CHAPTER 6

Machine Learning

Machine learning is a computational process to discover the underlying models of system behavior. Machine learning takes datasets, processes them, and attempts to discover causal variables. Machine learning techniques were big in the 1980s during the first artificial intelligence. Machine learning is having a major resurgence in the last decade because of the immense amounts of data available and the significant advancements in computational power.

The absence of a robust and unified theory of cyber dynamics presents challenges and opportunities for using machine learning–based data-driven approaches to further the understanding of the behavior of such complex systems. Analysts can also use machine learning approaches to gain operational insights. In order to be operationally beneficial, cyber security machine learning–based models need to have the ability to: (1) represent a real-world system, (2) infer system properties, and (3) learn and adapt based on expert knowledge and observations. Probabilistic models and probabilistic graphical models provide these necessary properties and are further explored in this chapter. Bayesian networks (BNs) and hidden Markov models (HMMs) are introduced as an example of a widely used data-driven classification/modeling strategy.

This chapter is organized as follows: We begin with an introduction of machine learning concepts and techniques. This is followed by a discussion of validating models derived through machine learning. Finally, we will explore using BNs and HMMs.

CHAPTER OBJECTIVES

- Introduce machine learning
- Discuss model validation
- Explore the use of Bayesian networks and hidden Markov models in cyber security research

Research Methods for Cyber Security. DOI: http://dx.doi.org/10.1016/B978-0-12-805349-2.00006-6

WHAT IS MACHINE LEARNING

Machine learning is a field of study that looks at using computational algorithms to turn empirical data into usable models. The machine learning field grew out of traditional statistics and artificial intelligences communities. From the efforts of mega corporations such as Google, Microsoft, Facebook, Amazon, and so on, machine learning has become one of the hottest computational science topics in the last decade. Through their business processes immense amounts of data have been and will be collected. This has provided an opportunity to re-invigorate the statistical and computational approaches to autogenerate useful models from data.

Machine learning algorithms can be used to (a) gather understanding of the cyber phenomenon that produced the data under study, (b) abstract the understanding of underlying phenomena in the form of a model, (c) predict future values of a phenomena using the above-generated model, and (d) detect anomalous behavior exhibited by a phenomenon under observation. There are several open-source implementations of machine learning algorithms that can be used with either application programming interface (API) calls or nonprogrammatic applications. Examples of such implementations include Weka,[1] Orange,[2] and RapidMiner.[3] The results of such algorithms can be fed to visual analytic tools such as Tableau[4] and Spotfire[5] to produce dashboards and actionable pipelines.

Cyber space and its underlying dynamics can be conceptualized as a manifestation of human actions in an abstract and high-dimensional space. In order to begin solving some of the security challenges within cyber space, one needs to sense various aspects of cyber space and collect data.[6] The observational data obtained is usually large and increasingly streaming in nature. Examples of cyber data include error logs, firewall logs, and network flow.

CATEGORIES OF MACHINE LEARNING

There are two dimensions around which machine learning is generally categorized: the process by which it learns and the type of output or problem it attempts to solve. For the first machine learning–based solution strategies can be broadly classified into three categories based on the mechanism used to perform learning namely, supervised learning, semisupervised learning, and unsupervised learning.[7] For the latter, machine learning algorithms can be broken into four categories: classification, clustering, regression, and anomaly detection.

The style of learning has an impact upon the question you are trying to solve. In some cases, you have data that you do not know the ground truth,

other times it is possible to label data with categories or classifications. Sometimes you know what a good result looks like but you may not know what variables are important to get there. By categorizing machine learning techniques by the learning style can help you in selecting the best approach for your research. Table 6.1 discusses the different styles and provides a sample set of machine learning algorithms.

Supervised learning involves using a labeled dataset (e.g., the outcomes are known and labeled). Unsupervised learning is used in cases where the labels of the data are unknown (e.g., when the outcomes are unknown, but some similar measure is desired). Examples of unsupervised learning approaches include self-organizing maps (SOMs), K-means clustering, expectation–maximization (EM), and hierarchical clustering.[8] Unsupervised learning approaches can also be used for preliminary data exploration such as clustering similar error logs entries. Results of unsupervised algorithms are frequently visualized using visual analytic tools. An important caveat on using an unsupervised approach is to make sure one knows the numeric space that the data encompasses as well as the type of distance measure applied. Semisupervised approaches are a hybrid of unsupervised and supervised approaches. Such approaches are used when only some of the data is unlabeled. Semisupervised approaches are used when a portion of the data is unlabeled. Such approaches can be inductive or transductive.[9]

While it is sometimes helpful in picking algorithms based on what type of input data is available, it is equally helpfully to break them out along the

Table 6.1 Learning Style Categorization of Machine Learning Algorithms

Style	Definition	Example Algorithms
Unsupervised	In **unsupervised learning**, no extra or meta data is provided to the algorithm and it is forced to discover the structure data and the relationship of variables by observing a raw dataset.	K-means clustering, hierarchical clustering, principal component analysis
Supervised	In **supervised learning**, input data is annotated with expert information detailing what the expected output or answer would be. The process of annotating data for supervised learning is called **labeling**.	Neural network, Bayesian networks, decision tree, support vector machine
Semi-supervised	In **semisupervised learning** a small set of learning data is labeled but large gaps in labeling are present. This is largely used when it is known that a small number of variables led to a result, but the full extent of the variables involved is unknown. A special case of semisupervised learning is called **reinforced learning** where an expert informs the algorithm if its output is correct or not.	Expectation–maximization, transductive support vector machine, Markov decision processes

