On String Classification in Data Streams

Charu C. Aggarwal
IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
charu@us.ibm.com

Philip S. Yu
IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
psyu@us.ibm.com

ABSTRACT

String data has recently become important because of its use in a number of applications such as computational and molecular biology, protein analysis, and market basket data. In many cases, these strings contain a wide variety of substructures which may have physical significance for that application. For example, such substructures could represent important fragments of a DNA string or an interesting portion of a fraudulent transaction. In such a case, it is desirable to determine the identity, location, and extent of that substructure in the data. This is a much more difficult generalization of the classification problem, since the latter problem labels entire strings rather than deal with the more complex task of determining string fragments with a particular kind of behavior. The problem becomes even more complicated when different kinds of substrings show complicated nesting patterns. Therefore, we define a somewhat different problem which we refer to as the *generalized classification problem.* We propose a scalable approach based on hidden markov models for this problem. We show how to implement the generalized string classification procedure for very large data bases and data streams. We present experimental results over a number of large data sets and data streams.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms

Keywords

String, Classification, Hidden Markov Models

1. INTRODUCTION

In recent years, a number of applications such as market basket analysis, customer tracking, DNA analysis and text use data which are in string format. In many cases, it is desirable to classify particular kinds of fragments of these strings. This is a generalization of the traditional classification problem for strings in which we associate a class label with an unmarked string. This does not address the significantly more complex problem of finding particular kinds of substructures in strings. In many applications, the problem of finding substructures in strings is significantly more important than that of classifying the string itself. Some examples of such domains are as follows:

- In a genome application, we would like to find different kinds of protein sequences which are embedded in very long patterns. A given pattern may contain one or multiple occurrences of such a substructure. Such a problem is especially difficult because of the considerable length of the strings in a typical biological application.
- In a credit card application, a particular subset of transactions may correspond to fraudulent behavior of a set of customers. While the problem of finding fraudulent transactions has been well studied in prior work, our framework provides a more general technique to find particular portions of the transactions which are fraudulent in nature.

The standard classification problem for strings was been studied in the context of a number of domains [1, 4, 5, 6, 9, 10, 12, 13, 14, 15, 17]. The nature of substructure mining is inherently more difficult than the standard version of the classification problem because of the added difficulty of determining the substructure extent. Therefore, standard classification methods cannot be easily extended to the generalized classification problem. In many applications, an additional level of complexity arises from the need to apply the procedure to very large databases and data streams. The data stream problem imposes additional one-pass constraints on the classification process. Many classification algorithms for a variety of domains have also been extended to the data stream problem [2, 7, 11]. The generalized string classification problem is a particularly difficult one for the stream domain because of the complexity of the Hidden Markov Model construction process. Most natural training approaches for Hidden Markov Models are inherently multi-scan methods. We will develop an approach in which is I/O efficient and requires only one pass over the data stream.

In order to solve the generalized classification problem, we will utilize a stochastic modeling approach, which is referred to as *hidden markov models.* We will first discuss how to find simple kinds of substructures in a string which does not have any complicated nesting patterns. Then, we will discuss the extension of this technique to substructures with different kinds of nesting patterns. It is a challenging problem to use the hidden markov model approach for classification of very large databases. This is because the parameter estimation procedure of a hidden markov model usually requires an iterative expectation maximization procedure which is computationally expensive and may require a large number of passes on disk resident data. In this paper, we are able to avoid this problem by designing the topology of the Markov Models effectively. Specifically, we construct the models in such a way that the entire training procedure requires only a single database scan with simple additive operations.

This paper is organized as follows. In the remainder of this section, we will define the generalized classification problem. In section 2, we will discuss the basic hidden markov model (HMM) approach to the generalized string classification problem. Section 3 discusses extensions to more complex variants. The empirical results are discussed in section 4. Section 5 contains the conclusions and summary.

1.1 The Generalized Classification Problem

The generalized classification problem for strings is defined as follows. We given a database $\mathcal{D}$ containing N strings $s_1 \ldots s_N$. We also have a set k class labels denoted by $C_1 \ldots C_k$. Each string s_i may have one or more substrings which satisfy the following properties:

• The substring has a *begin marker* which indicates its first position in s_i.
• The substring has an *end marker* which indicates its last position in s_i.
• The substring has a label drawn from $\mathcal{C}_1 \ldots \mathcal{C}_k$ which indicates its class.

The database $\mathcal{D}$ is used for training a model to identify the substructures in the strings. We note that the database $\mathcal{D}$ may also take on the form of a data stream in which new records are received continuously. For a given test instance T, the model is used to determine the following: (1) The identity of the classes present in T as substructures. (2) The location and extent of these substructures. The classification algorithm automatically inserts the appropriately labeled markers at various positions in the string in order to identify the corresponding substructures to the user.

The above definition can be generalized to data streams by assuming that the data contains separate training and test streams. These separate training and test streams contain records which are differentiated only by the fact that the markers are present in the training data. In practice, the process of training and testing is a separate and continuous process by which the training model is continuously updated with additional data, whereas the testing process uses this continuously constructed training model.

2. HIDDEN MARKOV MODEL APPROACH

In this section, we will first provide a brief description of hidden markov models, and then discuss how the concepts can be suitably leveraged in order to solve the generalized classification problem. A hidden markov model describes a series of observations by a "hidden" stochastic process called a Markov chain. This model has a number of underlying states, which have transitions into one another with predefined probabilities. A probability distribution of symbols

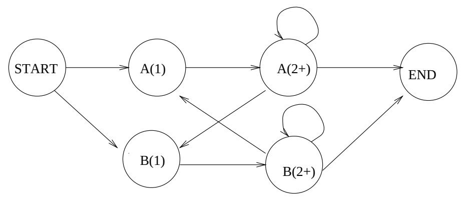


Figure 1: HMM Example

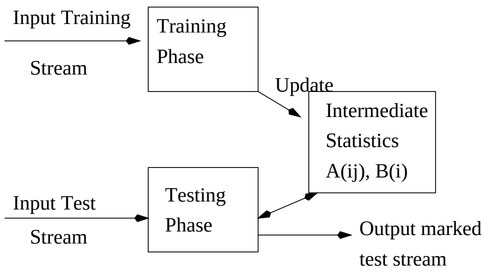


Figure 2: Stream Model Updates

is associated with each state. For each transition, a symbol is generated by using the probability distribution of symbols at the state that the system transitioned from. While the actual sequence of states is not observable (and is only a conceptual model), the sequence of symbols generated by the hidden markov model are observed explicitly. The particular topology of the model, state transition probabilities and symbol generation probabilities create a class of strings which are generated by that markov model.

