Supplementary Information: Probabilistic Prediction of Oceanographic Velocities with Multivariate Gaussian Natural Gradient Boosting

Michael O'Malley[1], Adam M. Sykulski[2], Rick Lumpkin[3], and Alejandro Schuler[4]

[1]STOR-i Centre for Doctoral Training, Department of Mathematics and Statistics, Lancaster University, Lancaster, U.K.
[2]Department of Mathematics, Imperial College London, London, U.K.
[3]NOAA/Atlantic Oceanographic and Meteorological Laboratory, Florida, U.S.A.
[4]Center for Targeted Machine Learning, University of California, Berkeley, California, U.S.A.

This supplementary information provides the following:

1. Extra details on the setup of the probabilistic neural network.
2. Details of the grid search which we used for both the application and simulation.
3. Extra details on the timings of all the methods.
4. Extended simulation results, where for the simulations we show the metrics introduced for the application. We also show the unmodified simulation as it was presented in Williams (1996).
5. A sample code snippet using the package.

A Probabilistic Neural Networks

In the paper, an existing neural network approach (referred to as NN) is used to fit conditional multivariate Gaussian distributions. The method fits a neural network taking inputs $\mathbf{X} \in \mathcal{X}$ where the output layer has M units, which are used as $\boldsymbol{\theta}$ in the probability density function parameterization in Section 3.1 (Williams, 1996; Sützle and Hrycej, 2005). The loss function used when optimizing the neural network parameters is the negative log-likelihood.

Instead of fitting M independent models as in Section 2.1, when using probabilistic neural networks we fit one model which has M outputs, one for each parameter of the probability density function. Throughout the paper, a fully connected neural network structure is used such as the one in Figure S1. We describe the structure search in Section B of this document.

Note that, unlike NGBoost, the natural gradient is not used in the optimization of the probabilistic neural networks to account for the geometry of the distribution space.

B Machine Learning Model Details

We cease the model fit when the validation score has not improved for 50 iterations. For the neural network approach, an iteration is one *epoch* which is a full pass of the training dataset. For boosting approaches, an iteration is the growth of one base learner $f^{(m)}$ as explained in Section 2.1. If early stopping does not occur, we specify a maximum of 1000 iterations, however in our results this was only reached in the application section when using the GB method. The validation score used is the same as the scoring rule or loss for the method, i.e., root mean squared error for skGB, negative log-likelihood for all other methods.

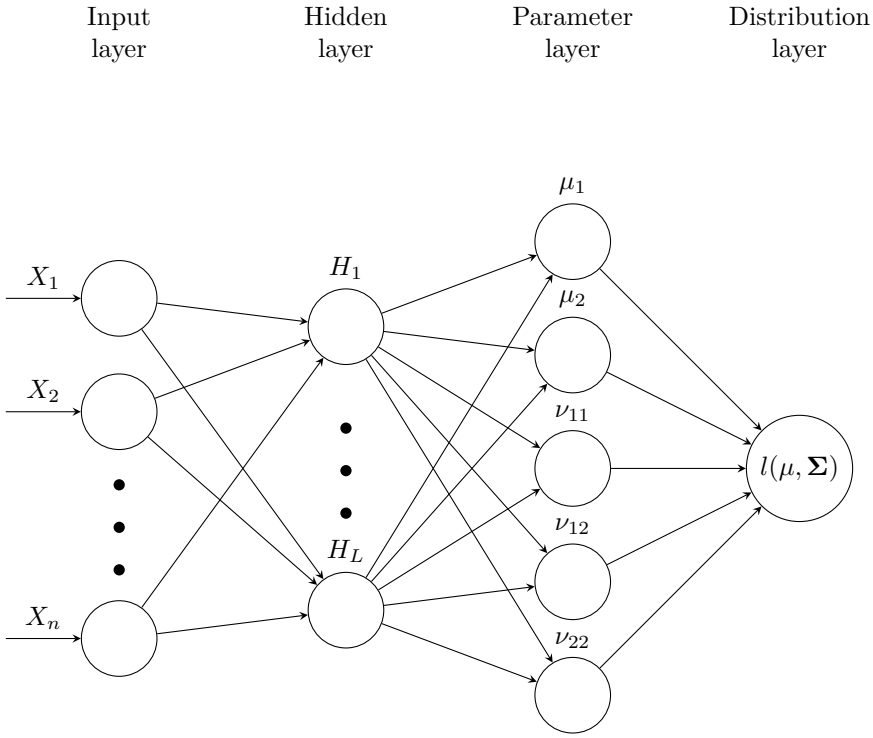

Figure S1: Neural Network structure with one hidden layer with L neurons and a 2 dimensional output.

