A cross-validation package driving Netica with python

ARTICLE *in* ENVIRONMENTAL MODELLING AND SOFTWARE · JANUARY 2015

Impact Factor: 4.54 · DOI: 10.1016/j.envsoft.2014.09.007

2 AUTHORS:

Michael N Fienen
United States Geological Survey
55 PUBLICATIONS **685** CITATIONS
SEE PROFILE

Nathaniel G. Plant
United States Geological Survey
106 PUBLICATIONS **1,276** CITATIONS
SEE PROFILE

Environmental Modelling & Software 63 (2015) 14–23

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

A cross-validation package driving Netica with python

Michael N. Fienen [a, *], Nathaniel G. Plant [b]

[a] US Geological Survey, Wisconsin Water Science Center, 8505 Research Way, Middleton, WI 53562, USA
[b] US Geological Survey, St. Petersburg Coastal and Marine Science Center, 600 Fourth Street South, St. Petersburg, FL 33701, USA

ARTICLE INFO

Article history:
Received 19 April 2014
Received in revised form
3 September 2014
Accepted 10 September 2014
Available online 3 October 2014

Keywords:
Cross-validation
Bayesian networks
Uncertainty
Probability
Python
Netica
Prediction

ABSTRACT

Bayesian networks (BNs) are powerful tools for probabilistically simulating natural systems and emulating process models. Cross validation is a technique to avoid overfitting resulting from overly complex BNs. Overfitting reduces predictive skill. Cross-validation for BNs is known but rarely implemented due partly to a lack of software tools designed to work with available BN packages. CVNetica is open-source, written in Python, and extends the Netica software package to perform cross-validation and read, rebuild, and learn BNs from data. Insights gained from cross-validation and implications on prediction versus description are illustrated with: a data-driven oceanographic application; and a model-emulation application. These examples show that overfitting occurs when BNs become more complex than allowed by supporting data and overfitting incurs computational costs as well as causing a reduction in prediction skill. CVNetica evaluates overfitting using several complexity metrics (we used level of discretization) and its impact on performance metrics (we used skill).

Published by Elsevier Ltd.

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, 1990–2013) and Hugin (Hugin Expert A/S, 2013) among many others. Applications in water resources have included groundwater management (Martín de Santa Olalla et al., 2007; Molina et al., 2010, 2013), and model emulation (Plant and Holland, 2011a, 2011b; 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 should have formal tests and validation of prediction performance (Chen and Pollino, 2012; Marcot, 2012). BNs, as described in detail below, are made up of nodes representing variables. These nodes are discretized into bins and connected by edges. The arrangement of nodes, edges, and bins all impact the ability of a BN to fit a dataset descriptively and to predict new data accurately. There is always a tradeoff—better descriptive ability often is attained at the expense of predictive power. When descriptive ability is too high, the BN is fitting noise rather than true variability and this leads to the well-known result of overfitting. To protect against this, it is important to evaluate both descriptive ability and predictive power.

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., 2001–2014) and Python can access C libraries and functions. We discuss the technical challenges associated with running C APIs using Python and describe the toolbox of validation metrics included in this work.

There are many methods and metrics available to evaluate model performance (Bennett et al., 2013). The choices made in this work provide a starting point within a framework that other users can easily extend. 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. The number of nodes

* Corresponding author. Tel.: +1 608 821 3894.
E-mail addresses: mnfienen@usgs.gov (M.N. Fienen), nplant@usgs.gov (N.G. Plant).

http://dx.doi.org/10.1016/j.envsoft.2014.09.007
1364-8152/Published by Elsevier Ltd.

and the nature of their connections through edges are other aspects of BN design that define the complexity of a BN and impact both descriptive and predictive power. Our framework allows for consideration and analysis of other validation metrics and techniques beyond those presented here, including any user-defined configuration of BNs that controls their complexity.

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 include Boolean, 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 makes 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 conditional probabilities learned by the BN are ambivalent with respect to direction. Nodes can also be intermediate if they act as constraints or model coefficients. Once a BN is created and calibrated, it can be used to make predictions by selecting bins corresponding to input values and noting the changes in probability distributions of the output nodes. A user can also interact with a BN in an inverse-modeling context by specifying specific outputs and evaluating which are the most likely combinations of inputs that would result in the selected outputs.

