CVNetica–A cross-validation package driving Netica with Python

Michael N. Fienen

US Geological Survey, Wisconsin Water Science Center, 8505 Research Way, Middleton WI 53562 USA

+001 (608) 821-3894

Nathaniel G. Plant

US Geological Survey, St. Petersburg Coastal and Marine Science Center, 600 Fourth Street South, St. Petersburg, Florida, 33701, USA.

Abstract

Bayesian networks (BNs) are powerful tools for probabilistically simulating natural systems and emulating process models. Cross validation is an important technique to avoid overfitting that can result from overly complex BNs. Overfitting results in a reduction of true predictive skill. Formal cross-validation for BNs has been discussed but rarely implemented. The lack of widespread cross-validation is due partly to a lack of software tools designed to work with available BN packages. CVNetica is an open-source package written in Python that extends the Netica software package to perform cross-validation and to read, rebuild, and learn BNs from data. Optimal BN complexity can be identified through exploration of BN complexity using predictive skill as a performance metric. Insights gained from cross-validation and implications on predictive versus descriptive skill are illustrated with two examples: a data-driven oceanographic application; and a model-emulation application.

Keywords: Cross-validation; Bayesian networks; uncertainty; probability; Python; Netica; prediction

---

Email addresses: `mnfienen@usgs.gov` (Michael N. Fienen), `nplant@usgs.gov` (Nathaniel G. Plant)

Review Disclaimer

This manuscript draft is preliminary, deliberative, and predecisional, released for the sole purpose of peer review. This document should be treated as confidential by journal staff and reviewers until a final version is approved for publication by the United States Geological Survey. This disclaimer will be removed if and when a publication version of the manuscript is approved.

1. Introduction

Over the past two decades, the use of Bayesian Networks (BN; Jensen and Nielsen, 2001) has increased greatly, in large measure due to the availability of commercial software packages such as Netica (Norsys Software Corp., 2013) and Hugin (Hugin Expert A/S, 2013) among many others. Applications in water resources have included groundwater management (Martin de Santa Olalla et al., 2007, Molina et al., 2010, 2013), and model emulation (Plant and Holland, 2011a,b, Fienen et al., 2013). This builds on a history of applications in national security, economics, and ecology.

An important topic that is not always discussed in the literature is that applications of BNs need formal tests and validation of prediction performance (Chen and Pollino, 2012, Marcot, 2012). Some validation metrics are calculable by the commercial software packages, but substantial gaps in capabilities remain. Fortunately—at least in the case of Netica—an application programming interface (API) exists with versions in multiple programming languages. To create a toolbox of performance metrics, we used Python (Rossum, 1995) with the Netica C APIs. These APIs expose most of Netica's functionality, through functions, to external programming. Among the languages available, C was chosen because one of our goals was to interface with Python 2.7.6 (Rossum, 1995), Numpy 1.8 and Scipy 0.13.2 (Jones et al., 2014). We discuss the technical challenges associated with running C APIs using Python and describe the toolbox of validation metrics included in this work.

Building on techniques introduced by Fienen et al. (2013), we developed tools addressing two fundamental questions of Bayesian network performance: how does predictive performance compare with descriptive calibration quality?; and how does the complexity of the underlying network impact predictive and descriptive performance? Cross-validation is used to answer both questions, and the number of bins per node is used as a metric of complexity to answer the second. These specific questions are evaluated in this work but our framework allows for consideration and analysis of other validation metrics and techniques beyond those presented here.

2. Bayesian Networks

This background section on Bayesian networks (BNs) is derived from Fienen et al. (2013). A Bayesian network is a directed acyclic graph (Korb and Nicholson, 2004), composed of nodes and edges. Nodes represent variables whose parameter values may includeBoolean, discrete states, or, for continuous variables, discrete ranges that are discretized into bins. Edges form the connections between nodes and represent a correlated connection between the properties represented by the nodes. The entire catalog of these correlations make up conditional probability tables (CPTs). In a predictive context, nodes can be thought of as either input (e.g. forcing) or output (e.g. response) , although this distinction is not a sharp one as the correlations learned by the BN are ambivalent with respect to direction. Nodes can also be intermediate if they act as constraints or model coefficients.

An example of a BN created and visualized using Netica is presented in figure 1.

Calculations are made using the BN based on conditional probabilities using

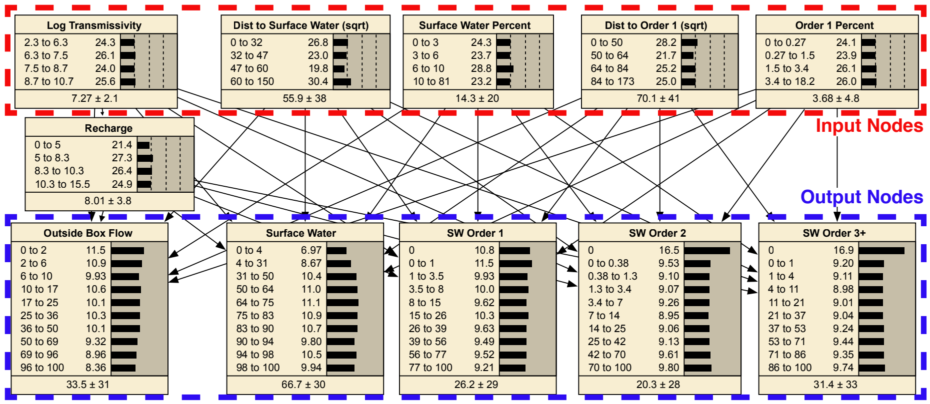

Figure 1: Example groundwater application of a Netica BN showing input (outlined in a red box) and output (outlined in a blue box) nodes, edges (black lines) and, in this case, a single intermediate node (recharge).