Boosting Grid Search In our reported results, the boosting methods (NGB, GB, skGB, and Indep NGB) all used the same base learner, an sklearn decision tree, resulting in a direct mapping between hyper-parameters. Hence, we carried out the same grid search for these four models. We conducted a grid search over the following sets:

- Max depth in $\{8, 15, 31, 64\}$.
- Minimum data in leaf in $\{1, 15, 32\}$.

which resulted in 12 hyper-parameter sets for each method, and all other parameters were left at the default other than the learning rate. We did not tune the learning rate, we used a learning rate of 0.01 for the simulation and 0.1 for the application. The larger learning rate was chosen for the application to allow us to conduct replicated fits in a reasonable amount of time.

Neural Network grid search We carried out the following grid search for the neural network architecture in both the simulation example and application in Sections 4 and 5. We considered the following structures for the hidden layers in the grid search:

- A 20 unit layer
- A 50 unit layer
- A 100 unit layer (best for simulation when $N \in \{1000, 3000, 5000, 10000\}$)
- A 20 unit layer followed by another 20 unit layer
- A 50 unit layer followed by another 20 unit layer. (best for simulation when $N \in \{500, 8000\}$)
- 100 units followed by another 20 unit layer (best for the application).

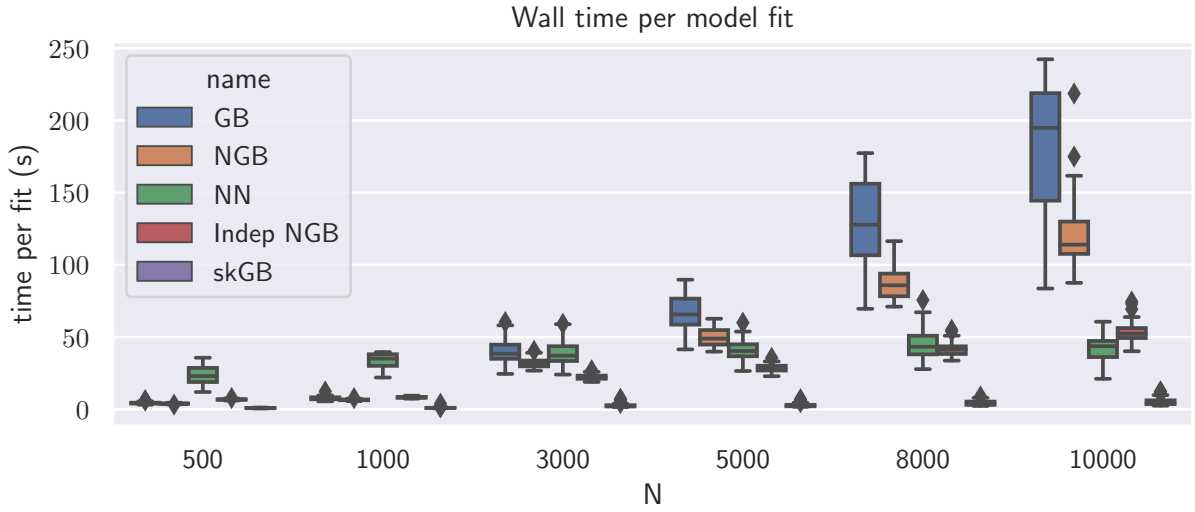

Figure S2: Wall Times per model fit over the 50 replications of Section 4. Box plot drawn in the standard way (boxes being a 50% inter-quartile range, with lines drawn at median and at the minimum or maximum excluding 'outliers'). NN model fits were given access to 10 threads, all other methods ran on a single thread. The plotting times are for a single model fit. It does not show the times relating to the grid search carried out for the NN method.

After each layer, we applied a RELU activation on each node aside from the output nodes. A fixed batch batch size of 256 was used. For the simulation, a learning rate of 0.01 worked well for the Adam optimizer. In the application, we added an extra parameter into the grid search, a learning rate of 0.001 and 0.01. Adam's other parameters were left as the defaults in tensorflow. For the application, the best learning rate and structure pair was 0.001 and a 100 unit layer followed by a 20 unit layer respectively.

