Chapter 2

Memory-based System Classification

In this chapter, we study the methodology of the On-line MEmory-based GenerAl purpose system classification technique (OMEGA). OMEGA combines a series of classifications in the framework of likelihood analysis and hypothesis testing. In this chapter, we will introduce likelihood analysis and hypothesis testing first, then discuss efficiency issues. Afterwards, we will summarize pre-processing method and briefly discuss alternative memory-based classification and prediction methods.

2.1 Likelihood Analysis

As defined in the last chapter, a system classifier should estimate the underlying generator of a set of observation signals $\boldsymbol{O}_q$, under the assumption that the generator must be one of a finite number of candidate systems, $S_1, ..., S_n$. For example, given a time series of a vehicle's behavior in traffic, $\boldsymbol{O}_q$, the task of system classification is to tell the sobriety state of the driver, S_p, assuming that we have sufficient knowledge of the behavior of sober drivers, sleepy drivers and even intoxicated ones.

Average residuals

If we treat a driver as a system, the outputs are the control actions of the driver: the positions of the steering wheel and the gas and brake pedals. A driver chooses his control actions accord-

ing to the state of the vehicle, the road condition and the traffic condition, as well as his previous actions, hence the inputs of the system are the speed of the vehicle, its orientation, its distance to the center of the road, the road curvature, the distances from the vehicle to the neighboring ones in traffic, as well as the driver's previous control of the steering angle and the gas/brake throttle. Usually an observation sequence consists of a series of input-output data points $\{x_{qi}, y_{qi}\}$. Temporarily let's assume that at any time instant, the output y_{qi} is fully controlled by the input x_{qi}. We will come back to this topic in Section 2.4.

We do not know which candidate system generated the observation sequence $\boldsymbol{O}_q$, but let's guess it is the first system, S_1. Assuming somehow we have sufficient knowledge about S_1, so that given a specific input x, we can predict the output $\hat{y}|S_1$. Since $\boldsymbol{O}_q$ consists of a series of data points $\{x_{qi}, y_{qi}\}$, $i = 1, ..., N_q$, if we pick up one input x_{qi} from them, we can predict the corresponding output, $\hat{y}_{qi}\big|S_1$. If S_1 is indeed the real underlying generator of $\boldsymbol{O}_q$, $\hat{y}_{qi}\big|S_1$ is expected to be close to the observed output, y_{qi}. In other words, the smaller the residual between $\hat{y}_{qi}\big|S_1$ and y_{qi}, the more likely the unlabeled data points $\{x_{qi}, y_{qi}\}$, $i = 1, ..., N_q$, were generated by S_1. If there are N_q data points in $\boldsymbol{O}_q$, we will get N_q such residuals. We can use the average of these residuals as a measure of the likelihood.

If there are n candidate systems, we can calculate n such averages of residuals. The smallest one indicates the particular candidate system which is most likely to be the generator of the unlabeled data points, $\{x_{qi}, y_{qi}\}$, $i = 1, ..., N_q$, or equivalently, the observation sequence $\boldsymbol{O}_q$.

The average residual is a useful metric, but it treats every residual equally. This is not desirable, because some $\hat{y}_{qi}\big|S_p$'s have better quality than the others, and they should therefore have stronger impact on the likelihood measurement. Hence, we explore the likelihood approach in next subsection.

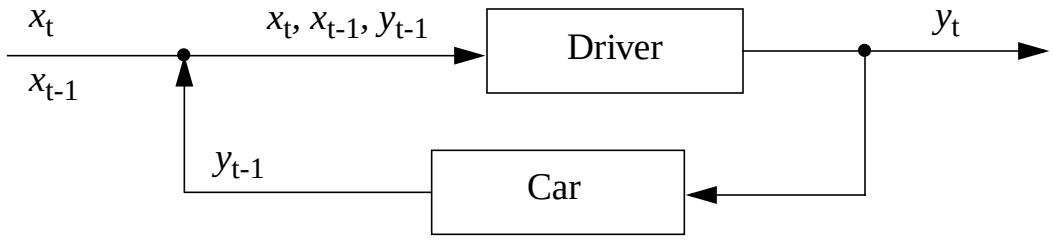

Figure 2-1: A driving system with one delay and feedback.

Likelihood

From the Bayesian point of view, the system classification problem can be structured as calculating $P(S_p \mid \boldsymbol{O}_q), p = 1, ..., N_s$, which is the probability that given an observation sequence $\boldsymbol{O}_q$, the underlying generator is the p'th candidate system. The biggest $P(S_p \mid \boldsymbol{O}_q), p \in \{1, \ldots, N_s\}$, indicates the most likely system which generated $\boldsymbol{O}_q$.