Bayes' Theorem

$$p\left(F_i|O_j\right) = \frac{p\left(O_j|F_i\right)p\left(F_i\right)}{p\left(O_j\right)} \tag{1}$$

where $p\left(F_i|O_j\right)$ is the posterior (updated) probability of a forecast (F_i) given (conditional on) a set of observations (O_j); $p\left(O_j|F_i\right)$ is the likelihood function, $p\left(F_i\right)$ is the prior probability of the forecast, and $p\left(O_j\right)$ is a normalizing constant. The posterior probability reflects an updating that is achieved by considering the entire chain of conditional probabilities of all bins connected to the node representing F_i. The likelihood function represents the probability that the observations (O_j) would be observed given that the forecast was perfectly known. This is a metric of the ability of the BN to function as a forecasting device and imperfections in such forecasts are a function of epistemic uncertainty. Epistemic uncertainty includes uncertainty due to model imperfection, data errors, data paucity, and other sources. The prior probability of the forecast, $p\left(F_i\right)$, is the probability of a forecast without the benefit of updated observations and the BN (or a process model or other experiment). $p\left(F_i\right)$ may be calculated by using expert knowledge, or may be assumed relatively uninformative to make the entire process as objective as practical (similar to an ignorance prior (Jaynes and Bretthorst, 2003)). A common prior often used in BNs is the division of a node into bins of equal probability. This results in bins of equal probability or ''belief'' although it is not exactly an ignorance prior because the probability mass in each bin may differ due to variable bin widths. It is possible to evaluate the contribution to all uncertainty values calculated by the BN by expressing the uncertainty in the prior probabilities. In Figure 1, the horizontal bars correspond to relative probabilities associated with bins outlined by the numbers listed to the left of them. These bars form a histogram and are referred to as "belief bars."

Once a system is cast in a BN, new observations of system state are applied and propagated through the BN using Bayes' theorem such that all forecasts made in the model are contingent upon the specific observations of system state. In other words, each forecast is associated with a specific configurations of system state. In our approach, observations are indicated by selecting a bin and forcing the probability of a value in the node to be 100%. This implies that observational uncertainty does not exceed the width of the specified bin (for continuous variables) or that the discrete or Boolean state is known perfectly. (It is straightforward to relax this assumption to consider inputs that are uncertain.) When this operation is performed, the Bayesian update propagates in each direction among nodes that are d-connected (Jensen and Nielsen, 2001), updating the probabilities regardless of causal direction. In this way, correlations are expressed as well as causal responses. By selecting a suite of observations of state, the BN acts like a transfer function by providing an estimate of the forecast of interest and associated uncertainty.

A key piece of a priori information is the establishment of edges connecting the nodes. Edges should reflect a cascade of causality grounded in an understanding of the underlying process being modeled. If multiple processes from different models are to be linked, the selection of edge relationships defines the linkage. While machine learning can be used to teach a BN which parameters are connected to each other and to outputs, we adopt a Bayesian approach in which expert system understanding is used to specify these connections through the identification of nodes and edges. In this way, the BN honors the physical conditions known by the modeler and these are incorporated as soft knowledge.

In Figure 1, arrows on the edges indicate the direction of causal dependence. When all nodes are d-connected, the direction of the edge arrows serve no purpose. However, in the context of d-separation, the direction of causality has important ramifications on the propagation of uncertainty from observations to forecasts.

When computational conditions and problem size permit, a conditional probability table (CPT) can be created that directly enumerates the conditional probabilities of all nodes in the BN. This becomes impractical rapidly, however, because the size of the CPT scales on the order of $n \times d^{k+1}$ where n is the number of nodes, d is the number of bins, and k is the number of parents for a node. In the case where full enumeration is impractical due to this rapid increase in computational expense with complexity, an iterative expectation-maximization (EM) algorithm is used (Dempster et al., 1977) to calculate approximate probabilities and maximum-likelihood values for the BN without full enumeration of the CPT. The EM algorithm iterates between estimating the maximum log likelihood of the function and finding the set of parameters resulting in that maximum log likelihood.

3. Cross validation tool

CVNetica is a Python module that performs cross-validation and calculates other performance metrics on BNs created with the Netica software package.

Netica is a commercial package with more power than open-source alternatives. However, CVNetica is open-source and freely available. The APIs for Netica are described in (Norsys Software Corp., 2010) and are provided as a dynamic linked library (DLL) for Windows. Static libraries are also available for Macintosh and *nix platforms, but to use them with Python, dynamic interface wrappers would be necessary in addition to the Python function wrappers written in CVNetica.