Table 6.2 Categories of Machine Learning Algorithms Separated by Problems they Address[10]

Problem	Definition	Example Algorithms
Classification	**Classification** algorithms take labeled data and generate models that classify new data into the learned labels.	Hidden Markov models, support vector machines (SVMs), random forests, naïve bayes, probabilistic graphical models, logistic regression, neural networks [9]
Clustering	**Cluster analysis** attempts to take a dataset and define clusters of like items.	K-means, heirarchical, density-based (DBSCAN)
Regression	**Regression** attempts to generate a predictive model by optimizing the error in learned data.	Linear,logistic, ordinary least squares, multivariate adaptive regression splines
Anomaly detection	**Anomaly detection** takes a dataset of "normal" items and learns a model of normal. This model is used to determine if any new data is anomalous or low probability of occurring.	One-class SVM, linear regression and logistic regression, frequent pattern growth (FP-growth), a priori

result types provided. Variables within a dataset can be numeric (i.e., discrete or continuous), ordinal (i.e., order matters), cardinal (i.e., integer valued), nominal/categorical (i.e., used as an outcome class name). Machine learning algorithms can also be categorized based on the type of problem they solve. An example of such a breakdown of algorithms is listed in Table 6.2.

Decision tree algorithms: classification trees (e.g.,C4.5) can be used in cases of a nominal class variable while regression trees can be used for continuous numeric valued outcome variables.

As discussed by Murphy et al.,[11] several issues affect the alternative learning schemes, including:

- Dynamic range of the features
- Number of features
- Type of the class variable
- Types of the features
- Heavily correlated features

In order to be operationally beneficial, cyber security machine learning–based models need to have the ability to: (1) represent a real-world system, (2) infer system properties, and (3) learn and adapt based on expert knowledge and observations. Probabilistic graphical models have wide applications for assessing and quantifying cyber security risks.[12,13] These models contain desirable properties including representation of a real-world system, inference about queries of interest related to the system, and learning from expert knowledge and past experience.[14] The probabilistic terms in these models may be estimated or learned from historical data, generated from simulation experiments, or elicited through informed judgments of subject matter experts.

DID YOU KNOW?

A common application of anomaly detection machine learning algorithms is credit card fraud detection. Machine learning is used to generate models of each customer's behavior and usage pattern. If the activity appears that is deemed anomalous by the model then a fraud alert is triggered. So when you go on vacation and get a fraud alert from using your credit card, you should know this means you deviated enough from your schedule such that it appears anomalous.

Owing to the adaptive nature of cyber threats, probabilistic cyber risk models need to accommodate efficient updating of model structure and parameter estimates as new intelligence and information becomes available. Also, understanding relationships between factors influencing the occurrence and impacts of such events is a critical task. Bayesian networks, or probabilistic directed acyclic graphs, have mathematical properties for characterizing relationships between dynamic event and system factors, can be updated using probabilistic theories, and produce inference and predictions for unobserved factors given evidence. Past research indicates the potential for the application of BNs along with attack graphs for real-world cyber defenses.[15,16,17,18] HMMs have been widely used to generate data-driven models for several cyber security solutions.

DEBUGGING MACHINE LEARNING

One of the challenges with machine learning is the problem of overfitting or underfitting called *variance* and *bias* respectively. **Model variance** or **overfitting** is when a machine learning developed model fits the training dataset very well but fails to generalize to new datasets. Model bias or underfitting is when the machine learning generated model has high error in fitting the training set. This can commonly occur when you are learning over too many features There are two general options to address overfitting: reduce the number of features or use regularization. **Regularization** is the process to reduce the magnitude of values of a large feature set.

If you find that a model you have developed through machine learning is making large errors in predictions what should you do? One approach is to develop a diagnostic. A **machine learning diagnostic** is a test designed to gain insight into what isn't working in an algorithm or how to improve its performance. Diagnostics can be difficult to build but they are well worth it in the long run.

Another good approach to validating your model is to use cross-validation. **Cross-validation** is a process to evaluate the generalizability of your developed model. The process of doing cross-validation largely starts with dividing

your initial dataset into three; a training set, a cross-validation set, and a test set. You want to have a sufficient amount of data for your training set so a good rule of thumb is to make the divisions at 60% training, 20% cross validation, and 20% test. The cross-validation set is used to tune or find the best-fit model parameters. Then the test set is used to determine the generalizability of the generated model.

There are a few more tips for addressing. To fix high variance you can get more training data or try to learn around a smaller set of features or variables. To fix high bias try adding more features or polynomial features. In both cases tuning your parameters can help.

DID YOU KNOW?

In 2011 the IBM Watson super computer competed in two Jeopardy matches with Brad Rutter and Ken Jennings, two of the most successful Jeopardy players in history. The technology underlying Watson's ability to parse and answer questions, called DeepQA, leveraged over 100 machine learning algorithms.[19] But it wasn't just the algorithms alone but also the pipeline and structured sequence of using the machine learning algorithms that helped Watson best two of the best Jeopardy contestants.

