Towards Matrix Langevin Process Inference and a Geometry of Belief

John Whitamore

Abstract

TBA

1 Introduction

This paper started life as a tutorial exploring connections between well-known parametric and non-parametric regression and classification models in statistics, probabilistic machine learning and neural networks. Through a focus on feature selection, sparsity and regularisation, a unifying framework emerged. Upon closer inspection, that framework proved not only to be powerful and general but also interpretable and amenable to control inputs such as human interactions.

The framework treats all transformations and all hypotheses as rotations - orthogonal matrices drawn from a subset of the Lie group $\mathrm{SO}(n)$. The Matrix Langevin distribution provides a natural way to express probability densities over these rotations and thus over the set of all transformations and hypotheses. The dynamics of transformation are modelled using two stochastic differential equations. The first, known as the Left-Invariant Stochastic Flow or Brownian Motion on a Lie Group, is familiar to mathematical physicists, differential geometers, topologists and statisticians working with Stiefel manifolds and orientation data. The second equation, which additionally incorporates a fracture field - analogous to jumps or control inputs - appears to be novel. For want of a better name, we could refer to these models collectively as Matrix Langevin Processes.

The paper begins, innocently enough, with practical consideration of what we would like from machine learning models. And then proceeds through the foothills of OLS regression, ridge regression and so on. It is in these foothills that the framework begins to take shape. Once created, we demonstrate the application of the framework to a variety of models including Bayesian regression, the Relevance Vector Machine, Haar wavelets and polynomial splines. This portion of the paper is designed to ask what we want from machine learning models, to create a framework that can deliver those *desiderata* and to demonstrate its use.

The final section formalises the framework. It is here that we introduce and discuss the framework in terms of Matrix Langevin distributions and stochastic processes, with particular attention to the geometry of transformations. This section aims to provide a more formal foundation for the framework and to show how it might be used to construct machine learning models that are powerful, general, interpretable and which interact well with other models and with human users.

2 Desiderata

From a practical business point of view, an ideal machine learning model would be able to connect to a data lake, select, clean and structure relevant data, consider all useful hypotheses about the data and then reach reliably sensible conclusions including making predictions (in the regression sense) and classifications. The models should be reliable enough that they can be connected together to

do things, in an agentic flow. And they should be able to communicate with users in a way that can be clearly understood and which allows users to inform the model outputs. That's the dream. Let's consider it in detail so that we can write down a list of specific requirements.

A machine learning (ML) model pipeline typically has three stages. The first stage is responsible for extracting and collecting raw data and transforming it into a shape that is sufficiently clean and structured for the purpose at hand. We can write $g : \mathcal{D} \to \mathcal{D}^*$ in which $\mathcal{D}$ represents raw data in a data lake, $\mathcal{D}^*$ represents clean, structured data (sometimes in a data warehouse) and g represents the function (or ETL pipeline) responsible for data extraction, cleaning and structuring.

The second stage transforms structured data into the form that we want to use within our ML model. We can write $\phi : \mathcal{D}^* \to X$. In the context of regression, we might have a number of basis functions that map clean data $\boldsymbol{D}^*$ onto a design matrix X. In the context of text and other data, X might represent a carefully constructed ontology.

Finally, the third stage is the ML model itself, in which the model learns the mapping $f : X \to \boldsymbol{y}$. This is the part that I originally intended to write about.

Overall, we have a compositional pipeline $h = f \circ \phi \circ g : D \to y$ in which $g : \mathcal{D} \to \mathcal{D}^*$ maps raw to structured data, $\phi : \mathcal{D}^* \to X$ maps structured data onto a representation suitable for an ML model and $f : X \to \boldsymbol{y}$ maps the representation onto observed outputs. The task of the machine learning model is to learn f (or - in a Bayesian setting - a distribution over f) so that useful predictions and classifications can be made.

There are several practical problems with such a pipeline. The first is that the construction of g (ETL pipeline from data lake to data warehouse) and ϕ (creation of the data representation used by the ML model) are typically manual, time-consuming and subjective. The second is that the results of each stage are fixed with respect to the other stages. There is no sense of compositional flow through the whole process that can be used to achieve global optimisation. The third is that the results of each stage constrain the set of inputs available to the next stage. The fourth and fifth are that ML models tend not to explain themselves well to humans, nor to encourage (or even permit) human insights to shape their outputs.