According to Bayes rule, $P(S_p \mid \boldsymbol{O}_q)$ is proportional to $P(\boldsymbol{O}_q \mid S_p)$, when the prior probability $P(S_p)$ is fixed. Let's assume $\boldsymbol{O}_q$ can be transformed into a set of data points, $\{x_{qi}, y_{qi}\}, i = 1, ..., N_q$, so that temporal order is not important. If so, the following equations hold:

$$P(\boldsymbol{O}_q|S_p) = \prod_{i=1}^{N_q} P(x_{qi}, y_{qi}|S_p) = \prod_{i=1}^{N_q} P(y_{qi}|S_p, x_{qi})P(x_{qi}|S_p) \qquad (2\text{-}1)$$

However, temporal order is important for most system because of the system's delays and feedback. Figure 2-1 illustrates a symple driving system with one delay and feedback. For such a system, $P(y_{qi} \mid S_p, x_{qi})$ in Equation 2-1 should be changed to $P(y_{qi} \mid S_p, x_{qi}, x_{q,i-1}, y_{q,i-1})$, because the current system output is not only determined by the input at the moment, but also the delay $x_{q,\,i-1}$ and the feedback $y_{q,\,i-1}$. To be convenient, we use X_{qi} to represent the conjunction of x_{qi}, $x_{q,i-1}$ and $y_{q,i-1}$. $P(x_{qi} \mid S_p)$ should be changed to $P(X_{qi} \mid S_p, X_{q,i-1})$. The reason for the appearance of $X_{q,i-1}$ is that the two components of X_{qi}: $x_{q,i-1}$ and $y_{q,i-1}$, may be partially dependent on

their ancestors: $x_{q,i-2}$ and $y_{q,i-2}$. Theoretically, $P(X_{qi} \mid S_p)$ is no bigger than $P(X_{qi} \mid S_p, X_{q,i-1})$; however, in practice, we find that in many cases that we can substitute $P(X_{qi} \mid S_p)$ for $P(X_{qi} \mid S_p, X_{q,i-1})$, and the classification results are still satisfactory. Therefore, for a system with one delay and feedback, the following equations hold:

$$P(\boldsymbol{O}_q|S_p) = \prod_{i=1}^{N_q} P(y_{qi}|S_p, X_{qi})P(X_{qi}|S_p, X_{q,i-1}) \approx \prod_{i=1}^{N_q} P(y_{qi}|S_p, X_{qi})P(X_{qi}|S_p)$$

(2-2)

Therefore, to calculate $P(S_p \mid \boldsymbol{O}_q)$, an approach is to approximate $P(X_{qi} \mid S_p)$ and $P(y_{qi} \mid S_p, X_{qi})$. To explain their physical meanings, let's study the driving domain again. Suppose we want to distinguish a certain driver's different driving behaviors under two sobriety conditions: sober and intoxicated. We notice that corresponding to the same scenario, X_{qi}, the driver's response when he is intoxicated tends to be different from that when he is sober; in other words, facing a certain situation X_{qi}, the probability that the driver gives a certain response y_{qi} while he is intoxicated, i.e. $P(y_{qi} \mid S_{intoxicated}, X_{qi})$, may be different from the probability when he is sober, i.e. $P(y_{qi} \mid S_{sober}, X_{qi})$. Therefore, we believe that the probability $P(y_{qi} \mid S_p, X_{qi})$ is a good metric of the driver's sobriety condition.

Besides, we also notice that an intoxicated driver may encounter situations which are not familiar to him when he is sober. For example, an intoxicated driver may let his car be very close to other vehicles in traffic, but when he is sober, the driver may realize that this situation is so dangerous that he would try to avoid it. In other words, the probability that a sober driver encounters a certain scenario X_{qi}, i.e. $P(X_{qi} \mid S_{sober})$, may be different from the probability that he faces the same situation when he is intoxicated, i.e. $P(X_{qi} \mid S_{intoxicated})$. Notice that if a sober driver intentionally does something new, our system classifier may misunderstand him as being drunk. But, we do not have to blame our system classifier for that. Tom Hanks' performance in Forrest Gump is highly appreciated. Why? Because Tom mimicked the dummy's behavior

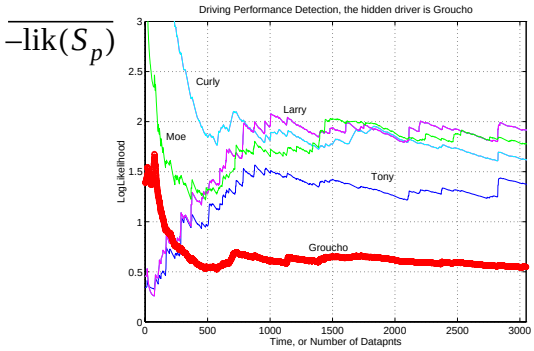

Figure 2-2: The X-axis is the number of observation data points. The Y-axis is the average of the negative log likelihood. To find the underlying system, one should compare the tails of the curves. Because Groucho's tail is closest to the X-axis, Groucho is most likely the underlying generator of the observation sequence.

seamlessly. Hence, we believe that by combining the two probabilities, $P(X_{qi} \mid S_p)$ and $P(Y_{qi} \mid S_p, X_{qi})$, we can have a good chance to distinguish the different underlying mechanisms, S_p, $p = 1, 2, ..., N_s$.

