Chapter 1

Introduction

1.1 What is system classification?

With the dramatic development of computer science and technology, we are on the edge of making many machines intelligent by embedding computer systems in them. For example, people have known how to cook rice for thousands of years, but only in the last two decades was the neuro-controlled automatic rice cooker invented. In the near future, by embedding computer chips in other kitchen devices, people will be further liberated from the tedious and exhausting cooking tasks which their predecessors have suffered for many centuries. Similar things will also happen to vehicles. In next century, we expect cars will become autonomous. Once the passengers tell the vehicle where to go, they can go to sleep or watch television. In the short term, cars will become smarter, if not completely autonomous. The smart car's intelligence has many aspects, including the ability to tell if the human driver's sobriety level is good enough for further operation. If necessary, the monitoring system may warn the driver to stop for a break. This is important because inattention may lead to the fatal accidents. In the U.S. 1996, there were over 37,000 automobile accidents involving fatalities, in which 42,000 people were killed. Among these cases, over 21,000 were single vehicle accidents resulted in 22,500 fatalities [Batavia, 98].

The technique we explore in this thesis is useful for driving sobriety monitoring, as well as other applications. Let's imagine that we have a vehicle full of smart sensors which can tell the velocity of the vehicle, its orientation, its lateral distance to the center of road, and the distances to the other vehicles nearby, etc. If we regard a driver as a system, the above variables are the inputs to the system. Based on the inputs, the driver has to properly steer and control the gas and brake pedal. Thus the outputs of the system are the vehicle's steering angle and its acceleration. Suppose we also measure the outputs. Let's take a record of both the input and output values every time unit, say 0.1 second. We will get a multi-dimensional time series. The driving time series varies from case to case, even if the driver is the same person and his/her sobriety condition is identical. The reason is that road conditions and traffic may be different, and these differences will make the driver's response (system outputs) differ from case to case. However, we believe if the driver is sober, his driving behavior time series should be consistent with his historic "sober" driving time series. Otherwise, if the driver is intoxicated, his driving (system outputs) may differ from those normal cases in memory. In addition, an intoxicated driver may create some unusual input scenarios because of his careless behavior.

How can we formalize the informal discussion above into a useful and reliable algorithm? In statistical terms, to classify the driving style we want to calculate $Prob(S_{normal} \mid \boldsymbol{O}_q)$, which is the probability that a driver's sobriety is normal, as inferred from the observation of their driving behavior. $\boldsymbol{O}_q$ represents the current driver's driving behavior time series; q stands for *query*, implying that the underlying state of the driver's sobriety condition is unknown. S_{normal} is the event of the driver being sober. To calculate $Prob(S_{normal} \mid \boldsymbol{O}_q)$, we compare the unclassified time series $\boldsymbol{O}_q$ with those time series in memory generated by the same driver when he was sober. If $Prob(S_{normal} \mid \boldsymbol{O}_q)$ is higher than a certain threshold, the driver seems to be sober. Otherwise, they are intoxicated. Sometimes, the task can be more complicated. For example, the police department may want to distinguish drowsiness from drunkenness. In this case, we should calculate $Prob(S_{intoxicated} \mid \boldsymbol{O}_q)$ or $Prob(S_{drowsy} \mid \boldsymbol{O}_q)$, as well as $Prob(S_{normal} \mid \boldsymbol{O}_q)$, the largest value indicates the driver's most likely sobriety condition.

Generally speaking, we define the task of a *system classifier* as the following: given a set of observations of a system's inputs and outputs, a system classifier is to figure out the underlying mechanism which generates these observations.

1.2 The applications of system classification

- System diagnosis:

  No machine can work perfectly all the time. People need to know when to fix the machines and how to fix them. This is the purpose of system diagnosis. System diagnosis can be done by human experts. However, in some cases an on-line autonomous system diagnosis tool is preferred, because for some complicated machines, no single human expert can understand every detail. Also, it is hard to ask the human expert (or a group of them) to do the diagnosis job twenty-four hours a day, seven days a week, in all possible situations including dangerous environments.

- Surveillance

  With the progress of video tracking and speech signal processing, we are on the edge of implementing an autonomous system to liberate human operators from surveillance jobs which may be tedious and last long hours. We expect that these autonomous systems will have better performance than that of a sleepy human operator. Similarly, we expect to apply this technique to make some military surveillance devices more intelligent. For example, we can invent an automatic radar monitoring system so that the soldiers can be liberated from the radar desk, especially during the tedious period when nothing unusual happens.