Timings & Computation All model fits were run on an internal computing cluster using only CPUs. We note that computational time was not a key consideration in this work. If it was of key importance, using a more efficient base learner for all boosting based approaches would be advisable for large N, such as LightGBM (Ke et al., 2017) or XGBoost (Chen and Guestrin, 2016).

Simulation We show the wall times for the simulation which we ran for Section 4 in Figure S2. Generally, NGB and GB are the slowest and skGB is fastest.

Application We did not explicitly time each model fit in the application, hence we only give rough estimates based on the total running time for the best selected models from the grid search. The neural networks were given access to 15 CPU threads. All other methods were limited to 1 CPU thread, the difference between wall time and CPU time was negligible for all boosting approaches, hence we only report the wall time for those here. The estimated fitting times are shown below (reported as total time for the 10 random replications divided by 10):

- NGB: 1.45 hours
- GB: 6 hours
- skGB: 1.3 hours
- NN: 0.7 hours wall time, 2.1 hours CPU time.

The GB method was the slowest method. This is because early stopping did not occur, meaning the algorithm would run for the full 1000 boosting iterations. In contrast, for NGB, early stopping usually occurred around 250 boosting iterations, which reduced the run time.

We reiterate that these times are rough estimates. Both the simulation and the application were run on a cluster with various nodes with shared resources which the model fitting would be assigned to. For the timings shown in the list above, the NN approach was fit on a node with a 2.6GHz AMD Opteron Processor 6238 and the NGB, GB and skGB models were fit on a 2.30GHz Intel Xeon CPU E5-2699 v3.

C Extended Simulation Results

We now report the metrics introduced in Section 5 for the simulation study in Section 4. These were omitted from the main manuscript due to page length considerations. These are shown in Table S1. We can see that the RMSE metric is very poor for the GB method, which explains the poor NLL and KL divergence metrics for GB. With all methods, NLL and KL divergence show similar patterns, however, if we used the NLL metric it would choose NN for lower values of N over NGB.

We give the results for the unaltered simulation setup of Williams (1996) in Table S2. As noted in the main paper, we see that NN does best for $N \in \{500, 1000, 3000\}$ and NGB does best for $N \in \{5000, 8000, 10000\}$ according to the KL divergence from the truth metric. Moreover, we note GB does not do as poorly as in Table S1. We believe this is because the mean is fit better as can be seen in the RMSE metric. Otherwise, the general patterns seen in both Tables S1 and S2 are very similar.

D Sample Code Snippet

To emphasize how straightforward it is to use the model we show a minimal code snippet below to show how the package can be used:

```
# pandas is used to read the data
import pandas as pd

# NGBoost is the package for prediction
import ngboost

data = pd.read_hdf("path to dataset")

# Split into covariates X:
coordinates = data[["lon", "lat"]]
# and response Y:
velocities = data[['u', 'v']]

# Initiate the model and specify the response is 2 dimensional
# n_estimators should be larger in practice.
multivariate_model = ngboost.NGBoost(
    Dist=ngboost.distns.MultivariateNormal(2),
    n_estimators=10
    )
#Fit the ensemble
multivariate_model.fit(X=coordinates, Y=velocities)

# Obtain the predicted distribution at the coordinates
predicted_distributions = multivariate_model.pred_dist(coordinates)

# The predicted mean and covarince arrays respectively:
predicted_distributions.loc # Shape N x 2
predicted_distributions.cov # Shape N x 2 x 2
```

Table S1: Metrics used in Section 5, on the simulation run in Section 4. We also include the KL divergence from Table 1 in the paper for comparison (with one less decimal point shown here to allow the table to fit on the page). The average over the 50 replications is reported. Standard error estimates reported after ±.