BAYESIAN NETWORK MATHEMATICAL PRELIMINARIES AND MODEL PROPERTIES

A **BN** is a graphical model that represents uncertainties (as probabilities) associated with discrete or continuous random variables (nodes) and their conditional dependencies (edges) within a directed acyclic graph.[20,21]

BNs model the relationships among variables and may be updated as additional information about these variables becomes available. Mathematically, if nodes represent a set of random variables, $X = X_1, X_2, ..., X_n$ then a set of links connecting these nodes $X_i \rightarrow X_j$ represent the dependencies among the variables. Also, each node is conditionally independent of nondescendant nodes given its parent nodes and has an attached probability function. As a result, the joint probability of all nodes, $P(X)$ may be represented as: $\prod_{i=1}^{n} P\left(X_i \mid parents(X_i)\right)$.

Strengths and Limitations

Key strengths and limitations of BNs are listed below:

- Models dependencies among random variables as directed acyclic graphs
- Allows probabilistic inference of unobserved variables
- Graphical representation may be intuitive for users

- Incorporate data/expert judgments and update structure/parameters as "new" data/knowledge becomes available
- Inferring structure of an unknown network may be computationally demanding from a scalability perspective
- Identifying reliable prior knowledge is a challenge

Data-driven Learning and Probabilistic Inference within Bayesian Networks

Structure learning within **Directed Acyclic Graphs** (DAGs) may be broadly classified as: (1) *constraint-based* and (2) *score-based*. Constraint-based algorithms use conditional independence tests using the data to build causal graphs that satisfy constraints. A challenge associated with constraint-based approaches is identifying independence properties and optimizing network structure. Also, these approaches do not account for a well-defined objective function and may result in nonoptimal graphical structures. Score-based algorithms assigns a score function to the entire space of causal graphs and typically use greedy search among various potential DAGs to identify the structure with the highest score. These approaches are optimization-based and tend to scale better.

Data-driven statistical learning methods (e.g., *Hill Climbing* and *Grow-Shrink* algorithms) may be adopted to infer Bayesian network structures for various parameters of interest across geographic regions. **Hill Climbing** is a score-based algorithm that uses greedy heuristic search to maximize scores assigned to candidate networks.[22] **Grow-Shrink** is a constraint-based algorithm that uses conditional independence tests to detect *blankets* (comprised of a node's parents, children, and children's other parents) of various variables.

DIG DEEPER: BAYESIAN NETWORK PROVENANCE

The probabilities and process underlying BNs was first defined by Thomas Bayes in mid-17th century. The Bayes rule determines the probability of an event by updating prior probabilities with new information. It wasn't until the 1980s that Judea Pearl made the distinction between evidence and causality. In the Bayesian network defined by Pearl, the nature and uncertainty of evidence is taken into account before updating probabilities.

Parameter Learning

Probabilistic parameters associated with these network structures may be estimated using *expectation maximization* and *maximum likelihood* estimation techniques. **Expectation maximization** is useful with not fully observed data, and is an iterative algorithm where in the "Expectation" step the probability of unobserved variables given observed and current parameters are estimated. Thereafter, in the "Maximization" step the current parameters are updated through log-likelihood maximization.

Maximum likelihood approach is useful with fully observed data and involves estimating probabilistic parameters θ, for each node in the graph such that the log-likelihood function, $log\big(P(X|\theta)\big)$, is maximized.

Bayesian estimation is also an option, where θ is treated as a random variable, a prior probability $p(\theta)$ is assumed, and data is used to estimate the posterior probability of $p(\theta|X)$.

Probabilistic Inference

BNs apply Bayes' theorem for inference of unobserved variables. **Variable elimination** (by integration or summation) of unobserved, nonquery variables is a widely used exact inference method. Approximate inference methods include stochastic *Markov Chain monte Carlo (MCMC)* simulation, *logic sampling*, *likelihood weighting*, and others.

Notional Example with bnlearn Package in R

R, in conjunction with the RStudio integrated development environment, provides a powerful platform for data analysis. The default setup of R provides several libraries including the stat library. R and RStudio need to be installed separately.[23,24] The example described below uses the R package "*bnlearn*"[25] for structure learning, parameter learning, and probabilistic inference. The code begins by loading the "learning" dataset. This data has discrete levels for each of the following random variables: In a realistic cyber setting, these random variables (e.g., A, B, etc.) may represent the time-varying health of system components and the levels (e.g., a, b, c) may represent the discrete states of health.

```
#install.packages("bnlearn")
library(bnlearn)
data(learning.test)
str(learning.test)
##  'data.frame': 5000 obs. of 6 variables:
##   $ A: Factor w/ 3 levels "a","b","c": 2 2 1 1 1 3 3 2 2 2 ...
##   $ B: Factor w/ 3 levels "a","b","c": 3 1 1 1 1 3 3 2 2 1 ...
##   $ C: Factor w/ 3 levels "a","b","c": 2 3 1 1 2 1 2 1 2 2 ...
##   $ D: Factor w/ 3 levels "a","b","c": 1 1 1 1 3 3 3 2 1 1 ...
##   $ E: Factor w/ 3 levels "a","b","c": 2 2 1 2 1 3 3 2 3 1 ...
##   $ F: Factor w/ 2 levels "a","b": 2 2 1 2 1 1 1 2 1 1 ...
head(learning.test)
##  A B C D E F
##  1 b c b a b b
##  2 b a c a b b
##  3 a a a a a a
##  4 a a a a b b
##  5 a a b c a a
##  6 c c a c c a
```

Structure learning using *Hill Climbing* and *Grow-Shrink* results in the following.