- Human behavior monitoring

  Every year in the U.S., thousands of people die in traffic accidents. Some of these accidents are caused by the exhaustion of the drivers. It would be desirable to have a way to monitor the behavior of the human operators and give them warnings if necessary.

Another possible application is that with the booming of virtual reality stores on the Internet, more and more customers will go shopping via the Internet. Technically the e-stores' server is capable of tracking the behaviors of the visitors, to detect the customers' purpose and/or preference. This prospect does raise many moral, ethical, and social issues which are beyond the scope of this thesis.

- Human skill transition and evaluation

  Sometimes people want to learn physical skills from the masters. Some skills should be passed on before the old masters die. Some skills should also be transferred to robots, because robots can work in remote or inhospitable environments. Therefore, we need some ways to transfer skills and evaluate the learned performance.

- Financial monitoring

  We can apply the techniques of this thesis to keep an eye on the financial climate, which is useful and rewarding.

1.3 The assumptions of OMEGA

In this thesis, we investigate and extend memory-based learning for general propose on-line system classification. We name this new technique On-line MEmory-based GenerAl purpose system classifier, (OMEGA). OMEGA calculates $Prob(S_p \mid \boldsymbol{O}_q)$, which is the probability that the underlying mechanism of a set of observations $\boldsymbol{O}_q$ is system S_p. It has following the assumptions:

1. OMEGA does not approximate the closed-form mechanism of the underlying system. We also assume that the unknown underlying generator of $\boldsymbol{O}_q$ must be one of a *finite* set of candidate systems. This assumption is not so bad as it looks. For the example mentioned above, it is unnecessary to require every police officer to know the psychological and physiological processes underlying intoxication. Instead, if a traffic police officer can cor-

rectly detect any unusual driving behavior, his job is well done.

2. For the same example, to calculate the probability $Prob(S_{normal} \mid \boldsymbol{O}_q)$, we compare the query driving time series $\boldsymbol{O}_q$ with those "sober" driving time series in memory. In other words, we assume that we have collected some training observations of each candidate system's behavior before the classification job for $\boldsymbol{O}_q$ comes. Notice that if there are only a few sober driving time series samples in memory, it is still possible to approximate $Prob(S_{normal} \mid \boldsymbol{O}_q)$. Of course, the fewer the sober samples in memory, the less reliable the approximated $Prob(S_{normal} \mid \boldsymbol{O}_q)$ is.

3. Originally motivated to classify time series, our research ends up with a general purpose technique which is also capable of general pattern classification. In other words, the observation $\boldsymbol{O}_q$ may be a time series, but this is not necessary. As defined, $\boldsymbol{O}_q$ is in fact a set of observation data points, while a data point consists of the inputs of the concerned system at a certain time instant and their corresponding outputs. When $\boldsymbol{O}_q$ is not a time series, we can shuffle its data points randomly.

4. OMEGA works best for those systems whose input and output are fully observable, and the output are fully determined by the input. Note that this assumption is often violated in practice. For example, in driving domain, a driver's control action may be influenced by some of his hidden psychological and physiological factors. However, like other machine learning methods, we assume a driver control action is somehow predictable by some observable input variables.

5. The inputs and outputs of any candidate systems can be of any type. They can be continuous or discrete, (including categorical), or even a mixture of the two. However we assume the types of the input and output of all candidate systems are the same.

6. We study stochastic systems; in other words, given a certain input, the corresponding output is stochastic. The conditional distribution of the output given a certain input can be of any type. For some systems, the outputs corresponding to an identical input may scatter

around a center, so that the conditional distribution can be roughly formed as Gaussian. However, as a general purpose approach, OMEGA does not require this uni-modal assumption.

1.4 Related fields

Conventionally, classification is to detect to which category a single data point belongs. However, since a time series consists of a sequence of data points, system classification involves a sequence of classifications, then summarize them so as to draw an overall conclusion.

System classification is different from *system identification*. The latter estimates the configuration and the parameters of an unknown system, but system classification's task is to recognize an unknown system, without necessarily estimating its parameters.