The core functionality of CVNetica is based around the concept of using cross-validation (Hastie et al., 2009, Marcot, 2012, Fienen et al., 2013) metrics to asses the quality of predictions made by a BN. In k-fold cross validation used in this work, the calibration dataset is, randomly without replacement, divided into k folds or partitions where k typically is between 2 and 10. For each fold, the BN is trained using the dataset without the data in the fold, then the BN is used to make predictions on the left-out data. In this way, performance of the BN is evaluated on data not used in calibration to simulate performance in true future prediction. Several performance metrics can be used for this purpose, as discussed in Norsys Software Corp. (2013), Plant and Holland (2011a), and Fienen et al. (2013). In this work, we will focus on skill

$$sk = \left[1 - \frac{\sigma_e^2}{\sigma_o^2}\right] \tag{2}$$

where σ_e^2 is the mean squared error between observations and BN predictions, and σ_o^2 is the variance of the observations (Gutierrez et al., 2011, Plant and Holland, 2011a, Weigend and Bhansali, 1994). Skill is evaluated by comparing BN predictions to observations with a value of unity indicating perfect correspondence and a value of zero indicating substantial discrepancy between BN predictions and observations.

CVNetica also reports log loss, error rate, experience, quadratic loss, mutual information (entropy), variance reduction (sensitivity) all of which are described by Norsys Software Corp. (2013, 2010). Expected values are reported either as mean or most likely (ML). For ML values, the value corresponding to the center of the bin with the highest predicted probability is reported. The mean values, are computed as the product of the bin centers and the probability in each bin, consistent with a typical expectation operation.

By evaluating skill over both the calibration data sets and prediction data sets, the value of a BN as a descriptive or predictive tool can be evaluated. As BN complexity increases, so does the calibration sk and with sufficient complexity, calibration sk approaches unity (perfection). However, greater descriptive value in a BN comes at a cost in predictive value. This is the classic condition of overfitting as cast in the context of information theory by Fienen et al. (2013).

One way to systematically evaluate BN complexity is to adjust the number of bins for each node with more bins meaning a greater level of complexity. CVNetica has the capability to make this type of analysis efficient by allowing the user to specify an original BN and a configuration of bins for each node. CVNetica then builds a new BN with the requested number of bins and assigning equiprobable prior distributions for each bin. In Fienen et al. (2013) the number of bins was assumed the same for each node. Using CVNetica the number of bins in each node can be varied independently to allow for exploration of various assumptions of complexity. The user can also establish scenarios manually varying the number and nature of edges connecting nodes and even the number of nodes themselves. A group of these scenarios is defined by CVNetica as a "set." Each set can be evaluated as a batch and then tabulated and graphical results are generated of performance metrics across the sets.

3.1. Details about program structure

There are two levels at which CVNetica performs. At the highest level, a script in `CV_driver.py` performs the cross validation protocol described below.

This script is driven by an XML-based configuration file and should generally require minimal editing, save for identifying the configuration file to use in the `parfile` variable name. At a lower level, `pythonNeticaTools.py` provides the `pyneticaTools` class that interacts with the Netica DLL via wrappers around many essential Netica functions. Examples of how these methods work are discussed in Section 4.1. At an intermediate level, `pythonNetica.py` provides the `pynetica` class that combines several Netica functions for tasks such as starting a Netica environment, rebinning nodes, and other intermediate level tasks.

3.1.1. Cross Validation Driver

The `CV_driver.py` script drives a cross-validation exercise specified in the XML based configuration file (Figure 2). If no rebinning is requested ($<$ `rebin_flag` $>$ `False` $<$ `/rebin_flag` the BN specified in the `baseNET` element is used for analysis along with the casefile identified by the `baseCAS` element and metrics of performance. If the `rebin_flag` element is True, then the nodes from the BN identified in the `originalNET` element are rediscretized using the information on rebinning provided at the end of the input file. For each node listed, if `numbins` $>$ `0` the node is discretized into bins `numbins` bins of equal probability. In the special case where `numbins` $=$ `0`, the node is not rediscretized but it is used either as input or response as described by the `input` and `response` elements above. This special case allows for other discretization strategies (such as thresholds) to be implemented for nodes that are to be treated as input or response nodes but without equiprobable discretization. Nodes that are not identified as either input or response should not have `node` elements provided and are unaltered by CVNetica in the analysis.

If the `CVflag` element is False, only a single run using all the data in the `baseCAS` file and the BN identified in the `baseNET` is performed and metrics are calculated. The predictions for each configuration of input are recorded in a compressed Python pickle file.

If the `CVflag` element is True, then k-fold cross validation is performed using the number of folds indicated in the `numfolds` element. For each fold, $\frac{n}{k}$ (where n is the total number of data points and k is the number of folds) data points are separated from the rest of the data points to be left out of the calibration, selecting from a randomized list such that each fold samples across the training set to span spatial or temporal trends or patterns. The BN is then retrained on the $n - \frac{n}{k}$ retained data and metrics of performance are calculated for both the left out data (referred to as "validation") and the training data (referred to as "calibration").