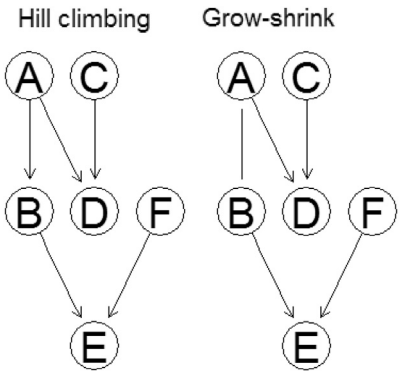
```
raw <- data.frame(learning.test)
bn.h <- hc(raw) #hill climbing
bn.g <- gs(raw) #grow-shrink
bn.h
##
##   Bayesian network learned via score-based methods
##
##   model:
##     [A][C][F][B|A][D|A:C][E|B:F]
##   nodes:                                 6
##   arcs:                                  5
##     undirected arcs:                     0
##     directed arcs:                       5
##   average markov blanket size:           2.33
##   average neighbourhood size:            1.67
##   average branching factor:              0.83
##
##   learning algorithm:                    Hill-Climbing
##   score:                                 BIC (disc.)
##   penalization coefficient:              4.258597
##   tests used in the learning procedure:  40
##   optimized:                             TRUE
```

Bayesian Information Criterion (BIC) score above is a measure of relative model quality, and provides a mechanism for model selection by balancing goodness-of-fit and complexity through a *penalty term*. Lower *BIC* scores are preferred; however, these scores do not represent model quality in the absolute sense and must be interpreted carefully.

The resulting structures are given below. The dependencies among certain nodes learned solely from data may or may not make intuitive sense and can be updated based on expert inputs (e.g., *blacklist* or *whitelist*). *Blacklist* indicates an expert-provided absence of a relationship between nodes that is indicated prior to data-driven structure learning; and *Whitelist* is the presence of such a relationship. The different approaches can result in different outcomes including directionality or the lack thereof between nodes.

```
#source("https://bioconductor.org/biocLite.R")
#biocLite("Rgraphviz")
library(Rgraphviz)
##  Loading required package: graph
##  ## Attaching package: 'graph'
##  The following objects are masked from 'package:bnlearn':
```

```
##
##    degree, nodes, nodes<-
##  Loading required package: grid
par(mfrow = c(1, 2))
graphviz.plot(bn.h, main = "Hill climbing")
graphviz.plot(bn.g, main = "Grow-shrink")
```



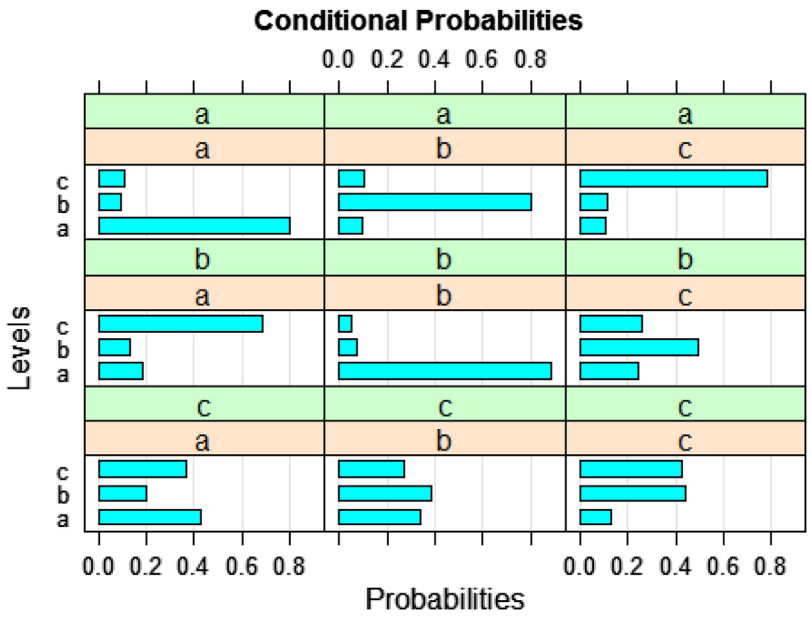
In this notional example, the *hill climbing* and *grow-shrink*-based BN structures above are similar except the directionality between nodes A and B. Grow-shrink results in an undirected edge between nodes A and B, whereas hill climbing results in the learning of a dependence of node B on node A. Once the network structure is determined, one can learn the model parameters (i.e., *conditional probability* tables (CPT) for the *discrete* case) associated with each node in the BN. The results below display parameters of node D based on maximum likelihood and Bayesian estimation methods.

```
fit.bnm <- bn.fit(bn.h, data = raw, method = "mle")
fit.bnm$D
##
##    Parameters of node D (multinomial distribution)
##
##  Conditional probability table:
##
##  , , C = a
##
##    A
##  D    a    b    c
##    a 0.80081301 0.09251810 0.10530547
##    b 0.09024390 0.80209171 0.11173633
##    c 0.10894309 0.10539019 0.78295820
##
##  , , C = b
##
##    A
```

```
##  D    a    b    c
##    a 0.18079096 0.88304094 0.24695122
##    b 0.13276836 0.07017544 0.49390244
##    c 0.68644068 0.04678363 0.25914634
##
##  , , C = c
##
##    A
##  D    a    b    c
##    a 0.42857143 0.34117647 0.13333333
##    b 0.20238095 0.38823529 0.44444444
##    c 0.36904762 0.27058824 0.42222222
fit.bnb <- bn.fit(bn.h, data = raw, method = "bayes")
fit.bnb$D
##
##    Parameters of node D (multinomial distribution)
##
##  Conditional probability table:
##
##  , , C = a
##
##    A
##  D    a    b    c
##    a 0.80039110 0.09273317 0.10550895
##    b 0.09046330 0.80167307 0.11193408
##    c 0.10914561 0.10559376 0.78255696
##
##  , , C = b
##
##    A
##  D    a    b    c
##    a 0.18126825 0.88126079 0.24724285
##    b 0.13339591 0.07102763 0.49336034
##    c 0.68533584 0.04771157 0.25939680
##
##  , , C = c
##
##    A
##  D    a    b    c
##    a 0.42732811 0.34107527 0.13577236
##    b 0.20409051 0.38752688 0.44308943
##    c 0.36858138 0.27139785 0.42113821
```