An example of a BN created and visualized using Netica is presented in Fig. 1. In this example, each input node is connected to each output node and the “Recharge” node serves as an intermediate node between “Transmissivity” and all the output nodes.

Calculations are made using the BN based on conditional probabilities using Bayes' Theorem.

$$p(F_i|O_j) = \frac{p(O_j|F_i)p(F_i)}{p(O_j)} \quad (1)$$

where $p(F_i|O_j)$ is the posterior (updated) probability of a forecast (F_i) given (conditional on) a set of observations (O_j); $p(O_j|F_i)$ is the likelihood function, $p(F_i)$ is the prior probability of the forecast, and $p(O_j)$ 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. This “chain” is made up of all the nodes connected directly or through other nodes, as represented by edges. 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, $p(F_i)$, is the probability of a forecast without the benefit of updated observations and the BN (or a process model or other experiment). $p(F_i)$ may be estimated 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 the probability distribution represented by 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. When the underlying problem responds to thresholds in the data, it is often important to incorporate these thresholds as bin boundaries when defining the structure of the BN. Similarly, in some cases, specific categories even within a continuous distribution have important meaning and should be incorporated into the node structure.

It is possible to evaluate the contribution to all uncertainty values calculated by the BN by expressing the uncertainty in the prior probabilities. In Fig. 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 as 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. In other words, each forecast is associated with a specific configuration of system state. In our approach, observations are indicated by selecting the bin associated with that observation 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. Nodes are d-connected if there is an unblocked path of edges connecting them. 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. It is possible to learn the structure of a BN from data [e.g. (Jensen and Nielsen, 2001), Chapter7] rather than from expert understanding and in doing so, the resulting BN is more similar to an Artificial Neural Network [e.g. Samarasinghe, 2006]. In this work, the structure is always specified *a priori* through expert system understanding.

In Fig. 1, arrows on the edges indicate the direction of causal dependence. When all nodes are d-connected, the direction of the edge arrows serves no purpose in the calculations of conditional probabilities. However, in the context of d-separation, the direction of causality has important ramifications on the propagation of uncertainty from observations to forecasts and in either case, directionality of the edges helps users keep track of causal relationships when visually interpreting a BN.

The final step in calibrating a BN is parameter estimation. In this step, conditional probabilities are learned through enumeration or approximation. 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. This rapid growth of computational expense (especially with respect to parents) provides motivation for designing as simple networks as feasible. Beyond the techniques discussed in this work, intermediate nodes (latent nodes) or other techniques to combine the effects of multiple parents (through principal component