Metric	N	NGB	Indep NGB	skGB	GB	NN
KL div	500	**0.56 ± 0.02**	1.63 ± 0.04	17.19 ± 0.30	126.23 ± 2.58	1.28 ± 0.55
	1000	**0.26 ± 0.00**	1.15 ± 0.02	17.96 ± 0.27	114.11 ± 1.62	0.32 ± 0.02
	3000	**0.11 ± 0.00**	0.88 ± 0.01	19.61 ± 0.25	97.68 ± 1.40	0.15 ± 0.00
	5000	**0.08 ± 0.01**	0.88 ± 0.01	20.31 ± 0.17	90.10 ± 1.29	0.13 ± 0.01
	8000	**0.05 ± 0.00**	0.87 ± 0.01	20.61 ± 0.17	79.01 ± 1.17	0.10 ± 0.00
	10000	**0.04 ± 0.00**	0.83 ± 0.01	20.55 ± 0.15	74.80 ± 1.19	0.13 ± 0.00
NLL	500	1.35 ± 0.02	1.27 ± 0.01	2.05 ± 0.01	2.49 ± 0.01	**1.05 ± 0.02**
	1000	1.02 ± 0.01	1.10 ± 0.01	1.92 ± 0.01	2.25 ± 0.01	**0.89 ± 0.01**
	3000	0.84 ± 0.01	1.02 ± 0.01	1.90 ± 0.01	2.01 ± 0.01	**0.83 ± 0.01**
	5000	**0.79 ± 0.01**	0.98 ± 0.01	1.87 ± 0.01	1.89 ± 0.01	0.81 ± 0.01
	8000	**0.78 ± 0.01**	0.97 ± 0.01	1.87 ± 0.01	1.80 ± 0.01	0.80 ± 0.01
	10000	**0.77 ± 0.01**	0.97 ± 0.01	1.88 ± 0.01	1.75 ± 0.01	0.82 ± 0.01
RMSE	500	0.66 ± 0.00	0.65 ± 0.00	0.65 ± 0.00	1.91 ± 0.01	**0.65 ± 0.00**
	1000	0.63 ± 0.00	0.63 ± 0.00	0.63 ± 0.00	1.85 ± 0.01	**0.62 ± 0.00**
	3000	0.63 ± 0.00	0.62 ± 0.00	0.63 ± 0.00	1.76 ± 0.01	**0.62 ± 0.00**
	5000	0.62 ± 0.00	0.62 ± 0.00	0.62 ± 0.00	1.70 ± 0.01	**0.62 ± 0.00**
	8000	0.62 ± 0.00	**0.62 ± 0.00**	0.62 ± 0.00	1.65 ± 0.01	0.62 ± 0.00
	10000	0.62 ± 0.00	**0.62 ± 0.00**	0.62 ± 0.00	1.64 ± 0.01	0.62 ± 0.00
90% PR area	500	1.90 ± 0.02	2.59 ± 0.03	4.27 ± 0.05	8.57 ± 0.10	3.02 ± 0.07
	1000	1.97 ± 0.01	2.52 ± 0.02	4.54 ± 0.04	7.74 ± 0.07	2.64 ± 0.02
	3000	2.22 ± 0.01	2.60 ± 0.01	4.96 ± 0.03	7.21 ± 0.06	2.58 ± 0.02
	5000	2.30 ± 0.01	2.65 ± 0.01	5.09 ± 0.02	6.89 ± 0.05	2.61 ± 0.03
	8000	2.36 ± 0.01	2.69 ± 0.01	5.14 ± 0.02	6.49 ± 0.05	2.65 ± 0.03
	10000	2.38 ± 0.01	2.69 ± 0.01	5.19 ± 0.02	6.37 ± 0.05	2.63 ± 0.03
90% PR cov	500	0.76 ± 0.00	0.83 ± 0.00	0.81 ± 0.00	0.84 ± 0.00	0.88 ± 0.00
	1000	0.80 ± 0.00	0.85 ± 0.00	0.83 ± 0.00	0.86 ± 0.00	0.89 ± 0.00
	3000	0.85 ± 0.00	0.86 ± 0.00	0.85 ± 0.00	0.88 ± 0.00	0.89 ± 0.00
	5000	0.87 ± 0.00	0.87 ± 0.00	0.86 ± 0.00	0.89 ± 0.00	0.90 ± 0.00
	8000	0.87 ± 0.00	0.88 ± 0.00	0.87 ± 0.00	0.90 ± 0.00	0.90 ± 0.00
	10000	0.88 ± 0.00	0.88 ± 0.00	0.86 ± 0.00	0.90 ± 0.00	0.90 ± 0.00

References

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3146–3154.

Sützle, E. A. and Hrycej, T. (2005). Numerical method for estimating multivariate conditional distributions. *Computational statistics*, 20(1):151–176.

Williams, P. M. (1996). Using neural networks to model conditional multivariate densities. *Neural computation*, 8(4):843–854.

Table S2: Same as Table S1 except we use the same simulation as Williams (1996), i.e. without adding the $+x$ and $-x^2$ terms of equation (3) in the main paper. Note that the difference between NGB and GB or NN is not as large as in the original table. All values rounded to two decimal places.