The code below generates a plot of the CPT.

```
bn.fit.barchart(fit.bnm$D, xlab = "Probabilities", ylab = "Levels",
main = "Conditional Probabilities")
##  Loading required namespace: lattice
```



With the BN structure and parameters, we can perform probabilistic inference. *Logic sampling* and *likelihood weighting* are currently implemented options. For example, P(B = = "b"| A = = "a") ~ 0.025.

```
cpquery(fit.bnm, event = (B == "b"), evidence = (A == "a"))
##  [1] 0.02622852
cpquery(fit.bnm, event = (B == "b"), evidence = (A == "a" & D == "c"))
##  [1] 0.02760351
```

BNs are also useful for in-sample and out-of-sample predictions. In-sample predictions are useful for model evaluation and out-of-sample predictions help with model testing and validation. An out-of-sample prediction example is given below with training and testing datasets and out-of-sample predictive performance of 90%.

```
train <- raw[1:4990, ]
test <- raw[4991:5000, ]
bn.train <- hc(train)
fit <- bn.fit(bn.train, data = train)
pred <- predict(fit, "D", test)
#library(xtable)
#print(xtable(cbind(pred, test[, "D"])), type = 'html')
#print(xtable(table(pred, test[, "D"])), type = 'html')
cbind(pred, test[, "D"])
##    pred
```

```
##  [1,]   1 1
##  [2,]   3 3
##  [3,]   2 2
##  [4,]   1 2
##  [5,]   2 2
##  [6,]   1 1
##  [7,]   2 2
##  [8,]   2 2
##  [9,]   3 3
##  [10,]   1 1
table(pred, test[, "D"])
##
##  pred a b c
##    a 3 1 0
##    b 0 4 0
##    c 0 0 2
```

HIDDEN MARKOV MODELS

In this section, we discuss HMMs,[26] which are a type of dynamic BN models. HMMs have found applications in biological sequence analysis for gene and protein structure predictions,[27,28,29] multistage network attack detection[30], and in pattern recognition problems[31] such as speech,[32] handwriting,[33] and gesture[34] recognition. HMMs model the generation of a sequence of states that can only be inferred from a sequence of observed symbols. The symbols can be discrete (e.g., events, tosses of a coin) or continuous. In a HMM, a hidden Markov process generates the sequence of states, which are in turn used to explain and characterize the occurrence of a sequence of observable symbols. Therefore, the generation of the observable symbols is probabilistically dependent on the generation of the unobservable Markov states.

To illustrate the process of generating the observation sequence in a HMM, let's denote a time-ordered sequence of observed symbols of length T, as $Y = \{Y_1, Y_2, \ldots, Y_T\}$, and the associated hidden sequence of Markov states, as $X = \{X_1, X_2, \ldots, X_T\}$. Each observed element,$Y_i$, can be a symbol describing the outcome of a stochastic process. Let $O = \{O_1, O_2, \ldots, O_M\}$ represent a discrete set of M such possible outcomes (observed symbols). Similarly let $S = \{S_1, S_2, \ldots, Y_N\}$ represent a discrete set of N distinct Markov states. Besides specifying the number of observed symbols, M, and the number of distinct Markov states, N, a HMM specification involves specifying three probability distributions: (1) the transition state probability distribution, A; (2) the probability distribution to choose the observed symbol from a state, B; and (3) the initial state distribution, π. The compact notation, $\lambda = (A, B, \pi)$, is normally used to represent an HMM.