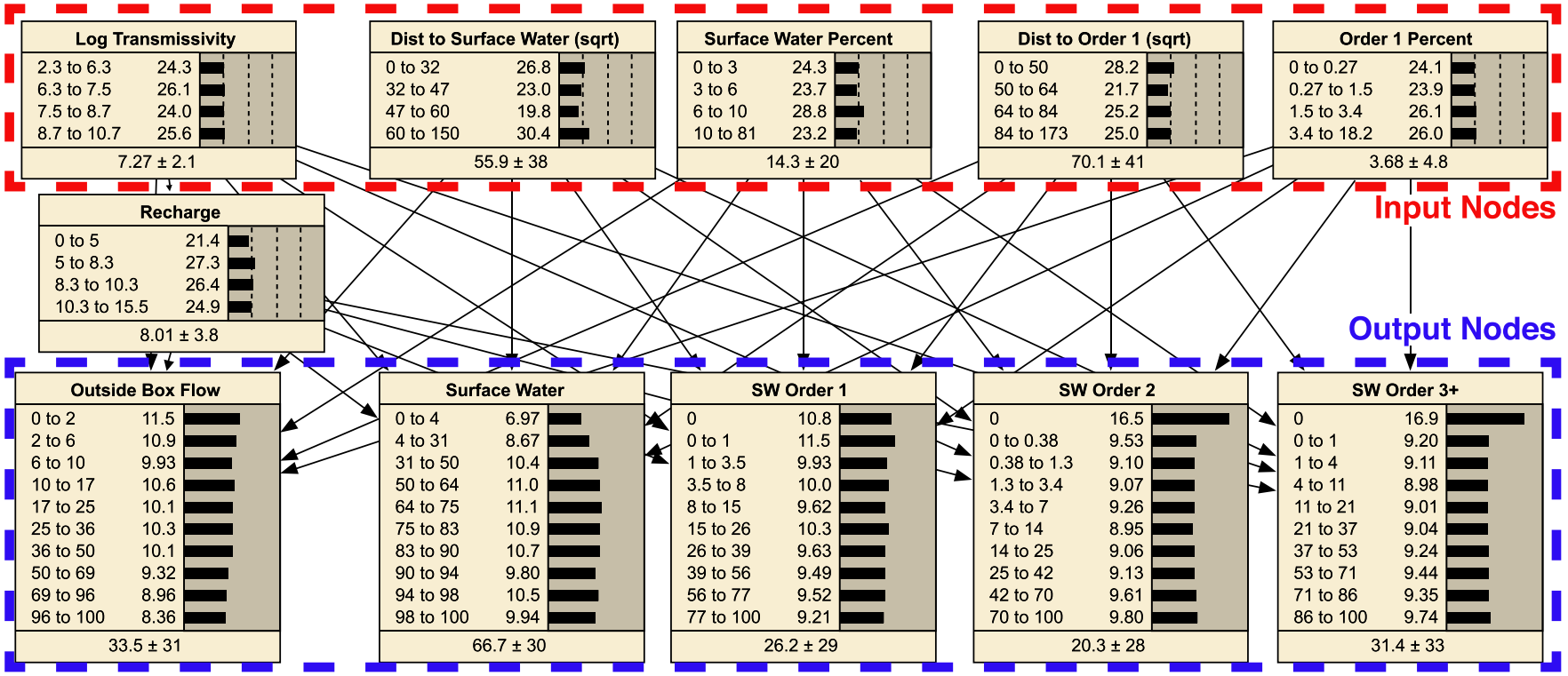

Fig. 1. Example groundwater application of a Netica BN showing input (outlined in a red box) and output (outlined in a blue box) nodes, edges (black lines) and, in this case, a single intermediate node (recharge). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

analysis, for example) have the potential to reduce the number of edges and nodes. This can enhance network function without intractable computational expense. Nonetheless, in many cases, full enumeration is impractical due to this rapid increase in computational expense with complexity, so 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. CVNetica

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 so requires no additional license. The APIs for Netica are described in (Norsys Software Corp, 1990–2010) and are provided as a dynamic linked library (DLL) for Windows. Static libraries are also available for Macintosh and Unix-based platforms, but to use them with Python, dynamic interface wrappers around the static functions would be necessary in addition to the Python function wrappers written in CVNetica. In this section we first describe the metrics and methods used to evaluate BNs and then describe the program structure of CVNetica.

3.1. Metrics and methods

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 assess the quality of predictions made by a BN. An accurate assessment of BN predictions requires predictions to be made on data that were not included in the dataset used to calibrate the BN (Hastie et al., 2009). This can be accomplished either by reserving a single subset of available data for evaluating predictions or by using cross validation. 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. This approach was selected both to make the most of data available (e.g. every data point is used at some stage in both calibration and prediction) and to smooth over any bias that may result in the random selection of a single withheld subset of the data. As a result, when metrics are reported over the k folds, the standard deviation of the metrics is also reported to acknowledge the variability in selection among the fold samples.

Several performance metrics can be used for this purpose, as discussed in (Norsys Software Corp, 1990–2013; Plant and Holland, 2011a; and Fienen et al., 2013). In this work, we focus on skill. Skill is evaluated based on a single representative expected value for each prediction as compared to a single observation corresponding to the same input values used to make the prediction. In the examples considered in this work, a single representative prediction is often considered as the key output of the BN in practice. Additional metrics take posterior variance into more explicit consideration. Skill is defined as.

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

where σ_e^2 is the mean squared error between observations and transformed BN predictions, and σ_o^2 is the variance of the observations (Gutierrez et al., 2011; Plant and Holland, 2011a; Weigend and Bhansali, 1994). Because the BN predictions are actually in the form of probabilities of the output variable falling within discrete ranges (bins), these probabilities are transformed to the expected values. Expected values may be computed either as mean or most likely (ML). The mean values are computed as the product of the bin centers and the probability in each bin, consistent with a typical expectation operation. For ML values, the value corresponding to the center of the bin with the highest predicted probability is reported. Additionally, since the transformation to an expected value does not capture all of the probabilistic information of the BN prediction, the expected values are fit to the observation via weighted linear regression, where the weights are set equal to the reciprocal of the variance of the BN prediction in order to give more weight to more confident predictions and σ_e^2 is computed as the mean-square regression error. A skill value of unity indicates perfect correspondence, a value of zero indicates substantial discrepancy between BN predictions and observations. Skill is closely related to the Nash-Sutcliffe Model Efficiency metric (Nash and Sutcliffe, 1970).