4. Working with Ctypes

The Netica software provides APIs for accessing and using the functions within it. Several versions of these APIs are available as precompiled libraries. To interface with Python, the C programming language APIs can be interfaced using the `ctypes` module which is built-in to Python 2.5+. The `ctypes` module enables the use of functions from a dynamic library of C code (a DLL on Windows) in the Python environment. In addition to making the functions accessible, some translation of variables is required–for example, C often refers to data using pointers whereas Python does not explicitly do so. C functions often return pointers to memory space of the resulting arrays so `ctypes` must be used to read the correct amount of data from memory to populate an array for further use in Python.

CVNetica provides Python functions wrapped around Netica C functions and helper functions to translate data to and from the Python environment. In the remainder of this section, the main aspects of interfacing with the Netica APIs are discussed in general terms. These examples use code snippets from

```xml
<data>
     <control_data>
          <baseNET>glacial_bins4_5_0.neta</baseNET> <!-- name of main .neta file -->
          <baseCAS>glacial.cas</baseCAS>            <!-- name of main data file -->
          <rebin_flag>True</rebin_flag>             <!-- flag determining if
                                                         rebinning should be performed -->
          <originalNET>glacial.neta</originalNET>   <!-- original .neta file providing node
                                                         structure and bins of numbins=0 below-->
          <pwdfile>mikeppwd.txt</pwdfile>           <!-- name of Netica license file -->
     </control_data>
     <kfold_data>
          <CVflag>True</CVflag>    <!-- flag indicating if k-fold cross validation
                                        should be carried out -->
          <numfolds>10</numfolds>  <!-- number of folds for cross validation -->
     </kfold_data>
     <scenario>
          <name>glacial_set1</name>     <!-- scenario name for output files -->
          <input>sqrt_SW_MIN</input>    <!-- input tags identify nodes as used for input -->
          <input>sqrt_RIVMIN1</input>
          <input>PCTORD1</input>
          <response>EXT_FLOW</response> <!-- response tags identify nodes as used for output -->
          <response>SW_SRC</response>
     </scenario>
     <sensitivity>
          <report_sens>True</report_sens> <!-- flag indicating if Netica sensitivity and
                                               other built-in metrics should be reported -->
     </sensitivity>
     <learnCPTdata>
          <voodooPar>100</voodooPar>       <!-- fitting parameter for learning CPTs -->
          <useEM>True</useEM>              <!-- use EM to learn CPTs if True. Else, use
                                                incporporate casefile method -->
     </learnCPTdata>
     <rebinning>
     <!-- if rebin_flag is True, then bin_setup.py will read in the
          rebin_name to write out the rebinned .neta file and will
          use the newbins information for that purpose.
          Nodes will be rediscretized into numbins equiprobable bins.
          Special case when numbins = 0, the node is not redisretized from originalNET -->
               <newbins>
                    <node numbins="4">sqrt_SW_MIN</node>
                    <node numbins="4">sqrt_RIVMIN1</node>
                    <node numbins="4">PCTORD1</node>
                    <node numbins="5">EXT_FLOW</node>
                    <node numbins="0">SW_SRC</node>
               </newbins>
     </rebinning>
</data>
```

Figure 2: Example XML configuration file for defining problem parameters. Blue text identifies syntax of element names, green text indicates comments in the file, and bold black text indicates element values. In the special case of the `node` element, an attribute (numbins) is indicated in red text.

the CVNetica codebase. Further documentation about `ctypes` is available from the official documentation (http://docs.python.org/2/library/ctypes.html).

4.1. Accessing the DLL

The first task when accessing the Netica DLL is to make the functions available to Python by assigning the DLL to an object. Note that the filename is not in quotes, nor is the `.dll` extension required. The `ctypes` module is imported as `ct` so in future code descriptions, `ct.<>` implies a method or property from `ctypes`.

```
import ctypes as ct
self.n = ct.windll.Netica
```

After this, `self.n` is an object with all of the Netica API functions available. To call a function from the DLL, the function name is dereferenced from `self.n` and in CVNetica, a wrapper function is created as an interface to the Netica function. In the following example, the Netica function to be called is `EnterNodeValue_bn`. This function takes two arguments as indicated in the function definition by Netica: `void EnterNodeValue_bn (node_bn* node, double value)` (Norsys Software Corp., 2010). The two arguments are of the custom C type defined by Netica as `node_bn* node` and a double-precision float `double value`. A wrapper around this function must then make type conversions as appropriate. The CVNetica variable `cnode` was returned by a Netica function, so it is already of the type required (a pointer). However, the CVNetica variable `cval` is a Python float and must be converted to a C double using a `ctypes` conversion.

```
def EnterNodeValue(self,cnode,cval):
        self.n.EnterNodeValue_bn(cnode,ct.c_double(cval))
        self.chkerr()
```

The chkerr method polls the Netica DLL for current error status and, if an error is encountered, kills CVNetica and displays the error from Netica to standard error.

4.2. Exchanging information with the Netica DLL

The functions in Netica can accept a variety of argument types. In the pyneticaTools class, methods that function as wrappers around Netica functions are written. The names are the same as the Netica functions with the _bn, _cs, and _ns suffixes removed. This class is not specific to cross validation applications and is meant to also serve as a starting point for other applications in which Netica functions must be used in Python.