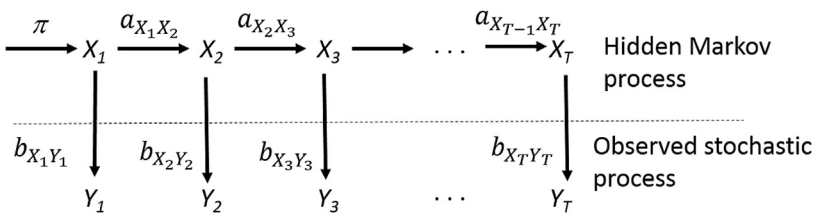
Fig. 6.1 illustrates a general example of an HMM. The process of generating the observation sequence is described below.

Generation of observation sequence in HMM

1. Initialize time index, $t = 1$
2. Choose the initial state, X_t, according to the initial state distribution, π.
3. Choose the observed symbol, Y_t, according to the probability distribution, B in state X_t.
4. Choose a new state, X_{t+1}according to the state transition probability distribution, A for the current state, X_t.
5. Set $t = t + 1$
6. **if** $t < T$ **then**
7. **go to** step 3
8. **end if**

There are three types of problems that need to be solved, in order for HMMs to be useful in real-world applications:

- Problem 1: Given the observation sequence, $Y = \{Y_1, Y_2, \ldots, Y_T\}$, and the model parameters, $\lambda = (A, B, \pi)$, compute the probability



Notations

X – sequence of hidden Markov states
Y – sequence of observed symbols
S – set of N distinct Markov states ($S = \{S_1, S_2, \ldots, S_N\}$)
π – initial state distribution vector
O – set of M possible observed symbols ($O = \{O_1, O_2, \ldots, O_M\}$)
A – transition state probability matrix
$a_{X_tX_{t+1}} = A_{ij}; X_t = S_i, X_{t+1} = S_j$, for any $i = 1,2,\ldots,N$ and $j = 1,2,\ldots,N$

B – observation probability matrix
$b_{X_tY_t} = B_{ik}; X_t = S_i, Y_t = O_k$, for any $i = 1,2,\ldots,N$ and $k = 1,2,\ldots,M$

FIGURE 6.1
General process modeled by hidden Markov models.

(likelihood), $P(Y|\lambda)$, that the observed sequence was produced by the model. Problem 1 aims to evaluate the model.

- Problem 2: Given the observation sequence, $Y = \{Y_1, Y_2, \ldots, Y_T\}$, and the model parameters, $\lambda = (A, B, \pi)$, determine the most optimal state sequence of the underlying Markov process, $X = \{X_1, X_2, \ldots, X_T\}$. Problem 2 aims to uncover the hidden part of the model.
- Problem 3: Given the observation sequence, $Y = \{Y_1, Y_2, \ldots, Y_T\}$, and the dimensions M and N, adjust the parameters of the model, $\lambda = (A, B, \pi)$, to maximize $P(O|\lambda)$. Problem 3 aims to find the best model that fits a training sequence of observed symbols.

Notional Example with HMM Package in R

This example uses the R package "*HMM*" for: (1) computing most probable path of states given a HMM and (2) inferring optimal parameters to a HMM. The Viterbi algorithm for state path estimation is implemented below:

```
#install.packages("HMM")
#source: https://cran.r-project.org/web/packages/HMM/HMM.pdf
library(HMM)
##Viterbi algorithm for computing most probable path of states given an
HMM
# HMM Initialization
hmm = initHMM(c("A","B","C"), c("o1","o2"), startProbs = matrix(c
(.25,.5,.25),1), transProbs = matrix(c
(.3,.4,.6,.4,.4,.3,.3,.2,.1),3), emissionProbs = matrix(c
(.5,.4,.9,.5,.6,.1),3))
print(hmm)
##  $States
##  [1] "A" "B" "C"
##
##  $Symbols
##  [1] "o1" "o2"
##
##  $startProbs
##  A    B    C
##  0.25 0.50 0.25
##
##  $transProbs
##    to
##  from    A    B    C
##    A 0.3 0.4 0.3
##    B 0.4 0.4 0.2
##    C 0.6 0.3 0.1
##
##  $emissionProbs
##    symbols
```

```
##  states o1 o2
##      A 0.5 0.5
##      B 0.4 0.6
##      C 0.9 0.1
#  Sequence of observations
observations = c("o1","o2","o2","o1","o1","o2")
print(observations)
##  [1] "o1" "o2" "o2" "o1" "o1" "o2"
#  Calculate Viterbi path
viterbi = viterbi(hmm,observations)
print(viterbi)
##  [1] "C" "A" "B" "A" "C" "A"
```

