3

Overview of Prominent Machine Learning and Data Mining Methods with Example Applications to the Medical Domain

Christopher M. Gifford

CONTENTS

3.1 Introduction

Modern medicine, health informatics, and bioinformatics systems create and leverage massive volumes of rich, complex, and highly coupled data sets. Hospitals, treatment centers, and doctors' offices capture a diverse collection of personal health data, health history, symptoms and diagnosis data, disease and treatment data, blood work and test results, and imaging products. Medical research institutions produce even further experimental data, metadata, and results at various resolutions, from cell-level tissue samples, to disease progression and propagation behaviors, to the sequencing of the human genome. These data are produced by a variety of devices, sensors, and information systems all over the world every day.

The speed and scale at which these data are produced frame a highly nonlinear, hyperdimensional feature space that is far too large and complex for doctors and scientists to fully understand. Humans can no longer perform exhaustive tasks or ask even simple questions of these data in an efficient manner. This creates the need for increased automation, decision support mechanisms, and nontrivial algorithm development to relieve data overwhelm and enable practitioners to focus on higher-level tasks, analytical reasoning, and making informed decisions. Automated, algorithmic solutions are attractive for several important reasons:

- Enable sensemaking of high-volume, complex data sets
- Offer efficient summarization, modeling, and analysis of an enormous amount of data
- Provide compact, intuitive, and in many cases human-readable outputs
- Assist in the identification of abnormalities
- Provide greater automation, speed, and effectiveness of doctors, scientists, and officials

Artificial intelligence, primarily machine learning and data mining, has played and will continue to play a key role in the advancement of technologies, research, understanding, and treatment in the medical domain. This chapter is aimed at introducing the reader to many prominent artificial intelligence techniques being employed to improve modern medicine and practice. Section 3.2 provides an overview of machine learning and data mining, including descriptions of classes of algorithms and secondary decision methods utilized in the broader community. Section 3.3 discusses popular software suites and resources to support development for various applications and domains. Section 3.4 discusses several relevant and illustrative examples of medical applications of artificial intelligence. Section 3.5 concludes the chapter with closing remarks, leading the reader into the research chapters of the book.

3.2 Overview of Machine Learning and Data Mining

Machine learning and data mining are major fields within artificial intelligence that are heavily used in many domains for a variety of applications. This section describes some of

the prominent methods and classes of algorithms. In many cases, the data set properties, expected output usage, and/or application domain drive the algorithm(s) to use for any particular problem.

3.2.1 Knowledge Discovery and Data Mining

The goal of data mining is to discover inherent and previously unknown information from data to represent as knowledge. In many cases, such methods are used to summarize and model very large data sets, capturing what salient (high support or occurring frequently) and interesting patterns reside in the data that may not have otherwise been discovered. Many of these methods were motivated by the growing demand to mine consumer transaction data to determine which items were purchased together, so that retail stores could better position their products within the store and track buying patterns. These methods therefore extend to online transactions and have therefore been heavily used by the Web community. As such, many algorithms leverage the transaction model for the core data structure and operating algorithmic assumptions. Two such methods are Association Rule Mining and Sequential Pattern Discovery.

3.2.1.1 Association Rule Mining

The methodology of association rule mining produces knowledge in rule form, called an *association rule*. An association rule describes a relationship among different attributes. Association rule mining was introduced by Agrawal et al. [1] as a way to discover interesting co-occurrences in supermarket data (the market basket analysis problem). It finds frequent sets of items (i.e., combinations of items that are purchased together in at least N database transactions) and generates from the frequent itemsets (such as $\{X, Y\}$) association rules of the form $X \rightarrow Y$ and/or $Y \rightarrow X$.

More formally, let $D = \{t_1, t_2, ..., t_n\}$ be the transaction database, and let t_i represent the ith transaction in D. Let $I = \{i_1, i_2, \ldots, i_m\}$ be the universe of items. A set $X \sqsubseteq I$ of items is called an itemset. When X has k elements, it is called a k-itemset. An association rule is an implication of the form $X \rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \cap Y = \varnothing$. Various statistical measures exist that offer metadata about the rule. *Support*, *confidence*, and *lift* are three such measures.

The *support* of an itemset X is defined as

$$\text{Support}(X) = \frac{\text{number records with } X}{\text{number records in } D}$$

The *support* of a rule $(X \rightarrow Y)$ is defined as

$$\text{Support}(X \rightarrow Y) = \frac{\text{number records with } X \text{ and } Y}{\text{number records in } D}$$

The *confidence* of a rule $(X \rightarrow Y)$ is defined as

$$\text{Confidence}(X \rightarrow Y) = \frac{\text{number records with } X \text{ and } Y}{\text{number records with } X}$$

Confidence can be treated as the conditional probability ($P(Y|X)$) of a transaction containing X and also containing Y. A high confidence value suggests a strong association rule. However, this can be deceptive. For example, if the antecedent (X) or consequent (Y) has a high support, it could have a high confidence even if it was independent. This is why the measure of lift was suggested as a useful metric.

The lift of a rule ($X \rightarrow Y$) measures the deviation from independence of X and Y. A lift greater than 1.0 indicates that transactions containing the antecedent (X) tend to contain the consequent (Y) more often than transactions that do not contain the antecedent (X). The higher the lift, the more likely that the existence of X and Y together is not just a random occurrence but, rather, due to the relationship between them.

$$\text{Lift}(X \rightarrow Y) = \frac{\text{Confidence}(X \rightarrow Y)}{\text{Support}(Y)}$$

A limitation of traditional association rule mining is that it only works on binary transaction data [e.g., an item is either purchased in a transaction (1) or not (0)]. In many real-world applications, data are either categorical (e.g., blue, red, green) or quantitative (e.g., number of molecules). For numerical and categorical attributes, Boolean rules are unsatisfactory. Extensions have been proposed to operate on these data, such as quantitative association rule mining [2] and fuzzy association rule mining [3,4].

Apriori [1] is the most widely used algorithm for finding frequent *k*-itemsets and association rules. It exploits the downward closure property, which states that if any *k*-itemset is frequent, all of its subsets must be frequent as well. The Apriori algorithm proceeds as follows:

Calculate the support of all *1*-itemsets and prune (i.e., remove) any that fall below the minimum support, specified by the user.