The easiest type is a pointer to an object returned by another Netica function. In this case, a Python variable represents the pointer–just a memory address–so no conversion is necessary. For single Python floats and ints, the conversions are ct.c_double(cval) and ct.c_int(cval), respectively, where cval is the Python variable.

Some Netica functions return a double value but also write another result to memory at a location indicated by a pointer passed to the function. An example is GetNodeExpectedValue_bn. The structure of this function in C is

```
double GetNodeExpectedValue_bn (node_bn* node,
          double* std_dev, double* x3, double* x4)
```

where the returned value is the expected value (double precision) of the node identified by node_bn*, the standard deviation is written to the memory location identified by the pointer double* std_dev, and x3 and x4 are NULL pointers reserved for future implementation. To collect the main returned value of the function, we must set restype of the function–accomplished through making an alias temporary function–and accepting the value as normally with a function.

To make use of the returned second value in Python–the value written to a memory location identified by a pointer–we must pass a `double` variable by reference (in other words, a pointer to the `double`). The Python wrapper for `GetNodeExpectedValue_bn` illustrates this process

```
def GetNodeExpectedValue(self,cnode):
        std_dev = ct.c_double()
        tmpNeticaFun = self.n.GetNodeExpectedValue_bn
        tmpNeticaFun.restype=ct.c_double
        expected_val = tmpNeticaFun(cnode,ct.byref(std_dev),
                                    None,None)
        self.chkerr()
        return expected_val, std_dev.value
```

Some Netica functions return either a character array or a numerical array. In both cases, the C code in Netica returns a pointer to the data. The Python code must, then, read a specified amount of data from that pointer location. Unlike pure Python, it is possible to read off the end of the information starting at the pointer location, so we must also specify the number of values to read from the memory location. Helper functions in `cthelper.py` read the character pointers, and single and double precision pointers. An example of this being used in CVNetica is in the `ReadNodeInfo` method of the `pyneticaTools` class.

5. Example Applications

The CVNetica code was applied to two different applications to evaluate predictive performance and guide the appropriate level of complexity for BN design. The two applications are (1) a data-driven prediction of ocean wave evolution and (2) a model emulation using a BN to make predictions trained on results of a groundwater flow model.

5.1. Data driven ocean waves

Weather forecasting and modeling has achieved sufficiently high accuracy that it is possible to replace observations with models if models are initialized well and have good boundary condition data. However, weather forecasts are not routinely available for periods extending more than a few days ahead, and they become less accurate. We would like to allow the climatological prior information to inform predictions when observations or forecasts are not available or are uncertain. As an example, we would like to predict wave height just offshore of the coast where there are not persistent observations. This could be done with laborious Monte Carlo simulations using models and previous climatology for model initialization and boundary-condition forcing. Or, we could use extant model output or observations to learn both the sensitivity of a specific prediction to changes in boundary conditions and include uncertainty in this sensitivity (the joint correlation) as well as uncertainty in the boundary conditions. This approach has been implemented before using Bayesian networks (Plant and Holland, 2011a,b), but the fidelity of the of the resulting BN models was not examined in detail, other than to note that to maximize the consistency of the BN predictions with new observations, the BN inputs should include input parameter-value or data uncertainties. Were these BN models over-fit to the data?

We explore the level of overfitting with a simplified ocean-wave prediction model based on a BN. The specific BN, illustrated in figure 3, has been used to drive subsequent predictions of morphologic evolution of a man-made sand berm constructed near the Chandeleur Islands (Plant et al., 2014). Here, we simplify the original model, which included information from two wave buoys, one tide gage, and Monte Carlo simulation of a wave-runup model (Stockdon

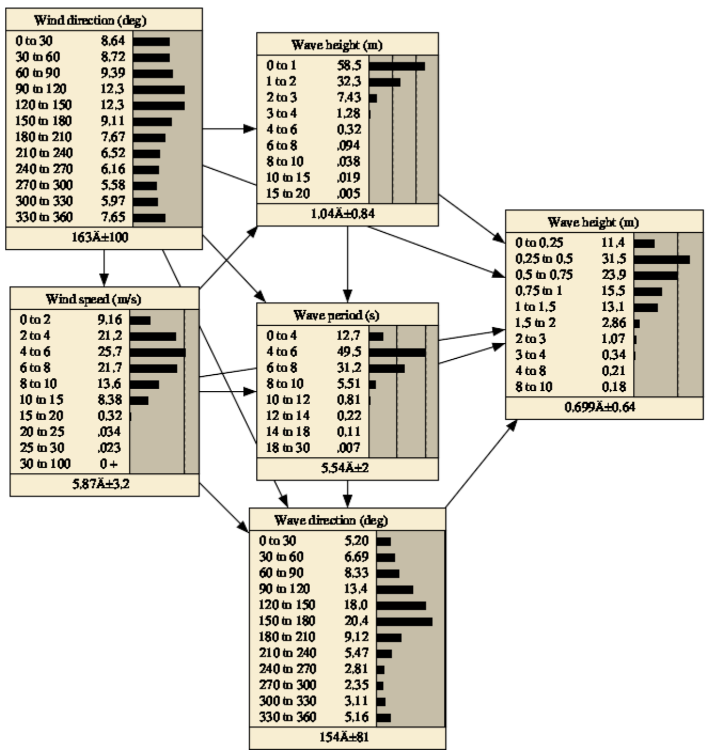

Figure 3: BN for the wave height prediction model.