Another closely related field is *fault detection*, which is also referred to as *novelty detection*. The task of fault detection is to tell whether or not a system's current behavior is out of the tolerance of its normal performance. System classification is different from fault detection because system classification concerns multiple systems, and it assumes that every system always works normally. The difficulty of fault detection is that its training data is usually unbalanced; in other words, the majority of the training data is collected when the system works normally. However, it is still straightforward to apply OMEGA to solve the fault detection problem: we approximate $Prob(S_{normal} \mid \boldsymbol{O}_q)$, if this probability value is lower than a certain threshold, the system is abnormal; in another case, even if the value of $Prob(S_{normal} \mid \boldsymbol{O}_q)$ is higher than the threshold, but it is not reliable (its confidence interval is too large), the state of the system is uncertain. The threshold can be decided by hypothesis testing methods.

1.5 The system classification approaches

There are two approaches to system classification: comparing the system parameters, or comparing the predictions.

Comparing the system parameters

This approach is similar to system identification: we approximate the unknown system's parameters first, then classify the system based on the comparison of the system parameters. For example, suppose we have a collection of observations $(x_1, y_1), (x_2, y_2), ..., (x_T, y_T)$, where x's are the system's inputs, and y's are the outputs. Temporarily, let's assume based on prior knowledge that we know these signals were generated by a linear system:

$$y = \beta_o + \beta_1 x + \xi$$

If there are sufficient observations, we are able to approximate the system parameters, β_0 and β_1. To detect if the observation signals $(x_1, y_1), (x_2, y_2), ..., (x_T, y_T)$ were generated by a particular one-input-one-output linear system whose parameters are α_0 and α_1, we can straightforwardly check if the α's and β's are close to each other respectively.

This approach looks simple, but it has three problems: (1) We need the prior knowledge of the closed-form formula of the system. (2) Before we employ this approach, we should make sure that identical systems must have the same parameters. When the system is more complicated than a linear one, different sets of parameters may correspond to the same system. Section 1.7 gives an example.

In some circumstances like chemical manufacturing process, it is hard to get precise mathematical models of the systems. Therefore, to design a robust, general purpose system classification package, we will resort to the other approach.

Comparing the predictions

Given a set of observations whose underlying generator is unknown, the prediction approach temporarily assumes the unknown underlying system is a certain candidate one. Based on our knowledge of this assigned candidate system, we can predict the outputs corresponding to the inputs of the observations. If the candidate system is indeed the real underlying system, the predictions must be close to those observed outputs. Otherwise, the assumption is not correct.

In more details, let's suppose there is a collection of observations, $(x_1, y_1), (x_2, y_2), \ldots, (x_T, y_T)$. To figure out whether or not they were generated by a certain linear system,

$$y = \alpha_o + \alpha_1 x + \xi$$

with particular α_0 and α_1 values, we can use the above formula to predict the y value given a certain x. Therefore, we will get a sequence of predictions, $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T$. The difference between them and the observed values $y_1, y_2, \ldots, y_T$ are the residuals. If the residuals are close to zero, the system with α_0 and α_1 as parameters is likely to be the underlying system which generated the observations.

Even with only one observation, the prediction-based approach can still start to work, though the result will be unreliable. With more observations, this approach can be expected to have improved performance. Therefore, the prediction-based approach is ideal for on-line applications.

Up to now, we have assumed the system is linear. The linear system model has been popular for several decades because it is simple and in many cases it is reasonable. For non-linear systems, we can apply non-linear function approximators such as neural network to do the prediction job, so that the prediction-based system classification approach still works [Petridis et al., 96].

The neural prediction approach uses neural networks to approximate every candidate system. If there are one hundred candidate systems, there will be one hundred neural networks. To calculate $Prob(S_p \mid \boldsymbol{O}_q)$, we compare the outputs of $\boldsymbol{O}_q$ with the predictions of the neural net, which represents S_p, given the corresponding inputs.

Although a neural classifier is capable of starting its job to detect the unknown underlying generator of $\boldsymbol{O}_q$ with very few data points in $\boldsymbol{O}_q$, we should clarify that it does need a large amount of training data to train the neural net to precisely represent the candidate system, say S_p. The training data are collections of observations similar to $\boldsymbol{O}_q$, but they are labeled by their underlying systems, say S_p.