For example, consider the markov model illustrated in Figure 1. Other than the START and END states, the model contains 4 states, which we have labelled $A(1)$, $A(2+)$, $B(1)$, and $B(2+)$ respectively. This model represents sequences drawn from the alphabet $\{A, B\}$. We assume that a transition from either of the states $A(1)$ or $A(2+)$ result in the generation of the symbol A, whereas a transition from either of the states $B(1)$ or $B(2+)$ result in the generation of the symbol B. The topology of the model ensures that all sequences generated from the model contain alternating subsequences of two or more symbols of the same type. Thus, a valid sequence generated from this model would be AABBBAA, but not AABAA. For any given string, the maximum probability path (product of corresponding transition probabilities and symbol generation probabilities) through the model reflects the level of fit of the string to the model. It is clear that in the case illustrated in Figure 1, the probability of any path generating the string AABAA is 0. By changing the symbol distribution and transition probabilities at the states, it is possible to change the nature of the sequences generated. We note that it is often possible to model a given class of strings with different topologies of the markov model. However, it is often a matter of skill and experience to design the topology in such a way that the parameter estimation for a particular topology reflects the class behavior well. In this paper, we will develop a topological class of models which are suited to string classification both in terms of accuracy and efficiency.

In order to develop further descriptions of the algorithmic techniques used in this paper, we will introduce some notations and definitions. We assume that the states of the

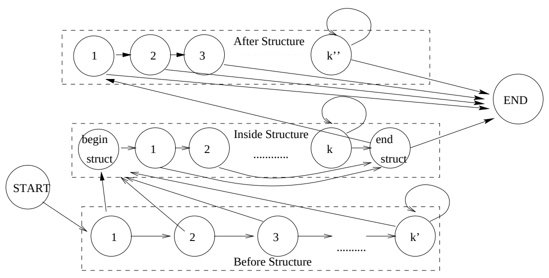


Figure 3: The Basic Substructure Problem

markov model are denoted by $\mathcal{S} = \{q_i \dots q_N\}$. The initial state probabilities are denoted by $\{\pi(1) \dots \pi(N)\}$. Therefore $\pi(i)$ is the probability that the system is in state i at the beginning of the stochastic process. It is clear that $\sum_{i=1}^{N} \pi(i) = 1$.

The set of symbols from which the strings are constructed are denoted by $\Sigma = \{\sigma_1 \dots \sigma_l\}$. For each state $q_j \in \mathcal{S}$, a probability distribution characterizes the symbol being generated at the corresponding transition. We shall denote this probability of the symbol σ_i being generated in state q_j by $b_j(\sigma_i)$. Thus, for a given state q_j, We have $\sum_{i=1}^{l} b_j(\sigma_i) = 1$. The probability of a transition from state q_j to state q_k is denoted by a_{jk}. Since the probability of transition from any state is one unit, we have $\sum_{j=1}^{N} a_{ij} = 1$.

Each sequence of transitions in the markov model creates a sequence of symbols generated at the various states. Let the corresponding path P for the sequence of transitions be denoted by $i_0 \dots i_{T+1}$, and let the set of symbols generated at those states be denoted by $SS = \theta_1 \dots \theta_{T+1}$. Then, the probability of the transitions following path P is given by the following expression:

$$\mathcal{P}(i_0, i_{T+1}, T+1, G) = \pi_{i_0} \cdot a_{i_0 i_1} \cdot b_{i_1}(\theta_1) \dots$$
$$\dots b_{i_{T-1}}(\theta_{T-1}) \cdot a_{i_{T-1} i_T} \cdot b_{i_T}(\theta_T) \cdot a_{i_T i_{T+1}}$$

The above expression is simply the product of the probabilities of state transitions and corresponding symbol generations. A higher value of this probability indicates a greater fit of the particular string $\theta_1 \dots \theta_{T+1}$ to the class of strings generated by this model. Therefore, hidden markov models provide the ability to model families of strings by using a particular layout of the model, symbol generation probabilities, and state transition probabilities. Conversely, for a given training set of strings, the parameters such as a_{ij} and $b_j(\cdot)$ need to be estimated from the data.

2.1 The Basic Substructure Problem

In this section, we will discuss the hidden markov model for the most simple case when there is no nesting of the class labels within one another. In later sections, we will show how to generalize it to the case when the underlying structures may be complex variations on the basic model.

In order to model the simple case, we divide the string into three portions: (1) The portion before the begin marker outside the class substring. (2) The portion within the class substring itself. (3) The portion outside the class substring. As illustrated in Figure 3, there are three kinds of states in this model, corresponding to symbols encountered before, inside or after the class substring. In order to transition from one class of states to another, either a *begin* or *end* marker

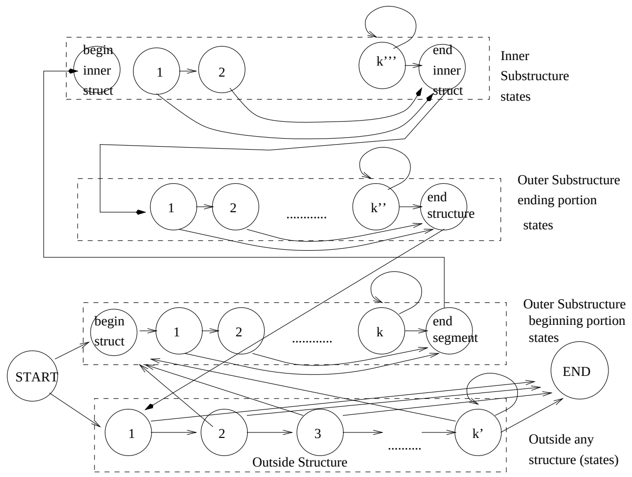

Inner Substructure states
begin inner struct
end inner struct
k'''
Outer Substructure ending portion states
end structure
k''
Outer Substructure beginning portion states
begin struct
end segment
k
START
END
Outside Structure
Outside any structure (states)
k'

Figure 4: Nested Substructures (Example)

must be encountered. We denote these markers by *beg_str* and *end_str* respectively. This is achieved by ensuring that the probability distribution of the symbols generated by the first and last states of the second segment of states are the deterministic distributions corresponding to the "begin" and "end" markers. Therefore, the state marked as *begin struct* in Figure 3 always generates the *beg_str* symbol, whereas the state marked *end struct* always generates the *end_str* symbol. Thus, the symbol set for the markov model is given by $\Sigma' = \Sigma \cup \{beg\_str, end\_str\}$. In addition, we impose two constraints on matching a string with the markov model:
(1) The initial set of state probabilities π are be chosen so that the beginning state is the START state. This is achieved by setting $\pi(i) = 0$ for each state other than the START state.
(2) After generation of all the symbols of the hidden markov model, the system finishes in the END state. Therefore, only those sequences in the Markov Model which begin at the START state and end at the END state are of interest. We note that the constraint on the model finishing in the END state can also be imposed by appending a special termination marker at the end of each string which is generated only by the END state.

