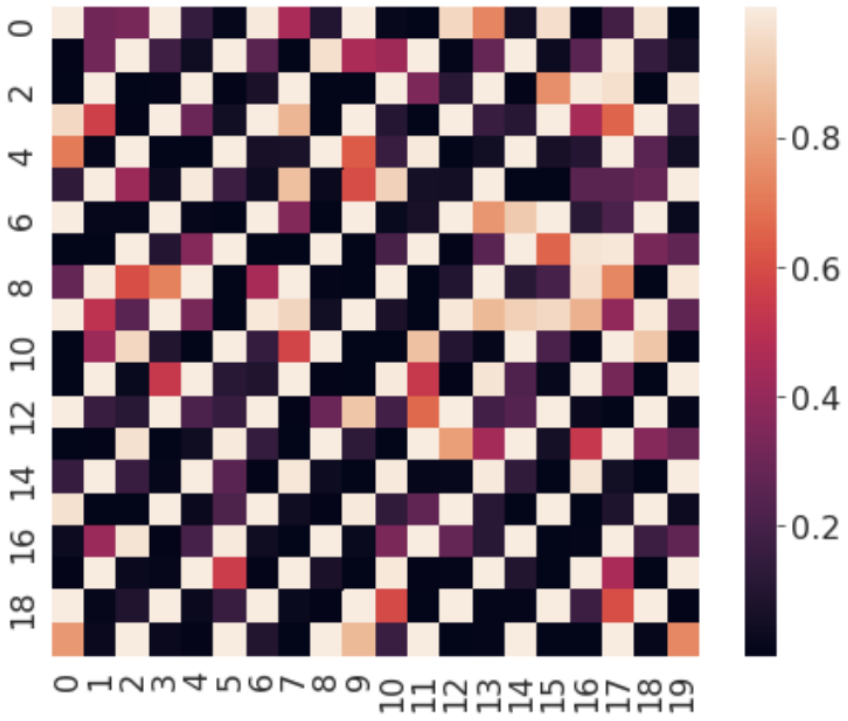

Figure 4.7: R reconstruction post training: C=[3] with R noise $\sim U[0, 0.3]$

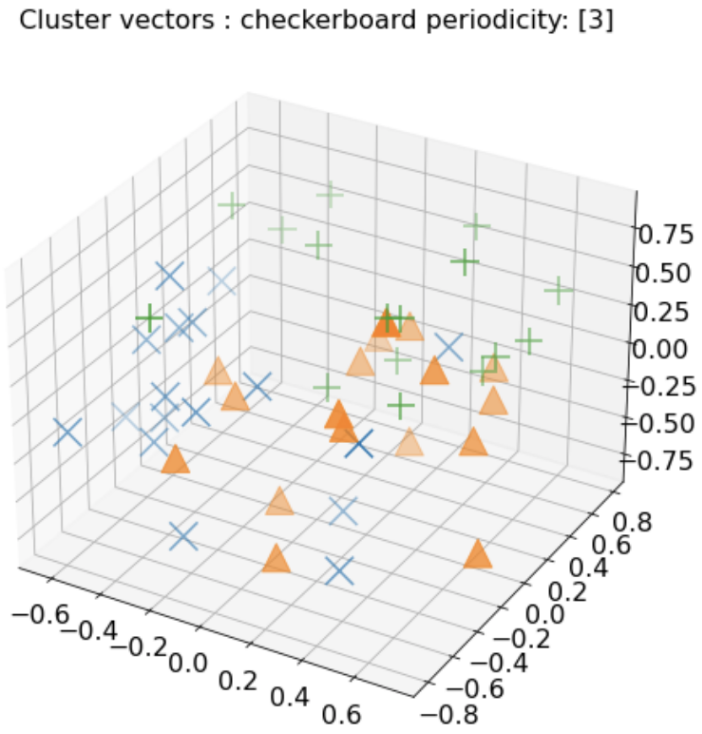

Figure 4.8: *Investor* cluster reconstruction post training: C=[3] with R noise $\sim U[0, 0.3]$