In addition to skill, CVNetica also reports log loss, error rate, experience, quadratic loss (Brier score), mutual information (entropy), and variance reduction (sensitivity) all of which are described by (Norsys Software Corp, 1990–2013, 1990–2010). These additional metrics in varying degree consider the probabilistic characteristics of the predictions made by a BN. Bennett et al. (2013) provide some guidance as to which metrics and methods are most appropriate for characterizing model performance in various contexts.

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) in the extreme case with a bin for every unique value in each node. 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). Cross validation allows users to assess the cost in predictive value imposed by the addition of complexity through more bins, additional nodes, or edges.

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 and the examples in this work are evaluated using bin configuration to represent complexity. CVNetica then builds a new BN with the requested number of bins and assigns 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. Graphical results are generated of performance metrics across the sets.

3.2. Program structure

There are three 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 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. The intermediate level depends on the lowest level, `pythonNeticaTools.py`, which 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. Call graphs in the supplemental online material also illustrate the relationships among these levels.

The `CV_driver.py` script drives a cross-validation exercise on an existing BN developed using Netica specified in the XML based configuration file (Fig. 2). The script reads in the existing BN, imposes the specified alterations, runs the resulting BN, and gathers and summarizes the results. 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.

```
<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>
```

Fig. 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. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

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, 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 $n-n/k$ 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 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, 1990–2010). 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 make type conversions as appropriate. The CVNetica variable `cnode` is of the custom C type `node_bn*`, and it was returned by a Netica API function, so it is already of the required type (in this case 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 argument 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 value returned by the function 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() # use ctypes conversion to declare
                                # std_dev a double precision variable
        tmpNeticaFun = self.n.GetNodeExpectedValue_bn # make a
                                # temporary function
    tmpNeticaFun.restype=ct.c_double # use restype to define
                            # that a double variable is returned
                            # by the function
    # note that in the call to the temporary function
    # the returned value (expected_val) is set by the
    # function, and std_dev is set in memory and dereferenced
    # with the method .value
    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 shown in Fig. 1.

5.1. Data driven ocean waves

Weather forecasting and modeling have 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, 2011b), but the fidelity 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 overfit 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 Fig. 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 et al., 2006) and the full dataset consisted of 50,385 entries. The first two columns of network nodes (Fig. 3) correspond to observations from an offshore buoy (NOAA 42040) collected from 1996-2011 that is located far offshore of the location where predictions are needed. The offshore 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 an onshore buoy between 2000 and 2008. The onshore 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 better numerical efficiency.

Our calibration/validation skill analysis was applied to this BN by varying the number of bins in all variables except for the wave heights. We chose to resolve wave heights consistently with the original model to ensure that probability predictions spanned a wide range of 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 (Fig. 4). However, the validation skill, averaged over 5 folds, reached its peak value at 4 bins and then decreased dramatically after 6 bins. It is likely that the optimal bin resolution 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 disadvantageous for others. The rapid decline after 6 bins suggests that none of the variables needed to be better resolved past this point.

5.2. Model emulation to determine 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 pumping 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 different 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 were evaluated and mappable characteristics of each well location were used to create the BN in Fig. 1. In all, 4905 wells were simulated, requiring nine model runs. For further discussion of the model used in this work see (Feinstein et al., 2013).