et al., 2006). The first two columns of network nodes (figure 3) correspond to observations from an offshore buoy (NOAA 42040) collected from 1996 to 2011. The variables are wind speed and direction and wave height, period, and direction. The third column has just one variable–wave height–that was observed at a nearshore buoy between 2000 and 2008. The nearshore buoy was subsequently lost. The BN describing the prior probabilities of each variable and the conditional probabilities among variables was designed to resolve boundary conditions at the offshore location and the prediction at the nearshore location accurately enough to support the morphologic evolution application. While this BN has very good hindcast skill (about 0.8), it is not clear that it has equally good forecast skill and whether fewer probability bins could be retained to give a skillful prediction with optimal numerical efficiency.

Our calibration/validation skill analysis was applied to this net by varying the number of bins in all variables except for the wave heights. We chose to resolve these variables consistently with the original model to ensure that probability predictions spanned a wide range conditions, rather than focusing on the most probable but extremely low wave-height range that was most common (i.e., 1-3 m). The number of bins ranged from 2 to 10 for the remaining variables. The calibration (i.e., hindcast) skill increased for all choices of bin numbers (figure 4). However, the validation skill, averaged over 5 folds, reached its peak value at 4 bins and then decreased dramatically after 6 bins. The optimal bin resolution likely varied for each variable type (wind speed, wind and wave directions, wave period) and this may explain the flattening of the validation curve between 4 and 6 bins, as it is possible that increasing bin resolution was advantageous, adding necessary resolution, for some variables but a disadvantage for others. The rapid decline after 6 bins suggests that none of the variables needed to be better resolved past this point.

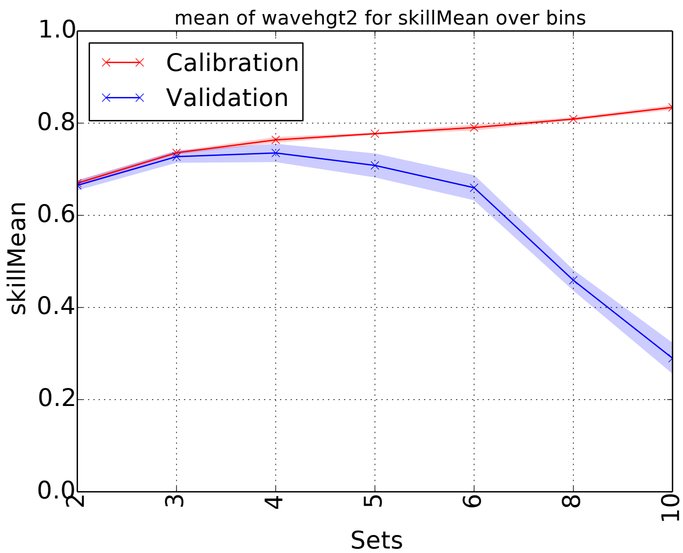

Figure 4: Calibration (descriptive) and Validation (predictive) skill values for various arrangements of bins on the wave prediction BN. For sets identified by a single number, that number of bins was used to discretize all nodes. The color shading indicates the 95% credible interval based on adding and subtracting 2σ to and from the median value of each metric over the 10 folds evaluated.

5.2. Model emulation source of groundwater to wells

In the Great Lakes Region of the United States, understanding the interactions between groundwater and surface water are important inputs to ecological management interests. Specifically, as extraction wells are installed, the baseflow in streams can be reduced and these reductions can affect fish habitat and associated societal and economic concerns (Ruswick et al., 2010, Barlow and Leake, 2012, Watson et al., 2014). An efficient method to determine the source of water to wells has the potential to improve management in the region by quickly screening proposed wells. If the source of water emanating from surface water (either through diversion or depletion) reaches a management threshold, then further management actions may be triggered. Using a numerical groundwater model, managers could conceivably explicitly assess the impact of each proposed well location. But computational run times and technical background may be prohibitive for that task. A more efficient option is model emulation as performed on a groundwater model by Fienen et al. (2013).

In this case–using MODFLOW-USG (Panday et al., 2013)–extraction wells were simulated on multiple staggered grids at sufficient distances that they would not interact in individual model runs. A base case was also simulated without extraction wells and, through superposition, the sources of water to the wells was evaluated and mappable characteristics of each well location were used to create the BN in Figure 1. For further discussion of the model used in this work see Feinstein et al. (tion).

An important question–similar to that evaluated by Fienen et al. (2013)–is what level of complexity provides the best tradeoff between descriptive and predictive power of the BN. Using CVNetica, it was possible to quickly evaluate k–fold cross validation for a variety of combinations of bins in the node arrangement depicted in Figure 1. For the most important response variable–SW_SRC, which is surface water source–Figure 5 depicts the change in both calibration and validation performance for 10-fold cross validation performed over an increasingly complex set of bin configurations. While increasing complexity (e.g., number of bins per node) monotonically improves calibration (description) over the training set, the skill improves at first for validation (prediction) but then degrades dramatically after four bins on the input nodes. In Figure 5, sets identified by a single value indicate the number of bins used to discretize each node. When a second number is present, the first number indicates bins used for the input nodes while the second indicates bins used for the output nodes. Little degradation and possibly a slight improvement in validation skill is seen with increasing output bins for a given complexity of input. This highlights that the main fitting of the BN takes place with respect to input and output complexity is more a matter of convenience than a source of real BN complexity.

