Chapter 10

Outlier Detection in Discrete Sequences

"Poets write the words you have heard before, but in a new sequence." – Brian Harris

10.1 Introduction

The previous chapter discusses anomaly detection from the perspective of continuous time series. A related setting is one in which the individual elements at each time stamp are discrete-valued (i.e., categorical). Such discrete time-series are also referred to as *sequences*. Discrete-valued temporal scenarios arise in numerous systems diagnosis, intrusion detection, and biological applications. In some domains such as intrusion detection and systems diagnosis, the discrete sequences are caused by *temporal ordering*, whereas in other domains such as biological data, the discrete sequences are caused by *physical ordering*. Nevertheless, at a logical level, the differences in the problem definitions for the two cases are relatively minor. The primary difference is that temporal data often has a specific direction to the analysis in real scenarios (i.e., forward in time), whereas this may not be the case for data based on placement relationships. At the analytical level, the models for the two cases are different in minimal ways and typically have cross-applicability. Some examples of applications that generate discrete data sequences are as follows:

- **Systems diagnosis:** Many automated systems continuously generate data about the system state. These could correspond to UNIX system calls, aircraft system states, mechanical systems, or host-based intrusion detection systems. The last case is particularly common, and is an important research area in its own right.
- **Biological data:** Biological data typically contains sequences of amino-acids, in which anomalous subsequences can provide interesting information about unusual properties of the genome sequences and their impact on different types of genetic conditions [365].
- **User-action sequences:** Such sequences are created by user *actions* in different application domains:

C.C. Aggarwal, *Outlier Analysis*, DOI 10.1007/978-3-319-47578-3_10

– Web logs contain long sequences of visits to Web sites by individuals. It is often desired to identify interesting subsequences that are indicative of anomalous or intrusive activity.

– Customer transactions may contain sequences of buying behavior. The symbols may correspond to the identifiers of the different items that are bought. Unusual temporal patterns may correspond to changes in buying patterns [122].

– User actions on Web sites such as online banking sites are frequently logged. This is similar to the aforementioned case of Web logs, except that the logs of banking sites are often more detailed. Anomalous subsequences of actions provide insights into unusual user activity, such as an attempt to break into the system.

In order to ease further discussion, some notations will be defined. A sequence is defined as an ordered set of symbols $a_1 a_2 \ldots a_r$ drawn from the symbol set $\Sigma = \{\sigma_1 \ldots \sigma_{|\Sigma|}\}$. In the most general form, a sequence can also be defined as an ordered set of sets $S_1 S_2 \ldots S_r$, where each S_i is a subset of Σ. In this chapter, the more common (and simpler) case will be examined in greater detail, where each element a_i is drawn directly from Σ. The more complex case of set-based sequences will also be examined briefly.

Discrete sequences are different from continuous time-series data, because it is difficult to directly compare two different symbols from the alphabet Σ in terms of distances. Therefore, commonly used regression-modeling methods for deviation detection in continuous data are not easily generalizable to this case. Nevertheless, even in this case, outliers can be defined in terms of either deviations from predicted values at specific time stamps, or in terms of unusual *successive combinations of sequence values*. The key is to define an appropriate predictive or regularity model in the discrete case that is analogous to its continuous counterpart. As in the case of continuous data, outliers are of two types, depending on whether *specific positions* are considered outliers, or whether *combinations of symbols* are considered outliers.

- **Position outliers:** In position-based outliers, the values at specific positions are predicted by a model. This prediction is used in order to determine the *deviation* from the model, and predict specific *positions* as outliers. Typically, Markovian methods are used for predictive outlier detection. This is analogous to deviation-based outliers that are discovered in time-series data with the use of regression models. Unlike regression models, Markovian models are better suited to discrete data. These correspond to *contextual* outliers.

- **Combination outliers:** In combination outliers, an entire test sequence is deemed to be unusual because of the combination of symbols in it. This is because this combination may rarely occur in a sequence database (frequency-based), or its distance (similarity) to most other subsequences of similar size may be very large (small). More complex models such as hidden Markov models can also be used to model the frequency of presence in terms of generative probabilities. For a longer test sequence, smaller subsequences are extracted from it for testing, and then the outlier score of the entire sequence is predicted as a combination of these values. This is analogous to the determination of unusual shapes in time-series data. These correspond to *collective* outliers.

The rarity in the combination of values can be defined in different ways depending on the specific choice of model. The most commonly used models are as follows:

- **Distance-based:** In this case, the nearest-neighbor distance of a subsequence to other candidate subsequences in the base data is computed. This provides an outlier score for the subsequence. The score of a longer sequence can be computed by combining the outlier scores of its smaller subsequences.

  For the case of shorter sequences, methods such as clustering can be used for the outlier analysis process. Such methods are analogous to distance-based techniques for outlier analysis in multidimensional data.

- **Frequency-based:** In this case, the frequency of the occurrence of different subsequences of values is compared between a test instance and the base training data. Sequences that show unusual differences in frequency distributions of subsequences between the test and training sequence are presumed to be outliers. Such methods are useful in cases in which the compared subsequence is relatively small, and the alphabet size is small. In such cases, a limited number of distinct values of the subsequence is possible, and it is meaningful to talk in terms of frequencies. As in the case of distance-based models, smaller subsequences are extracted for meaningful frequency comparisons.

- **Model-based:** In this case, a probabilistic generative model is constructed that generates the subsequences. A specific example is the *hidden Markov model*. Subsequences that have low probability of being generated by the model are predicted to be outliers. Such methods are analogous to expectation-maximization algorithms for outlier detection in multidimensional data. The advantage of such models in the discrete case is that a well-designed generative Markov model can encode both the user understanding of the sequences, and can also capture highly likely sequences that are not explicitly present in the data.

- **Transformation-based:** The sequences are transformed into a new space with the use of kernel methods. A pairwise object-to-object similarity matrix is constructed. Methods like kernel-PCA and one-class support vector machines can be used in conjunction with the resulting similarity matrix.

Supervision can be incorporated when labels are available. This requires the design of rare-class adaptations of sequence classification algorithms.

This chapter is organized as follows. Section 10.2 discusses predictive models for outlier detection of individual positions in sequences. Section 10.3 discusses methods for determining combination outliers in discrete sequences. Complex sequences correspond to set-valued symbols, and multivariate sequences. The detection of outliers in such sequences is discussed in section 10.4. Online and early outlier detection is also discussed in this section. Methods for supervised outlier detection are discussed in section 10.5. The conclusions and summary are presented in section 10.6.

10.2 Position Outliers

In continuous time-series data (Chapter 9), it is discussed how one can identify point outliers as significant deviations from *expected* values at time stamps. Thus, these methods intimately combine the problems of *forecasting* and *deviation-detection*. In the case of continuous data, regression models are utilized for forecasting. A similar principle applies to discrete sequence data, in which the discrete values at specific positions can be predicted.

The main difference is in the choice of models used for predicting *discrete* values rather than continuous ones. When a position has very low probability of matching its forecasted value, it is reported as an outlier. For example, consider a Radio-Frequency Identification (RFID) application, in which event sequences are associated with product items in a superstore with the use of semantic extraction from RFID tags. A typical (normal) event sequence might appear as follows:

```
PlacedOnShelf, RemovedFromShelf, CheckOut, ExitStore.
```

On the other hand, in a shoplifting scenario, the event sequence may be *unusually* different. An example of event sequence in the case of the shoplifting scenario is as follows:

```
PlacedOnShelf, RemovedFromShelf, ExitStore.
```

Clearly, the sequence symbol *ExitStore* is anomalous in the second case but not in the first case, because it does not depict the *expected* or *forecasted* value for that position. It is often desirable to detect such anomalous *positions* on the basis of expected values. Such anomalous positions may appear anywhere in the sequence and not necessarily in the final element of the sequence (as in the aforementioned example). The basic problem definition for position outlier detection is as follows:

Definition 10.2.1 (Semi-Supervised) *Given a set of training sequences $\mathcal{D} = T_1 \ldots T_N$, and a test sequence $V = a_1 \ldots a_n$, determine if the position a_i in the test sequence should be considered an anomaly based on its expected value.*

Some formulations do not explicitly distinguish between training and test sequences. This is because a sequence can be used for both model construction and prediction, especially when it is very long. This is analogous to the case of unsupervised models in which training and test data are not differentiated. For example, such an unsupervised formulation may be as follows:

Definition 10.2.2 (Unsupervised) *Given a long sequence $V = a_1 \ldots a_n$, determine if the position a_i in the test sequence should be considered an anomaly based on its expected value.*

The differences between these two models are relatively small, because the latter model simply needs to use the sequence V in order to construct the model. For ease in discussion, the first definition will be used in this chapter.

Typically, position a_i can be predicted in temporal domains only from the positions before a_i, whereas in other domains such as biological data, both directions may be relevant. The discussion below will assume the temporal scenario, although generalization to the placement scenario (as in biological data) is achieved in a straightforward way by examining windows on both sides of the position.

In many applications such as Web logs (for anomalous Web page-access prediction), the training sequence might exist as a single long sequence. Furthermore, the training and test sequences may not be cleanly separated. In such settings, it may be desirable to use the recent history of accesses to identify anomalous positions. The models discussed in this chapter are general enough to be applicable to these variations with small modifications.

Just as regression modeling of continuous streams uses small windows of past history, discrete-sequence prediction also uses small windows of the symbols. It is assumed that the prediction of the values at a position depends upon this short history. This is known as the *short-memory property* of discrete sequences, which generally holds true across a wide variety of temporal application domains [465]:

Definition 10.2.3 (Short Memory Property) *For a sequence of symbols* $V = a_1 \ldots a_i \ldots$, *the value of the probability* $P(a_i|a_1 \ldots a_{i-1})$ *can usually be accurately approximated as* $P(a_i|a_{i-k} \ldots a_{i-1})$ *for some small value of* k *in most application domains.*

Another observation is that the influence of a_{i-j} on a_i is typically greater at smaller values of j. Intuitively, this is because recent sequence symbols are more likely to be relevant for predicting the current symbol. After the value of $P(a_n|a_{n-k} \ldots a_{n-1})$ has been estimated, a position can be flagged as an outlier when the *observed* symbol in a test sequence has very low predicted probability of occurrence.

This section will discuss two types of methods for position outlier detection, known as *rule-based* and *Markovian* models. Both Markovian and rule-based models, which are technically equivalent, exploit the *small memory* property of sequences in order to explicitly model the sequences as a set of states in a Markov chain. This equivalence is not obvious at first sight. When the size of the history used for prediction is large, the number of states in the Markovian model increases to potentially $|\Sigma|^k$. Rule-based models can be used to provide a pruned and incomplete heuristic description of the transition behavior implied by the Markovian model. This provides a simpler framework for understanding the behavior of the sequences. Thus, rule-based models can be considered heuristic simplifications of (more formal) Markovian models.

10.2.1 Rule-Based Models

The primary goal in rule-based models is to estimate the value of $P(a_i|a_{i-k} \ldots a_{i-1})$ from the training database of sequences $\mathcal{D}$. When the value of $P(a_i|a_{i-k} \ldots a_{i-1})$ is large, the following rule can be expressed:

$$a_{i-k} \ldots a_{i-1} \Rightarrow a_i$$

The predictive probability $P(a_i|a_{i-k} \ldots a_{i-1})$ is often referred to as the *confidence* of the rule. Rules may be generated either lazily for specific test sequences, or they may be generated using pre-processing on the training data. In the latter case, a relevant subset of rules (to the specific test sequence) is identified and used for prediction.