Loop:

1. Form candidate *k*-itemsets by taking each pair (*p*, *q*) of (*k*-1)-itemsets, where all but one item match. Form each new *k*-itemset by adding the last item of *q* onto the items of *p*.
2. Prune the candidate *k*-itemsets by eliminating any itemset that contains a subset not in the frequent (*k*-1)-itemsets.
3. Calculate the supports of the remaining candidate *k*-itemsets and eliminate any that fall below the specified minimum support threshold. The result is the frequent *k*-itemsets.

The extension of apriori to fuzzy apriori was proposed by Kuok et al. [3]. Both apriori and fuzzy apriori need to repeatedly scan the database when they construct the *k*-itemsets and calculate their support. This requirement caused these methods to be criticized by many researchers for being inefficient. The frequent pattern (FP)-growth algorithm was proposed by Han et al. [5] as a fast method for generating frequent itemsets. When generating frequent itemsets, FP-growth avoids generating a significant amount of candidates that are generated by apriori. FP-growth builds and operates on a data structure called the FP-tree and passes over the data set only twice. The FP-growth algorithm operates as follows:

FP-tree construction

1. Calculate the support of all 1-itemsets and prune any that fall below the minimum support threshold, specified by the user.
2. Sort the remaining frequent 1-itemsets in descending global frequency.
3. Build the FP-tree, tuple by tuple, from the first transaction to the last.

Deriving frequent itemsets from the FP-tree

1. A conditional FP-tree is generated for each frequent itemset, and from that tree, the frequent itemsets with the processed item can be recursively derived.

FP-growth mines the complete set of frequent patterns regardless of the length of the longest pattern. Lin et al. [6] proposed a partial fuzzification of FP-growth, including a fuzzy FP-tree (a tree structure for frequent fuzzy regions). The first step in their method is to transform quantitative values into fuzzy membership function values (as in fuzzy apriori). Their extension is only partially fuzzy, as later in the process, it only uses the fuzzy region (i.e., membership function) with maximum cardinality among all the transformed fuzzy regions of a variable.

3.2.1.2 Sequential Pattern Discovery

Traditional association rule mining was designed for mining salient patterns within a set of transactions, ignoring the temporal order of the transactions. Thus, association rule mining becomes inefficient for temporal and sequential applications that require ordered matching rather than simple subset testing. Although comparatively less mature, sequential pattern discovery methods have been developed specifically for these types of problems.

Here, we define a *sequential pattern* as a temporally ordered set of items or events. As with association rule mining, combinatorial explosion and the number of data passes affect speed and memory requirements of these algorithms. Further challenges are pattern summarization (i.e., providing an expressive, compact result set), the ability to naturally handle and analyze temporal data (e.g., activity sequences), and achieving acceptable performance for long sequences and reduced support (low frequency). Many early sequence mining algorithms were able to achieve good performance for short frequent sequence applications but suffered degraded performance for long sequences and low support thresholds.

Examples of existing methods in this space are generalized sequential pattern (GSP) [7], PrefixSpan [8], bi-directional extension (BIDE) [9], mining top-K closed sequential patterns (TSP) [10], and CloSpan [11]. In many cases, the full set of subsequences is not necessary to capture the expressive power of the data set. CloSpan, in particular, leverages an important concept to enable the efficient mining of long sequences with reduced support: *closed subsequences* [11]. A subsequence S is considered *closed* if no superset (extension) of S exists with the same support. By removing these superset sequences from consideration, redundant subsequences with the same support are eliminated. This dramatically reduces the result set, improves algorithm efficiency, and allows the mining of long sequences to be a tractable process.

Result set summarization is necessary for usability of the outputs of such methods. High-support thresholds are typically used to reduce the result set to a manageable size. Interestingness measures have also been developed to aid in post-pruning (i.e., selective filtering) the result set. Closed sequences provide a more compact result set to reduce analyst overwhelm. Sequential patterns can be efficiently summarized by leveraging *partial orders* [12]. Partial orders are created by collapsing and combining overlapping sequences

into a directed graphical representation, viewable as a temporal "left-to-right" network. Thus, partial orders provide both an intuitive visualization of the result set to facilitate data understanding and a representative data model. By representing the result set in this manner, new incoming data can be evaluated against the sequential model as a basis for a variety of novel purposes, including the following:

- Anomaly detection (e.g., if an observed sequence does not fit the network, or events are observed out of order, an anomaly indicator can be triggered)
- Intervention/prediction (e.g., probable/expected future observations can be suggested using the temporal nature of the sequential network)
- Possible causes (e.g., probable prior observations can be suggested that lead to the currently observed event by exploiting the temporal nature of the sequential network)
- Further analytics and statistical measures

Potential medical applications of sequential pattern discovery include, but are not limited to, the following:

- DNA sequence alignment
- Motifs and tandem repeats in DNA sequences
- Modeling clinical visit patterns
- Analyzing infectious disease and micro-level disease observations
- Studying the evolution of a disease and agent/host interaction patterns
- Analysis of multi-omics experiment data involving temporal treatments

3.2.2 Machine Learning

The field of machine learning is concerned with algorithms and mechanisms to allow computers to learn (or model) data, behavior, or an environment. Learning is induced by automatically extracting information from data, typically in a statistical manner, and adjusting/adapting the underlying computational data structure, model, or resulting action/output over time. As a subfield of artificial intelligence, machine learning is highly related to other subfields of data mining, pattern recognition, classification, and optimization.

Enabling computers to model and provide automated decision support can eliminate the need for extensive human intervention or *a priori* assignment. It also provides a way to quickly and intelligently search through or summarize data to learn underlying patterns that are not evident or easily observable. The manner in which learning takes place, as well as the output of the learning process, depends on the underlying algorithm(s). These algorithms typically fall into one of several categories, depending on the task, underlying knowledge structure and extraction procedure, and desired output.

Many techniques exist within each category (e.g., unsupervised versus supervised). Some learning algorithms are *unstable*, meaning they experience major output changes with small input changes. Examples of unstable algorithms include artificial neural networks, decision trees, and rule-based methods. An example of a *stable* algorithm is a fuzzy system that makes use of non-crisp feature and decision boundaries. Machine learning algorithms are typically trained on known data and tested on (applied to) unknown/

unseen data. The following sections introduce a variety of machine learning algorithms, as well as multi-classifier decision fusion and meta-learning secondary decision methods.