It is assumed that the portion inside the structure contains k states. The portions before and after the structure contain k' and k'' states respectively. The last symbol of each segment contains a self-looping state which is used to model the case when the length of the string in that segment is larger than the number of available states. The values of k, k' and k'' are estimated using the training data. We will discuss this issue in more detail in a later section.

While other Markov model designs are certainly possible, we choose this particular design for several reasons: (1) The symbol generation and transition probabilities are sensitive to the distance of the state from the beginning of the first state of each segment. This models the fact that in most real data sets, the distribution of symbols and position of the class begin and end markers is sensitive to its position. (2) This class of Markov models can be easily extended to more complex structures containing nested sequences. These methods will be introduced at a later stage in this paper. (3) This class of Markov models creates a formulation which allows an efficient method for the parameter estimation problem for very large databases. We will discuss this issue in the next section.

2.2 Parameter Estimation for the HMM

The hidden markov model discussed in the previous section has a number of parameters which need to be estimated from the training data. The effectiveness of the model depends considerably upon how these parameters are estimated. Furthermore, since the method is to be used for massive databases and data streams, it is important to be able to perform these estimations efficiently.

The first parameter which needs to be estimated is the number of states for each segment of the hidden markov model. This is because these parameters are needed to actually set up the physical layout of the markov model. In Figure 3, we have denoted these parameters by k, k' and k''. In order to perform this computation, we find the mean and standard deviation of the number of symbols inside, before, and after the class substring. We denote the corresponding means by μ, μ' and μ'' respectively. The standard deviations are denoted by σ, σ' and σ'' respectively. Then the parameters estimations for k, k' and k'' are $\mu + r \cdot \sigma$, $\mu' + r \cdot \sigma'$, and $\mu'' + r \cdot \sigma''$ respectively. Here r is a user-defined parameter.[1] For the case of a massive database, we can either use a sample of the data or the entire database to perform this computation. In the case of a data stream, this computation is done only once for an initial segment of the data stream. At that point the markov model is constructed and its parameters are estimated for the training process.

Once the values of k', k'' and k''' have been determined, the physical layout of the states of the markov model can be finalized. We use this physical layout to estimate the transition and symbol generation probabilities. We note that this part of the parameter estimation is often a non-trivial task for a given hidden markov model. It is not practical to use an approach such as the Baum-Welch algorithm, because it requires multiple passes over the data. Such a process is inefficient for most large databases because of the considerable disk I/O required for multiple data passes. In the case of a data stream, such a procedure is essentially infeasible.

In the family of models discussed in this paper, the parameter estimation problem is simplified because of the particular topology of the models we have constructed. We note that the *begin struct* and *end struct* states generate a particular (marker) symbol with probability one. All markov models discussed in this paper are designed in such a way that the state to which each transition occurs belongs to one of two kinds of states: (1) The *begin struct* or *end struct* states which generate the begin markers and end markers deterministically. (2) Any of the other states which generate one of the symbols from the base alphabet.

Since the training data is provided with the *begin markers* and *end markers* specified in the strings, the exact sequence of transitions for each training data point can be known exactly irrespective of the transition and symbol generation probabilities. This is not the case for most markov models in which the paths are only known probabilistically, and the parameter estimation method tries to use iterative methods in order to maximize the accuracy of estimation. In the family of models discussed in this paper, the parameters can be easily estimated by finding the unique path through the model for each training example. We note that such a process requires one examination of the string, after which it never needs to be used again. This is important from a computational point of view for the data stream problem. Once these paths have been determined, we compute the following set of aggregate statistics from the training data:

A_{ij} : Number of times that a transition from state q_i to state q_j occurs over all unique paths for the different training sequences.

$B_i(\sigma_k)$: Number of times that symbol σ_k is observed in state q_i over all unique paths for the different training sequences.

One way of estimating the transition probability a'_{ij} is defined as $a'_{ij} = A_{ij}/\sum_{j=1}^{N} A_{ij}$. This is because the statistic in the numerator represents the number of transitions from state i to state j, whereas the statistic in the denominator represents the total number of transitions from state i. We note that while a'_{ij} is the maximum likelihood value of the transition probability without any prior knowledge of the data, this may often lead to unstable estimates when the amount of training data is small. Therefore, some amount of smoothing of the estimates using the default initial probabilities is useful.

If no data is available for a particular state, then we assume by default that each possible transition out of that state has the same probability. Similarly, for a given transition, since l possible symbols can be generated, we assume that the probability of each possible symbol being generated by default is given by $1/l$. When the amount of data available is small, we would like these default probabilities to be incorporated in the final estimation. Therefore, we make the implicit assumption that the observed transitions correspond to only a fraction $\alpha(i)$ of the entire set of observations. The remaining fraction of $1 - \alpha(i)$ observations for state i correspond to the default case in which the transition probability to every other state is equally likely. Then, the transition probabilities for the markov model are estimated as $a_{ij} = \alpha(i) \cdot a'_{ij} + (1 - \alpha(i)) \cdot (1/N)$.