6. Discussion and Conclusions

CVNetica is an open-source Python module using the `ctypes` module to drive APIs for the Netica BN software. The purpose is to implement cross-validation techniques for evaluating descriptive (calibration) versus predictive (validation) performance of BNs.

We show that CV provides an objective method for determining when a BN is being overfit to the data. And, it appears that overfitting is likely when the BN is designed to resolve physical processes, either observed or modeled. In the cases presented here, the BN design was guided by distribution of the priors of each variable as well as by the intended application of the BN predictions. For instance, in the ocean wave example, the intended application focused on resolving large storm events that were likely to cause erosion. This guided a a choice of fairly high resolution of the input data a the offshore buoy. While physically consistent, a price needed to be paid for the over-resolved output

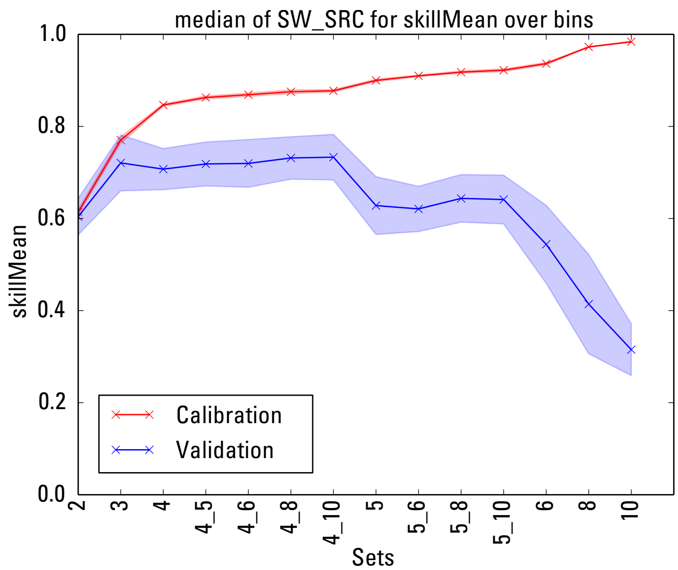

Figure 5: Calibration (descriptive) and Validation (predictive) skill values for various arrangements of bins on the glacial aquifer BN. For sets identified by a single number, that number of bins was used to discretize all nodes. For sets identified by two numbers separated by an underscore, the first and second numbers indicate the number of bins the input and output nodes, respectively, were discretized. The color shading indicates the 95% credible interval based on adding and subtracting 2σ to and from the median value of each metric over the 10 folds evaluated.

in order to achieve true predictive skill. That price was to greatly reduce the resolution of the input variables from as many as 12 bins to no more than 6 bins. While a price is paid in terms of input detail, there is a numerical as well as statistical benefit to reducing the bin resolution. For instance, the original ocean wave BN maintained over a million possible combinations of inputs and outputs scenarios within its conditional probability tables while the optimal 4-bin BN maintained 44 times fewer scenarios, reduced memory requirements, and had increased training and prediction speeds. For instance, the CV processing took 25 minutes for the 4-bin net compared to over 8 hours for the original net (a factor of 20 difference).

In the model emulation case, fewer input bins were supported while maintaining good predictive power. Four input bins resulted in good performance while a degradation of predictive skill started with five input bins. Predictive skill was relatively consistent with respect to output bins between 5 and 10 for a given set of input bins. This allows a resource manager to convey outcomes with some flexibility beyond the level of complexity supported by the data on the input side.

CVNetica is available for download at (https://github.com/mnfienen/NETICA_CV_GENERAL) and the authors welcome proposed contributions to code development going forward. These diagnostics and others have the potential to improve the validity of BNs used for prediction in natural resources and other applications.

7. Acknowledgements

This work was funded by (ASIS and GLAS others depending on which example application we use) and the USGS Coastal and Marine Geology Program. We are deeply grateful to Steven Mascaro for his initial PyNetica.py code which he kindly shared as a starting point for this work.

8. Disclaimer

Any use of trade, product, or firm names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

9. References

F. V. Jensen, T. D. Nielsen, Bayesian networks and decision graphs, Statistics for engineering and information science, Springer, New York, 2001.

Norsys Software Corp., Netica Version 5.12, http://www.norsys.com/, 1990–2013.

Hugin Expert A/S, HuginExpert Version 2013.0.0, http://www.hugin.com, 2013.

F. Martin de Santa Olalla, A. Dominguez, F. Ortega, A. Artigao, C. Fabeiro, Bayesian networks in planning a large aquifer in Eastern Mancha, Spain, Environmental Modelling and Software 22 (8) (2007) 1089–1100, doi: 10.1016/j.envsoft.2006.05.020.

J. L. Molina, J. Bromley, J. L. García-Aróstegui, C. Sullivan, J. Benavente, Integrated water resources management of overexploited hydrogeological systems using Object-Oriented Bayesian Networks, Environmental Modelling & Software 25 (4) (2010) 383–397, doi:10.1016/j.envsoft.2009.10.007.