3.2.2.1 Logistic Regression

Logistic regression is a statistical model that predicts the probability of an event occurring by fitting a logistic curve to the data. Several predictor variables are typically used to describe the relationship between risk factors and outcomes. Logistic regression has seen use in medical and social fields, including customer trending.

Logistic regression is based on the logistic function

$$f(Z) = \frac{1}{1 + e^{-Z}}$$

where Z represents the input (set of risk factors) and the returned value represents the output (probability of a particular outcome). Z normally takes on the following form:

$$Z = a + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n$$

where a is the intercept (where all risk factors are zero) and b_i represent the regression coefficients (risk factor contributions). Positive coefficients translate to that specific risk factor increasing the probability of an outcome, while negative coefficients translate to a decrease in probability of an outcome. Similarly, larger values correspond to higher influence. A multinomial logistic regression model with a ridge estimator is described in the work of le Cessie and van Houwelingen [13], which iterates until convergence.

3.2.2.2 Naïve Bayes

Statistical and probabilistic approaches to machine learning have been used for many years, achieving notable success in the medical domain. Several such approaches have become popular, such as Bayesian networks and the naïve Bayesian classifier, due to their simplicity and clear probabilistic semantics [14]. The naïve Bayes classifier operates on the assumption that numeric attributes result from a single Gaussian (normal) distribution. For real-world problems, this approximation is unlikely to be accurate but offers comparable performance to decision trees and other induction methods. It also assumes that the predictive attributes are conditionally independent given the class, working on the assumption that no concealed attributes influence prediction.

Naïve Bayes prediction typically takes place using the following process, where C represents a random variable indicating an instance's class, A represents a vector of random variables indicating the attribute values that are observed, c represents the label of a class, and a represents a vector of observed attribute values:

1. Use Bayes' rule to compute the probability of each class given a:

$$p(C = c \mid A = a) = \frac{p(C = c) \prod p(A_i = a_i \mid C = c)}{Z}$$

2. Predict the most probable class

The product in the numerator is due to the conditional independence assumption, and the denominator is a normalization factor, so the sum of $p(C = c|A = a)$ overall classes is 1. Normally, discrete attributes are modeled using a single probability value, while numeric/continuous attributes are modeled using a continuous probability distribution. A popular density estimation method is a single Gaussian. Kernel density estimation has been proposed so that continuous variables can be better estimated [14], especially in the case where the distribution is multimodal rather than normal.

3.2.2.3 Instance-Based Learners: k*-Nearest Neighbor and* K*

Some algorithms make use of specific instances rather than precompiled abstractions (trees, networks, etc.) during prediction. These methods are categorically termed instance-based learning or example-based learning. They have a strong underpinning as a nearest neighbor method and generally use some form of similarity measure to determine "distance" or "match" between instances [15]. One of the more popular instance-based learning algorithms is the *k*-nearest neighbor classifier, which utilizes the *k* most similar (nearest) training examples in various ways for prediction. In general, these algorithms work on the assumption that similar instances belong to similar classes. An advantage of such methods is their simplicity, which makes them applicable to many domains and problems. For instance, one could utilize $k = 5$ nearest neighbors for prediction and weigh the neighbors by the inverse of their distance when voting on class membership.

*K** is another flavor of instance-based learners that utilizes entropy (motivated by information theory) as a distance measure [16]. This allows for a more consistent approach to handling missing values and symbolic and real attribute values. The *K** distance method's general approach is similar to work done in comparing DNA sequences, where the distance between two instances is computed as a measure of the sum of all possible transforming paths (represented by transformation probabilities). The selected number of instances is termed the "sphere of influence," which dictates how the instances are weighed. During classification, the category corresponding to the highest probability is selected.

3.2.2.4 Decision Tree (Pruned and Unpruned)

Decision trees are one of the most well-known and widely used machine learning methods and have seen successful use in a variety of real-world problem domains. Their utility as a simple and fast learning approach is widely accepted. Many decision tree algorithms and variations exist, the most popular of which are Quinlan's ID3 and successor C4.5 [17]. Decision trees produced by these algorithms are typically small/shallow and accurate, making for fast and reliable classification.

The process of building a decision tree begins with a set of example cases or instances, each containing a series of numerical or symbolic attributes and a membership class. Each internal node of the tree represents a test that determines the branch to travel down. For example, if an internal node's test is "$x > 42$" and x is 30, the test returns false and proceeds down the right branch of the tree at that node (tests returning true proceed down the left branch). The tree's leaf nodes represent the possible classes that instances can belong to, which is the prediction produced by the tree for test instances.

C4.5 utilizes formulas based on information theory (information gain and gain ratio) to measure the "goodness" of tests for nodes. Thus, tests are chosen that extract the maximum

information given a single attribute test and the set of cases. Overfitting is reduced by estimating the error rate of each subtree and replacing it with a leaf node if its estimated error is smaller, a process termed "pruning" [17]. Finally, the decision tree can be transformed into a set of rules by traversing the tree paths from the leaf nodes. The resulting set of rules can then be simplified in a variety of ways (and some rules completely removed) to arrive at a final set of rules for classification.

3.2.2.5 Random Forest

Rather than using a single decision tree for classification, many of them can be used in conjunction. Such a method is termed a random forest [18]. Random forests have their roots in work involving feature subset selection, random split selection, and ensemble classifiers. A random forest is constructed by selecting a random feature vector that is independent of the previous chosen feature vectors (but identically distributed) and growing a tree-structured classifier using this vector and the training set. This is done T times to create a random forest of T decision trees, each of which independently votes on a class. The most popular class from all trees is used to label the test instance.

Two primary measures are used to determine the accuracy and interdependence of the trees both individually and as a whole. The goal is to minimize tree correlation while maintaining individual tree strength. The accuracy of random forests, which has been found to be comparable to and sometimes better than AdaBoost, depends on the strength of the individual tree classifiers and the correlation/dependence between them [18]. Typically, choosing one or two random features to grow on each tree can provide near-optimum results. Random forests allow for any number of trees, which in some cases can be in excess of 100, considering one of a variety of feature subsets (e.g., $\log M + 1$ features, where M is the number of inputs).