The value of $\alpha(i)$ heavily depends upon the number of data points which result in a transition from state i. When there are a larger number of transitions from state i, the available data provides an accurate estimation of the state transition probabilities, and the value of the fraction $\alpha(i)$ should be close to 1. Let us assume that the total number of transitions out of state i is denoted by $M(i) = \sum_{i=1}^{N} A_{ij}$. If p_{ij} be the true transition probability from state i to state j, then the standard deviation of the observed value $a'_{ij} = A_{ij}/M(i)$ is approximately $\sqrt{p_{ij} \cdot (1 - p_{ij})}/\sqrt{M(i)}$ by the central limit theorem. Correspondingly, we estimated the value of $\alpha(i)$ to $1 - 1/\sqrt{M(i)}$. This ensures that the maximum error contributed by the smoothing term is no larger than that of estimating the parameters using purely the training data. The symbol generation probabilities can be estimated in a similar way to the state transition probabilities using the following expression:

$b_i(\sigma_k) = \beta(i) \cdot B_i(\sigma_k)/\sum_{k=1}^{l} B_i(\sigma_k) + (1 - \beta(i)) \cdot (1/N)$

In this case $\beta(i)$ represents the fraction of the symbol generations at state i which are reflected by the available data. The value of $\beta(i)$ is estimated in a similar way as $\alpha(i)$. Specifically, $\beta(i)$ is estimated using the number of times that a transition into state i occurs over the entire training data

[1] For real applications, a choice of $r = 3$ suffices, since most of the data points are located within 3 standard deviations from the mean under the normal distribution assumption.

set. Let $P(i) = \sum_{k=1}^{l} b_i(\sigma_k)$ be the corresponding number of transitions. As in the case of $\alpha(i)$, the value of $\beta(i)$ was chosen[2] to be $1 - 1/\sqrt{P(i)}$.

We note that the training process results in the estimation of the parameters A_{ij}, and $B_i(\cdot)$. These parameters are continuously updated over time and are stored in a dynamically updated database along with the Markov Model topology. We note that all other parameters can be directly estimated from the above parameters which need to be updated incrementally. This database is used by the test procedure to estimate other parameters which are used for test stream classification. The overall architecture for test stream classification is illustrated in Figure 2. It is clear that the training procedure dynamically updates the intermediate statistics which is used by the test data stream. In the event that we are dealing with databases instead of data streams, then the statistics at the end of the training procedure are used for test example classification. In the next section, we will discuss how the training parameters are used for the purpose of classifying a test example. We also note that while the training procedure needs to sequentially update a centralized parameter database, this is not the case for a test example for which the processing can be easily parallelized or distributed as long as the parameter database is available.

2.3 The Testing Phase

While the testing phase is an inherently more difficult process from a computational standpoint, the advantage is that each test example can be classified separately. Therefore, the process scales up well with increasing computational speeds, and can be parallelized if desired. The method requires little disk I/O since the parameter set can be stored in memory during the classification process and updated only periodically. The procedure discussed in this section requires dynamic programming and is quite efficient in practice because of the nature of the underlying model.

The test instances are strings in which the positions of the begin and end markers have not been specified. We have already seen that there is a single non-zero probability path through the markov model, once these positions have been specified. Therefore, the classification phase of the problem can be posed as follows:

For a given test instance, where should the begin and end markers be inserted so that the probability of the unique path through the markov model is maximized.

We note that a similar problem arises when trying to identify a class of strings using the markov model technique [16]. However, in that case since the problem is only to identify the class of the string rather than its location and extent, the technique required is somewhat simpler. In that case, multiple paths exists through the model, and the identity of the class label is decided by the maximum probability path. Here, the problem posed is somewhat different:

For an incompletely specified string, where do we insert the marker labels so as to maximize the probability of the corresponding path through the markov model.

[2]An interesting point to note here is that the total number of transitions out of state i is equal to $\sum_{j=1}^{N} A_{ij}$. This is also equal to the number of symbols encountered at that state, and is equal to $\sum_{k=1}^{l} b_i(\sigma_k)$. This means that $P(i) = M(i)$. Therefore, for a given state i, the values of $\alpha(i)$ and $\beta(i)$ are the same.

While the problem of finding the maximum probability path through the markov model can be solved using the Viterbi algorithm [16], this version of the problem is subtly different, and requires a new solution. It turns out that it is possible to use a dynamic programming algorithm which can optimize these placements. Furthermore, this dyanmic programming algorithm only needs to use the set of parameters estimated by the training phase of the algorithm. Since these parameters are estimated dynamically using a one-pass algorithm, the testing process can be applied at any point in the stream computation, and not just at the end of the entire phase. Therefore, the procedure is naturally amenable to an *on-demand* classification process [2] for the string data stream.

Let us denote the test instance by $\theta_1 \ldots \theta_T$. We note that this test instance corresponds to a path through the markov model. This path will contain states which generate the actual symbols $\theta_1 \ldots \theta_T$ as well as the markers *beg_str* and *end_str*. Thus, the entire path of states through the markov model will have length greater than T. Let us denote the states which generate the symbols $\theta_1 \ldots \theta_T$ by $q_{i_1} \ldots q_{i_T}$. In addition, we assume that the first node on the path is the START state which corresponds to q_{i_0} and the last node $q_{i_{T+1}}$ is the END state. The sequence of states $q_{i_1} \ldots q_{i_{T+1}}$ may not necessarily be contiguously visited in the markov model in order to generate the string $\theta_1 \ldots \theta_T$. This is because the markers *beg_str* and *end_str* are also generated by the transitions from the *begin struct* or *end struct* states. Thus, if a marker state is present between two such states q_{i_k} and $q_{i_{k+1}}$, then an edge is not present in the markov model between the two states. This is a particular characteristic of the topology of the markov models discussed in this paper. Therefore, the following is always true of markov models discussed in this paper:

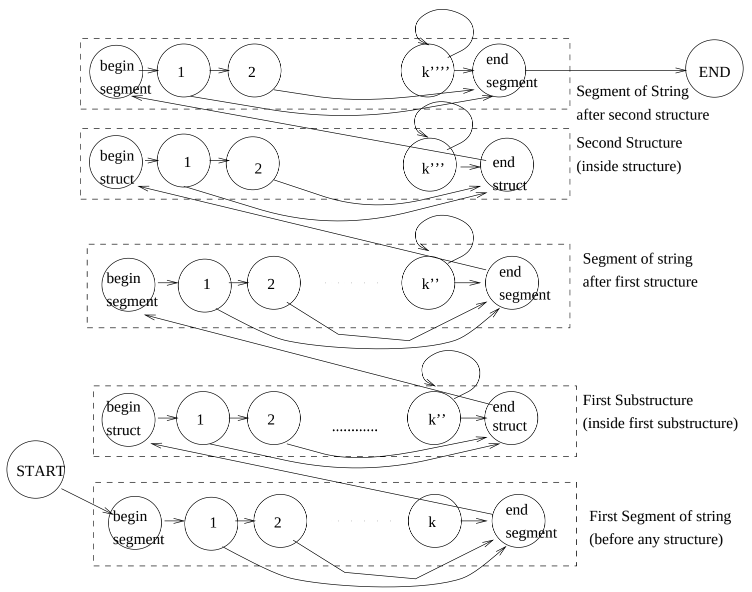
For any pair of non-marker states q_i and q_j, if a marker state q_b exists such that (q_i, q_b) and (q_b, q_j) are non-zero transition probability edges, then the following properties are true: (1) No other state q' exists such that the edges (q_i, q') as well as (q', q_j) exist. (2) An edge does not exist between q_i and q_j.