What are the challenges in robust estimation of the rules from the training data? The main problem is the large number of possibilities for the antecedent even at small window sizes (value of k). In fact, a particular sequence $a_{i-k} \ldots a_{i-1}$ may not even occur in a modestly sized training data set, and therefore rules containing such antecedents may not occur. When the number of occurrences is small, the corresponding rule confidence cannot be estimated accurately. Therefore, for a particular antecedent subsequence in the test sequence, it may not be possible to estimate the probability of the next forecasted symbol in a robust way. In order to improve robustness, a number of heuristic relaxations and variations of the basic probability-estimation approach have been developed:

- **Support criterion:** The rules should not be generated using only the confidence criterion. Both support *and* confidence criteria should be used. The *support* of a rule is defined by the number of times that the antecedent of the rule occurs in the training data. Only rules with large support may be relevant. This condition ensures that one does not use noisy rules that are caused by chance.

- **Variable antecedent size:** Not all relevant rules may correspond to a fixed window size. In many cases, the lengths of the antecedents of the rules may vary considerably.

- **"Don't care" positions:** Allowing "don't care" positions in the antecedents of the rules allows for the possibility of "noise" symbols that are not relevant to prediction. For example, a rule may be specified as follows:

$$a_1 a_2 * * a_5 \Rightarrow a_6$$

Here, the asterisk represents a "don't care" position, which may be noisy and may not have predictive power for the symbol a_6.

Although the support-confidence criterion of frequent and sequential pattern mining [33] provides a natural framework for rule-generation, many other off-the-shelf methods are available. For example, classification rules can be generated by extracting windows from the training sequence and treating the last symbol as a class label. Many off-the-shelf classification-rule generators such as *RIPPER* are available for this purpose [148]. An example of such a rule-based model is provided in [349]. For a given test subsequence, the first *fired* rule is examined. A rule is said to be fired by a test subsequence if it matches the antecedent of the rule. If the right-hand side of the fired rule corresponds to a *different* symbol as the value in the test sequence and the rule has high confidence, then this position is flagged as an anomaly. The confidence of the rule may be used as the outlier score. Numerous off-the-shelf rule generators are available for sequence data in the literature. Refer to the bibliographic notes for a discussion of these models.

10.2.2 Markovian Models

These models represent the sequence-generation process with the use of transitions in a Markov chain. This is essentially a special kind of *Finite State Automaton*, in which the states are defined by a short (immediately preceding) history of the sequences generated. Such models correspond to a set of states A, which represent the different types of memory about history of the sequence. For example, in *first-order* Markov models, each state represents the symbol from the alphabet Σ, which is generated as the final element of the sequence being modeled. Thus, the word "first-order" refers to the fact that the memory of these models is only 1. In kth order Markov models, each state corresponds to the subsequence of the final k symbols $a_{n-k} \ldots a_{n-1}$ in the sequence being modeled. Each transition in this model corresponds to an event a_n, representing the addition of the element a_n at the end of the sequence. As a result of the addition of this element, the Markovian model transitions from state $a_{n-k} \ldots a_{n-1}$ to the state $a_{n-k+1} \ldots a_n$. The probability of this transition is denoted by $P(a_n | a_{n-k} \ldots a_{n-1})$. It is noteworthy that this transition probability is the same as the confidence of a rule discussed in the previous section:

$$a_{n-k} \ldots a_{n-1} \Rightarrow a_n$$

This connection shows why Markovian models are essentially the same as rule-based models with a few minor differences in terms of formalization.

A Markovian model can be expressed as a set of nodes representing the states, and a set of edges representing the events, which cause movement from one state to another. Each state corresponds to the immediately preceding sequence of k states in a kth order Markov model. The probability of an edge provides the conditional probability of the corresponding event. Clearly, the order of the model encodes the memory that the model retains for the generation process. First-order models contain the least amount of retained memory. From the perspective of the (equivalent) rule-generation model, first-order models simply encode

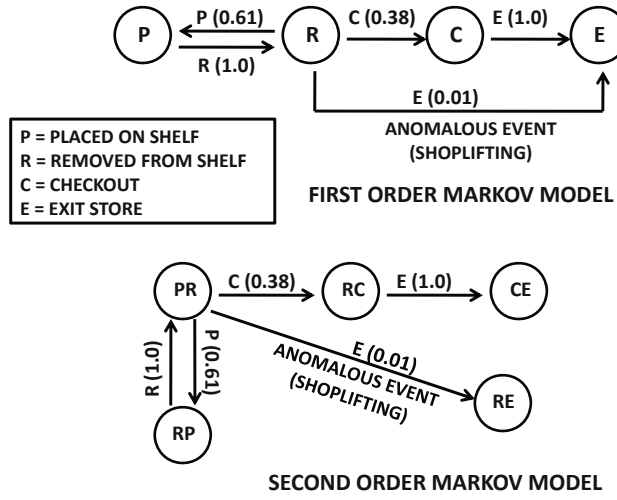

Figure 10.1: Markov model for the RFID-based shoplifting anomaly

the case, where the antecedent of the rule $a_{n-1} \Rightarrow a_n$ is of unit length. The kth order models correspond to rules, whose antecedents are of length k. For a given test sequence, windows of size $(k+1)$ are extracted from the sequence. The first k symbols in it are used to determine the relevant state in the model and determine the probability of a particular event at the $(k+1)$th position in the test sequence. Note that this corresponds to the firing of a rule in the equivalent rule-based models. In the event that the transition impled by the test subsequence is not defined, its probability is estimated to be 0. A simple first-order model is proposed in [588] and a kth order model is proposed in [396].

To understand how Markov models work, the previous example of tracking items with RFID tags will be revisited. An example of the different states of an item along with transitions is illustrated in Figure 10.1. Both a first-order and a second-order model have been illustrated in the figure. The edge transition probabilities are represented along with the edges, and are typically estimated from the training data. The transitions corresponding to the shoplifting anomaly are marked in both models, and they have a low probability of occurrence. This probability can also be viewed as an outlier score. This is a particularly simple example, in which a memory of one event is sufficient to completely represent the state of an item. In this case, identical results may be obtained from the first-order and second-order models in terms of the transition probabilities. This is not the case in general. For example, consider the case of a Web log in which the Markov models correspond to sequences of Web pages visited by users. In such a case, the probability distribution of the next Web page visited depends not just on the last page visited, but also on the other preceding visits by the user [169].

An observation from Figure 10.1 is that the number of states in the second-order model is larger than that in the first-order model. This is not a coincidence. In general, as many as $|\Sigma|^k$ states may exist in an order-k model. Of course, many of these subsequences may either not occur in the training data, or they may simply be invalid possibilities in a particular domain-specific context. For example, a PP state would be invalid in the example of Figure 10.1, since the same item cannot be sequentially placed twice on the shelf without removing it at least once. Higher-order models represent complex systems more accurately

if sufficient data is available. With limited data, higher-order models result in overfitting, which degrades accuracy. Furthermore, higher-order models tend to be inefficient. It is noteworthy that the transition probabilities need to be estimated from the training data, and each transition now represents a conditional on a sequence of length $k > 1$. Such sequences may be few when the training data is limited, as a result of which the *estimation* process may be very inaccurate. This type of over-fitting significantly impacts the overall accuracy of the model in a negative way. Furthermore, the larger number of states will also make the estimation process much slower. Therefore, a number of relaxations and variations of order-k models can be constructed for greater robustness. These are analogous to the variations of rule-based methods.

- **Variable-order models:** In these methods, a state in the model might correspond to different orders, depending on its frequency in the data. Higher-order states with very low frequency can be pruned from the model, and replaced with lower-order generalizations. A method to achieve this goal with the use of *Probabilistic Suffix Trees (PST)* is presented in [524].

- **"Don't care" subsequences:** Each state of the Markov model represents a subsequence of length k. Allowing a "don't care" as a valid symbol in the subsequence significantly generalizes the state. This reduces the number of states, and also increases the number of training subsequences matching a state. This increases the robustness and efficiency of the approach. Such models are referred to as *Sparse Markov Transducers (SMT)* [189].

The aforementioned variations are computationally challenging, since the number of states is likely to be larger in a variable order model. Therefore, efficient data structures are required for representation and processing.

10.2.3 Efficiency Issues: Probabilistic Suffix Trees

It is evident from the discussion in the previous sections that the Markovian and rule-based models are equivalent, with the latter being a simpler and easy-to-understand heuristic approximation of the former. Nevertheless, in both cases, the challenge is that the number of possible antecedents of length k can be as large as $|\Sigma|^k$. This can make the methods rather slow, when a lookup for a test subsequence $a_{i-k} \ldots a_{i-1}$ is required in order to determine the probability of $P(a_i|a_{i-k} \ldots a_{i-1})$. If these probability values are not organized properly, it can significantly slow down the approach, when retrieving the probability values for a particular subsequence. In many cases, the conditional probability needs to be estimated for each position in the sequence, which can be extremely slow.

Suffix trees [239] are a classical data structure, which store all subsequences in a given database. Probabilistic suffix trees (PST) represent a generalization of this structure, which also stores the conditional probabilities of generation of the next symbol for a given sequence database. For the case of order-k Markov models, a suffix tree of depth at most k will store all the required conditional probability values for the kth order Markovian models, including the conditionals for all lower-order Markov models. Therefore, such a structure encodes all the required information for variable-order Markov models as well. Of course, a key challenge is that the number of nodes in such a suffix tree can as large as $\sum_{i=0}^{k} |\Sigma|^k$. This issue is usually addressed with selective pruning.

A probabilistic suffix tree is a hierarchical data structure representing the different suffixes of a sequence. A node in the tree with depth k represents a suffix of length k, and

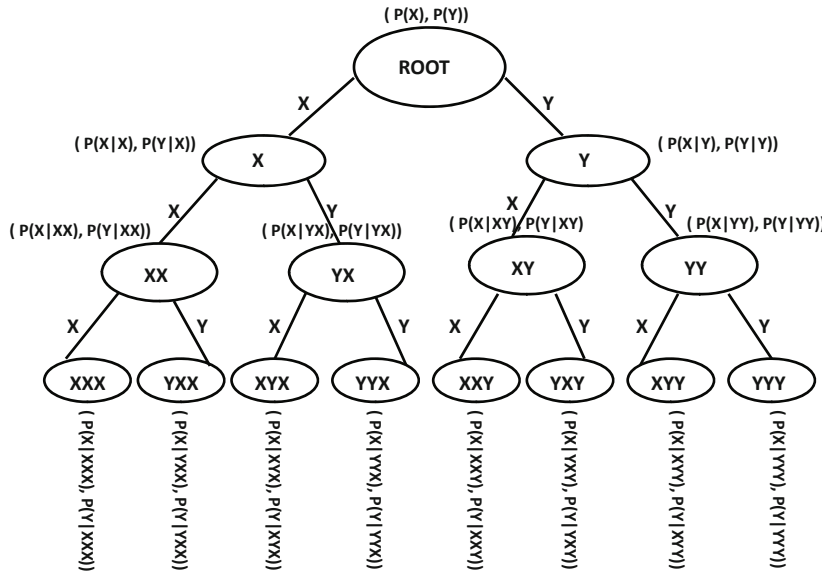

Figure 10.2: Probabilistic Suffix Tree

is therefore labeled with a sequence of length k. The parent of a node $a_{i-k} \ldots a_i$ corresponds to the sequence $a_{i-k+1} \ldots a_i$, which is obtained by removing the *first* symbol from the sequence. Each edge is labeled with the symbol that needs to be removed in order to derive the sequence at the parent node. Thus, a path in the tree corresponds to *suffixes* of the same sequence. Each node also maintains a vector of $|\Sigma|$ probabilities, whose elements correspond to the conditional probabilities of the generation of each symbol from $\Sigma = \{\sigma_1 \ldots \sigma_{|\Sigma|}\}$ after the sequence relevant to that node. Therefore, for a node corresponding to the sequence $a_{i-k} \ldots a_i$, and for each $j \in \{1 \ldots |\Sigma|\}$, the values of $P(\sigma_j | a_{i-k} \ldots a_i)$ are maintained. As discussed earlier, this corresponds to the conditional probability that σ_j appears immediately *after* $a_{i-k} \ldots a_i$, after the latter sequence has already been observed. This also provides the generative probability for quantification of position outlier scores. An example of a suffix tree with symbol set $\Sigma = \{X, Y\}$ is illustrated in Figure 10.2. The two possible symbol-generation probabilities at each node (for either X or Y) are shown. It is also evident that a probabilistic suffix tree of depth k encodes *all* the transition probabilities for Markovian models up to order k. Therefore, such an approach can also be used for variable-order Markovian models.