3.2.2.6 Rule Learners: Repeated Incremental Pruning to Produce Error Reduction and PART

Rule-learning systems offer several desirable properties, including their human-understandable format and general efficiency. However, they are known to not scale well in relation to data size. One dominant rule-learning variant is the repeated incremental pruning to produce error reduction (RIPPER) propositional rule learner [19]. RIPPER is based on iterative reduced-error pruning (IREP), which, for rules, involves the training data being split into a growing set and a pruning set. A rule set is constructed, one rule at a time, in a greedy manner. When a rule is constructed, all examples that are covered by the rule are deleted via a pruning method.

The uncovered examples are randomly partitioned into two subsets: a growing set and a pruning set. A rule is grown by adding conditions that maximize an information gain criterion until the rule covers no negative examples. The rule is then pruned by deleting condition(s) that maximize a pruning function until the function's value cannot be improved by any deletion. This process is repeated to generate, prune, and optimize the rule set until there are no positive examples remaining, or a rule is found with a large error rate. RIPPER is generally considered to be competitive with C4.5 rules in terms of accuracy and is able to efficiently operate on very large (hundreds of thousands of examples) and noisy data sets. More details are provided on RIPPER's performance and comparison to C4.5 in the work of Cohen [19].

Another successful rule-learning algorithm is PART [20]. PART is a decision-list algorithm based on partial decision trees, combining the advantages of both C4.5 and RIPPER while eliminating some of their disadvantages. It is considered a separate-and-conquer algorithm, meaning that it generates one rule at a time, removes covered instances, and repeats. PART differs in the general construction of rules by creating a pruned decision tree for all current instances, building a rule corresponding to the leaf node with the largest coverage, then discarding the tree and continuing. Partial decision trees are constructed in a manner similar to C4.5. This is said to help avoid the problem of generalizing without knowing its implications [20]. PART avoids global optimization that is present in C4.5 and RIPPER but still produces accurate rule sets. In comparison, PART compares favorably to C4.5 in accuracy and outperforms RIPPER at the cost of a larger rule set.

3.2.2.7 Decision Table

Another rule-learning method is the decision table classifier, with a default rule mapping to the majority class [21]. A decision table has two main components: a schema (set of features) and a body (set of instances with feature values and class labels—the training set). An optimal feature set is determined by transforming the problem into a state space (feature subset) search, using best-first search with the heuristic being *k*-fold cross-validation to estimate the future prediction accuracy. Incremental cross-validation can be used to provide a speedup for algorithms that support incremental addition and deletion of rules [21].

Prediction for a test instance takes place by finding all labeled training examples that exactly match the features of the test instance. If this set is empty, the majority class of the entire training set is used for labeling the test instance; otherwise, the majority class of the set of matching training examples is used for labeling. Decision tables are more ideal for discrete attributes, as they attempt to find exact matches of the feature set for an instance for class assignment. A perfect match would be highly unlikely for continuous attributes in practice but may be more likely if the number of values is low. Also, as the accuracy prediction method is heuristic based, the best feature set may not be found. For some domains, the decision table was found to perform comparably with the C4.5 decision tree algorithm [17].

3.2.2.8 Artificial Neural Network

Artificial neural networks are computational models consisting of a network of interconnected neurons. The original concept for this approach was inspired by attempts to analyze and model the human brain, focusing on its nodes/neurons, dendrites, and axons. They are typically used to model, discover, or learn patterns in complex relationships between inputs and outputs. They operate on a connectionist approach, where neurons are fired, and their output is adjusted by weighted connections en route through the network via other neuron interfaces and layers.

Neurons represent simple processing entities, which, when connected in concert with other neurons, can exhibit complex global behavior. Each neuron utilizes a transfer function to adjust input for forwarding through connections to other neurons in subsequent neuron layers. A neural network is typically broken down into layers of neurons that behave similarly in terms of transfer function or connection interfaces. There is typically an input layer representing the nodes for inputs into the neural network, some hidden

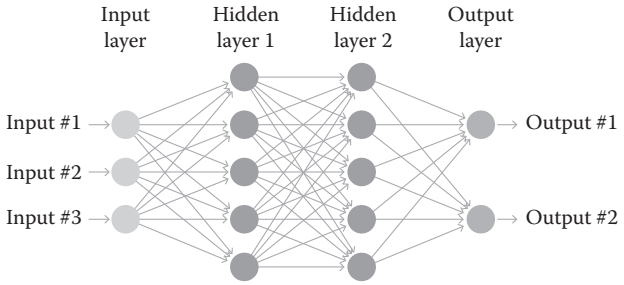

FIGURE 3.1
Fully connected neural network configuration with three input neurons, two hidden layers with five neurons each, and two output neurons.

intermediate layer(s) containing one or more neurons each, and an output layer, which produces the outputs from passing the input through the network. Figure 3.1 illustrates a fully connected neural network configuration, consisting of three input neurons, two hidden layers with five neurons each, and two output neurons.

Neural networks are trained by repeatedly providing the network with inputs and iterating until the error goal for network output has been achieved or a certain iteration threshold has been met. This occurs by providing the network with outputs corresponding to the inputs that it should attempt to learn. The network learns the association between inputs and outputs by passing each input through the network and calculating the error the network produces with its current connection weights. This error is then typically back-propagated to adjust connection weights in reverse, from output nodes to input nodes. By adjusting connection weights, the network can then adjust its output for given input. This process is repeated until a convergence goal is achieved. The connection weights of the network therefore represent the neural network's knowledge, given the instances it has seen and the output it has been provided to train toward. Neural networks exhibit primary disadvantages of being computationally slow in training and application phases, offering nondeterministic convergence (which can also be advantageous), and having difficulty in determining how/why the model made a particular decision. This affects both the use of a neural network model as well as its trustworthiness and acceptance of model outputs.