In order to facilitate further discussion, we will define a pseudo-transition probability between two such non-adjacent states. Therefore, for two such states q_i and q_j, we define the pseudo-transition probability $R(i, j)$ as the transition probability of the unique path from q_i to q_j.

$$R(i,j) = \begin{cases} a_{ij} & \text{if an edge exists between } q_i \text{ and } q_j \\ a_{ib} \cdot a_{bj} & \text{if an edge does not exist between } q_i \text{ and } q_j \text{ but } q_b \text{ is a marker state and the path } q_i q_b q_j \text{ exists in the model.} \end{cases}$$

Thus, $R(i, j)$ can be defined in a consistent way because of the afore-mentioned properties of the models in this paper. Let us consider the optimal probability path in the markov model from q_{i_0} to $q_{i_{T+1}}$ containing T marker states which generates the string segment G by successive transitions. Let us denote the probability of such a string generation by $\mathcal{P}(i_0, i_{T+1}, T+1, G)$. Let us denote the last symbol of the string G by g_l. Let us also denote the string G with its last symbol removed by G^-. Then, if $q_{i_0}, q_{i_1}, q_{i_2}, \ldots q_{i_{T+1}}$ be the optimal path, we have:

$$\mathcal{P}(i_0, i_{T+1}, T+1, G) = \pi_{i_0} \cdot R(i_0, i_1) \cdot b_{i_1}(\theta_1) \ldots \\ \ldots b_{i_T}(\theta_T) \cdot R(i_T, i_{T+1})$$



begin segment
1
2
k''''
end segment
END
Segment of String after second structure
begin struct
1
2
k'''
end struct
Second Structure (inside structure)
begin segment
1
2
k''
end segment
Segment of string after first structure
begin struct
1
2
k''
end struct
First Substructure (inside first substructure)
START
begin segment
1
2
k
end segment
First Segment of string (before any structure)

Figure 5: Seq. Substructures (Example)

We note that since $\pi_0 = 1$, it can be ignored from the equation. Furthermore, i_0 and i_{T+1} are fixed to be the START and STOP states. In order to reduce rounding errors, we define the logarithmic probability $\delta(i_0, i_{T+1}, T+1, G) = \log(\mathcal{P}(i_0, i_{T+1}, T+1, G))$. We obtain the following expression for the value of $\delta(i_0, i_{T+1}, T+1, G)$:

$$\mathcal{P}(i_0, i_{T+1}, T+1, G) = \log(R(i_0, i_1)) + \log(b_{i_1}(\theta_1)) + \\ +\log(R(i_1, i_2)) + \log(b_{i_2}(\theta_2)) + \ldots \log(R(i_{T-1}, i_T) + \\ +\log(b_{i_T}(\theta_T)) + \log(R(i_T, i_{T+1}))$$

The use of logarithms changes the multiplicative expression into an additive one as illustrated above. The optimal sequence of intermediate states may be obtained by using an iterative dynamic programming approach. First, we will set up an iterative condition to determine i_T, so that the corresponding path probability is maximized. The value of i_T which maximizes the probability of the corresponding path is as follows:

$$i_T = \text{argmax}_{j \in \mathcal{S}}\{\delta(i_0, j, T, G^-) + \log(b_j(g_l)) + \log(R(j, i_{T+1}))\}$$

This condition defines the natural iteration of a dynamic programming solution to the problem. In general, when we have to find out the optimal value of δ for any pair of nodes p and q for a given path of length n, the following expression for $\delta(\cdot,\cdot,\cdot,\cdot)$ turns out to be helpful for the dynamic programming solution:

$$\delta(p, q, n, G) = \max_{j \in \mathcal{S}}\{\delta(p, j, n-1, G^-) + \\ +\log(b_j(g_l)) + \log(R(j, q))$$

The corresponding intermediate node j can be computed by using the value of j which maximizes the above equation. The value of $\delta(\cdot,\cdot,\cdot,\cdot)$ for $n = 1$ is special and needs to be calculated separately as an initialization procedure. Specifically, we have:
$\delta(p, q, 1, \{\alpha\}) = R(p, q) \cdot b_j(\alpha)$

We note that this process requires $T + 1$ iterations, and each iteration requires $O(|\mathcal{S}|^2)$ computations. Therefore, the total time required by the process is $O(|\mathcal{S}|^2 \cdot T)$. Thus, while the testing phase is more expensive than the training phase, it is to be noted that these requirements are only in terms of the CPU requirements, cand can easily be overcome for strings of modest length. We note that all the computations in the testing phase can be accomplished with the use of in-memory parameters which are estimated during the training phase. Therefore, no disk accesses are required during the test phase, which can be performed using an on-demand approach.

3. EXTENSIONS TO COMPLEX CLASSIFICATION

The model discussed in this paper can be easily extended to a number of different problems. For example, nested structures can be modeled with this approach. In this section, we will provide a brief overview of different kinds of models. First, we will discuss the case of direct nesting of substructures. We would like to create the markov model which is able to model the basic hierarchy in the nesting. More complex nesting patterns can then be replicated using this technique. Let us assume that the structure SE is embedded within the structure SS. In Figure 4, we have illustrated the markov model for the simple nesting of two structures. We note that there are four basic kinds of states in Figure 4 which we describe in order from bottom to top: (1) The lowest set of states corresponds to string positions outside either structure. (2) The set of states which are next to the lowest correspond to the string positions outside the outer structure SE, but within the inner structure SS and preceding SE. (3) The next set of states correspond to string positions within both SE and SS. (4) The topmost set of states correspond to string positions outside SE but succeeding SS.