The probabilistic suffix true is pruned to improve its compactness. For example, suffixes that correspond to very low counts in the original data can be pruned. Furthermore, nodes with low generative probabilities can be pruned. The generative probability of a node, corresponding to sequence $a_1 \ldots a_n$ is approximated as follows:

$$P(a_1 \ldots a_n) = P(a_1) \cdot P(a_2|a_1) \ldots P(a_n|a_1 \ldots a_{n-1}) \tag{10.1}$$

For Markovian models of order $k < n$, the value of $P(a_r|a_1 \ldots a_{r-1})$ in the aforementioned equation is approximated by $P(a_r|a_{r-k} \ldots a_{r-1})$ for any value of k less than r. In order to create Markovian models of order k or less, it is not necessary to keep portions of the tree with depth greater than k. These observations have been used to create an efficiently pruned suffix tree for outlier analysis in [524].

Consider the sequence $a_1 \ldots a_i \ldots a_n$, in which it is desired to test whether position a_i is a position outlier. Then, it is desired to determine $P(a_i|a_1 \ldots a_{i-1})$. It is possible that the suffix $a_1 \ldots a_{i-1}$ may not be present in the suffix tree, because it may have been pruned from consideration. In such cases, the *short memory* property is used to determine the longest suffix (denoted by $a_j \ldots a_{i-1}$) present in the suffix tree. The corresponding probability is estimated by $P(a_i|a_j \ldots a_{i-1})$. Thus, the probabilistic suffix tree provides an efficient way to store and retrieve the relevant probabilities. The length of the longest path that exists in the suffix tree containing a non-zero probability estimate of $P(a_i|a_j \ldots a_{i-1})$ also provides an idea of the level of rarity of this particular sequence of events. Positions that contain only short paths preceding them in the suffix tree are more likely to be outliers. Thus, outlier scores may be defined from the suffix tree in multiple ways:

- If only short path lengths exist in the (pruned) suffix tree corresponding to a particular position in the sequence, then such a position is more likely to be an outlier.
- For the paths of lengths $1 \ldots r$, which do exist in the suffix tree for position a_i, a combination score may be used based on the models of different orders. In some cases, only lower-order scores are combined. In general, the use of lower-order scores is preferable to ensure robustness.

One property of the suffix tree is that it can also be used to approximate the generative probability of local subsequences of the data with the use of Equation 10.1. This can be used to determine short *segments* of sequences, which are anomalies, by determining the segments with low generative probabilities. These are referred to as *combination outliers*, which will be discussed in the next section.

Position outliers can also be used for *correcting* noise in sequences by replacing particular positions with symbols that have higher generative probabilities. Such positions and symbols can be extracted from the suffix tree, and such an approach has been discussed in [465]. These methods can also potentially be used to correct grammatical errors in text sentences, where each sentence is treated as a sequence of words, and the training data contains a large number of grammatically correct sentences.

10.3 Combination Outliers

In combination outliers, the goal is to determine unusual combinations of symbols in a given sequence with respect to other sequences. There are several versions of the problem corresponding to the size of the compared sequences, and whether the base data contains multiple sequences or whether it consists of a single long sequence. Interestingly, all these different versions of the problem can be reduced to the same primitive family of sequence anomaly-detection problems, except that different *wrappers* need to be used for different scenarios. Many of these formulations are treated very differently in the literature, and are often not connected with one another, because of the diverse problem domains in which they arise. A number of selective reductions between some of these formulations are discussed in [126]. The treatment in this book is more unified because the *specific model used* (e.g., frequency-based method or distance-based method or hidden Markov model) has been cleanly separated from the methodology.

One challenge is that sequence-based outlier detection arises in rather diverse scenarios. Sequence outlier analysis can be formulated very differently, because the data may be present in many different formats, such as a single sequence, multiple sequences, long or

short sequences. However, these scenarios often turn out to be similar with the appropriate data preprocessing. The most common formulation scenarios are discussed below:

Unsupervised versus semi-supervised: In the unsupervised version of the problem, all anomalous sequences are determined within a database of sequences. In the semi-supervised version of the problem, an anomaly score is determined for a *test* sequence, with respect to a *training* database of *normal* sequences. At the formulation level, there is no difference between these problems except for the fact that the semi-supervised case makes a distinction between training and test data. The implicit assumption is that the training data does not contain anomalies and therefore a one-class model should be constructed on it. For discussion purposes, the semi-supervised version will be used, since the model of comparing a test sequence with a set of training sequences is more fundamental and can be generalized to the other version as a repetitively used primitive subroutine.

Long versus short test sequence: When the test sequence is relatively short, the sequence can be directly compared to windows of the training sequences in order to determine its *rarity*. In cases in which both the test sequences and training sequences are short, and of comparable length, multidimensional methods for anomaly detection such as the k-nearest neighbor method can be adapted to this case. These are referred to as *point-based* techniques.

When the test and training sequences are long, it is not meaningful to compare whole sequences. In such cases, the curse of dimensionality prevents an informative computation of distances over long sequences. Similar to subspace methods, windows of the test sequence are extracted as small subsequences. Such small subsequences will henceforth be referred to as *comparison units*. Then, the *relative behavior* of the comparison units is compared in both the training data and test sequence. If this comparison unit (or a small variation of it) is much less prevalent in the training sequence, as compared to the test sequence, it is considered an anomaly. The final anomaly score for a given test sequence can be derived as a *combination score* from all the subsequences extracted from it.

Single long training sequence versus many training sequences: The sequence database may contain a single long training sequence, or it may contain many training sequences. At a logical level, there is little difference between these cases because all distance-based, frequency-based, and Markov models can be derived equivalently either from a single long training sequence or multiple training sequences. This is because the models describe the behavior of only a very small window of the data corresponding to the *comparison unit*. For example, when a comparison unit is compared to training sequences in a k-nearest neighbor approach, the anomaly score is the kth nearest neighbor distance among all (similar size) subsequences in the training data, whether they are derived from the same series or multiple series. For greater generality, it will henceforth be assumed that the training data contains multiple sequences.

Single long sequence that is undifferentiated between training and test data: It is also possible to create difficult combinations of the aforementioned cases. A particularly common case is that of Web logs, in which a single long sequence can be extracted, and it might be desirable to determine unusual *portions* of this sequence. The additional step required for this case is to extract portions of the undifferentiated sequence as test sequences. A multi-granularity approach can be used in order to extract test sequences of different lengths, possibly in geometrically increasing sizes. Then, a *second level* of smaller subsequences is extracted from the test sequences. These subsequences correspond to the

comparison units. These are used to model the *relative difference* between the derived test sequences and the training sequences in terms of the smaller window-based comparison units.

Presence or absence of domain knowledge about relevant comparison units for anomaly detection: In all the aforementioned cases, it is assumed that the comparison unit subsequences are derived as contiguous windows from the test sequences. In many applications, specific domain knowledge may be available about important comparison units. For example, in a security application designed to detect unusual login attempts, a sequence such as *Login Password Login Password Login Password* may be very relevant for detecting interesting anomalies. In such cases, the models can be easily evaluated and combined in terms of these *domain-dependent comparison units*, rather than the units extracted from the test sequences. This is a form of supervision with domain-knowledge. In general, domain knowledge can significantly improve the quality of the overall results.

All the aforementioned variations are also relevant to time-series data, although discrete sequence data seems to exhibit a wider variation among these different possibilities in real application domains.

10.3.1 A Primitive Model for Combination Outlier Detection

From the aforementioned discussion, it is clear that in the most general case, a model needs to be constructed on the basis of relative comparisons between three different types of sequences: (i) the training sequences, (ii) the test sequence, and (iii) the comparison units. One of these three different types of sequences (the comparison unit) is usually extracted from the test sequence, and may not directly be a part of the input on a stand-alone basis, unless specific domain knowledge is available. However, it will still be included in the primitive formulation, since it provides a very general way to define a primitive formulation for combination outlier detection. This formulation and its minor variants are repeatedly used in various forms of sequence anomaly detection. Some notations and definitions will be used in order to distinguish between the training database, test sequence, and the comparison units.

- The training database is denoted by $\mathcal{D}$, and contains sequences denoted by $T_1 \ldots T_N$.
- The test sequence is denoted by V.
- The comparison units are denote by $U_1 \ldots U_r$. Typically, each U_i is derived from small contiguous windows of V. In domain-dependent cases, $U_1 \ldots U_r$ may be provided by the user.

  In the case of small test sequences, a single comparison unit $U_1 = V$ may be used. Furthermore, in cases in which the test sequence is of similar (short) length as the training sequences, this reduces to the easier special case of point-anomaly detection.

This sets the stage for defining the primitive sequence anomaly detection problem.

Definition 10.3.1 (Primitive Sequence Anomaly (Relative)) *Given a training database of discrete sequences $\mathcal{D}$, a test sequence V, and a comparison unit U_i, determine the anomaly score of U_i in terms of relative rarity in $\mathcal{D}$ with respect to V with the use of model $\mathcal{M}$.*

The model $\mathcal{M}$ may be a distance-based, frequency-based or hidden Markov model. Each of these will be discussed in subsequent sections. The comparison units may either be specified

by the user, or they may be extracted from small sliding windows of the test sequence. In cases, where the comparison unit U_i is extracted directly from V, the *absolute* rarity (rather than *relative* rarity) of U_i is computed with respect to the training sequences, since the comparison unit is already known to belong to the test sequence. For example, for a distance-based model, the comparison unit will have zero distance from at least one window in the test sequence V, and absolute comparisons to the training sequence will provide the same results as a relative comparison between the training and test data. In such cases, the absolute distances to windows of the training sequence are used as the anomaly score. Even in the case of frequency-based and Markov models, relative comparisons are possible, although they are not generally performed in domain-independent scenarios. Thus, the issue of *relative* surprise is more relevant in scenarios where the comparison units are provided as part of the input. For the case where the comparison units are extracted from the sequences, the primitive sequence anomaly detection problem may be restated as follows:

Definition 10.3.2 (Primitive Sequence Anomaly (Absolute)) *Given a training database of discrete sequences $\mathcal{D}$, a test sequence V, and a comparison unit U_i extracted from V, determine the absolute anomaly score of U_i with respect to $\mathcal{D}$ with the use of model $\mathcal{M}$.*

Multiple comparison units are extracted from V using a sliding window method, where each U_i corresponds to a contiguous window of V.

10.3.1.1 Model-Specific Combination Issues

For each of the two formulations introduced in the previous section, the anomaly score of the model is computed with respect to a comparison unit. However, multiple comparison units are extracted from a test sequence. As a post-processing step, it is needed to compute the overall anomaly score of the test sequence by combining the results from different comparison units. The precise method for combining the different scores will be discussed along with the model in the following sections. These methods are similar in spirit and principle, although differences may exist at the detailed level because of the differences in how the anomaly score for each comparison unit is quantified by the different models.