There are three concerns with a neural prediction-based system classification approach. (1) It is computationally expensive to train a neural network. Things become worse when new training data is constantly becoming available. (2) Even if we can afford a supercomputer which is capable of updating the neural networks quickly, we will have another trouble: *interference*. The neural networks will evolve to fit the new data, and the old data will eventually lose their impact. (3) Every candidate system's neural network, should be included in the competition, until there is convincing evidence that a certain candidate's neural net is less competitive. Therefore, when there are a huge number of candidates, the computational cost becomes prohibitively expensive, especially in the early stage when all the candidates are involved in the process.

To overcome these problems, the memory-based learning approach is a good choice. A memory-based learning system stores all the training data in memory. When new data arrive, they will be stored into the memory together with the old data. All processing of the training data is deferred until a prediction query is made. Therefore, less interference happens. Second, as we will introduce in the later chapters, the memory-based learning methods do not require any parametric model of the system. Hence, there is no model which needs to be trained off-line. Third, by reorganizing the memory in kd-tree form and caching some information into the tree nodes, the memory-based learning process can be done very quickly. Fourth, also with the help of kd-tree, we can focus on the more promising candidates from the very beginning.

1.6 Thesis outline:

The thesis research consists of four parts: (1) The top-level principle of OMEGA, which is to combine a series of classifications in the context of likelihood analysis and hypothesis testing. (2) A new memory-based classifier, which has many improvements over existing classifiers. (3) Efficient memory information retrieval and regression using the cached kd-tree technique. (4) Cross-validation for feature selection and parameter tuning. Although (2) (3) (4) are three independent research topics, they act as components in the OMEGA approach.

Chapter 2 introduces the principle and framework of OMEGA to give the readers a birds-eye view of the whole approach and the relationship of the various components. As a demonstration, in Chapter 3 we use OMEGA to classify different styles of tennis playing, and compare OMEGA's performance with those of other methods. From Chapter 4 to Chapter 7, we discuss the components of OMEGA in details. Chapter 4 explores the new memory-based classifier, and compares it with other classification methods. In Chapter 5 and 6, we discuss a technique to re-organize the memory so as to improve the efficiency of information retrieval and regression. In Chapter 7, we talk about cross-validation, which is useful for feature selection and parameter tuning for the learning process. After that, we combine all the techniques into the OMEGA toolkit, and apply it to classify different driving styles, using both simulation data and real world data, referring to Chapter 8 and 9. Finally, Chapter 10 is a summary of all the research work, the contributions, and the open questions.

Figure 1-1 illustrates the structure of OMEGA system and the organization of the thesis.