In Figure 4, we have marked each set of corresponding states accordingly. While this is an example of a simple nest-

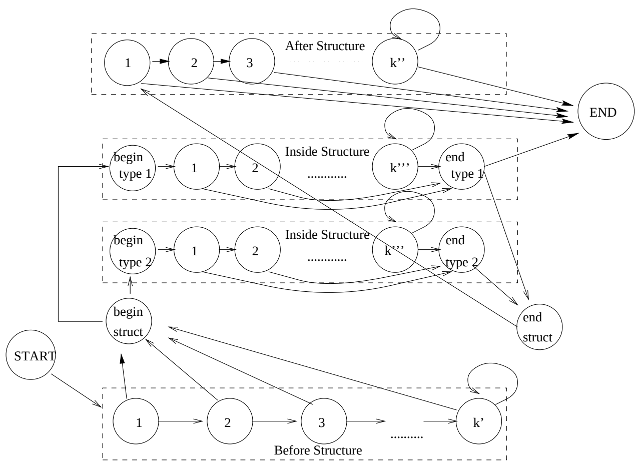


Figure 6: Multiple Labels (Example)

ing, the method can be easily extended to recursive nesting by using similar kinds of building blocks.

Often, there may be multiple substructures which may follow one another in the string. Such substructures are referred to as sequential substructures. In combination with the nesting approach discussed above, a powerful tool can be constructed for different categories of structures. In Figure 5, we have illustrated an example in which we have two sequential substructures which follow one another. Therefore, the figure shows five distinct sets of states which we have ordered from bottom to top in Figure 5: (1) The lowest sequence of states refers to the positions just before both structures. (2) The second sequence refers to the positions inside the first structure. (3) The third sequence refers to the positions inside the two structures. (4) The fourth sequence refers to the positions inside the second structure. (5) The final sequence refers to the positions after both structures.

In many cases, different kinds of substructures may be present in the same string. In such a case, only a minor modification is needed to the results of Figure 3. In this case, for the set of states inside the structure, we may have k different possibilities corresponding to the different classes that the structures may be drawn from. For example, in the Figure 6, we have illustrated the case when there are two kinds of classes in the strings. These two classes are labelled Type 1 and Type 2 respectively. When the substring belongs to the class of type 1, then this corresponds to a path through the markov model using the states between those labelled “begin type 1” and “end type 1”. In this case, the maximum probability path through the markov model also provides information on the identity of the class to which the corresponding substring belongs.

4. EXPERIMENTAL RESULTS

In order to test the effectiveness of the system, we used a number of synthetic and real data sets. We will show that the generalized classification method is not only effective, but is extremely efficient for very fast data streams.

4.1 Data Set Descriptions

Two classes of data sets and streams were generated by postprocessing synthetic and real data sources. These classes are as follows:

Data	**Base String**	**Embed. String**
MStream1	T20.I5.D1M	T10.I2.D1M
MStream2	T20.I4.D1M	T5.I2.D1M

Table 1: Market Basket Data Characteristics

Data Set	NNC	MMGC
WD1	14.3%	43.4%
WD2	15.7%	41.5%
MStream1	36.7%	75.4%
MStream2	34.6%	73.1%

Table 2: Classification Accuracy Results

(1) Market Basket Data Streams: The market basket data streams were generated in order to simulate the presence of different kinds of transactions within the same sequence. Such a situation may arise in unsegmented sequences from a variety of domains. These data sets were constructed by modifying a generator discussed in [3]. We will first discuss how to convert a transaction generated in [3] into a string format. Each transaction consisted of a string of sorted item ids from 1 to 100. Each of these item ids were randomly mapped onto a symbol from $\Sigma = \{\sigma_1 \ldots \sigma_l\}$ with $l = 100$. By replacing the item id with the corresponding symbol, a string was obtained using the symbols from Σ. In order to generate a string with a marker delimited substring portion, two separate strings were generated using different parameter values of the data generator of [3]. The second string was then embedded within the first string at a random position. The begin and end markers of the structure correspond to the beginning and ending positions of the embedded string. We shall refer to the first string as the *base string*, and the second string as the *embedded string*. Two separate data streams were generated, each corresponding to the different parameter values of the outer and embedded string. These data streams are referred to as MStream1 and MStream2 respectively. In Table 1, we have illustrated the underlying itemset distributions which were used to generate the base and embedded strings. The notations TxIyDz as indicated in Table 1 are directly adapted from [3]. As evident from the notation, 1 million (1 M) base and embedded strings were generated and were paired off to create 1M structurally embedded strings. Each such set was then divided in a ratio of 9 : 1 as the training and test data stream respectively. For the case of the market basket data streams, the training process was performed in parallel with the classification of the test data stream in order to test effectiveness.

(2) Web Access Data Sets: We wanted to test how well the system worked in finding embedded patterns of a temporal nature. In order to do so, we marked all accesses which occurred within a pre-specified time interval of the data in a stream of accesses. The original traces were obtained from [18]. Two sets of 30 traces (each corresponding to one day of access) were used. From each trace, we sampled 100 user accesses repeatedly in the same temporal sequence as the original data. These were then processed in order to convert them into string format. The strings were composed of the symbols drawn from $\{.com, .edu, .gov\}$, depending upon which web sites were accessed. By using this procedure, a total of 30,000 strings were generated from each set of 30