10.3.1.2 Easier Special Cases

An important special case is one in which the test sequence and training sequences are short, and of comparable size. In such cases, extraction of comparison units becomes irrelevant. Furthermore, it is much easier to compute similarity measures between pairs of *short* sequences in a more robust way, because the curse of dimensionality becomes less relevant. Such cases can be reduced to point-anomaly detection. Some of the models discussed below can also be applied to these simplified cases. Since this is a common special case, a separate formulation is defined for this problem.

Definition 10.3.3 (Whole Sequence Anomaly (Semi-supervised)) *Given a database $\mathcal{D}$ of short training sequences $\{T_1 \ldots T_N\}$, and a test sequence V of comparable size, determine the anomaly score of V with respect to $\mathcal{D}$ with the use of model $\mathcal{M}$.*

The unsupervised variation of the above problem is very similar, and can be stated as follows:

Definition 10.3.4 (Whole Sequence Anomaly (Unsupervised)) *Given a database $\mathcal{D}$ of short training sequences $\{T_1 \ldots T_N\}$, determine all anomalous sequences with the use of model $\mathcal{M}$.*

As discussed earlier in this chapter, the unsupervised and semi-supervised versions are almost identical, and therefore do not need separate discussion. Therefore, both of these cases will be discussed in a unified way.

Note that both formulations above are missing the comparison unit. Short sequences do not require the extraction of smaller comparison units for robust distance computations in anomaly detection. In the following sections, the problem of whole sequence anomaly detection will also be discussed. Although certain models such as distance-based models and hidden Markov models can be used effectively in this case, frequency-based models cannot be used in such scenarios. This is because frequency-based models are dependent on the repetitive frequencies of short comparison units within long sequences. This is not possible in scenarios where all sequences are short.

10.3.1.3 Relationship between Position and Combination Outliers

Although the most popular and robust methods for detecting combination outliers use window-based comparison units, it is sometimes possible to determine the outlier score for smaller test sequences by repeated use of combination outliers. Specifically, the anomaly score of a sequence $a_1 \ldots a_n$ can be estimated as the product of the following conditionals:

$$P(a_1 \ldots a_n) = P(a_1) \cdot P(a_2|a_1) \ldots P(a_n|a_1 \ldots a_{n-1}) \tag{10.2}$$

Note that each of these probability values is available in a probabilistic suffix tree of order (depth) n. For cases in which only a depth of k is maintained, the short-memory property of sequences can be used to approximate $P(a_i|a_1 \ldots a_{i-1})$ with $P(a_i|a_{i-k} \ldots a_{i-1})$. Such an approach has been used in [524] for efficient determination of sequence outliers.

10.3.2 Distance-Based Models

In distance-based models, the absolute distance of the comparison unit is computed to equivalent windows of the training sequence. The distance of the k-th nearest neighbor window in the training sequence is used to determine the anomaly score. In the context of sequence data, some of the proximity-functions are *similarity* functions, whereas others are distance functions. An example of the former is the use of the *longest common subsequence (LCSS)*, whereas an example of the latter is the *edit distance*. In the following, we provide a brief discussion of these proximity function, although more details may be found in [33, 23]:

- **Simple matching coefficient:** This is the simplest possible function and determines the number of matching positions between two sequences of equal length. This is also equivalent to the Hamming distance between a pair of sequences.

- **Longest common subsequence (and its normalized variants):** Let T_1 and T_2 be two sequences. Any sequence obtained by dropping some elements of a sequence is referred to as its *subsequence*. Therefore, the longest common subsequence of T_1 and T_2 is the longest sequence that is a subsequence of both T_1 and T_2. Let the length of the longest common subsequence be denoted by $L(T_1, T_2)$. Then, the value $NL(T_1, T_2)$ of the normalized longest common subsequence is computed by normalizing $L(T_1, T_2)$ with the underlying sequence lengths in a similar way to the cosine computation between unordered sets:

$$NL(T_1, T_2) = \frac{L(T_1, T_2)}{\sqrt{|T_1|} \cdot \sqrt{|T_2|}} \tag{10.3}$$

The advantage of this approach is that it can match two sequences of unequal lengths. However, as with many sequence-based proximity function, a dynamic programming approach is required to compute the value of $L(T_1, T_2)$, which is rather slow. Details of this dynamic programming algorithm are provided in [33].

- **Edit distance:** The edit distance is one of the most common similarity functions used for sequence matching [239]. This function measures the distance between two sequences by the minimum number of edits required to transform one sequence to the other. The computation of the edit distance can be computationally very expensive.

- **Compression-based dissimilarity:** This measure is based on principles from information theory. Let W be a window of the training data, and $W \oplus U_i$ be the string representing the concatenation of W and U_i. Let $DL(S) < |S|$ be the description length of any string S after applying a standard compression algorithm to it. Then, the compression-based dissimilarity $CD(W, U_i)$ is defined [312] as follows:

$$CD(W, U_i) = \frac{DL(W \oplus U_i)}{DL(W) + DL(U_i)} \tag{10.4}$$

  This measure always lies in the range $(0, 1)$ and lower values indicate greater similarity. The intuition behind this approach is that when the two sequences are very similar, the description length of the combined sequence will be much smaller than that of sum of the description lengths. On the other hand, when the sequences are very different, the description length of the combined string will be almost the same as the sum of the description lengths.

- **Other methods:** A variety of other methods may be used to compute similarity. These methods vary in terms of how two sequences may be aligned or the importance given to different lengths of the alignment. Some examples of such methods include counting mismatches among lookahead pairs [198], and a length-sensitive recursive computation of subsequence similarity [338, 339]. The latter method is of particular interest and is discussed below.

  An important observation in the context of many sequence applications such as intrusion detection is that *contiguous mismatches are more important than non-contiguous mismatches.* This is because anomalous events usually cause *bursts* of anomalous symbols, which can be distinguished from the occasional noise. Let U_i be a comparison unit, and W be a window of the training sequence of the same length as U_i. For each position l in U_i, the length p of the longest contiguous set of positions to the left of position l is determined, so that the position indices $\{l-p+1, \ldots l\}$ in both sequences match exactly. This length is aggregated over the different values of l from 1 to $|U_i|$. The intuition here is that long contiguous matches are rewarded more than distributed and scattered matches.

In order to compute the anomaly score of a comparison unit U_i with respect to the training sequences in $T_1 \ldots T_N$, the first step is to extract equivalent windows from $T_1 \ldots T_N$ as the size of the comparison unit. The k-th nearest neighbor distance is used as the anomaly score *for that comparison unit.* It now remains to explain how the results for the different comparison units $U_1 \ldots U_r$ extracted from V are combined to provide a unified anomaly score for the test sequence V.

10.3.2.1 Combining Anomaly Scores from Comparison Units

The anomaly scores from different comparison units extracted from a test sequence can be combined together in order to create a global anomaly score for the test sequence V. For ease in discussion, the default assumption is that higher scores correspond to greater outlierness. Some combination methods are as follows:

- **Number of anomalous units:** For each comparison unit, its anomaly score is compared to a specific threshold. The number of anomalous units among $U_1 \ldots U_r$ is reported as the final anomaly score. Note that this method is not specific to the distance-based model for anomaly detection, and can be used in the context of any of the models. For example, such an approach has been used in the context of intrusion detection in [197, 272]. However, it does lose some information because it does not use the values of the scores of the various units.

- **Aggregate anomaly score:** This aggregates the anomaly score over all comparison units. Although such an approach accounts for varying levels of anomaly scores, it can be impacted from the noisy scores of windows that are not anomalous.

- **Selective aggregate anomaly score:** This approach combines the virtues of the two aforementioned methods. In the first step, windows with anomaly score greater than a particular threshold are identified. The anomaly scores over these windows are aggregated to yield a final anomaly score.

- **Clustered anomaly scores:** It is generally recognized that *contiguous anomalous windows are generally more significant in application-specific scenarios than anomalous windows that are arbitrarily distributed over the sequence.* This is because in many motivating applications such as intrusion detection, anomalies are rare, but they often occur over locally clustered windows when they do occur.

  - **Locality frame count:** A method known as *Locality Frame Count (LFC)* proposed in [197] examines each possible set of η contiguous comparison units, as a *super-unit*. If number of anomalous units within this super-unit is larger than a given threshold, then the entire super-unit is deemed anomalous. The total number of anomalous super-units is reported as the outlier score. Note that this method is not specific to distance-based methods. In fact, a similar method has been used for HMM-based models, which will be discussed later in this chapter [211].
  - **Leaky bucket:** A method based on a similar principle has been proposed in [217]. In this method, the comparison units are scanned in temporal order, and a running count is maintained of the difference between the number of anomalous comparison units and the number of normal units. However, the running count is never allowed to fall below 0. The highest value of the running count over the entire sequence is reported as the outlier score. Such an approach also favors highly clustered anomalous comparison units. For example, interleaved normal and abnormal units will have a (combined) anomaly score no larger than 1. Note that the use of the leaky bucket is not specific to the use of distance-based methods, but can be used with virtually any model.

Different extreme-value analysis techniques can be used in order to identify thresholds at which scores should be considered anomalous. Such methods are discussed in Chapter 2.

10.3.2.2 Some Observations on Distance-Based Methods

Distance-based methods are more suitable for cases in which the comparison units are directly extracted from the test sequence. On the other hand, in many scenarios, comparison units may be provided by a domain expert and may correspond to significant semantic events (e.g., known intrusion patterns, shop-lifting patterns, hacking attack patterns, and so on). These correspond to the formulation in Definition 10.3.1. In such cases, other classes of models such as frequency-based or model-based methods (HMM) should be used. The latter classes of methods are suitable *both* for relative comparisons based on domain-dependent subsequences, and for absolute comparisons to the training data based on comparison units extracted directly from test sequences.

10.3.2.3 Easier Special Case: Short Sequences

The use of extracted comparison units or domain-specific comparison units is necessitated by the curse of dimensionality in distance computations over longer sequences. However, when the data contains a set of relatively short sequences of comparable size, straightforward generalizations of multidimensional methods can be used. This scenario corresponds to Definitions 10.3.3 and 10.3.4. Any standard k-nearest neighbor technique or clustering method discussed in Chapter 4 may be used. The major distinction is that the distances need to be computed directly on the sequences. For that purpose, any of the aforementioned distance (or similarity) functions in this chapter can be used. Specific examples of algorithms of the two types are as follows:

- A k-nearest neighbor approach, which uses the inverse of the similarity value as the anomaly score, is proposed in [127]. Such an approach has been shown to be quite effective in spite of its simplicity.
- The work in [103, 104] uses a k-medoid based clustering approach in order of determine relevant outliers. A method called *CLUSEQ* for clustering with the use of probabilistic suffix trees is proposed in [581]. The core idea is that a probabilistic suffix tree provides an efficient representation of a cluster, and this can be used for efficient similarity computations in the clustering process. The probabilistic suffix tree is pruned of infrequent nodes in order to ensure a more compact tree for efficiency. The approach is designed for clustering as the primary goal, but is also able to determine outliers as a side product.

10.3.3 Frequency-Based Models

Frequency-based models are typically used with domain-specific comparison units specified by the user. In this case, the relative frequency of the comparison unit needs to be measured in the training sequences and the test sequences, and the level of surprise is correspondingly determined. Such methods are primarily designed for user-specified comparison units; however, they can easily be extended to the setting of extracted comparison units.

10.3.3.1 Frequency-Based Model with User-Specified Comparison Unit

When the comparison units are specified by the user [310], a natural way of testing the anomaly score is to test the frequency of the comparison unit U_j in the training and test patterns. For example, when a sequence contains a hacking attempt, such as a sequence of *Login* and *Password* events, this sequence will have much higher frequency in the test

sequence, as compared to that in the training sequences. The specification of such relevant comparison units by a user provides very useful domain knowledge to an outlier analysis application.