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 Fig. 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 Fig. 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

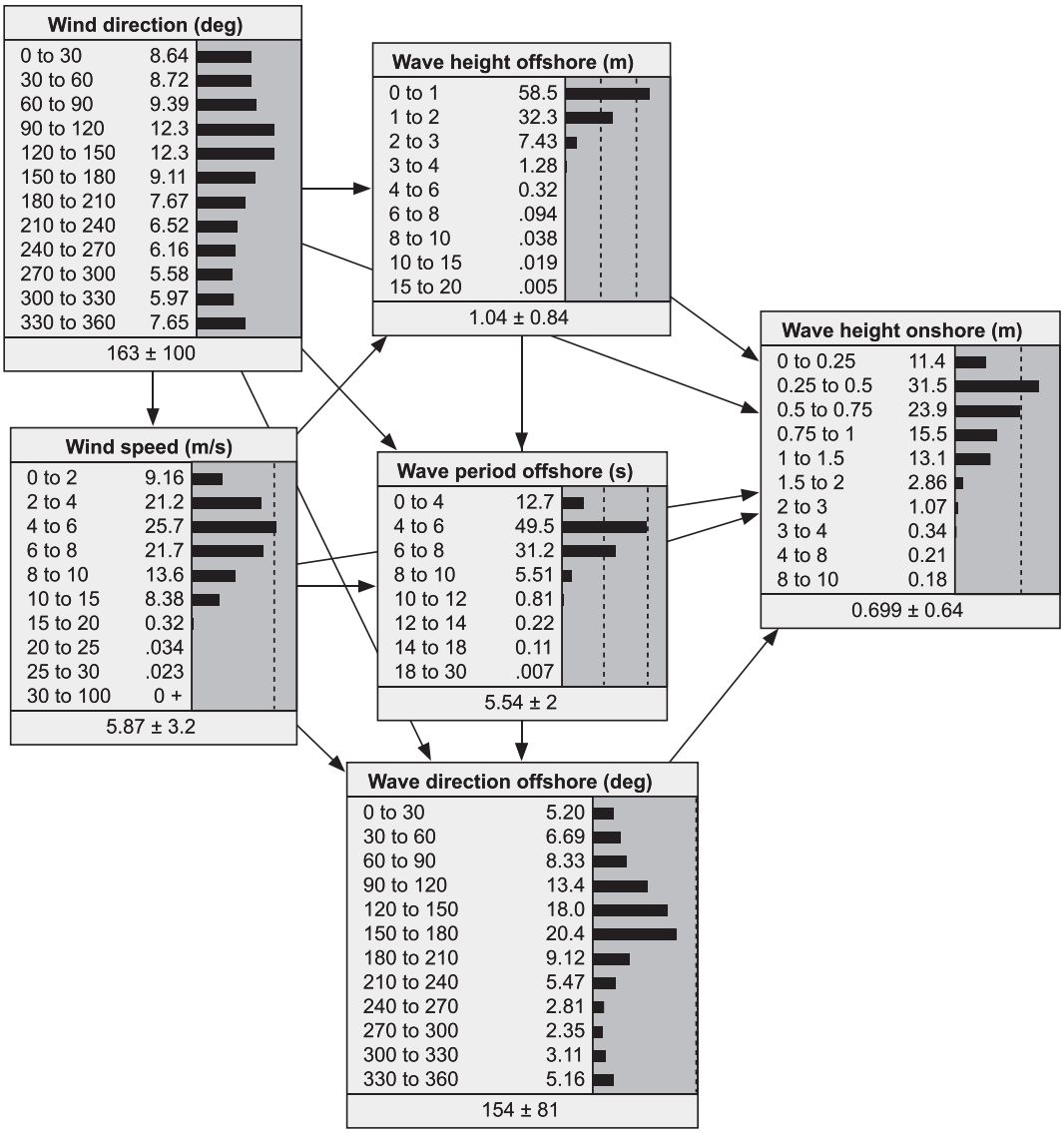

Fig. 3. BN for the wave height prediction model.

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 while 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 CVNetica 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 choice of fairly high resolution of the input data at the offshore buoy. While physically consistent, a price needed to be paid for the over-resolved output 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 min for the 4-bin net compared to over 8 h 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