3.2.2.9 Support Vector Machine

Support vector machines are kernel-based learning machines that make no assumptions on the data distribution. Support vector machines are backed by strong mathematical rigor, are robust to noise, and are currently popular in the literature. They are designed for binary (two-class) problems but can be extended for multiclass classification data sets. The support vector Machine's goal is to find separators in the form of a hyperplane. The data are transformed into another feature space (including nonlinear features), followed by hyperplane construction. The best hyperplane is the one that represents the largest separation, or margin, between the two classes. Support vectors are samples on that margin. When new data are provided, a prediction is made based upon which portion of the margin the data point lays on. Various kernels have been developed for this method, one of which is the popular polynomial kernel.

3.2.3 Multi-Classifier Decision Fusion

Each machine learning paradigm has its own strengths and weaknesses stemming from its underlying theory, mathematical underpinning, and assumptions about the data and/or decision space. Therefore, the accuracy of each algorithm is dependent upon how well those assumptions apply to the input data. Different algorithms achieve different levels of accuracy on the same data, but specific classifiers can develop levels of expertise on portions of the decision space. These aspects provide support for investigating decision fusion and meta-learning approaches.

Since different learning methods typically converge to different solutions, combining decisions from different learning paradigms can be leveraged for improved accuracy [22,23]. Classifier combination is one such area in machine learning that has offered advances in classification accuracy for complex data sets. It has been termed differently in the literature, namely, classifier fusion, mixture of experts, committees, ensembles, teams, collective recognition, composite systems, and so forth. Generally, when predictions from multiple classifiers are combined, they are said to form an ensemble that is then used to classify new examples. Figure 3.2 illustrates the concept of combining decisions from N classifiers to arrive at a fused, consensus decision and confidence.

When developing a multi-classifier system, its members can be a mixture of weak (i.e., high-error-rate) and strong (i.e., low-error-rate) classifiers. Weak classifiers are typically simple to create, at the expense of their accuracy on complex data sets. Strong classifiers are typically time consuming and expensive to create, as their parameters are fine-tuned and tweaked for optimal performance. Combining weak/strong or homogeneous/heterogeneous classifiers offers the benefit of encompassing different levels of expertise and knowledge bases [23].

An important aspect related to differences in learners is their level of error correlation relative to one another, and as a unit. The more correlated (i.e., less disjoint) individual learners are, the less complementary they may be. If they are uncorrelated, they likely will misclassify different instances, and combining them better enables the system to correctly classify more instances. A significant improvement over a single classifier can only happen if the individual classifier theories are substantially different. It is desired to obtain a balance between high performance and complementarity in a team where decisions are combined. If one learner does not predict correctly, the other learners should be able to do so. Diverse models are therefore more likely

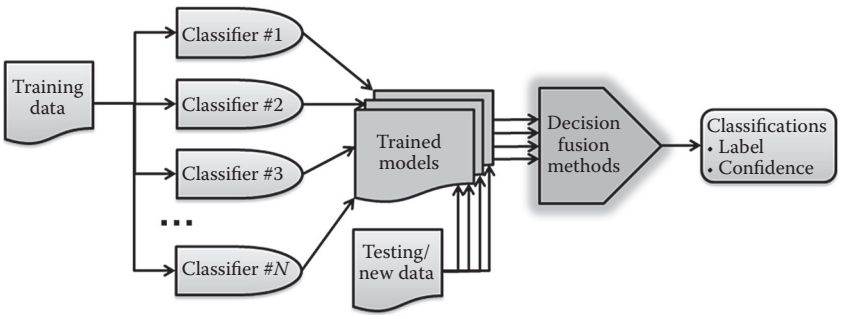

FIGURE 3.2
Decision fusion process for a multi-classifier system, where decisions are collected from a set of classifiers and combined in some way to arrive at a single classification label and confidence for each instance.

to make errors in different ways. Methods to accomplish this typically involve introducing diversity in terms of learning paradigms, feature subsets, or training sets. As modern data volumes grow in size, attempts at distributed processing and parallel machine learning are gaining popularity.

Various approaches exist for combining decisions from multiple classifiers, some of which perform better under certain circumstances. Pure voting methods are among the simplest ways to combine decisions from multiple classifiers. Each entity classifies (votes) that an input instance belongs to one of multiple classes. These votes are tallied in one of a number of ways to arrive at the final decision of the classifiers being fused in the form of a single class label. Examples of pure voting methods are majority vote, maximum confidence, average confidence, and product of confidences.

More intelligent voting methods are accuracy driven. Accuracy-driven voting methods are those that take into account the accuracy, confidence, or probability of each learner offering its vote. This moves a step beyond simple voting, where votes from classifiers that have exhibited strong classification accuracies are weighted more, and weak classifier votes are either discarded or weakly weighted. A popular accuracy-driven voting method is weighted voting, where each classifier is assigned a voting weight (usually proportional to its operational accuracy), and the class with the highest weighted vote is used for classification.

Data manipulation methods, which manipulate the training and testing data to attempt to achieve optimal classification accuracy, are by far the most popular means to train and combine multiple classifiers. They have seen widespread use due to their simplicity and mathematical background and have produced some of the best accuracies on challenging data sets. Some of the popular data manipulation methods include the following:

- **Input decimation:** Classifier correlation can be reduced by purposefully withholding some parts of each pattern (i.e., only using a subset of the features for certain classifiers). Feature inputs can be "pruned" by measuring how each affects the classifier output. Those features that have the least effect on the output can be selected for removal without compromising overall classifier performance. One example is to have one classifier per class, where each classifier only uses the features with high correlation to that class.
- **Boosting:** Boosting adaptively changes the training set distribution based on performance of previous classifiers, attempting to reduce both bias and variance. Based on results of previous classifiers, diverse training samples are collected such that instances that were incorrectly predicted play a more important role in training (i.e., further learning focuses on difficult examples). This method relies on multiple learning iterations. At each iteration, instances incorrectly classified are given greater weight in the next iteration. Thus, the classifier in each iteration is forced to concentrate on instances it was unable to correctly classify in previous iterations. All classifiers are then combined after all iterations have been processed, or a threshold is met.
- **Bagging:** Bagging creates a family of classifiers by training on stochastically different portions of the training set. N training "bags" are initially created, each obtained by taking a training set of size S and sampling the training set S times with replacement. Some instances could occur multiple times, while others may not appear at all. Each bag is then used to train a classifier, and classifiers are then combined using an equal weight for each.