Let $f(T, U_j)$ represent the number of times that the comparison unit U_j occurs in the sequence T. Since the frequency $f(T, U_j)$ depends on the length of T, the normalized frequency $\hat{f}(T, U_j)$ may be obtained by dividing the frequency by the length of the sequence:

$$\hat{f}(T, U_j) = \frac{f(T, U_j)}{|T|} \tag{10.5}$$

Then, the anomaly score of the training sequence T_i with respect to the test sequence V is defined by subtracting the relative frequency of the training sequence from the test sequence. Therefore, the anomaly score $A(T_i, V, U_j)$ is defined as follows:

$$A(T_i, V, U_j) = \hat{f}(V, U_j) - \hat{f}(T_i, U_j) \tag{10.6}$$

The absolute value of the average of these scores is computed over all the sequences in the database $\mathcal{D} = T_1 \ldots T_N$. This represents the final anomaly score.

A useful output of this approach is the specific subset of comparison units specified by the user that are the most anomalous. This provides intensional knowledge and feedback to the analyst about *why* a particular test sequence should be considered anomalous. A method called *TARZAN* [310] uses suffix tree representations to efficiently determine all the anomalous subsequences in a comparative sense between a test sequence and a training sequence.

One issue with this model is that since the comparison units are specified by the user, they may not always be of the compact size required for effective frequency computations (unlike the case in which comparison units are extracted from test sequences with compactness in mind). When the comparison units themselves are long, the frequencies of these units in the training sequences may be small or zero. As a result, the anomaly scores may no longer remain significant. A number of methods have been proposed in the literature in order to address these issues, most of which have to do with examining the behavior of smaller portions of the comparison units. In particular, a method proposed in [310] uses subsequences of the comparison units, and computes an aggregate quantity as a function of the frequencies of the underlying subsequences. A different method in [243] allows relaxation of the definition of "subsequence" in counting frequencies by allowing permutations of the comparison unit. The assumption is that the exact ordering of the symbols within a short window is not as important in certain application-specific scenarios. In fact, in such cases, relaxations can add to the robustness of the similarity computations.

In order to improve the *efficiency* of frequency computation, the work in [244] suggests that the training sequences should be decomposed into smaller windows for subsequence computations. The number of windows in which the comparison unit occurs is used as a proxy for the frequency.

10.3.3.2 Frequency-Based Model with Extracted Comparison Units

The frequency-based model is mostly designed for the case of user-specified comparison units. However, it is also possible to generalize the model to the case where extracted comparison units are used. In this model, the comparison unit is extracted directly from the test sequence. For each comparison unit, the same model may be used, as in the case of user-specified comparison units. Although such methods are feasible, they have generally

not been used very frequently in the literature. Distance-based methods are more popular, when comparison units are extracted directly from the test sequences.

10.3.3.3 Combining Anomaly Scores from Comparison Units

Although frequency-based methods are generally used for surprise detection with user-specific comparison units, the results from multiple comparison units can also be combined into a single anomaly score. When many comparison units are specified by a user, this can be used to create a unified comparison score. The methodology for achieving this goal is the same as discussed for distance-based methods in section 10.3.2.1. This approach is also applicable when the comparison units are extracted directly from the test sequences. The former results in a domain-dependent anomaly score, whereas the latter results in an unsupervised anomaly score.

10.3.4 Hidden Markov Models

Hidden Markov models (HMMs) are probabilistic models that generate sequences through a sequence of transitions between states in a Markov chain. So how are hidden Markov models different from the Markovian techniques introduced earlier in this chapter? Each state in the Markovian techniques introduced earlier in this chapter is well defined by a particular position in the sequence. The state is uniquely defined by the previous k positions in the sequence from that position. This state is also directly visible to the user in terms of the precise order of transitions for a particular training or test sequence. Thus, the generative behavior of the Markovian model is always *fully visible* for a given sequence.

In a hidden Markov model, the states of the system are *hidden* by definition, which implies that they are not directly visible to the user. Only a sequence of (typically) discrete observations is visible to the user, which are generated by symbol emissions from the states after each transition. These observations correspond to the application-specific sequence data. In many cases, the states may be defined (during the modeling process) on the basis of an *understanding* of how the underlying system behaves, although the precise sequence of transitions may not be known to the analyst. This is why such models are referred to as "hidden."

Each state is associated with a *set of emission probabilities* over the symbol Σ. In other words, a visit to the state j leads to an emission of one of the symbols $\sigma_i \in \Sigma$ with probability $\theta^j(\sigma_i)$. Correspondingly, a sequence of transitions in a hidden Markov model corresponds to an *observed data sequence*. Hidden Markov models may be considered special cases of the mixture models discussed in Chapter 2. The main caveat is that the different components of the mixture are not independent of one another, but are related through sequential transitions. Thus, each state is analogous to a component in the multidimensional mixture model of Chapter 2. Each symbol generated by this model is analogous to a data point generated by the multidimensional mixture model. Furthermore, the successive generation of data items (sequence symbols) is also dependent on the previous history. This is a natural consequence of the fact that the successive states emitting the data items are not independent of one another, because they continually transition to one another. Unlike traditional mixture models, hidden Markov models are specifically designed to capture the temporal correlations in sequential data.

In order to better explain hidden Markov models, an illustrative example will be used. Consider the scenario where a set of students register for a course, and generate a sequence corresponding to the grades received in each of their weekly assignments. This grade is drawn

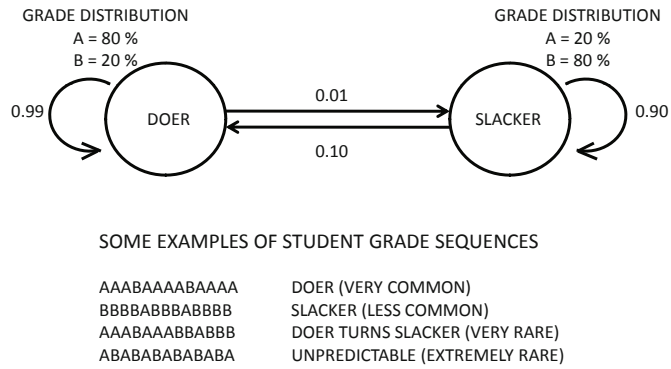

Figure 10.3: Generating grade sequences from a hidden Markov model

from the symbol set $\Sigma = \{A, B\}$. *The model created by the analyst is that* the class contains students who, at any given time, are either *doers* or *slackers* with different grade-generation probabilities. A student in a *doer* state may sometimes transition to a *slacker* state and vice versa. These represent the two states in the system. Weekly home assignments, which are given to the students, are graded with one of the symbols from Σ. This results in a *sequence* of grades for each student, and it represents the only *observable* output for the analyst. The (hidden) state of a student represents only a *model* created by the analyst to *explain* the grade sequences and is, therefore, not observable in of itself. It is important to understand that if this model is a poor reflection of the true generative process, then the accuracy of the learner will be poor.

Assume that a student in a *doer* state is likely to get an A grade in a home assignment with an 80% probability and a B with a 20% probability. For *slackers*, these probability values are reversed. Although we have explicitly specified these probabilities here for illustrative purposes, they are usually not known a priori and need to be *learned* or *estimated* from the observed grade sequences. The precise state of a student is not known to the analyst at any given time. These grade sequences are, in fact, the only *observable* outputs for the analyst. Therefore, from the perspective of the analyst, this is a *hidden* Markov model, which generates the sequences of grades from an *unknown* sequence of state transitions. This unknown sequence can only be probabilistically *estimated* for a particular observed sequence.

The two-state hidden Markov model for the aforementioned example is illustrated in Figure 10.3. This model contains two states corresponding to *doer* and *slacker*, which represent the state of a student in a particular week. It is possible for a student to transition from one state to another, although the likelihood of this is low. It is assumed that the set of initial state probabilities governs the a priori distribution of *doers* and *slackers*. These probabilities represent the a priori understanding about the students when they join the course. Some examples of *possible sequences* and their corresponding rarity levels are illustrated in Figure 10.3. For example, the sequence `AAABAAAABAAAA` is most likely generated by a student who is consistently in a *doer* state, and the sequence `BBBBABBBABBBB` is most likely generated by a student who is consistently in *slacker* state. The second sequence is typically rarer than the first because the population predominantly contains[1] *doers*. The sequence

[1]The assumption is that the initial set of state probabilities are approximately consistent with the steady state behavior of the model for the particular set of transition probabilities shown in Figure 10.3.

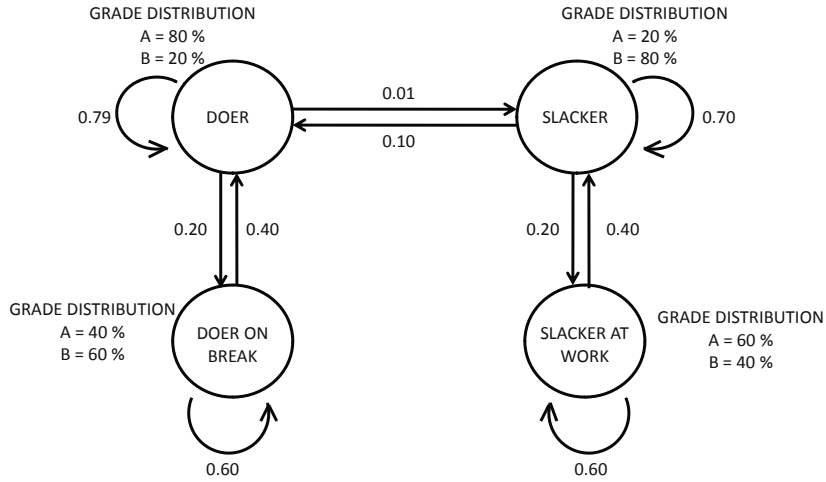

Figure 10.4: Extending the model in Figure 10.3 with two more states provides greater expressive power for modeling sequences

AAABAAABBABBB corresponds to a *doer* who eventually transitions into a *slacker*. This case is even rarer, because it requires a transition from the *doer* state to a *slacker* state, which has very low probability. The sequence ABABABABABABA is *extremely anomalous*, because it does not represent temporally consistent *doer* or *slacker* behavior that is implied by the model. Correspondingly, such a sequence has very low probability of fitting the model.

A larger number of states in the Markov model can be used to encode more complex scenarios. It is possible to encode domain knowledge to improve the basic design of the state layouts in the model. For example, *doer*s may sometimes slack off for short periods and then return to their usual state. Similarly, *slacker*s may sometimes become temporarily inspired to be *doer*s, but may eventually return to what they are best at. Such episodes will result in local portions of the sequence that are distinctive from the remaining sequence. These scenarios can be captured with the 4-state Markov Model illustrated in Figure 10.4. A larger number of states enable the modeling of more complex scenarios. In general, more training data is required to learn the (larger number of) parameters of such a model without overfitting. For smaller data sets, the transition probabilities and symbol-generation probabilities are not estimated accurately. As discussed in Chapter 2, this is a common problem of generative models. Therefore, a number of important issues arise about the initial design choices in a hidden Markov model. These will be discussed in the next subsection.