These points translate into widely-experienced practical difficulties. Endless meetings to discuss and decide which data should be pipelined and how it should be represented. Operational blockages because a data scientist wants to use a certain data set but discovers that, although available in the data lake, it is not yet available for use in the data warehouse. And models that generate outputs a) with no explanation of what features or hypotheses were used and b) with no opportunity for users to shape or inform the outputs of the model.

An ideal machine learning model would exhibit:

- End-to-end compositionality: models should support forward and backward training across the entire pipeline, enabling convergence through global optimization.
- Automated feature selection and regularization: at every stage, the system should autonomously select relevant features, progressively de-selecting less useful ones and applying principled regularization to avoid over-fitting.
- Hypothesis management: just as with features, the system should reason over hypotheses starting from a very broad space of hypotheses and narrowing the space of plausible explanations as it ascends the architecture.
- Interpretability and explanation: the system should be able to trace its reasoning from raw data to output, articulating which hypotheses were considered and which were ultimately used.
- Human interaction: the model should allow users to shape and inform its outputs.

3 Linear regression

A linear regression model is one in which the function $f : X \to \boldsymbol{y}$ is linear, so that $f(X) = X\boldsymbol{w}$, where $X \in \mathbb{R}^{N \times M}$ is called the *design matrix* and $\boldsymbol{w} \in \mathbb{R}^M$ is a vector of regression weights (or coefficients). We will use $\boldsymbol{\epsilon} \in \mathbb{R}^N$ to denote a vector of residuals $\boldsymbol{\epsilon} \equiv \boldsymbol{y} - X\boldsymbol{w}$ and $\boldsymbol{\theta}$ to denote the parameters of a model.

3.1 OLS regression

Given real-valued observed data $\boldsymbol{y}$, it can be appropriate to assume i.i.d. normally distributed residuals, giving log likelihood $\ell(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{\theta})$:

$$\ell(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{\theta}) = -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - \boldsymbol{x}_n^T\boldsymbol{w})^2$$

Maximising the log likelihood corresponds to minimising the sum of squared residuals $\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}$ given by

$$\boldsymbol{\epsilon}^T\boldsymbol{\epsilon} = (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - \boldsymbol{x}_n^T\boldsymbol{w})^2$$

Differentiating with respect to $\boldsymbol{w}$ and setting the gradient equal to (vector) $\mathbf{0}$ yields the maximum likelihood estimate of the regression weights:

$$\begin{aligned}\frac{d\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}}{d\boldsymbol{w}} &= 2X^TX\boldsymbol{w} - 2X^T\boldsymbol{y} = \mathbf{0} \\ \hat{\boldsymbol{w}} &= (X^TX)^{-1}X^T\boldsymbol{y}\end{aligned}$$

The maximum likelihood weights $\hat{\boldsymbol{w}}$ are used to obtain fitted values, $\hat{y} = X\hat{\boldsymbol{w}}$ and can be used to predict output values $\hat{\boldsymbol{y}}^* = \boldsymbol{x}^{*T}\hat{\boldsymbol{w}}$ given some new value of $\boldsymbol{x}^{*T}$.

3.1.1 Geometric interpretation

We have $\hat{\boldsymbol{w}} = (X^TX)^{-1}X^T\boldsymbol{y}$ and $\hat{\boldsymbol{y}} = X\hat{\boldsymbol{w}}$. Putting these together yields

$$\begin{aligned}\hat{\boldsymbol{y}} &= X(X^TX)^{-1}X^T\boldsymbol{y} \\ &= H\boldsymbol{y}\end{aligned}$$

where $H = X(X^TX)^{-1}X^T$ is known as the "hat" matrix. It is made up of an outer product XX^T transformed by a matrix $P = (X^TX)^{-1}$ to give XPX^T.

XX^T has dimensions $N \times N$ and operates in observation space. It encodes similarity between pairs of rows of the design matrix, so that element i, j of XX^T encodes the similarity between rows i and j of the design matrix and - to the degree that rows of the design matrix successfully model variation in the observed data - also encodes the similarity between data points.