The Viterbi-training algorithm for inferring optimal parameters is implemented below.

```
##Viterbi-training algorithm for inferring optimal parameters to an HMM
#  Initial HMM
hmm = initHMM(c("A","B","C"), c("o1","o2"), startProbs = matrix
(c(.25,.5,.25),1), transProbs = matrix
(c(.3,.4,.6,.4,.4,.3,.3,.2,.1),3), emissionProbs = matrix
(c(.5,.4,.9,.5,.6,.1),3))
# Sequence of observations
a = sample(c(rep("o1",100),rep("o2",300)))
b = sample(c(rep("o1",300),rep("o2",100)))
observation = c(a,b)
#  Viterbi-training
vt = viterbiTraining(hmm,observation,1000)
print(vt$hmm)
##  $States
##  [1] "A" "B" "C"
##
##  $Symbols
##  [1] "o1" "o2"
##
##  $startProbs
##  A    B    C
##  0.25 0.50 0.25
##
##  $transProbs
##    to
##  from        A         B         C
##    A 0.0000000 0.2981928 0.7018072
##    B 0.4230769 0.5769231 0.0000000
##    C 1.0000000 0.0000000 0.0000000
##
##  $emissionProbs
##    symbols
##  states o1 o2
```

```
##    A 0.5 0.5
##    B 0.0 1.0
##    C 1.0 0.0
```

DISCUSSION

BNs and HMMs provide a probabilistic framework to infer, predict, and gain insights into the dependencies between components within a cyber system. The choice of random variables, their types (discrete or continuous), and state information are critical for designing and interpreting BN and HMM results. Within BNs, the strength and direction of component dependencies (represented as conditional probabilities) learned from data may vary across systems and time periods. Additional cyber intelligence information related to pre-event conditions and postevent impacts can be valuable for enhancing what-if and forensic analyses. Modeling extensions of interest may include the use of dynamic BNs that allow evolution over time and hybrid networks that can accommodate mixed discrete and continuous random variables as nodes. Further, validation approaches may be incorporated to test various data-driven learning models using appropriate training and testing datasets.

SAMPLE FORMAT

In the following, we will provide you with a general outline for publishing your results. Every publication will provide their own formatting guidelines. Be sure to check your selected venue's submission requirements to make sure you follow any outline and formatting specifications. The outline provided here follows a common flow of information found in published papers and should meet the requirements of a larger number of publisher specifications.

Every paper is unique and requires some different ways of presentation; however, the provided sample flow includes all of the general information that is important to cover in a paper and is a general starting format we take when starting to write a paper and then modify it to support the topic and venue. We understand that every researcher has their own style of presentation, so feel free to use this as a jumping-off point. The discussions of each section are provided to explain what is important to include and why, so you can present the important information in whatever way best suits your style.

Abstract

The abstract is a concise and clear summary of the paper. The goal of an abstract is to provide readers with a quick description of what the paper discusses. You should only talk about what is stated in the rest of the paper and

nothing additional. Each submission venue will provide guidelines on how to provide an abstract. Generally, this includes the maximum length of the abstract and formatting instructions, but sometimes this will include information on the type and layout of the content to provide.

Introduction

The first section of a paper should always be an introduction for the reader into the rest of the paper. The introduction section provides the motivation and reasoning behind why the research was performed. This should include a statement of the research question and any motivating questions that were used for the research. If any background information is required, such as explaining the domain, environment, or context of the research, you would discuss it here. If the audience is significantly removed from some aspect of this topic, it may be worth it to create an independent background section. In machine learning papers, it is good to include the performance or learning criteria around which you will determine the quality of the machine learning application. It's also good to specify the category of machine learning you are describing in this paper; is it a new machine learning algorithm or an application of an existing machine learning approach to a new set of data?

Related Work

The related works section should include a quick summarization of the field's knowledge about this research topic. Are there competing solutions? Have other machine learning approaches been used in the past? Explain what is deficient about past applications. What was the gap in the solution space that motivated the use of the proposed machine learning approach defined in this paper.

Approach

The approach section of an applied paper is often the meat or largest portion of the paper. In this section you describe how your machine learning algorithm works and represents or systematizes knowledge. Describe your learning dataset and any processing or manipulation of the data to provide it to the algorithm.

Evaluation

In the evaluation explain how you are going to exercise the machine learning approach to generate a results dataset for performance characterization. Discuss what test datasets you will use. Explain if you do cross-validation and regularization.

Data Analysis/Results

In the results section of your paper, explain what you found after you performed your analysis. Present the performance and/or learning results around whatever metrics you define. Comparative analysis with past or competing applied solutions are very helpful in contextualizing your results. Without previous performance numbers it is difficult to understand the significance of your work. Creating tables to show results is an efficient and effective method. You can also show pictures of interesting results, that is if a data anomaly occurred or to display the distributions of the data samples.

Discussion/Future Work

The discussion/future work section is for you to provide your explanations for the results you received. Discuss additional tests you think should be performed. Discuss future directions in research that may result from the knowledge gained from the evaluation. If performance was less than expected should more observation research be performed?

Conclusion/Summary

In the final section of the paper, summarize the results and conclusions of the paper. The conclusion section is often a place readers jump to quickly after reading the abstract. Make a clear and concise statement about what the ultimate results of the experiments and what you learned from it.

Acknowledgments

The acknowledgments section is a place for you to acknowledge anyone who helped you with parts of the research that were not part of the paper. It is also good to acknowledge and funding sources that supported your research.