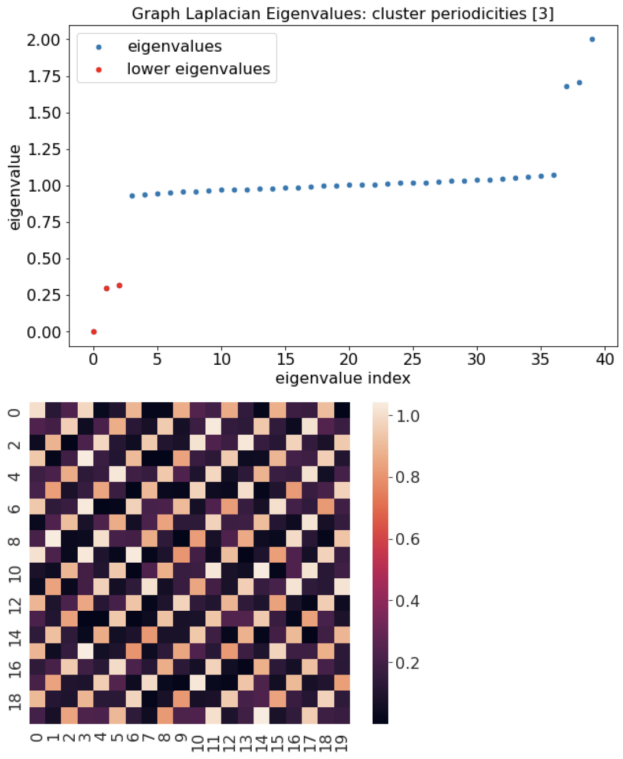

Figure 4.9: Graph Laplacian Eigenvalues: C=[3] with R noise $\sim U[0, 0.3]$: Eigenvalue gap respected

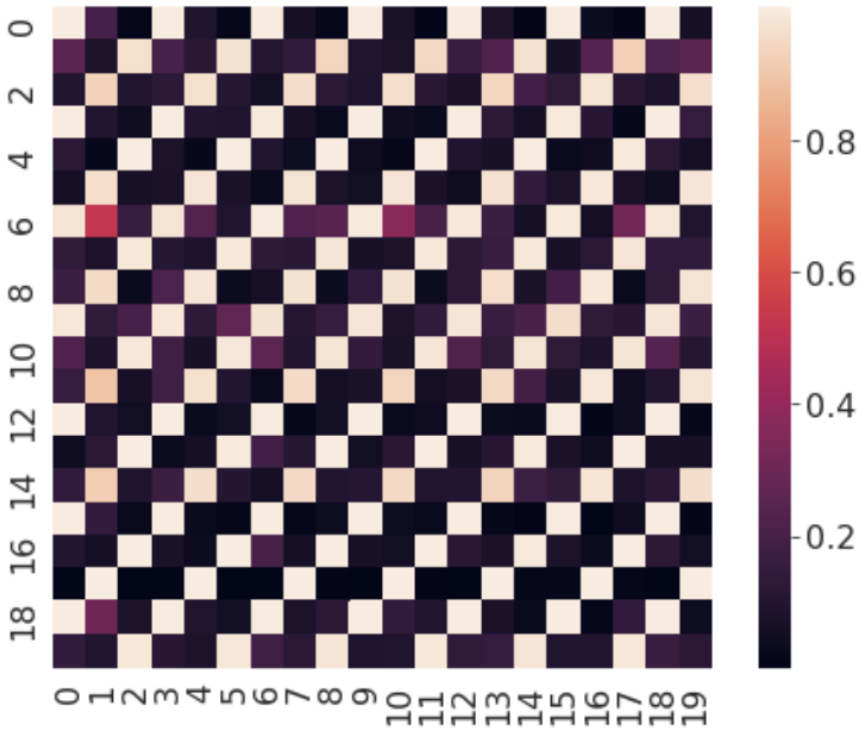

Figure 4.10: R reconstruction post training: C=[3] with R noise $\sim U[0, 0.3]$: Eigenvalue gap respected