3.2.4 Ensemble Learning, Meta-Learning, and Other Abstract Methods

One of the most active areas in supervised learning is constructing successful ensembles of classifiers. Ensemble methods construct a set of classifiers and then classify new data points by taking a (weighted) vote of their predictions [24]. Ensembles are well established as a method for obtaining highly accurate classifiers by combining less accurate ones. They can often outperform any single classifier. However, to be more accurate than a single classifier, the ensemble must be composed of both accurate and diverse classifiers [24].

Similarly, meta-learning uses a machine learning algorithm to model the decision patterns of a set of classifiers to construct a model that could yield increased accuracy. Figure 3.3 illustrates the meta-learning process for a multi-classifier system, where decisions are collected from a set of classifiers, used to create a model of the decision pattern of those classifiers, and that model is tested to arrive at a single classification for each instance. The goal of the meta-classifier is to learn the mapping of classifier decisions that provides the most correct labels based on the training data. The meta-classifier will then label input instances based on the decisions from the underlying individual classifiers. This process is also known as "stacking."

3.2.5 Evolutionary Algorithms

Evolutionary algorithms are a class of stochastic search and optimization methods that mimic natural biological evolution. The common underlying premise is that given a population of individuals, the environmental pressure causes natural selection, which further causes a rise in the quality of the population [25]. Given a quality function to be maximized (i.e., the *fitness function*), a random set of candidate solutions can be evolved that maximize or improve upon previous fitness. This fitness measure ensures that the better (more fit) candidates are selected to seed the next generation as baseline candidates, further evolved by applying reproduction operators such as *recombination* and *mutation* to them. Recombination is an operator that applies to two or more selected candidates (the parents) and results in one or more new candidates (the children). Mutation is applied to one candidate and results in one new candidate. Taken together, the operations of recombination and mutation produce a set of new candidate solutions (the offspring) that

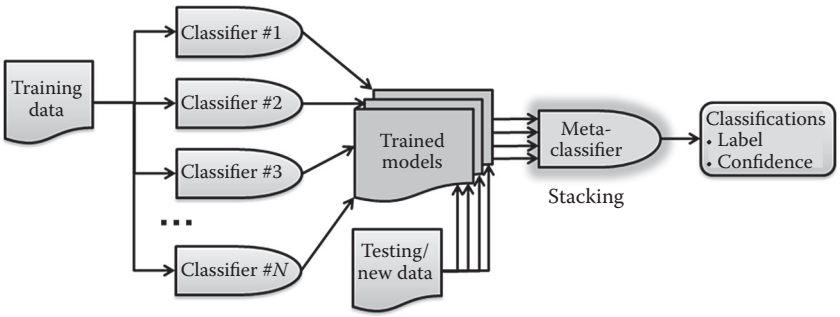

FIGURE 3.3
Meta-learning process for a multi-classifier system, where decisions are collected from a set of classifiers, used to create a model of the decision pattern of those classifiers, and that model is tested to arrive at a single classification label and confidence for each instance. This process is also known as "stacking."

compete—based on their fitness—with other candidates for inclusion in the next generation. This process, called a *generation*, is iterated until a termination condition is met. The termination condition can be the creation of a solution with sufficient quality, a time limit, or some computational limit [25]. The iterative process of computing and evolving a population is illustrated in Figure 3.4.

Evolutionary algorithms differ substantially from more traditional search and optimization methods [25]. The most significant differences are as follows:

- Use stochastic transition rules, not deterministic ones.
- Search multiple points in the solution space, not just a single point.
- Do not require problem-specific knowledge; only the fitness levels influence the directions of search [26].
- Are usually more straightforward to apply, as no restrictions for the definition of the objective function exist [26].
- Can provide a number of potential solutions to a given problem. The selection of the solution to utilize is left to the user. Thus, in cases where the particular problem does not have one individual/unique solution, as in the case of multiobjective optimization and scheduling problems, the evolutionary algorithm can be leveraged for identifying several high-quality alternative solutions.

The general pseudocode for an evolutionary algorithm can be summarized by the following:

```
BEGIN
       INITIALIZE Population with random Candidate solutions
       EVALUATE each Candidate;
REPEAT WHILE (TERMINATION CONDITION is not satisfied)
       SELECT Parents;
       RECOMBINE pairs of Parents;
       MUTATE resulting Offspring;
       EVALUATE new Candidates;
       SELECT Individuals for the next Generation;
END
```

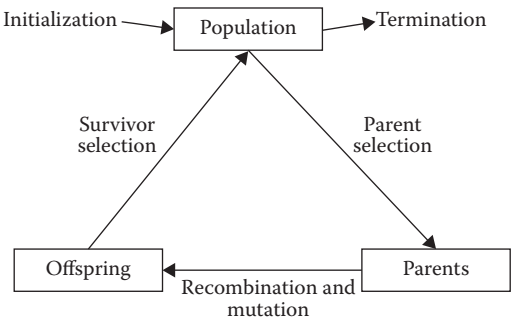

FIGURE 3.4
Iterative evolutionary process by which a population is created and evolved to arrive at a set of candidate solutions that maximize a fitness function.

3.2.5.1 Representation

The first step in defining any evolutionary algorithm is relating the paradigm to the problem at hand. The representation of individual candidate solutions is the manner by which the problem context is mapped to the problem-solving space where evolution can take place. Objects forming possible solutions within the problem context are referred to as *phenotypes*. The encodings of individuals in the population are referred to as *genomes* [25]. Thus, the design of any evolutionary algorithm must begin with specifying a representation, which is a mapping from the phenotypes onto a set of genotypes that represent candidate solutions. It is important to note that the phenotype space is often very different from genotype space [26]. For example, a binary string can be used as the genotype to describe a phenotype, which is a letter of the English alphabet. The entire evolutionary process occurs in the genotype space. The final solution is the phenotype that is obtained from decoding the fittest genotype after the termination condition.

