Short explanation of Treeffuser

March 2, 2024

Abstract

We present Treeffuser, a new model class for robust probabilistic prediction of arbitrarily complex conditional densities. Typical regression models return a point estimate conditional on covariates, but probabilistic regression models output a full probability distribution over the outcome space, conditional on the covariates. This allows for predictive uncertainty estimation — crucial in applications like healthcare and weather forecasting. However, standard methods for probabilistic predictions rely on stringent parametric assumptions about the form of likelihood, leading to poorly calibrated uncertainty estimates and bad predictions when these assumptions are not met. To remedy this issue we leverage recent advances in diffusion-based generative modeling to propose Treeffuser a new model for probabilistic predictions that achieves comparable or better uncertainty estimates, calibration, and conditional log-likelihood than existing methods, while providing considerable advantages in usability and robustness. We provide a implemention of treefuser in [repo/pipy]

1 Goals

Assume that $\mathbf{x} \in \mathbb{R}^{d_x}$, $\mathbf{y} \in \mathbb{R}^{d_y}$ and that there exists some distribution $p^*(\mathbf{x}, \mathbf{y})$ over both $\mathbf{x}$ and $\mathbf{y}$. The goal is to learn the full conditional distribution $p^*(\mathbf{y}|\mathbf{x})$ so that we can sample from it.
Once we have learned to sample from it we can use the samples as one would in a bayesian setting and report, mean, quantiles, variance, expectations of quantities, measures of risk and uncertainty (e.g CVAR, VaR, etc.), etc.
The requirements for this algorithm are:

1. It should be able to learn the full conditional distribution $p^*(\mathbf{y}|\mathbf{x})$ even if the distribution is multimodal, has complex dependencies, is reasonably high-dimensional, has heavy tails, etc.

2. It should be robust enough to be usable by non-experts with a scikit-learn like interface.

3. The algorithm should be tailored for tabular data (i.e not for images, text, etc.).

2 The algorithm

The idea of the algorithm is the following:

Use diffusion models to learn the conditional distribution $p^*(\mathbf{y}|\mathbf{x})$ *but instead of estimating the score function using a neural network use gradient boosted trees, with one tree for each dimension of* $\mathbf{y}$.

To be concrete, we first construct a diffusion process $\{\mathbf{y}(t)\}_{t=0}^{T}$, indexed by time $t \in [0, T]$, with initial condition $\mathbf{y}(0) \sim p^*(\mathbf{y}|\mathbf{x})$ and such that

$$d\mathbf{y} = \mathbf{f}(\mathbf{y}, t)dt + g(t)d\mathbf{W}$$

where $\mathbf{W}$ is a Wiener process and $\mathbf{f}$ and g are functions that we choose.

By a result in Anderson (I'll add the reference) then it is the case that by starting from samples $\mathbf{y}(T) \sim p^*(\mathbf{y}(T)|\mathbf{x})$ and reverting the process we can obtain samples from $p^*(\mathbf{y}|\mathbf{x})$. Moreover, the reverse process has the nice form of

$$d\mathbf{y} = [\mathbf{f}(\mathbf{y}, t) - g(t)^2 \nabla_{\mathbf{y}} \log p_t(\mathbf{y}|\mathbf{x})]dt + g(t)d\bar{\mathbf{W}}$$

where $p_t(\mathbf{y}|\mathbf{x})$ is the density of the diffusion process at time t, and $\bar{\mathbf{W}}$ is a reverse Wiener process.

Therefore if we can learn the score function $\nabla_{\mathbf{y}} \log p_t(\mathbf{y}|\mathbf{x})$ and we construct the diffusion process such that $p^*(\mathbf{y}(T)|\mathbf{x})$ is a simple distribution like a standard normal then we can sample from $p^*(\mathbf{y}|\mathbf{x})$ by sampling from the standard normal and then running the reverse process (this can be done with standard numerical methods for SDEs). The only thing that we need is an estimate for the score function.

It turns out that by results in Vincent (2010) it is possible to estimate this score function by selecting the mapping $\mathbf{s}(\mathbf{y}, \mathbf{x}, t) : \mathbb{R}^{d_y} \times \mathbb{R}^{d_x} \times \mathbb{R} \to \mathbb{R}^{d_y}$ that minimizes the following loss:

$$\mathcal{L}(\mathbf{s}) = \mathbb{E}_{p(t)} \mathbb{E}_{p^*(\mathbf{x})p^*(\mathbf{y}|\mathbf{x})p_t(\mathbf{y}(t)|\mathbf{y}(0))} \left[||\nabla_{\mathbf{y}} \log p_t(\mathbf{y}(t)|\mathbf{y}(0)) - \mathbf{s}(\tilde{\mathbf{y}}, \mathbf{x}, t)||^2 \right] \quad (1)$$

Where $p(t)$ is uniform over $[0, T]$, and $p_t(\mathbf{y}(t)|\mathbf{y}(0))$ is the density of the diffusion process at time t conditioned on the initial condition $\mathbf{y}(0)$.
This quantity is something that we can work with empirically because

1. $\nabla_{\mathbf{y}} \log p_t(\mathbf{y}(t)|\mathbf{y}(0))$ is computable analytically as we construct the diffusion process.
2. We can sample from $p(t)$ because it is just a uniform distribution.
3. We can motecarlo approximate the expectation over $p^*(\mathbf{x})p^*(\mathbf{y}|\mathbf{x})p_t(\mathbf{y}(t)|\mathbf{y}(0))$. by using a dataset $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{n}$ and sampling from $p(\mathbf{y}(t)|\mathbf{y}(0))$ which is ok because it is a quantity we construct ourselves when we create the diffusion process.

Therefore, what we can do is construct one gradient boosted tree for each dimension of $\mathbf{y}$ that takes as input $\mathbf{y}$ and $\mathbf{x}$ and t outputs the i-th coordinate of the score function $s(\mathbf{y}, \mathbf{x}, t)_i$ and we train them by minimizing the loss above. Once trained we have a function $\mathbf{s}(\mathbf{y}, \mathbf{x}, t)$ that we can plug-in to the reverse SDE (1) and use to sample from $p^*(\mathbf{y}|\mathbf{x})$ which is what we wanted.

3 Why Gradient Boosted Trees?

In principle this scheme could work with any model. However, gradient boosted trees are the method of choice for tabular datasets. Moreover, because we care about robustness and usability by non-experts they seem like the natural model class.
In some simple experiments that we have conducted in a dataset with $n = 10000$ and $d_x = 1$ and $d_y = 1$ it takes around 3 seconds to train and 20 seconds to sample 100 points for a 1000 test points (i.e 100000 samples total).

4 Why better calibration - UQ?

We make no parametric assumptions about the form of the likelihood and hence we can learn the conditional distribution $p^*(\mathbf{y}|\mathbf{x})$ regardless of how complex it is as long as we sample densely enough from it and our model is flexible enough.(We could even prove stuff about this but not sure it is helpful).

References

Vincent, P. (2010). A connection between score matching and denoising autoencoders.