10.3.4.1 Design Choices in a Hidden Markov Model

In the previous example, an *understanding* of the generating process was used to design the basic architecture of the hidden Markov model. Such an understanding may or may not be available in a given application. A good initial design (i.e., state layout) of the Markov model is crucial, because a poor design could result in the model either overfitting the data or not fitting the training sequences at all. Two primary design choices are key to this process:

- In its most general form, a hidden Markov model with n states will have n^2 possible

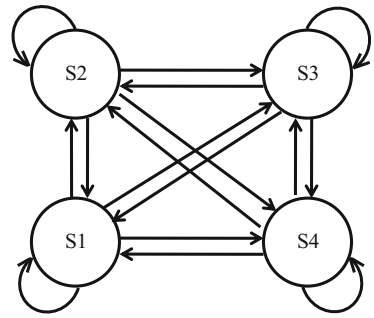

Figure 10.5: A black-box hidden Markov model with four states and no encoded domain knowledge: A larger number of transition probabilities need to be learned.

transitions between different states (including self-transitions), and $n \cdot |\Sigma|$ symbol generation probabilities, where one distribution is associated with each state. This is a black-box model, in which no prior knowledge is encoded about the generating process. In such cases, the initial Markov model is a clique with n states. In practice, the states may be designed with an intuitive understanding about the underlying generating process, and not all pairs of states may have transitions among them. This encodes significant *domain knowledge* about the sequence generation process. For example, in the case of Figure 10.4, the states are intuitively associated with characteristics of the generating process (students). Furthermore, transitions do not occur between all pairs of states. These states represent a priori domain knowledge of the analyst about the generative process for the sequences.

The equivalent four-state black-box Markov Model is illustrated in Figure 10.5. Note that the states are not labeled in this case, because they do not reflect an analyst's understanding of the underlying generating process. Clearly, different trade-offs exist between these choices. By selecting a particular topology of the model based on domain understanding, the number of parameters is greatly reduced. For example, fewer transition probabilities need to be learned in Figure 10.4 as compared to Figure 10.5. In the former case, many transition probabilities have already been set to zero before the training process, and this reflects the domain-specific understanding. This reduces the likelihood of overfitting in the case of Figure 10.4. On the other hand, if the specific topology of the domain-specific architecture of Figure 10.4 does not truly reflect the student behavior, then the resulting model will also poorly reflect the training data. In this sense, the model of Figure 10.5 has the advantage that it can be used as black-box for the sequences from any domain.

- The number of states represents the level of complexity in the Markov model. This is analogous to the number of components in the mixture model of Chapter 2. A larger number of states can encode greater complexity about variations in sequence patterns, but may also overfit a training data set, when it is small. Therefore, when the black-box approach is used, it needs to be decided up front, how many states should be used in the model. When the data sets available are small, it is advisable to use a smaller number of states.

The first of the aforementioned design choices is important, because hidden Markov models are particularly effective when they are used to encode domain-specific understanding. The

ability to encode a set of sequences with a Markov model of a particular architecture is dependent on the understanding of the analyst about the generating process. As discussed throughout the book, the ability to encode domain knowledge in an indirect way during the model construction process is crucial for effective outlier analysis.

10.3.4.2 Training and Prediction with HMMs

In this section, hidden Markov models will be defined formally, and the associated training methods will be introduced. It is assumed that a hidden Markov model contains n states denoted by $\{s_1 \ldots s_n\}$. The symbol set from which the observations are generated is denoted by $\Sigma = \{\sigma_1 \ldots \sigma_{|\Sigma|}\}$. The symbols are generated from the model by a sequence of transitions from one state to another. Each visit to a state (including self-transitions) generates a symbol drawn from a categorical probability distribution on Σ. The symbol emission distribution is specific to each state. The probability $P(\sigma_i|s_j)$ that the symbol σ_i is generated from state s_j is denoted by $\theta^j(\sigma_i)$. The probability of a transition from state s_i to s_j is denoted by p_{ij}. The initial state probabilities are denoted by $\pi_1 \ldots \pi_n$ for the n different states. The topology of the model can be expressed as a network $G = (M, A)$, in which M is the set of states $\{s_1 \ldots s_n\}$. The set A represents the possible transitions between the states. If the architecture of the HMM is constructed with a domain-specific understanding, the set A is not a complete network, and, therefore, many transition probabilities are implicitly set to 0. *The goal of training the HMM model is to learn the initial state probabilities, transition probabilities, and the symbol emission probabilities from the training database* $\{T_1 \ldots T_N\}$.

Three methodologies are commonly leveraged in creating and using a hidden Markov model:

- **Training:** Given a set of training sequences $T_1 \ldots T_N$, estimate the model parameters, such as the initial probabilities, transition probabilities, and symbol-emission probabilities with an expectation-maximization algorithm. The Baum-Welch algorithm is used for this purpose.

- **Evaluation:** Given a test sequence V (or comparison unit U_i), determine the probability that it fits the HMM. This process is used to determine the anomaly scores. A recursive forward algorithm is used to compute this.

- **Explanation:** Given a test sequence V, determine the most likely sequence of states that generated this test sequence. This is helpful for providing an understanding of why a sequence should be considered an anomaly, when the states correspond to an intuitive understanding of the underlying system. In the example of Figure 10.3, it would be useful to know that an observed sequence is an anomaly, because of the unpredictable switching of a student between *doer* and *slacker* states. This can provide the *intensional knowledge* for understanding the state of a system. The most likely sequence of states is computed with the Viterbi algorithm.

Since the description of the training procedure relies on technical ideas developed for the evaluation method, we will deviate from the natural order of presentation and present the training algorithms at the very end. The evaluation and explanation techniques will assume that the model parameters, such as the transition probabilities, are already available from the training phase.

10.3.4.3 Evaluation: Computing the Fit Probability for Observed Sequences

One approach for determining the fit probability of test sequence $V = a_1 \dots a_m$ would be to compute all the n^m possible sequences of states (paths) in the HMM, and compute the probability of each. This probability can be computed based on the observed sequence, symbol-generation probabilities, and transition probabilities. The sum of these values can be reported as the fit probability. Obviously, such an approach is not practical because it requires the enumeration of an exponential number of possibilities.

The computation can be greatly reduced by recognizing that the fit probability of the first r symbols (and a fixed value of the rth state) can be recursively computed in terms of the corresponding fit probability of the first $(r-1)$ observable symbols (and a fixed $(r-1)$th state). Specifically, let $\alpha_r(V, s_j)$ be the probability that the first r symbols in V are generated by the model, and the final state in the sequence is s_j. Then, the recursive computation is as follows:

$$\alpha_r(V, s_j) = \sum_{i=1}^{n} \alpha_{r-1}(V, s_i) \cdot p_{ij} \cdot \theta^j(a_r) \tag{10.7}$$

This approach recursively sums up the probabilities of all the n different paths for different penultimate nodes. The aforementioned relationship is iteratively applied for $r = 1 \dots m$. The probability of the first symbol is computed as $\alpha_1(V, s_j) = \pi_j \cdot \theta^j(a_1)$ for initializing the recursion. This approach requires $O(n^2 \cdot m)$ time. Then, the overall probability is computed by summing up the values of $\alpha_m(V, s_j)$ over all possible states s_j. Therefore, the final fit $F(V)$ is computed as follows:

$$F(V) = \sum_{j=1}^{n} \alpha_m(V, s_j) \tag{10.8}$$

This algorithm is also known as the *Forward Algorithm*.

10.3.4.4 Explanation: Determining the Most Likely State Sequence for Observed Sequence

One of the goals of outlier analysis is to provide an explanation for why a data point or sequence should be considered an outlier. Since the sequence of (hidden) generating states often provides an intuitive explanation for the observed sequence, it is sometimes desirable to determine the *most likely sequence of states* for the observed states. This is achieved with the *Viterbi algorithm*.

One approach for determining the most likely state path of the test sequence $V = a_1 \dots a_m$ is to compute all the n^m possible sequences of states (paths) in the HMM, and compute the probability of each of them, based on the observed sequence, symbol-generation probabilities, and transition probabilities. The maximum of these values can be reported as the most likely path. This algorithm would have exponential complexity. Note that this is a similar problem to the fit probability except that it is needed to determine the *maximum* rather than the *sum*, over all possible paths. Correspondingly, it is also possible to use a similar recursive approach as the previous case to determine the most likely state sequence.

Any sub-path of an optimal state path must also be optimal for generating the corresponding subsequence of symbols. This property, in the context of an optimization problem of sequence selection, normally enables dynamic programming methods. The best possible state path for generating the first r symbols (with the rth state fixed to j) can be recursively computed in terms of the corresponding best paths for the first $(r-1)$ observable

symbols and different penultimate states. Specifically, let $\delta_r(V, s_j)$ be the probability of the best state sequence for generating the first r symbols in V and ending at state s_j. Then, the recursive computation is as follows:

$$\delta_r(V, s_j) = MAX_{i=1}^{n} \delta_{r-1}(V, s_i) \cdot p_{ij} \cdot \theta^j(a_r) \tag{10.9}$$

This approach recursively computes the maximum of the probabilities of all the n different paths for different penultimate nodes. The approach is iteratively applied for $r = 1 \ldots m$. The first probability is determined as $\delta_1(V, s_j) = \pi_j \cdot \theta^j(a_1)$ for initializing the recursion. This approach requires $O(n^2 \cdot m)$ time. Then, the best path is finally computed by using the maximum value of $\delta_m(V, s_j)$ over all possible states s_j. This approach is, essentially, a dynamic programming algorithm.

10.3.4.5 Training: Baum-Welch Algorithm

The problem of determining the optimal parameters in the case of HMM is very difficult, and no known algorithm is guaranteed to determine the global optimum. However, a number of options are available to determine a reasonably effective solution in most scenarios. The Baum-Welch algorithm is one such method. It is also known as the *Forward-backward* algorithm, and it is an application of the expectation-maximization approach to the generative hidden Markov model. First, a description of training with the use of a single sequence $T = a_1 \ldots a_m$ will be provided. Then, a straightforward generalization to N sequences $T_1 \ldots T_N$ will be discussed.

Let $\alpha_r(T, s_j)$ be the *forward* probability that the first r symbols in a sequence T of length m are generated by the model, and the last symbol in the sequence is s_j. Let $\beta_r(T, s_j)$ be the *backward* probability that the portion of the sequence after *and not including the* r*th position* is generated by the model, *conditional on the fact that* the state for the rth position is s_j. Thus, the forward and backward probability definitions are not symmetric. The forward and backward probabilities can be computed from model probabilities in a similar way as the evaluation procedure discussed above in section 10.3.4.3. The major difference for the backward probabilities is that the computations start from the end of the sequence in the backward direction. Furthermore, the probability value $\beta_{|T|}(T, s_j)$ is initialized to 1 at the bottom of the recursion to account for the difference in the two definitions. Two additional probabilistic quantities need to be defined in order to describe the EM algorithm:

- $\psi_r(T, s_i, s_j)$: Probability that the rth position in sequence T corresponds to state s_i, the $(r+1)$th position corresponds to s_j.

- $\gamma_r(T, s_i)$: Probability that the rth position in sequence T corresponds to state s_i.

The EM procedure starts with a random initialization of the model parameters and then iteratively estimates $(\alpha(\cdot), \beta(\cdot), \psi(\cdot), \gamma(\cdot))$ from the model parameters, and vice versa. Specifically, the iteratively executed steps of the EM procedure are as follows:

- **(E-Step):** Estimate $(\alpha(\cdot), \beta(\cdot), \psi(\cdot), \gamma(\cdot))$ from currently estimated values of the model parameters $(\pi(\cdot), \theta(\cdot), p_{..})$.

- **(M-Step):** Estimate model parameters $(\pi(\cdot), \theta(\cdot), p_{..})$ from currently estimated values of $(\alpha(\cdot), \beta(\cdot), \psi(\cdot), \gamma(\cdot))$.