3.2.5.2 Fitness Function

The fitness function represents a measure by which individuals of a population are evaluated and compared, with the goal of maximizing the overall fitness of a population through the evolutionary process. This function therefore represents the basis for parent and survivor selection and must be able to distinguish between individuals that are worthless, average, good, and excellent. The fitness function, together with the representation, provides the only view into the true problem space that the evolutionary algorithm computationally operates in [25]. The fitness function should generally be linear in its relationship with the problem objective function. It is possible that for a given problem, the fitness function is not the same as the problem objective function, as is the case with a minimization problem. In these cases, the fitness function may be the inverted objective function. In most cases, converting an objective function to a fitness function is trivial. Without a good fitness function, an evolutionary algorithm is directionless.

3.2.5.3 Population

The collection of individuals on which the evolutionary algorithm operates is called the "population." The population is a set of genotypes where multiple copies of the same genotype are allowed. Individual genotypes are static and do not evolve, so the unit of evolution is the whole population itself [25]. In many traditional implementations, defining a population is as simple as specifying the number of genomes held within it. In more sophisticated evolutionary algorithms, multiple populations can be simultaneously represented and evolved. Multipopulation algorithms include additional operators such as migration and competition.

3.2.5.4 Genetic Algorithms

Genetic algorithms are a special case of the more general class of evolutionary algorithms [25]. Genetic algorithms are the most widely known type of evolutionary algorithm. The primary construct that distinguishes genetic algorithms from other evolutionary algorithm variations is the representation of the candidate solutions. The representation of an individual in a genetic algorithm is defined as a string over a limited alphabet. Various applications fit this representation and general optimization strategy, including automated drug design, sequence alignment, molecular structure/folding/docking optimization, and general computational biology.

3.3 Machine Learning and Data Mining Resources

Many resources, most of which are open source, are available to support the development and application of these techniques across various domains and development environments. Toolboxes are available for most modern languages (Java, C#, Python) and scientific computing platforms (MATLAB®, IDL, R). For instance, MATLAB offers a neural network toolbox and a global optimization toolbox (contains genetic algorithm methods), among others, that cover many of the algorithms discussed in this chapter. Many other custom toolboxes can be downloaded that offer a range of techniques, variations, and support tools.

The Waikato Environment for Knowledge Analysis (Weka), developed and managed by the University of Waikato, is a comprehensive open-source set of object-oriented machine learning and data mining algorithms written in Java [27]. In addition to classification, regression, clustering, and association rule algorithms, it also provides methods for data filtering, preprocessing, and visualization. The set of packages provides interfaces to preprocessing routines including feature selection, categorical and numeric learning tasks, performance enhancement of classifiers, evaluation according to different criteria such as root mean squared error, and experimental support for verifying the robustness of models. Functionalities such as training, testing, and using classifiers are provided for all algorithms. Additionally, all algorithms are implemented using the same software architecture, so comparing, combining, and cross-validating algorithms is straightforward. Weka has been extensively used in machine learning research to develop new techniques and compare to the state of the art. RapidMiner [28] is another free machine learning suite similar to Weka that has a large user community.

3.4 Example/Illustrative Medical Applications

Artificial intelligence is widely used in the medical domain, several representative examples of which are included in other chapters of this book. Machine learning and data mining are currently popular mechanisms for supporting medical research and creating models to better understand complex data sets, relationships, and associations. This section includes several relevant examples that illustrate how these methods can be used for medical applications.

3.4.1 Multiagent Infectious Disease Propagation and Outbreak Prediction

Anticipating, predicting, and monitoring disease outbreaks can assist public health officials in their response efforts. To this end, the following capabilities are desirable: (1) to predict when and where an outbreak will occur; (2) should an outbreak occur in a certain location, to predict how it will spread; and (3) to monitor an outbreak as it is occurring. Various efforts are focused on (1) and (3); however, there is a need to develop the second capability (i.e., to predict how an outbreak will spread). Understanding the communicability circumstances (the how) as well as the spatial (the where) and temporal (the when, and how fast) spread characteristics of an infectious disease can aid in preparation and active intervention efforts. Often, this capability is called "disease propagation modeling."

Historically, models based on differential equations have been used to predict how a disease will spread. These models emphasize the numbers of people who are susceptible to contracting the disease, are infected/infectious with the disease, and have recovered from the disease. However, they typically do not describe how the disease will spread spatially. Another technique for disease propagation modeling that has gained recent prominence is agent-based modeling, which is based on the multiagent system paradigm. Rather than consider broad groups of people, this technique represents people individually and attempts to characterize their behavioral patterns. It is hoped that first-hand encounters between people can then also be predicted, therein directly representing the opportunities for the disease to be transmitted from one person to another.

The power of such a system is that new diseases or disease variations can be specified, and many simulations can be run to understand how those diseases spread under certain population, travel, and social constraints. It also enables the evaluation of vaccination/intervention mechanisms by simulating the intervention and measuring the corresponding decrease in infection, death, or spread as a surrogate variable of the intervention policy's utility. Similarly, such systems can be leveraged for sensor placement optimization, in cases where a limited number of sensors are available and need to be optimally placed for sufficient impact.

Efforts to model disease propagation are extensive, for many years focusing on *SEIR* differential equations models [29], where *S* is the number of susceptible individuals, *E* the number exposed, *I* the number infectious, and *R* the number recovered. Within the past decade, greater attention within the field has been given to agent-based models, and Connell et al. [30] offer a comparison of the two methods. Such agent-based models, of course, are conducive to computer simulation development, as has been documented in [31]. Incidentally, agent-based models have also been applied to many other areas of study, including containment of terrorist bioattacks [32], the spread of fear [33] or civil unrest [34], and catastrophic-event evacuations [35].

Infectious disease outbreak prediction represents an area of current global importance and research interest. Data mining can be leveraged to reveal patterns of disease outbreaks and forecast the location and time frame of an outbreak before its emergence. Dengue fever, for example, is an endemic without a vaccine that has tens of millions of cases each year. Predictive modeling efforts are currently being explored, such as the use of logistic regression and fuzzy association rule mining, to discover complex relationships between confirmed disease outbreaks and environmental, biological, ecological, and sociopolitical variables [36]. Predicting dengue fever outbreak in a timely manner could enable active public health interventions that can aid in limiting, localizing, or eliminating the outbreak from occurring.