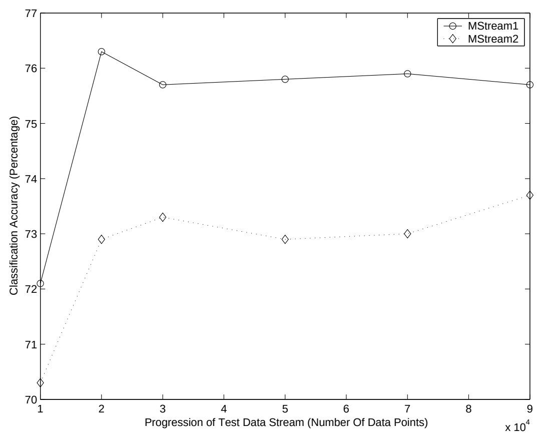


Figure 7: Effectiveness of Testing Phase with Data Stream Progression

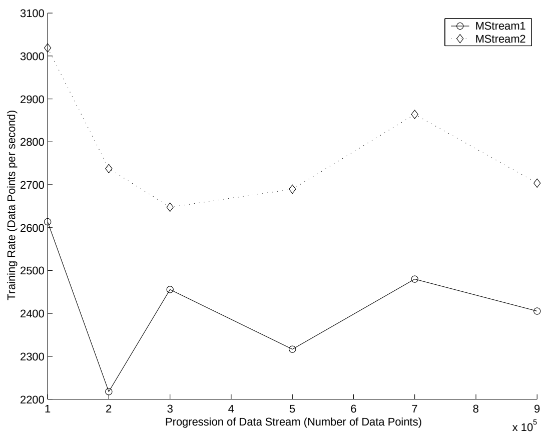


Figure 8: Scalability of Training Phase with Data Stream Progression

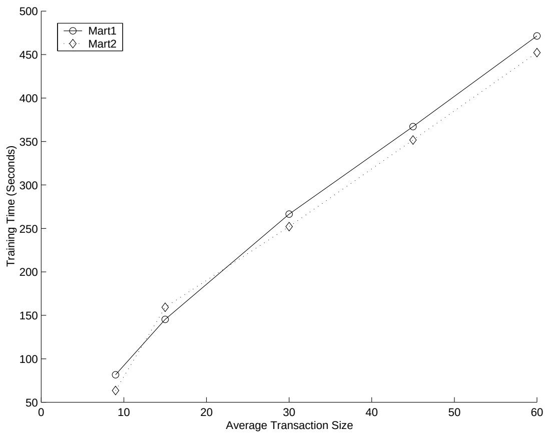


Figure 9: Scalability of Training Phase with Transaction Size

traces. All web page accesses occurring between 9:00 and 9:30 AM were marked in 90% of the strings. Two data sets labelled WD1 and WD2 were generated for the two different sets of traces.

4.2 Effectiveness Results

In order to test the effectiveness of the technique, we used the length of the intersection of the marked intervals between the true substructure and the substructure found by the algorithm. Specifically, if I_T be the true substructure, and I_F be the substructure found, then the accuracy $AC(I_T, I_F)$ is defined by $AC(I_T, I_F) = |I_T \cap I_F|/|I_T \cup I_F|$

One way of testing effectiveness was to use a generalization of a nearest neighbor classifier, which we will henceforth denote as NNC. In this technique all the substructures in the training data were stored. For a given test instance, all possible consecutive substructures of lengths within pre-defined thresholds were enumerated, and the closest substructure from the training data to any of these test substructures was used in order to define the classification markers. The measurements were computed using the edit distance. While such a system was severely limited in its scope for efficient classification of different kinds of complex structures, we will show that it does not perform as well as the markov model for even those cases where it is actually feasible to use it.

We have illustrated the accuracy of the two techniques in Table 2. It is clear that the hidden markov model based classifier significantly outperforms the nearest neighbor classifier in each case. This is because the brute force nearest neighbor classifier was unable to exploit the distributions of the symbols within the substructures. The accuracy of the nearest neighbor classifier was particularly poor in the case of the web access data sets. This is because the classification behavior was decided by deeply embedded temporal sequence fragments in the case of the web access data sets. In such cases, the brute force approach is unable to isolate the relevant subsequences for the purpose of classification. In the case of the market basket data streams, the quality of classification by both methods was less disparate, though the markov model technique did outperform the nearest neighbor classifier by an impressive margin in this case as well. In this case, the brute force wss partially able to isolate the relevant fragments, but was still not completely successful in the classification process. As a result, the hidden markov model technique continued to be significantly more effective.

We also tested the consistency of the classification process with progression of the data stream. In Figure 7, we have illustrated the classification accuracy with test stream progression for both the market basket data streams. In each case, it is clear that the accuracy of the classification process increased initially with data stream progression, but stabilized rapidly. The reason for this initial behavior is that the classification model takes a little while to construct a statistically robust model of the underlying data behavior. An interesting observation is that the improvement of the model to optimal accuracy requires only a few thousand records. Thus, the initialization time is relatively low. Furthermore, the consistency of the accuracy with stream progression shows that it is possible to maintain a stable and highly accurate classification process throughout the classification process.

4.3 Efficiency Results

We also tested the system for efficiency of the classification process. We have already noted that the process of model construction requires exactly one pass over the database in

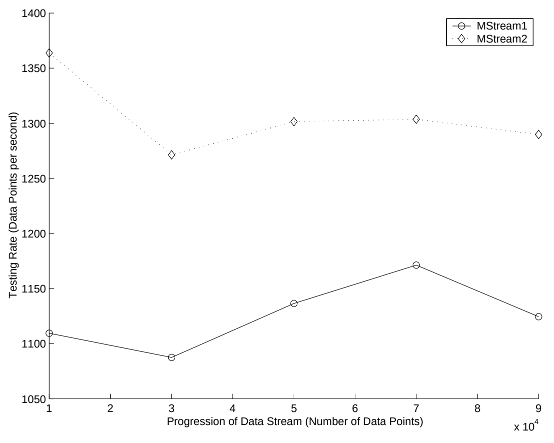


Figure 10: Scalability of Testing Phase with Data Stream Progression

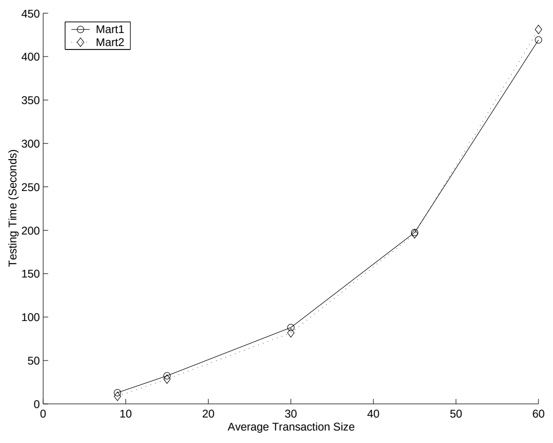


Figure 11: Scalability of Testing Phase with Transaction Size

each instance. In addition, it is useful to test the scalability of the model construction and testing phases with increasing string length. Specifically, we tested the following: (1) Scalability of model construction with progress of data stream. (2) Scalability of model construction phase with increasing string length. (3) Scalability of the testing phase with progress of data stream. (4) Scalability of the testing phase with increasing string length. The market basket data streams turned out to be quite useful for measuring the scalability of the technique, since it was easy to vary the average string length by modifying the parameters of the underlying transaction generation model [3].