By contrast, X^TX has dimensions $M \times M$ and operates in the lower dimensional feature space. It encodes how similar columns - or features - of the design matrix are to each other. Its inverse,

$P = (X^T X)^{-1}$ represents an operation of correcting for feature covariance, acting as a kind of normalising transformation in feature space.

Putting these pieces together, the hat matrix $H = XPX^T$ represents the pairwise similarity between observations (XX^T) being corrected by $P = (X^T X)^{-1}$ for pairwise similarity between features, obtaining fitted values that respect both the geometry of the data and the structure of the features.

3.1.2 Remarks towards the framework

A difficulty is that OLS regression will fail if $(X^T X)^{-1}$ cannot be inverted. Overt failure is, in a sense, a good type of problem because it is obvious when it has occurred and corrective steps can be taken. A more insidious problem is when the columns of X are not sufficiently linearly independent. This leads to the model appearing to work whilst generating regression weights that are unstable and dangerous to use. These two problems are manifestations of multicollinearity.

As a first step towards our framework, we therefore stipulate that the columns of X must be mutually orthogonal. We further stipulate that the columns must be normalised to unit length. (Given mutually orthogonal columns, normalising column lengths simply results in the corresponding regression weights being multiplied by the normalisation term, so the effect comes out in the wash). A matrix with mutually orthogonal columns, each normalised to unit length, is said to have *orthonormal* columns.

If X has orthonormal columns then $X^T X = I_M$. If X is square we also have $XX^T = I_N$. Let's use these properties in the context of our OLS regression results. We have

$$\begin{aligned}\hat{\boldsymbol{y}} &= X\hat{\boldsymbol{w}} \\ &= X(X^T X)^{-1} X^T \boldsymbol{y} \\ &= XX^T \boldsymbol{y}\end{aligned}$$

If X is square then we have $XX^T = I_N$ and we obtain the beautiful but useless result $\hat{\boldsymbol{y}} = \boldsymbol{y}$. This is not a model but a tautology: "If you tell me the true value of y, I'll tell you my estimate of y". This demonstrates that X must not be square. We need to project the observed data into a lower dimensional feature space in order to obtain useful results.

On the other hand, if X is "tall", with the number of rows N being larger than the number of columns M, we have a result which is useful ($\hat{\boldsymbol{y}} = XX^T \boldsymbol{y}$) and in which the matrix X does not appear directly but only as the outer product XX^T. This suggests that we could define a new matrix $K = XX^T$, subject to K having certain required properties, and re-cast our OLS regression simply as $\hat{\boldsymbol{y}} = K\boldsymbol{y}$.

Let's think carefully, because this formulation seems to contain some interesting possibilities. The first is that we might be able to write OLS regression directly in terms of a matrix K, rather than needing to use design matrix X (painstaking work requiring domain knowledge that we might not have). In particular, we might be able to create an architecture in which K is progressively refined across layers. And then to build this architecture all the way from raw data in a data lake through to model outputs $\hat{\boldsymbol{y}}$.

The second is that we might be able to construct links between OLS regression and other widely-used models, particularly non-parametric models. Doing so would be useful in itself. It would also suggest the ability of a layered architecture being able to represent a wide variety of model types.

Both of these possibilities turn out to be true. However, it's also worth noting that we probably wouldn't want to apply this formulation naively: attempting to estimate $N(N+1)/2$ elements of

symmetric matrix K is grossly inefficient compared to estimating M regression coefficients in the OLS formulation.

3.1.3 Decomposition

Here, K plays the role previously played by the hat matrix H but with the requirement that X now has orthonormal columns. We have $X^T X = I_M$ so that no feature space correction is made or required because the feature space has orthonormal columns. Nonetheless, we have already seen that we do need to project the observed data into a lower dimensional feature space in order to obtain useful results. Indeed, we require that matrix K is an *orthogonal projection matrix*.

In order to be an orthogonal projection matrix, K must be symmetric (which implies square), positive semi-definite and idempotent (so that $K^2 = XX^T XX^T = XIX^T = XX^T = K$ with $X^T X = I$). We require that the eigenvalues are either 0 or 1, which guarantees idempotency ($K^2 = K$) and implies that the determinant of K is either 0 or 1 depending on rank.