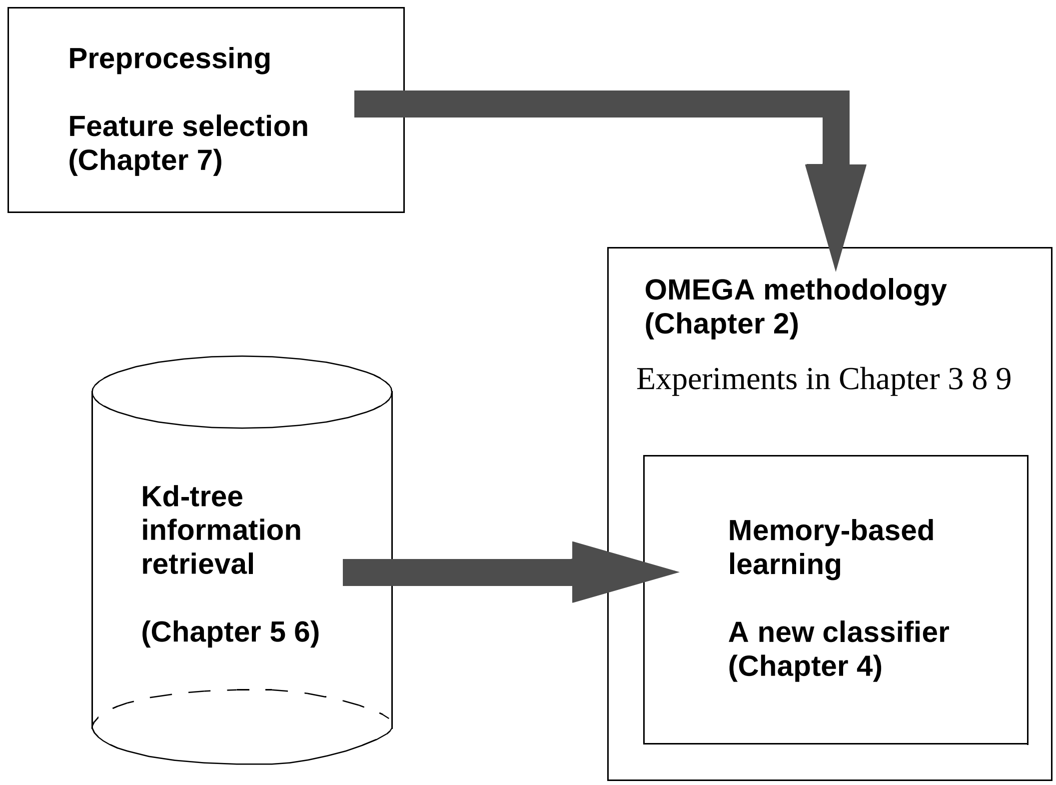


Figure 1-1: The structure of OMEGA system and the organization of the thesis.

1.7 *[1]: Hidden Markov Model (HMM)

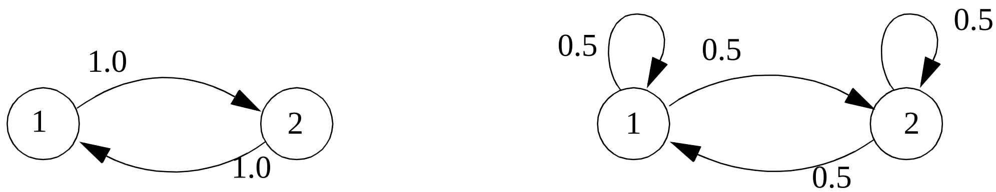
HMMs have been widely accepted as a time series analysis tool. They stand between the parameter comparison approach and the prediction approach. On one hand, it approximates the parameters of the hidden Markov model; on the other hand, it use a method similar to the prediction approach to evaluate whether or not two hidden Markov models with different parameters are in fact identical. There is no doubt HMM is an important and interesting technique, but it is questionable if it is a robust, general purpose system classification tool.

Before we argue the reasoning of our conclusion, let's give a brief introduction to HMM. HMM assume that a system has some internal hidden states. As time passes, the system jumps from

1. This section can be skipped if the reader does not have much interest in HMM.

Hidden State	Observation A	Observation B
1	0.5	0.5
2	0.5	0.5

Hidden State	Observation A	Observation B
1	1.0	0.0
2	0.0	1.0



An observation sequence: A A B A B A B B A B A B A A B B. The above two models have the same chance to be the generator of the observation sequence.

Figure 1-2: Two identical HMMs

one internal state to another. Each hidden state generates an observable signal, but it is possible that one state has several possible signals, and the same signal may be shared by several hidden states. The observation time series generated by a HMM is stochastic in two aspects: (1) The jumps are stochastically decided by the transition probabilities among the hidden states. (2) Even for the same hidden state, we may observe differing signals. Two two-state HMMs are illustrated in Figure 1-2. The numbers attached to the arc links are the transition probabilities. Since all the transition probabilities in Figure 1-2 (a) are 1.0, the system definitely switches its hidden state every time step. The system of Figure 1-2 (b) has a 50% chance to stay in the same hidden state, but has the other 50% chance to switch. The tables above the diagrams indicate the probabilities linking the hidden states to the observations, A and B.

If two time series are different, the underlying HMMs' parameters must be distinguishable. The HMM parameters include the transition probabilities and the probabilities linking the hidden states to the observations.

However, notice that an identical system may have a different structure and parameters. The system of Figure 1-2 (a) is in fact equivalent to that of Figure 1-2 (b), because both systems have exactly the same chance to generate the observation sequence written in Figure 1-2. Therefore, to detect if two HMMs are equivalent, we cannot simply compare their parameters. Instead, we should use the first HMM to generate a sample observation sequence, then find a way to measure how well the sample observation sequence fits the second HMM.

HMM were originally explored by the speech recognition community. For speech, there is no input, all the signals can be regarded as outputs. To extend HMM to systems which have both inputs and outputs, one solution is to enumerate every possible combination of input and output as a state. Thus, the number of states explodes as the number of possible input and output values increases. Therefore, in our opinion, HMMs did not easily fit our tastes.