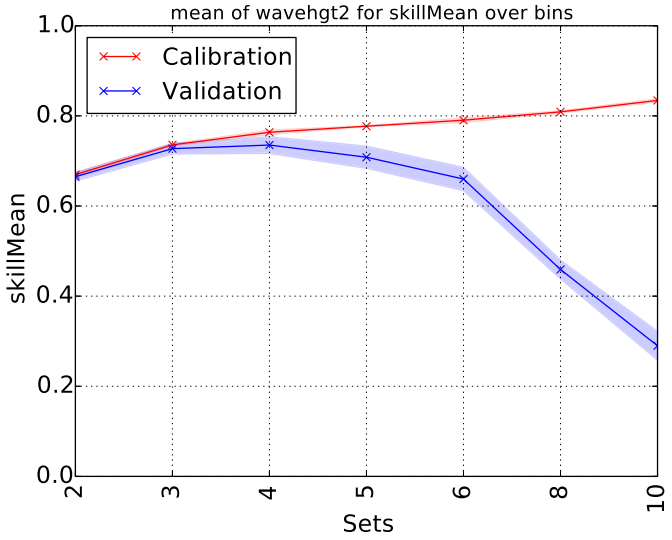

Fig. 4. Calibration (descriptive) and Validation (predictive) skill values for various arrangements of bins on the wave prediction BN. For sets identified by a single number, that number of bins was used to discretize all nodes. The color shading indicates an approximation of the 95% credible interval based on adding and subtracting 2σ to and from the median value of each metric over the 10 folds evaluated. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

some flexibility beyond the level of complexity supported by the data on the input side.

CVNetica is available for download at (http://mnfienen-usgs.github.io/CVNetica) 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.

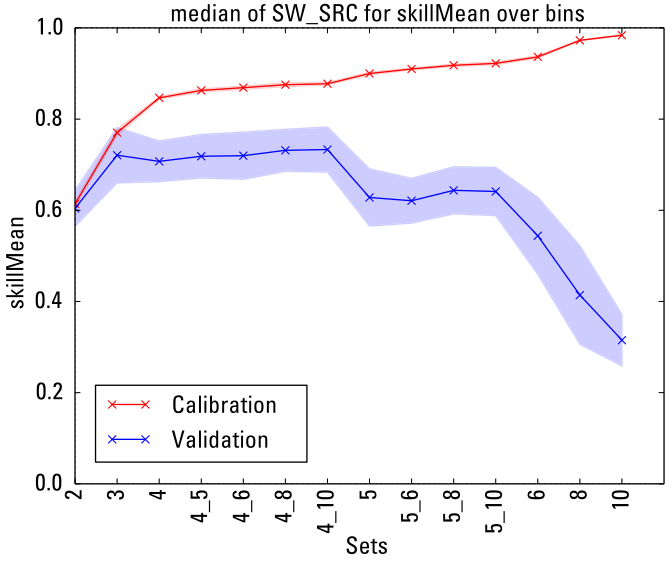

Fig. 5. Calibration (descriptive) and Validation (predictive) skill values for various arrangements of bins on the glacial aquifer BN. For sets identified by a single number, that number of bins was used to discretize all nodes. For sets identified by two numbers separated by an underscore, the first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

Disclaimer

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

Acknowledgments

This work was funded by the USGS Groundwater Resources Program 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. We also thank Howard Reeves, USGS, Tony Jakeman, Editor-in-Chief, and two anonymous reviewers for insightful comments that improved this manuscript.

Appendix A. Supplementary data

Supplementary data related to this article can be found at http://dx.doi.org/10.1016/j.envsoft.2014.09.007.

References

Barlow, P., Leake, S., 2012. Streamflow Depletion by Wells—Understanding and Managing the Effects of Groundwater Pumping on Streamf. United States Geological Survey.

Bennett, N.D., Croke, B.F.W., Guariso, G., Guillaume, J.H.A., Hamilton, S.H., Jakeman, A.J., Marsili-Libelli, S., Newham, L.T.H., Norton, J.P., Perrin, C., Pierce, S.A., Robson, B., Seppelt, R., Voinov, A.A., Fath, B.D., Andreassian, V., 2013. Characterising performance of environmental models. Environ. Model. Softw. 40, 1–20.

Chen, S.H., Pollino, C.A., 2012. Good practice in bayesian network modelling. Environ. Model. Softw. 37, 134–145.

Dempster, A., Laird, N., Rubin, D., 1977. Maximum likelihood from incomplete data via em algorithm. J. R. Stat. Soc. Ser. B-Methodol. 39 (1), 1–38.

Feinstein, D., Fienen, M., Reeves, H., Langevin, C., 2013. Application of a Semi-structured Approach with Modflow-usg to Simulate Local Groundwater/surface-water Interactions at the Regional Scale as Basis for a Decision-Support Tool, MODFLOW and More 2013. Colorado School of Mines, Golden, CO. June 3–5.

Fienen, M.N., Masterson, J.P., Plant, N.G., Gutierrez, B.T., Thieler, E.R., 2013. Bridging groundwater models and decision support with a Bayesian network. Water Resour. Res. 49 (10), 6459–6473.