3.4.2 Automated Amblyopia (Lazy Eye) Screening System

Amblyopia, commonly referred to as lazy eye, is a neurological vision disorder that studies show affects 2%–5% of the population. Current methods of treatment produce the best visual outcome if the condition is identified early in the patient's life. Several early screening procedures are aimed at finding the condition while the patient is a child, including an automated vision screening system (AVVDA) [37] that uses artificial intelligence algorithms to identify patients who are at risk for developing the amblyopic condition and should be referred to a specialist. AVVDA uses case-based reasoning, C4.5 decision tree, and artificial neural network classifiers to assist in making the decision. Various features are used by this system as input to the machine learning algorithms,

including color density, Hirschberg reflex, and iris and pupil color slopes. Continued efforts based on multi-classifier decision fusion have improved upon the existing AVVDA performance [38].

3.4.3 Anticancer Drug Design and High-Throughput Screening

Taxol is a drug that has been shown to offer benefits in treating breast and ovarian cancers, and represents one of a small set of anticancer drugs that has been discovered, designed, and tested. Thus, Taxol becomes a candidate of further study to determine what features of Taxol make it an anticancer drug and to design other (potentially new) drugs that may offer improved performance or broader applicability. Machine learning algorithms (e.g., artificial neural networks) can be used to learn a model of Taxol's composite features and measured performance on breast and ovarian cancer cases. The model can then be used to evaluate existing drugs as anticancer drugs to consider for testing (i.e., classifier output is "test" versus "don't test" based on candidate drug composite features). The model could also be used to suggest (or design) composite feature sets that are highly similar to, or better than, that of Taxol and its anticancer performance metrics. A primary challenge of this problem area is limited and/or highly imbalanced training data.

Given recent advancements in high-performance computing and sequencing technologies, high-throughput screening has become commonplace in the medical community. The premise of high-throughput screening is simple: distinguish between active and nonactive cancer drugs (e.g., cobalt) given a set of experimentally determined and known features for each drug. The goal of such systems is to maximize the number of correctly identified active drugs (AA) while minimizing the number of nonactive drugs identified as active (NA). If a drug researcher plans on purchasing and testing all of the drugs the screening system classifies as active, it is desired to maximize the percentage (AA/NA) of those purchased drugs that are actually active. By raising this percentage, we lower the number of drugs it is necessary to buy to achieve the desired amount of active compounds, which saves a significant amount of time and money. If researchers simply buy all of the drugs they encounter, they will find a very low percentage of them to be active. Bayes classifiers and artificial neural network systems have been created for this task that were able to ensure that, on average, 6% were active, some leveraging more advanced methods such as multi-nets (combining multiple artificial neural networks with different characteristics).

3.4.4 Genetic Sequence Classification

Many biological classification problems exist that require sophisticated methods to learn the differences between large, highly overlapping data classes. One example is genetic sequence alignment, where a class of sequences with biologically significant properties is leveraged for developing a system and model capable of automatically identifying significant sequences. This represents a two-class problem: significant versus insignificant genetic sequences. Inputs are generally of the genetic sequence form (A,C,G,T). Within genetic sequence alignment, one focus area could be the identification of sequences that are found in an exonic splicing enhancers (ESE) motif recognized by the human serine rich protein SC35. Selected sequences for artificial neural network model creation, for instance, could be functional and specific by being those that promote splicing in a nuclear extract complemented by SC35 but not by SF2/ASF. Input test sequences could then be identified as potentially sharing one or two matches to a short and highly degenerate octamer consensus. Once a model capable of distinguishing between significant and insignificant

sequences is established, it can be used to test new sequences for significance or to perform large-scale simulations to discover (potentially new) sequences that are deemed significant for further investigation.

3.5 Conclusions

This chapter has introduced many prominent machine learning and data mining methods and discussed considerations of their application and impact on the problem domain. Example applications to the medical domain were provided to illustrate how such methods can be applied to current complex problems. The field of artificial intelligence has and will continue to play an important role in the advancement of scientific research, modern health and human information systems, and the broader medical community. Upcoming chapters leverage many of these methods to further improve the state of the art and offer solutions to some of today's most challenging medical applications.

References

1. R. Agrawal, T. Imielinski, and A. Swami, Mining association rules between sets of items in large databases, In *Proceedings of the International Conference on Management of Data*, Washington, DC, pp. 207–216, 1993.
2. R. Srikant and R. Agrawal, Mining quantitative association rules in large relational tables, In *Proceedings of the International Conference on Management of Data*, Montreal, Quebec, Canada, pp. 1–12, 1996.
3. C.M. Kuok, A. Fu, and M.H. Wong, Mining fuzzy association rules in databases, In *ACM SIGMOD Record*, 27(1), New York, pp. 41–46, 1998.
4. A.L. Buczak and C.M. Gifford, Fuzzy association rule mining for community crime pattern discovery, In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining: Workshop on Intelligence and Security Informatics*, Washington, DC, 2010.
5. J. Han, J. Pei, and Y. Yin, Mining frequent patterns without candidate generation, In *Proceedings of the ACM International Conference on Management of Data*, New York, pp. 1–12, 2000.
6. C.-W. Lin, T.-P. Hong, and W.-H. Lu, Linguistic data mining with fuzzy FP-trees, *Expert Systems With Applications*, 37, pp. 4560–4567, 2010.
7. R. Srikant and R. Agrawal, Mining sequential patterns: generalizations and performance improvements, In *Proceedings of the International Conference on Extending Database Technology: Advances in Database Technology*, London, pp. 3–17, 1996.
8. J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu, Mining sequential patterns by pattern-growth: the PrefixSpan approach, *IEEE Transactions on Knowledge and Data Engineering*, 16(11), pp. 1424–1440, 2004.
9. J. Wang and J. Han, BIDE: efficient mining of frequent closed sequences, In *Proceedings of the International Conference on Data Engineering*, Washington, DC, pp. 79–90, 2004.
10. P. Tzvetkov, X. Yan, and Y. Han, TSP: mining top-k closed sequential patterns, In *Proceedings of the IEEE International Conference on Data Mining*, Melbourne, FL, pp. 347–354, 2003.
11. X. Yan, J. Han, and R. Afshar, CloSpan: mining closed sequential patterns in large datasets, In *Proceedings of the International Conference on Data Mining*, San Francisco, pp. 166–177, 2003.