In order to test the scalability of the testing phase with increasing string length, we generated two data sets which in which the parameter for transaction length was varied for the base and embedded strings. The base strings were denoted by Tx.I5.D1M and Ty.I4.D1M respectively according to the notations of [3]. The corresponding embedded strings were half the average lengths of the base strings. Thus, these transaction sets were T(x/2).I2.D1M and T(y/2).I2.D1M respectively. We refer to the corresponding (families of) generated strings as Mart1 and Mart2 respectively.

In order to test the scalability of the model construction phase with increasing training data size, we computed the rate of processing of the data points with progression of the data stream. The results are illustrated in Figure 8. On the X-axis, we have illustrated the progression of the stream in terms of the number of training data points. On the Y-axis, we have illustrated the processing rate in terms of the number of data points per second. It is clear that the training phase exhibited consistent processing performance throughout the progress of the data stream. For static databases, this also means that the algorithm illustrates linear CPU scalabality along with one database scan. The excellent efficiency of the model construction phase can be attributed to the effective design of the markov model topology for parameter estimation. We note that each parameter in the Markov model is estimated using simple and linearly additive operations. This makes the process easy to implement for a data stream. In each case, the algorithm was able to process thousands of data points per second.

Next, we tested the scalability of the training procedure with increasing string length. For this purpose, we utilized the two string database families Mart1 and Mart2 discussed earlier. The results for both families are illustrated In Figure 9. On the X-axis, we have illustrated the string length (including both the original string and the embedded string), whereas on the Y-axis, we have illustrated the running time for the entire training database of 900K records. While both datasets provided roughly comparable results (illustrating that the internal characteristics of the string mattered less than the string lengths), the primary observation was that the running times scaled linearly with transaction sizes. This is because of the number of states in the resulting markov model scaled linearly with string length. Thus, the time required for parameter estimation scaled linearly as well. The results on scalability of model construction are quite important, since many real domains contain very large databases with long strings. Therefore, these results validate the applicability of the generalized classification method to such problems.

Next, we tested the effectiveness of the testing procedure with progression of the data stream. The results for the data sets MStream1 and MStream2 are illustrated in Figure 10. On the X-axis, we have illustrated the progress of the test data stream in terms of the number of data points. On the Y-axis, we have illustrated the processing rate in terms of the number of data points per second. We note that the testing procedure slightly was slower than the training procedure, though it continued to process several thousand strings per second. We also tested the efficiency of the testing procedure with increasing string length. In Figure 11, we have illustrated the efficiency of the testing procedure with increasing string length. On the X-axis, we have illustrated the transactions size, whereas on the Y-axis, we have illustrated the testing time for the entire database of 100K test records. In the results of Figure 11, we found that while the behavior of the algorithm was super-linear, it was considerably better than the worst-case complexity of a dynamic programming algorithm. This was because of the fact that a large number of transition probabilities for a given model topology were implicitly set at zero. The corresponding paths could be ignored in the dynamic programming computation process. This greatly improved the efficiency of the testing procedure.

5. CONCLUSIONS AND SUMMARY

In this paper, we proposed a generalized classification model which in being able to identify particular fragments of the strings for the classification process. Our approach can support incremental model update and works well with very fast data streams. Such a technique has applicability in a number of practical scenarios such as text analysis, DNA mining, and web analysis. Standard classification methods cannot be applied easily to this case because of the additional complexity of finding the embedded substructures. We propose an approach based on hidden markov models for this problem, and make it scalable for very large databases. The results were tested over a number of large data sets and data streams, and were found to be qualitatively effective in addition to being very efficient from a computational point of view.

6. REFERENCES

[1] C. C. Aggarwal. On Effective Classification of Strings with Wavelets. *ACM KDD Conference*, 2002.

[2] C. C. Aggarwal, J. Han, J. Wang, P. S. Yu. On Demand Classification of Data Streams. *ACM KDD Conference*, 2004.

[3] R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules. *VLDB Conference*, 1994.

[4] G. J. Barton. Protein Multiple Sequence Alignment and Flexible Pattern Matching. *Methods in Enzymology*, 183, 403-428, 1990.

[5] G. A. Churchill. Stochastic Models for Heterogeneous DNA Sequences. *Bull. Math Biol*, 59, pp. 79-91, 1989.

[6] M. Deshpande, G. Karypis. Evaluation of Techniques for Classifying Biological Sequences. *Technical report, TR 01-33, University of Minnesota*, 2001.

[7] P. Domingos, G. Hulten. Mining High-Speed Data Streams. *ACM SIGKDD Conference*, 2000.

[8] D. Gusfield. Algorithms on Strings, Trees and Sequences. *Press Syndicate of the University of Cambridge*, 1997.

[9] D. Haussler, A. Krogh, I. S. Mian, K. Sjolander. Protein Modeling using Hidden Markov Models: Analysis of Globins. *Technical Report UCSC-CRL-92-23, University of California at Snta Cruz Computer Science*, Santa Cruz, CA, 1992.

[10] D. Haussler, A. Krogh, I. S. Mian, K. Sjolander. Protein Modeling Using Hidden Markov Models: Analysis of Globins. *Proceedings of the Hawaii International Conference on System Sciences*, IEEE Computer Society Press, 1993.

[11] G. Hulten, L. Spencer, P. Domingos. Mining Time-Changing Data Streams. *ACM KDD Conference*, 2001.

[12] K. Karplus, C. Barrett, R. Hughey. Hidden Markov Models for Detecting Remote Protein Homologies. *Bioinformatics*, 14(10): pp. 846-856, 1998.

[13] A. Krogh, M. Brown, T. S. Mian, K. Sjolander, D. Haussler. Hidden Markov Models in Computational Biology: Applications to Protein Modeling. *University of California Technical Report UCSC-CRL-93-32*, 1993.

[14] V. Borkar, K. Deshmukh, S. Sarawagi. Automatic Segmentation of Text into Structured Records. *SIGMOD Conference*, 2001.

[15] W. R. Taylor. The Classification of Amino Acid Conservation. *Journal of Theoretical Biology*, 119, 205-218, 1986.

[16] A. J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm. *IEEE Transactions on Information Theory*, IT-13, 260-269, 1967.

[17] M. S. Waterman. Sequence Alignments. *Mathematical Methods for DNA Sequences*, Waterman M. S. ed.. CRC Press, 1989.

[18] ftp://ircache.nlanr.net/Traces/