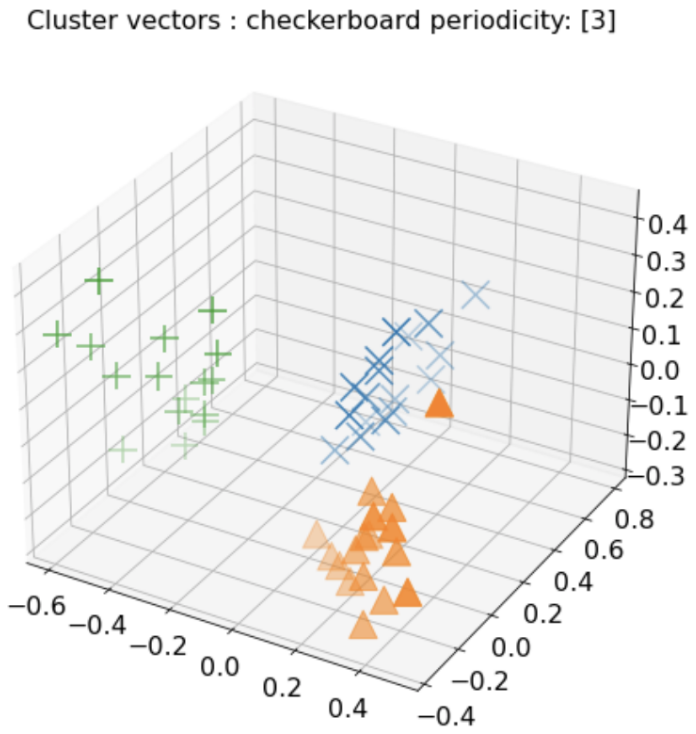

Figure 4.11: *Investor* cluster reconstruction post training: C=[3] with R noise $\sim U[0, 0.3]$: Eigenvalue gap respected

Chapter 5

Probabilistic Time Series Modelling

5.1 Variational Models

5.1.1 Non-linear State Space Models

XXX

Bibliography

[Hand2009] David J. Hand
Measuring classifier performance: a coherent alternative to the area under the ROC curve
Mach Learn (2009) 77: 103–123

[Caticha2007] Ariel Caticha
Information and Entropy
https://arxiv.org/pdf/0710.1068.pdf

[Samo2018] Yves-Laurent Kom Samo, Dieter Hendriks
What makes an Asset Useful?
https://arxiv.org/abs/1806.08444

[LDP2018] Marcos López De Prado
Advances in Financial Machine Learning
Wiley, 2018

[LDP2020] Marcos López De Prado
Machine Learning for Asset Managers
Cambridge University Press, 2020

[Lu2017] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, Liwei Wang
The Expressive Power of Neural Networks: A View from the Width, 2017
https://arxiv.org/abs/1709.02540

[Gabrielsson2020] Rickard Brüel Gabrielsson
Universal Function Approximation on Graphs, 2020
https://arxiv.org/abs/2003.06706

[Hutchinson2021] Michael Hutchinson, Charline Le Lan, Sheheryar Zaidi, Emilien Dupont, Yee Whye Teh, Hyunjik Kim
LieTransformer: Equivariant Self-Attention for Lie Groups, 2021
https://arxiv.org/abs/2012.10885

[Xia2021] Kevin Xia, Kai-Zhan Lee, Yoshua Bengio, Elias Bareinboim
The Causal-Neural Connection: Expressiveness, Learnability, and Inference, 2021
https://arxiv.org/abs/2107.00793

[Kalainathan2018] Diviyan Kalainathan, Olivier Goudet, Isabelle Guyon, David Lopez-Paz, Michèle Sebag
Structural Agnostic Modeling: Adversarial Learning of Causal Graphs, 2018
https://arxiv.org/abs/1803.04929

[Luxburg2007] Ulrike von Luxburg
A Tutorial on Spectral Clustering, 2007
https://arxiv.org/abs/0711.0189

[Spielman2019] Daniel A. Spielman
Spectral and Algebraic Graph Theory, 2019
http://cs-www.cs.yale.edu/homes/spielman/sagt/sagt.pdf

[Zheng2018] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, Philip S. Yu
Spectral Collaborative Filtering
https://arxiv.org/abs/1808.10523

[Ng2002] Ng, A., Jordan, M. and Weiss, Y.
On spectral clustering: analysis and an algorithm, 2002
Advances in Neural Information Processing Systems 14

[Rendle2009] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, Lars Schmidt-Thieme
BPR: Bayesian Personalized Ranking from Implicit Feedback, 2009
https://arxiv.org/abs/1205.2618