Let's decompose K so that we can see what is going on more clearly. We can write $K = Q\Lambda Q^T$ such that a) Q is $N \times N$ and orthogonal, with $Q^T Q = I_N$ and b) Λ is $N \times N$ and diagonal with diagonal entries in $\{0, 1\}$. The number of 1s in Λ determines the rank of K, i.e. the dimension of the subspace, which must always be less than N. These conditions are sufficient to guarantee that K is symmetric, positive semi-definite and idempotent.

The diagonal elements of K act as switches that encode binary choices about which parts of observation space are projected into feature space. They also serve to partition K into a part which is all zeroes (and effectively not used in the model) and a part which is used in the model and which is positive definite.

Somewhere in the distance, we can see the possibility of an algorithm which automatically determines a) which parts (for relevance) and b) how many parts (for sparsity) of observation space to project into feature space. However, we can also see that a naive application of the machinery we have so far would be grossly inefficient. We need to add something - which is regularisation - and, in doing so, let go of something else - which is the requirement for K to be idempotent.

3.2 Ridge regression

One of the tell-tale signs of an OLS model over-fitting to data is large absolute values of weights. For example, using a polynomial to model linear observed data will tend to result in a fitted polynomial that matches the data very accurately at the observed data points but which, in between data points, soars very high and swoops very low. We observe high absolute values of weights as the polynomial over-fits the data.

Ridge regression penalises weights using the square of the L2 norm over weights, so that we write a loss function, L as the sum of squared residuals (as for OLS regression) plus a multiple λ of the sum of squared weights. Note that this loss function balances accuracy (from reducing the sum of squared errors) against model complexity (rather loosely), by penalising high values of regression weights.

$$\begin{aligned} L &= \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} + \lambda \boldsymbol{w}^T \boldsymbol{w} \\ &= (\boldsymbol{y} - X\boldsymbol{w})^T (\boldsymbol{y} - X\boldsymbol{w}) + \lambda \boldsymbol{w}^T \boldsymbol{w} \end{aligned}$$

We find the regression weights that minimise the loss by differentiating the loss with respect to the weights $\boldsymbol{w}$ and setting the result equal to (vector) zero.

$$\begin{aligned}\frac{dL}{d\boldsymbol{w}} &= 2X^TX\boldsymbol{w} - 2X^T\boldsymbol{y} + 2\lambda\boldsymbol{w} = \mathbf{0} \\ (X^TX + \lambda I_M)\boldsymbol{w} &= X^T\boldsymbol{y} \\ \boldsymbol{w} &= (X^TX + \lambda I_M)^{-1}X^T\boldsymbol{y}\end{aligned}$$

An appropriate value of λ is typically found using cross-validation.

Note that the expression for the weights that minimise loss for some value of λ is very similar to its equivalent under OLS regression. Instead of $(X^TX)^{-1}$ for OLS we now have $(X^TX + \lambda I_M)^{-1}$ for ridge regression.

3.2.1 Going deeper

Let us again insist that X have orthonormal columns so that $X^TX = I_M$. This time we write $K = \frac{1}{1+\lambda}XX^T$ in order to incorporate our regularisation term. This enables us to write $\hat{\boldsymbol{y}} = K\boldsymbol{y}$, as before. And we perform the decomposition $K = Q\Lambda Q^T$ exactly as before but with one important difference. Whereas before, we required the diagonal elements of Λ to be either 0 or 1, we now require them to be in $[0, 1]$ so that we can accommodate the $\frac{1}{1+\lambda}$ term from regularisation, in which $\lambda \geq 0$.

Think of the diagonal terms of Λ (under OLS regression) as being binary switches that decide whether a feature of Q is excluded from (value 0) or included in (value 1) the feature subspace. The product of the non-zero diagonal terms represents the determinant of the features chosen to be included in the feature space. We now need to reduce that determinant from 1 (under OLS) to $\frac{1}{1+\lambda}$ under ridge regression. With $\lambda \geq 0$, this means that, for ridge regression, the diagonal elements must now be real values in $[0, 1]$ rather than the binary switches in $\{0, 1\}$ used for OLS regression.

And suddenly the door swings open to connect ridge regression with a wide variety of well-known models. Let's take a look.