Metric	N	NGB	Indep NGB	skGB	GB	NN
KL div	500	0.96 ± 0.03	2.27 ± 0.06	19.84 ± 0.39	2.16 ± 0.06	**0.40 ± 0.04**
	1000	0.46 ± 0.01	1.44 ± 0.03	20.08 ± 0.29	1.28 ± 0.03	**0.26 ± 0.04**
	3000	0.18 ± 0.01	1.06 ± 0.02	20.43 ± 0.19	0.75 ± 0.01	**0.13 ± 0.02**
	5000	**0.11 ± 0.00**	0.96 ± 0.02	20.65 ± 0.20	0.65 ± 0.01	0.13 ± 0.02
	8000	**0.07 ± 0.00**	0.91 ± 0.01	20.75 ± 0.17	0.55 ± 0.01	0.11 ± 0.02
	10000	**0.06 ± 0.00**	0.92 ± 0.02	21.10 ± 0.18	0.54 ± 0.01	0.13 ± 0.02
NLL	500	1.24 ± 0.01	1.25 ± 0.01	1.96 ± 0.01	1.27 ± 0.01	**0.93 ± 0.02**
	1000	1.03 ± 0.01	1.12 ± 0.01	1.91 ± 0.01	1.12 ± 0.01	**0.86 ± 0.01**
	3000	0.86 ± 0.01	1.02 ± 0.01	1.88 ± 0.01	0.98 ± 0.01	**0.82 ± 0.01**
	5000	0.82 ± 0.01	1.00 ± 0.01	1.86 ± 0.01	0.94 ± 0.01	**0.81 ± 0.01**
	8000	**0.78 ± 0.01**	0.97 ± 0.01	1.86 ± 0.01	0.90 ± 0.01	0.80 ± 0.01
	10000	**0.78 ± 0.01**	0.97 ± 0.01	1.86 ± 0.01	0.90 ± 0.01	0.81 ± 0.01
RMSE	500	0.64 ± 0.00	0.64 ± 0.00	0.63 ± 0.00	**0.63 ± 0.00**	0.64 ± 0.00
	1000	0.63 ± 0.00	0.63 ± 0.00	0.63 ± 0.00	**0.62 ± 0.00**	0.63 ± 0.00
	3000	0.62 ± 0.00	0.62 ± 0.00	0.62 ± 0.00	**0.62 ± 0.00**	0.62 ± 0.00
	5000	0.61 ± 0.00	0.61 ± 0.00	0.61 ± 0.00	**0.61 ± 0.00**	0.62 ± 0.00
	8000	0.62 ± 0.00	0.62 ± 0.00	0.62 ± 0.00	**0.61 ± 0.00**	0.62 ± 0.00
	10000	0.62 ± 0.00	0.62 ± 0.00	0.62 ± 0.00	**0.62 ± 0.00**	0.62 ± 0.00
90% PR area	500	2.03 ± 0.02	2.61 ± 0.03	4.68 ± 0.06	2.72 ± 0.03	2.79 ± 0.05
	1000	2.11 ± 0.02	2.53 ± 0.02	4.86 ± 0.05	2.65 ± 0.02	2.79 ± 0.04
	3000	2.31 ± 0.01	2.65 ± 0.01	5.10 ± 0.03	2.69 ± 0.01	2.71 ± 0.03
	5000	2.35 ± 0.01	2.67 ± 0.01	5.14 ± 0.02	2.70 ± 0.01	2.69 ± 0.02
	8000	2.41 ± 0.01	2.70 ± 0.01	5.20 ± 0.02	2.71 ± 0.01	2.73 ± 0.03
	10000	2.42 ± 0.01	2.72 ± 0.01	5.24 ± 0.02	2.72 ± 0.01	2.70 ± 0.03
90% PR cov	500	0.78 ± 0.00	0.83 ± 0.00	0.84 ± 0.00	0.84 ± 0.00	0.88 ± 0.00
	1000	0.81 ± 0.00	0.84 ± 0.00	0.85 ± 0.00	0.85 ± 0.00	0.89 ± 0.00
	3000	0.86 ± 0.00	0.87 ± 0.00	0.86 ± 0.00	0.88 ± 0.00	0.89 ± 0.00
	5000	0.86 ± 0.00	0.87 ± 0.00	0.87 ± 0.00	0.88 ± 0.00	0.90 ± 0.00
	8000	0.88 ± 0.00	0.88 ± 0.00	0.87 ± 0.00	0.89 ± 0.00	0.90 ± 0.00
	10000	0.87 ± 0.00	0.88 ± 0.00	0.87 ± 0.00	0.89 ± 0.00	0.90 ± 0.00