It now remains to explain how each of the aforementioned estimations is performed. The values of $\alpha(\cdot)$ and $\beta(\cdot)$ can be estimated using the forward and backward procedures, respectively. The forward procedure is already described in the evaluation section, and the backward procedure is analogous to the forward procedure, except that it works backward from the end of the sequence. The value of $\psi_r(T, s_i, s_j)$ is equal to $\alpha_r(T, s_i) \cdot p_{ij} \cdot \theta^j(a_{r+1}) \cdot \beta_{r+1}(T, s_j)$, since the sequence-generation procedure can be divided into three portions corresponding to the sequence portion up to position r, the generation of the $(r+1)$th symbol, and the portion after the $(r+1)$th symbol. The estimated values of $\psi_r(T, s_i, s_j)$ are normalized to a probability vector by ensuring that the sum over different pairs $[i, j]$ is 1. The value of $\gamma_r(T, s_i)$ is estimated by summing up the values of $\psi_r(T, s_i, s_j)$ over fixed i and varying j. This completes the estimations of the E-step.

The re-estimation formulas for the model parameters in the M-Step are relatively straightforward. Let $I(a_r, \sigma_k)$ be a binary indicator function, which takes on the value of 1 when the two symbols are the same, and 0, otherwise. Then the estimations can be performed as follows:

$$\pi(j) = \gamma_1(T, s_j), \; p_{ij} = \frac{\sum_{r=1}^{m-1} \psi_r(T, s_i, s_j)}{\sum_{r=1}^{m-1} \gamma_r(T, s_i)}$$

$$\theta^i(\sigma_k) = \frac{\sum_{r=1}^{m} I(a_r, \sigma_k) \cdot \gamma_r(T, s_i)}{\sum_{r=1}^{m} \gamma_r(T, s_i)}$$

The precise derivations of these estimations on the basis of expectation-maximization techniques may be found in [454]. This completes the estimations for the M-step.

As in all EM methods, the procedure is applied iteratively to convergence. More details on robust termination criteria may be found in [454]. The approach can be generalized easily to N sequences by applying the steps to each of the sequences, and averaging the corresponding model parameters in each step.

10.3.4.6 Computing Anomaly Scores

In theory, it is possible to compute anomaly scores directly for the test sequence V, once the training model has been constructed from the sequence database $\mathcal{D} = T_1 \ldots T_N$. However, as the length of the test sequence increases, the robustness of such a model diminishes because of the increasing noise resulting from the curse of dimensionality. Therefore, the comparison units (either extracted from the test sequence or specified by the domain expert) are used for computing the anomaly scores.

- When the comparison units are extracted from the test sequence (the absolute model of Definition 10.3.2), it is possible to compute the probability of fit of each comparison unit to the training data. The negative logarithm of the resulting probability provides an anomaly score in which large values are indicative of anomalous behavior for a particular comparison unit. At this point, the method of section 10.3.2.1 can be used to combine the final anomaly scores over different windows.

- When the comparison units are provided by a domain expert (the relative model of Definition 10.3.2), two separate Markov Models are computed for the training and test sequences, respectively. This approach will not work, if the test sequence is not sufficiently long to create a robust model without overfitting. The negative logarithm of the fit probability of the comparison unit to the two models is computed with the Viterbi algorithm. The difference in the two values is reported as the anomaly score.

The second of the two methods is used rarely, because of the tendency of a single test sequence to create a model which overfits the data.

A number of methods also use the Viterbi algorithm on the test sequence in order to mine the most likely state sequence. In some domains, it is easier to determine anomalies in terms of the state sequence rather than the observable sequence. Furthermore, low transition probabilities on portions of the state sequence provide anomalous localities of the observable sequence. The drawback is that the most likely state sequence may have a very low probability of matching the observed sequence. Therefore, the estimated anomalies may not reflect the true anomalies in the data when an *estimated* state sequence is used for anomaly detection. The real utility of the Viterbi algorithm is in providing an *explanation* of the anomalous behavior of sequences in terms of the intuitively understandable states rather than quantifying its anomaly score.

10.3.4.7 Special Case: Short Sequence Anomaly Detection

A special case arises, when database $\mathcal{D}$ of shorter sequences is available, and the anomalous sequences need to be determined, on the basis of their dissimilarity with other sequences. This corresponds to Definition 10.3.4. As discussed earlier, this problem is similar to point anomaly detection formulations, which are used in multidimensional data. Distance- and clustering-based methods for addressing this special case have been discussed in section 10.3.2.3. A question arises whether HMMs can also be used for probabilistic clustering and anomaly detection in sequences with the use of mixture models.

It turns out that it is possible to use hidden Markov models for clustering and anomaly detection. The idea is to represent the data as a *mixture of HMMs* [111, 500]. As discussed earlier, the HMM is itself a mixture model, in which each state can be considered a component of a mixture. This approach uses a *second* level of mixture modeling, where the observations correspond to individual sequences from $\mathcal{D}$ rather than the individual symbols of a sequence. A single HMM model for a training data set makes the implicit assumption that every sequence $T_i \in \mathcal{D}$ is generated from the same distribution (cluster). This is not true in practice, since the individual sequences might themselves belong to different clusters, each of which has its own generating HMM. This can be modeled using a second level of mixture modeling, in which the k independent HMMs are used, and each sequence $T_i \in \mathcal{D}$ is associated with a generative probability from this mixture. Sequences that have low generative probabilities, or which have low probability of assignment to their best matching component may be considered anomalies.

10.3.5 Kernel-Based Methods

An interesting approach for anomaly detection in whole sequences is to use kernel methods. Consider a setting, where we have N sequences, we would like to determine outliers among them, without using window-based analysis. In other words, each sequence is treated individually as a comparison unit.

The basic idea in kernel-based methods is to use linear models in a transformed multidimensional space. In other words, each sequence is implicitly transformed into a multidimensional space with the use of a kernel similarity function. The basic idea here is that we can construct an $N \times N$ matrix S of similarities between the various sequences. Typically, kernel functions are used to compute these similarities. Some examples of such kernels include the use of the *bag-of-words kernel*, *spectrum kernel*, and the *weighted degree kernel* [33]. All these kernel functions have the virtue of being positive-semidefinite which is essential for

extracting a valid embedding from the similarity matrix S. One can then extract a multidimensional (k-dimensional) embedding from the symmetric and positive semi-definite matrix S by extracting all the k strictly positive eigenvectors from the embedding as follows:

$$S \approx Q\Lambda^2 Q^T \tag{10.10}$$

Here, Q is an $N \times k$ matrix with orthonormal columns and Λ is a $k \times k$ diagonal matrix. The matrix $Q\Lambda$ in an $N \times k$ matrix containing the k-dimensional embeddings of the data points. Furthermore, each of these k dimensions are uncorrelated with one another, which makes it particularly easy to use the Mahalanobis method. The first step is to normalize each column of $Q\Lambda$ to unit norm, which yields the matrix Q. Subsequently, the k-dimensional mean of the rows of Q is computed. The Euclidean distance of each point to this mean provides the Mahalanobis outlier score in the embedded space.

It is noteworthy that this approach is implemented with the use of kernel similarity functions among strings, which may not always satisfy domain-specific criteria. Therefore, one might ask whether it is possible to use more traditional similarity functions like the use of the longest common subsequence or other heuristic methods. The main problem in using such methods is that the resulting matrix might not be positive semi-definite. In such cases, we might not be able to diagonalize the matrix with non-negative eigenvalues, which is required to extract the embedding $Q\Lambda$. Note that the matrix Λ^2 in Equation 10.10 is required to contain nonnegative eigenvalues. In such cases, one can still *heuristically* extract an approximate embedding. For many natural similarity functions, the negative eigenvalues are small in magnitude, and they may be truncated to 0 as an approximation. It can be shown that such an approach closely approximates the original similarity matrix when the negative eigenvalues are small.

Kernel functions can be used in combination with other types of linear models. For example, one can adapt the one-class support-vector machine to outlier detection in sequences by using the kernel similarity function. This approach boils down to a straightforward application of the technique in section 3.4 of Chapter 3.

10.4 Complex Sequences and Scenarios

In many real applications, the available sequences may be more complex than the ones that have been introduced so far in this chapter. For example, in a system diagnosis application, multiple sequences may continuously be produced by different types of sensors. In other cases, *each element* of the sequence may be a *set* drawn from Σ rather than a solitary symbol. In some scenarios such as online applications, it may be desirable to declare a sequence an anomaly as early as possible by observing only the early part of it. These cases may sometimes require more complex techniques than those discussed so far.

10.4.1 Multivariate Sequences

Multivariate sequences are quite common in system diagnosis applications, in which multiple sensors may record sequences drawn from possibly different alphabets. Consider the case, where r different sensors record sequences drawn from the (respective) alphabets $\Sigma_1 \ldots \Sigma_r$. Each generated symbol is associated with a time-stamp, which is particularly crucial in the multivariate case in order to determine the temporal correspondence between the sequences generated by the different sensors.

Much work has not been done in the literature on the multivariate case for *discrete* sequences, as compared to the continuous case [140]. However, a variety of solutions are possible for finding unusual *time-windows* by combining the results from univariate analysis of the different series. Let $q_i(t_c-h, t_c)$ represent the generative probability of the subsequence generated in the time-interval (t_c-h, t_c). This generative probability can be computed using Equation 10.1 on an individual sequence. The final generative probability for the anomaly score of the *time window* (t_c-h, t_c) over all the r different series is given by $\prod_{i=1}^{r} q_i(t_c-h, t_c)$.

The case in which the anomalies need to be discovered on the entire sequence is a bit simpler. In this setting, we have a database of N multivariate sequences and we wish to discover the anomalies among them. In that case, one can use an approach similar to that discussed in section 10.3.5. The only difference is that a *multivariate* similarity function needs to be used to construct an $N \times N$ similarity matrix and extract the kernel embedding. All other steps are identical to those discussed in section 10.3.5.

10.4.2 Set-Based Sequences

Most of the algorithms in this chapter are designed for the case of sequences in which each symbol is drawn from the symbol set Σ. In practice, each unit element of the sequence may be a set $S_i \subseteq \Sigma$. Such cases arise commonly in domains such as market-basket analysis. The analysis of anomalies in such complex sequences are different from the case of simpler sequences, because two set-based elements may have varying levels of similarity, depending upon the number of common elements between the sets. Furthermore, temporal correlations between the set-elements in such sequences are typically much weaker than symbol-only sequences because of the larger number of possibilities for a set element in the sequence. For example, in the market basket scenario, it is generally not meaningful to predict conditional probabilities of set-based symbols based on the previous history.

The problem of finding anomalies in set-based sequences is much closer to the problem of finding unusual novelties or change analysis in multidimensional data. Here, each set is treated as discrete multidimensional binary vector of dimensionality $|\Sigma|$, corresponding to whether or not an element of Σ is present in the set. A number of methods available in the literature are applicable to these scenarios:

- The models discussed in Chapter 8 for first-story detection [622] in text can be applied to this scenario by treating each set-element analogous to a document, and the symbol set Σ analogous to a text lexicon. In fact, a few methods [29] treat the problem of novelty detection of text, categorical data, and market-basket sequences in a unified way, with the use of clustering. The creation of new clusters in a streaming cluster generation process, corresponds to the relevant anomalies. It has been shown in [29], that such a method can simultaneously determine evolving clusters, anomalies, and newly forming trends in the underlying data.

- Some methods have also been designed to find surprising patterns with the use of the concept of *temporal description length* [122]. The goal is to determine itemsets and data segments which reflect significant changes in correlations over time. This technique borrows ideas from information theory. The itemsets are temporally segmented. For each segment, the minimum number of bits requited to encode each segment is determined. Segments that require a larger number of bits to encode are assumed to be less homogeneous and may contain surprising patterns or change points. Such segments may be deemed to be outliers.

In general, set-based sequences are much closer in spirit to sparse multidimensional data, and provide a much richer domain for analytical purposes. Numerous methods for aggregate change analysis discussed in Chapter 8 can also be easily generalized to this case.