J.-L. Molina, D. Pulido-Velázquez, J. L. García-Aróstegui, M. Pulido-Velázquez, Dynamic Bayesian Networks as a Decision Support tool for assessing Climate Change impacts on highly stressed groundwater systems, Journal of Hydrology 479 (2013) 113–129, doi:10.1016/j.jhydrol.2012.11.038.

N. G. Plant, K. T. Holland, Prediction and assimilation of surf-zone processes using a Bayesian network Part I: Forward models, Coastal Engineering 58 (1) (2011a) 119–130, doi:10.1016/j.coastaleng.2010.09.003.

N. G. Plant, K. T. Holland, Prediction and assimilation of surf-zone processes using a Bayesian network Part II: Inverse models, Coastal Engineering 58 (3) (2011b) 256–266, doi:10.1016/j.coastaleng.2010.11.002.

M. N. Fienen, J. P. Masterson, N. G. Plant, B. T. Gutierrez, E. R. Thieler, Bridging groundwater models and decision support with a Bayesian network, Water Resources Research 49 (10) (2013) 6459–6473, ISSN 0043-1397, doi:10.1002/wrcr.20496.

S. H. Chen, C. A. Pollino, Good practice in Bayesian network modelling, Environmental Modelling & Software 37 (2012) 134–145, doi:10.1016/j.envsoft.2012.03.012.

B. G. Marcot, Metrics for evaluating performance and uncertainty of Bayesian network models, Ecological Modelling 230 (2012) 50–62, doi:10.1016/j.ecolmodel.2012.01.013.

G. Rossum, Python reference manual, Tech. Rep., CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, The Netherlands, URL `http://www.python.org`, 1995.

E. Jones, T. Oliphant, P. Peterson, SciPy: Open source scientific tools for Python, URL `http://www.scipy.org/`, 2001–2014.

K. B. Korb, A. E. Nicholson, Bayesian artificial intelligence, Chapman & Hall/CRC series in computer science and data analysis, Chapman & Hall/CRC, Boca Raton, Fla., 2004.

E. T. Jaynes, G. L. Bretthorst, Probability theory : the logic of science, Cambridge University Press, Cambridge, UK ; New York, NY, 2003.

A. Dempster, N. Laird, D. Rubin, Maximum Likelihood From Incomplete

Data Via EM Algorithm, Journal of the Royal Statistical Society Series B-Methodological 39 (1) (1977) 1–38.

Norsys Software Corp., Netica API Programmer's Library: C Language Version Reference Manual Version 4.18, http://www.norsys.com/, 1990–2010.

T. Hastie, R. Tibshirani, J. H. Friedman, The elements of statistical learning : data mining, inference, and prediction, Springer series in statistics, Springer, New York, 2nd edn., 2009.

B. T. Gutierrez, N. G. Plant, E. R. Thieler, A Bayesian network to predict coastal vulnerability to sea level rise, Journal of Geophysical Research 116 (F2), doi:10.1029/2010jf001891.

A. S. Weigend, R. J. Bhansali, PARADIGM CHANGE IN PREDICTION, Philosophical Transactions of the Royal Society a–Mathematical Physical and Engineering Sciences 348 (1688) (1994) 405–420, doi:10.1098/rsta.1994.0100.

N. G. Plant, J. Flocks, H. F. Stockdon, J. W. Long, K. Guy, D. M. Thompson, J. M. Cormier, J. L. M. C. G. Smith, P. S. Dalyander, Predictions of barrier island berm evolution in a time-varying storm climatology, Journal of Geophysical Research Earth Surface 119 (2) (2014) 300–316, doi: 10.1002/2013JF002871.

H. F. Stockdon, R. A. Holman, P. A. Howd, J. S. A. H., Empirical parameterization of setup, swash, and runup, Coastal Engineering 53 (7) (2006) 573–588, doi:10.1016/j.coastaleng.2005.12.005.

F. Ruswick, J. Allan, D. Hamilton, P. Seelbach, The Michigan water withdrawal assessment process–Science and collaboration in sustaining renewable natural resources, Renewable Resources Journal 26 (4) (2010) 13–18.

P. Barlow, S. Leake, Streamflow depletion by wells–Understanding and managing the effects of groundwater pumping on streamflow, Circular 1376, United States Geological Survey, 2012.

K. A. Watson, A. S. Mayer, H. W. Reeves, Groundwater Availability as Constrained by Hydrogeology and Environmental Flows, Groundwater 52 (2) (2014) 225–238, doi:10.1111/gwat.12050.

S. Panday, C. Langevin, R. Niswonger, M. Ibaraki, J. Hughes, MODFLOW-USG version 1: An unstructured grid version of MODFLOW for simulating groundwater flow and tightly coupled processes using a control volume finite-difference formulation, Techniques and Methods, book 6, chap. A45, United States Geological Survey, 2013.

D. T. Feinstein, M. Fienen, C. Langevin, H. W. Reeves, Application of Unstructured MODFLOW to simulate local groundwater/surface-water interactions at the regional scale as basis for a decision-support tool, Groundwater .