References

Each publication will provide some guidelines on how to format references. Follow their guidelines and list all your references at the end of your paper. Depending on the length of the paper you will want to adjust the number of references. The longer the paper the more references. A good rule of thumb is 15–20 references for a 6-page paper. For peer-reviewed publications, the majority of your references should be other peer-reviewed works. Referencing web pages and Wikipedia doesn't generate confidence in reviewers. Also, make sure you only list references that are useful to your paper, that is don't inflate your reference count. Good reviewers will check and this will likely disqualify and reflect poorly on you.

Endnotes

1. Weka 3: Data Mining Software in Java. (n.d.). Retrieved February 25, 2017, from http://www.cs.waikato.ac.nz/ml/weka/Weka.
2. Shaulsky, G., Borondics, F., Bellazzi, R. (n.d.). *Orange—Data Mining Fruitful & Fun*. Retrieved February 25, 2017, from http://orange.biolab.si/BioLab, University of Ljubljana.
3. Data Science Platform. *Machine Learning* (2017, February 22). Retrieved February 25, 2017, from http://www.rapidminer.com/rapidminer.
4. Tableau Software (n.d.). Retrieved February 25, 2017, from http://www.tableau.com/.
5. Data Visualization & Analytics Software—TIBCO Spotfire. (n.d.). Retrieved February 25, 2017, from http://spotfire.tibco.com/.
6. Sheymov, V., *Cyberspace and Security: A Fundamentally New Approach*. 2012: Cyber books publishing.
7. Chapelle, O., B. Schölkopf, and A. Zien, *Semi-supervised Learning*. 2010: MIT Press.
8. Murphy, K.P., *Machine Learning: A Probabilistic Perspective*. 2012: MIT Press.
9. Chapelle, O., B. Schölkopf, and A. Zien, *Semi-supervised Learning*. 2010: MIT Press.
10. Borgelt, C., *An implementation of the FP-growth algorithm*, in *Proceedings of the 1st international workshop on open source data mining: Frequent pattern mining implementations*. 2005, ACM: Chicago, Illinois, pp. 1–5.
11. Murphy, K.P., *Machine Learning: A Probabilistic Perspective*. 2012: MIT Press.
12. Ezell, B.C., et al., *Probabilistic risk analysis and terrorism risk*. Risk Analysis, 2010. **30**(4): p. 575–589.
13. Koller, D. and N. Friedman, *Probabilistic graphical models: principles and techniques*. 2009: MIT press.
14. Koller, D. and N. Friedman, *Probabilistic graphical models: principles and techniques*. 2009: MIT press.
15. Frigault, M., et al. *Measuring network security using dynamic bayesian network*. in *Proceedings of the 4th ACM workshop on Quality of protection*. 2008. ACM.
16. Xie, P., et al. *Using Bayesian networks for cyber security analysis*. in *2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2010. IEEE.
17. Bode Moyinoluwa, A., K. Alese Boniface, and F. Thompson Aderonke. *A Bayesian Network Model for Risk Management in Cyber Situation*. in *Proceedings of the World Congress on Engineering and Computer Science*. 2014.
18. Shin, J., H. Son, and G. Heo, *Development of a cyber security risk model using Bayesian networks*. Reliability Engineering and System Safety, 2015. 134: p. 208–217.
19. https://www.aaai.org/Magazine/Watson/watson.php.
20. Team, R.C., *R: A language and environment for statistical computing*. 2013.
21. Nielsen, T.D. and F.V. Jensen, *Bayesian Networks and Decision Graphs*. 2007: Springer New York.
22. Bode Moyinoluwa, A., K. Alese Boniface, and F. Thompson Aderonke. *A Bayesian Network Model for Risk Management in Cyber Situation*. in *Proceedings of the World Congress on Engineering and Computer Science*. 2014.
23. Team, R.C., *R: A language and environment for statistical computing*. 2013.
24. *RStudio*. 2016: https://www.rstudio.com/.
25. Scutari, M., *Learning Bayesian networks with the bnlearn R package*. arXiv preprint arXiv:0908.3817, 2009.
26. Rabiner, L.R., *A Tutorial on Hidden Markov-Models and Selected Applications in Speech Recognition*. Proceedings of the Ieee, 1989. **77**(2): p. 257–286.
27. Yoon, B.-J., *Hidden Markov models and their applications in biological sequence analysis*. Current genomics, 2009. **10**(6): p. 402–415.
28. Krogh, A., et al., *Hidden Markov models in computational biology: Applications to protein modeling*. Journal of molecular biology, 1994. **235**(5): p. 1501–1531.

29. Eddy, S.R., *What is a hidden Markov model?* Nature biotechnology, 2004. **22**(10): p. 1315–1316.
30. Ourston, D., et al. *Applications of hidden markov models to detecting multi-stage network attacks*. in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*. 2003. IEEE.
31. Fink, G.A., *Markov models for pattern recognition: from theory to applications*. 2014: Springer Science & Business Media.
32. Gales, M. and S. Young, *The application of hidden Markov models in speech recognition*. Foundations and trends in signal processing, 2008. **1**(3): p. 195–304.
33. Plötz, T. and G.A. Fink, *Markov models for offline handwriting recognition: A survey*. International Journal on Document Analysis and Recognition (IJDAR), 2009. **12**(4): p. 269–298.
34. Moni, M. and A.S. Ali. *HMM based hand gesture recognition: A review on techniques and approaches*. in *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*. 2009. IEEE.