Gutierrez, B.T., Plant, N.G., Thieler, E.R., 2011. A Bayesian network to predict coastal vulnerability to sea level rise. J. Geophys. Res. 116 (F2).

Hastie, T., Tibshirani, R., Friedman, J.H., 2009. The Elements of Statistical Learning: Data Mining, Inference, and Prediction, second ed. In: Springer series in Statistics Springer, New York.

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

Jaynes, E.T., Bretthorst, G.L., 2003. Probability Theory: the Logic of Science. Cambridge University Press, Cambridge, UK; New York, NY.

Jensen, F.V., Nielsen, T.D., 2001. Bayesian Networks and Decision Graphs. In: Statistics for Engineering and Information Science. Springer, New York.

Jones, E., Oliphant, T., Peterson, P., 2001–2014. SciPy: Open Source Scientific Tools for Python. URL: http://www.scipy.org/.

Korb, K.B., Nicholson, A.E., 2004. Bayesian Artificial Intelligence. In: Chapman & Hall/CRC Series in Computer Science and Data Analysis. Chapman & Hall/CRC, Boca Raton, Fla.

Marcot, B.G., 2012. Metrics for evaluating performance and uncertainty of Bayesian network models. Ecol. Model. 230, 50–62.

Martín de Santa Olalla, F., Dominguez, A., Ortega, F., Artigao, A., Fabeiro, C., 2007. Bayesian networks in planning a large aquifer in Eastern Mancha, Spain. Environ. Model. Softw. 22 (8), 1089–1100.

Molina, J.L., Bromley, J., García-Aróstegui, J.L., Sullivan, C., Benavente, J., 2010. Integrated water resources management of overexploited hydrogeological systems using object-oriented Bayesian networks. Environ. Model. Softw. 25 (4), 383–397.

Molina, J.-L., Pulido-Velázquez, D., García-Aróstegui, J.L., Pulido-Velázquez, M., 2013. Dynamic Bayesian networks as a decision support tool for assessing climate change impacts on highly stressed groundwater systems. J. Hydrol. 479, 113–129.

Nash, J.E., Sutcliffe, J.V., 1970. River flow forecasting through conceptual models part I—a discussion of principles. J. Hydrol. 10 (3), 282–290.

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

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

Panday, S., Langevin, C., Niswonger, R., Ibaraki, M., Hughes, J., 2013. 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. In: Techniques and Methods, Book 6, Chap. a45. United States Geological Survey.

Plant, N.G., Flocks, J., Stockdon, H.F., Long, J.W., Guy, K., Thompson, D.M., Cormier, J.M., M, J.L., Smith, C.G., Dalyander, P.S., 2014. Predictions of barrier island berm evolution in a time-varying storm climatology. J. Geophys. Res. Earth Surf. 119 (2), 300–316.

Plant, N.G., Holland, K.T., 2011. Prediction and assimilation of surf-zone processes using a Bayesian network part i: forward models. Coast. Eng. 58 (1), 119–130.

Plant, N.G., Holland, K.T., 2011. Prediction and assimilation of surf-zone processes using a Bayesian network part II: inverse models. Coast. Eng. 58 (3), 256–266.

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

Ruswick, F., Allan, J., Hamilton, D., Seelbach, P., 2010. The michigan water withdrawal assessment process–science and collaboration in sustaining renewable natural resources. Renew. Resour. J. 26 (4), 13–18.

Samarasinghe, S., 2006. Neural Networks for Applied Sciences and Engineering: from Fundamentals to Complex Pattern Recognition, first ed. Auerbach Publications. URL: http://www.worldcat.org/isbn/084933375X.

Stockdon, Hilary F., Holman, Rob A., Howd, Peter A., Sallenger Jr., Asbury H., 2006. Empirical parameterization of setup, swash, and runup. Coast. Eng. 53 (7), 573–588.

Watson, K.A., Mayer, A.S., Reeves, H.W., 2014. Groundwater availability as constrained by hydrogeology and environmental flows. Groundwater 52 (2), 225–238.

Weigend, A.S., Bhansali, R.J., 1994. Paradigm change in prediction. Philos. Trans. R. Soc. a–Math. Phys. Eng. Sci. 348 (1688), 405–420.