10.4.3 Online Applications: Early Anomaly Detection

In many sequential data mining applications, such as those arising in Web click-streams, or systems diagnosis applications, it may be desirable to perform the analysis in online fashion, as more data is received. In such cases, most of the techniques discussed earlier can be adapted to the online case:

- **Position outliers:** For the case of position outliers, the prediction is dependent only on a short memory before the position being predicted. The current model can be used to make an efficient prediction. Furthermore, many of the commonly used training models such as probabilistic suffix trees can be efficiently updated as more sequences are added to the training database.
- **Combination outliers:** In these cases, the unit of prediction is typically the comparison unit. As long as the next comparison unit has been derived, it can be used in conjunction with the current model to make the prediction. The process of updating the training model may sometimes be more challenging. Some models such as k-nearest neighbors may slow down with increasing training data size. In such cases, the training data may need to sampled in order to retain efficient prediction.

Numerous examples of online sequential anomaly detection methods exist in domains as diverse as systems diagnosis and intrusion detection [10, 22].

10.5 Supervised Outliers in Sequences

Because of the complexity of sequence data, it is often hard to obtain meaningful outliers with the use of purely unsupervised methods. Many patterns that are discovered as outliers may correspond to noise, and may not represent true anomalies. It has been shown in diverse application domains such as system call anomalies [338], financial fraud [374], and Web sequence robot detection [531], that the nature of the anomalous sequences are highly specific and governed by the underlying application. Therefore, the ability to distinguish noise from anomalies can only be obtained by incorporating additional knowledge about previously known (and interesting) examples into the outlier analysis process. Such learning methods can significantly improve the quality and relevance of the underlying outliers in real application scenarios.

The problem of supervised outlier detection in sequence data is equivalent to sequence classification. As in all supervised outlier analysis methods, the major difference is in terms of class imbalance and cost sensitivity, which needs to be accounted for in the classification process. This ensures that rare classes can be meaningfully discovered. In these models, labels may be available in different ways:

- Labels may be associated with each position in a sequence. Such situations occur commonly in natural language data, in which the different words may need to be classified in a sentence [51, 337]. In the rare-class versions of this problem, most of the labels may belong to the normal category, but an occasional label may correspond to an anomaly (e.g., a word which is "out of place" or grammatically incorrect).

- Rare labels may be associated with small subsequences of a single large sequence [16]. Other subsequences are assumed to be normal by default.
- Labels are associated with the full sequences in a database of multiple sequences. Most labels are normal, but a small number of them may be anomalous. This is the most common scenario in real applications such as intrusion detection.

The second (window-based) formulation above can be transformed to the third (full-sequence) formulation by extracting windows of subsequences from the full sequence, and classifying them with the use of a full-sequence classifier. Furthermore, in many cases of temporal sequences, it is desirable to learn the label of a sequence by examining as little of it as possible from the early portions of the sequence. This is known as *early classification*. Detailed surveys on sequence classification may be found in [11, 575].

Although most of the methods for classification of sequence data are not designed for the imbalanced case, the generalization to the imbalanced case is straightforward with the use of the meta-algorithms discussed in Chapter 7. Thus, the issue of class imbalance is orthogonal to the actual techniques that are used for sequence classification. While sequence classification is too vast an area to be covered in this chapter, a brief review of the main classes of techniques are discussed below.

Feature Transformations

The first class of methods extracts symbolic sequences of size k from the base sequences. These are referred to as k-grams, and they form the "words" from the sequence vocabulary. These words are used as the features, and a variety of classifiers such as SVMs [354] can be used on this new representation. Another class of feature extraction methods includes *pattern-based methods*, in which relevant patterns with sufficient support and confidence are mined. Such an approach mines the *local* patterns from the underlying data, which are relevant to sequence classification. Other forms of feature extraction such as wavelet decomposition [13] can be used, which extract both the local and global properties of the underlying data. A rule-based classifier was constructed which relates the wavelet patterns to the underlying classes.

Distance-based Methods

As discussed in Chapter 9 on time-series outlier analysis, distance-based methods are particularly popular in the context of continuous time-series. In the case of continuous data, distance functions such as the Euclidean metric can be used. However, for discrete data sets, popular methods for sequence alignment such as the edit distance [239] can be used. Distance-based methods can also be used *in combination* with feature transformation methods. For example, the k-gram representation can be used in combination with distance-based methods in order to construct sequence classifiers.

Hidden Markov Models

These are extremely popular methods, which can be used in order to model either whole sequences or subsequences associated with class labels [16, 511]. In these methods, a hidden Markov model is used in order to create a generative model which is specific to each class. The major difference from the methodology discussed in this chapter is that the parameters of the HMM are learned separately for each class. This is done by using only the sequences

specific to a particular class in order to train the relevant model. For a given test sequence, the evaluation algorithm is used to determine the identity of the best fit class. In the rare class scenario, the cost-sensitive methods of Chapter 7 can be easily generalized to this case. Specifically, let $q_1 \ldots q_m$ be the probabilities generated by the forward algorithm, when applied to each of the k different HMMs, corresponding to the m different classes. Let the costs for the m different classes be $c_1 \ldots c_m$. Then, the cost-weighted probability f_i for the ith class is defined as follows:

$$f_i = \frac{c_i \cdot q_i}{\sum_{j=1}^{k} c_j \cdot q_j} \tag{10.11}$$

The class with the highest value of f_i is reported as the relevant one.

Kernel Support Vector Machines

Finally, one can use kernel support-vector machines by using string kernels to compute the similarities between pairs of strings. The kernel similarity matrix can be used to perform the classification of the sequences. The SVM needs to be modified slightly to account for the rare-class setting. These modifications are discussed in Chapter 7.

Comparative Studies

A natural question arises as to how the aforementioned classifiers compare with one another. A detailed performance study, which compares different sequence classifiers is provided in [168]. The results suggest that SVM classifiers can work well when they are used in combination with the right feature transformation methods. Furthermore, such classifiers can also provide high-levels of interpretability, when the feature space corresponds to k-grams extracted from the data.

10.6 Conclusions and Summary

Discrete sequences are extremely common in virtually all application domains such as biological data, user-generated data, systems diagnosis data and Web logs. This provides a massive source of temporal data, which is very rich, and is also very complex from the perspective of outlier analysis. The complexity of sequence data allows for diverse definitions of sequence outliers. The broad classes of models are similar to continuous time-series data, although the details of the models are different because of the effect of the different data type. As in the case of time-series data, a single position can be identified as a contextual outlier, or a set of positions may be identified as a collective outlier. Contextual outlier-detection methods typically use Markovian techniques and are significantly less computationally complex. This chapter also discusses some of the more common models and techniques for identifying collective outliers in sequence data. These are distance-based methods, frequency-based methods, and hidden Markov models. Many of these methods can also be generalized to the supervised case, when labels are available.

10.7 Bibliographic Survey

The problem of sequence outlier detection is useful in the context of a wide variety of applications such as host-based intrusion detection, system fault detection, biological sequences,

and Web click-streams [338, 374, 484, 506, 531]. A general survey on sequence outlier detection may be found in [126]. Another recent survey [232] and a related book [231] provide an integrated treatment of time-series and discrete-sequence outlier detection.

Outliers are of two types corresponding to *position outliers* and *combination outliers*. Position outliers correspond to specific events (symbols) in the sequence (string), which may be considered an outlier. Combination outliers correspond to unusual subsequences of symbols in the sequence, which differ from the overall trends in the data. Position outliers are computed using either Markovian techniques [396, 588] or rule-based systems [148, 588], which are equivalent. The short-memory property [465] can be used in order to predict the value of a sequence from the use of the last k positions. These models can be considered the discrete equivalent of continuous regression-based predictive models discussed in Chapter 9. The use of rule-based methods for finding position outliers is discussed in [349, 148]. The use of a suffix-tree to represent all order-k (or less) states for outlier prediction was proposed in [387]. Sparse Markov transducers, which are constructed using probabilistic suffix trees, are presented in [189]. The simplest Markov model of the first order is proposed in [588] and a kth order Markovian model is presented in [396]. A discussion of pruning techniques for improving the efficiency of predictive Markovian models is provided in [169]. Probabilistic suffix trees [524] can be used in order to greatly improve the speed of Markovian techniques. A method for determining outliers based in information theoretic measures was proposed in [312]. Given a sequence, the idea is to segment it into two parts, and determine the segment that has the larger MDL (Minimum Description Length). This segment is assumed to be more noisy and contain the outliers. This process is recursively repeated, until the appropriate outlier position in the sequence is determined. The actual computation of the description length is based on the concept of Kolmogorov complexity.

Combination outliers typically use windowing techniques in which comparison units are extracted from the sequence for the purpose of analysis [197, 272]. Proximity-based methods for outlier detection require the efficient computation of similarity functions in sequence data, such as the longest common subsequences [279] and the edit distance [239]. A number of different similarity functions are discussed in [198, 338, 339]. These methods are typically used in conjunction with window-based methods in order to compare short subsequences of the test sequence with the training sequence. Frequency methods provide a more effective way of computing the surprise of a user-specified comparison unit between a training and test sequence. The TARZAN algorithm [310] uses suffix trees in order to compute the surprise level for a user-specified comparison unit. Proximity methods are also used for point-anomaly detection with the use of k-nearest neighbor methods [127], k-medoid based clustering [103, 104], and probabilistic suffix tree-based clustering [581]. Markov Models for the detection of significant episodes and change points in sequences are proposed in [242, 501].

Hidden Markov models are a popular method for finding combination outliers. A tutorial on hidden Markov models may be found in [454]. While they support the direct determination of the outliers with the use of the underlying fit probabilities, methods are also designed to mine the state transition sequences with window-based methods for anomaly detection [197, 612]. Hidden Markov Models have also been used for point anomaly detection of small sequences [111, 500] by using a mixture of hidden Markov models.

Outlier detection is also a challenging problem in the context of complex sequences in which individual points are sets rather than symbols. Most methods for novelty detection in text, categorical, and market-basket streams can be generalized to this case [29, 122]. The techniques for first-story detection in text can also be applied to set-based sequential data [47, 48, 622].

Supervised methods are used frequently in a variety of applications such as intrusion detection, when labeled data is available [185, 216, 217, 561, 545]. Reviews of sequence classification methods can be found in [11, 575]. Position classification in sequences is useful for many natural language applications [51, 337], in which the specific properties of positions in sentences need to be learned. The most common method for sequence classification is use of feature extraction methods such as k-gram extraction and wavelets [13, 354]. Hidden Markov models are a natural technique for the problem of classification [16, 511]. An experimental evaluation of methods for sequence classification may be found in [168].

10.8 Exercises

1. Consider the discrete sequence denoted by ACGTACTACGTACGTACGT.

- Construct an order-1 Markovian model in order to determine the position outliers. Which position is found as the outlier?
- Now create an order-2 Markovian model in order to determine the position outliers. Which position is found as the outlier?

2. Determine all order-1 rules from the discrete sequence of Exercise 1. Which position is found as the outlier?

3. For the discrete sequence of Exercise 1, determine all subsequences of length 2. Use a frequency-based approach to assign outlier scores to objects. Which pairwise sequence of length 2 should be considered a combination outlier?

4. Repeat Exercise 3 with sequences of lengths 3, 4, and 5. What happens to the relative outlier scores with increasing subsequence length?

5. Repeat Exercises 3 and 4 with the use of a distance-based approach.

6. Construct a black box 2-state hidden Markov model in order to learn the state probabilities of the sequence in Exercise 1. Write a computer program to learn the state transition and symbol emission probabilities. Now apply the model to extracted subsequences of length 2. Which subsequences are predicted as the outliers by the model?