Let's assume we know how to approximate $P(X_{qi} \mid S_p)$ and $P(y_{qi} \mid S_p, X_{qi})$. The details will be covered by Section 4. To make the computation more convenient, usually we calculate *the average of the negative log likelihood* instead of $P(\boldsymbol{O}_q \mid S_p)$. The average of the negative log likelihood is defined as:

$$\begin{aligned}\overline{-\text{lik}(S_p)} &= -\frac{1}{N_q}\log(P(\boldsymbol{O}_q|S_p)) \\ &= -\frac{1}{N_q}\log\prod_{i=1}^{N_q} P(y_{qi}|S_p, X_{qi})P(X_{qi}|S_p) \\ &= -\frac{1}{N_q}\sum_{i=1}^{N_q}\log P(X_{qi}|S_p) - \frac{1}{N_q}\sum_{i=1}^{N_q}\log P(y_{qi}|S_p, X_{qi})\end{aligned} \tag{2-3}$$

Notice $\overline{-\text{lik}(S_p)}$ is a positive real number, because both $P(X_{qi} \mid S_p)$ and $P(y_{qi} \mid S_p, X_{qi})$ are between *0* and *1*.

For the example in Figure 2-2, we were given a sequence of unlabeled observations of the driving behavior. The driver is unknown, but he must be one of five candidates: Tony, Larry, Curly,

Moe and Groucho. Using all the 3,150 unlabeled data points, we calculated five averages of the negative log likelihood $\overline{-\text{lik}(S_p)}$, p = 1, ..., 5. Since OMEGA is an on-line approach, the 3,150 data points were not available at the early stage, we also study the $\overline{-\text{lik}(S_p)}$ with fewer data points. Therefore, we have five curves in the picture, the *X*-axis is the number of data points involved in the calculation of $\overline{-\text{lik}(S_p)}$, the *Y*-axis is $\overline{-\text{lik}(S_p)}$. At the very beginning, the values of $\overline{-\text{lik}(S_p)}$ are not reliable, because they were calculated using only a few data points; but with more and more data points involved, the $\overline{-\text{lik}(S_p)}$ curves become more consistent. The tails of the curves (to the right extreme) are the $\overline{-\text{lik}(S_p)}$ based on all the 3,150 data points. Among the five tails, the one closest to the horizontal axis indicates the generator of the observation sequences. In Figure 2-2, Groucho's tail is closest to the *X*-axis, thus Groucho seems to be the unknown driver.

2.2 Hypothesis Testing

Closely looking at the picture, $\overline{-\text{lik}(S_p)}$ of Groucho at the tail is 0.53, while that of Tony is about 1.40. Since 0.53 looks significantly smaller than 1.40, we claim that Groucho, not any other operator, seems to be the unknown driver.

However, we are not always lucky enough to be able to assign a unique candidate system to be the generator of $\boldsymbol{O}_q$. It is possible that more than one candidate's curves so close to each other that it is hard to tell which one is more likely to be the underlying generator. In Figure 2-3, Larry and Tony's tails are very close to each other. Larry's $\overline{-\text{lik}(S_p)}$ is 1.19, while Tony's is 1.21. Although Larry is a bit closer to the horizontal axis than Tony, we do not want to stake too much on Larry to be the only probable operator. Instead we say that the observation sequence $\boldsymbol{O}_q$ is confusing. It is important to distinguish the confusing situation from the exclusive one; because if the situation is confusing and we appoint a unique operator, we may end up with a severe mistake.

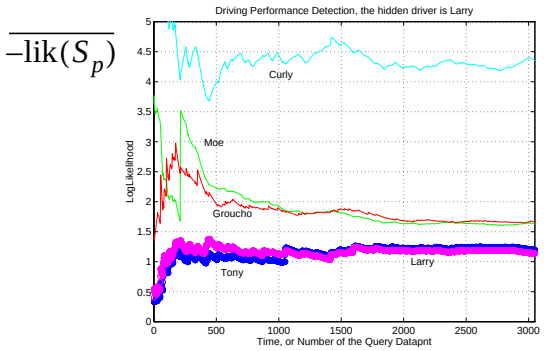

Figure 2-3: A confusing case. Since Larry and Tony's curves, especially their tails to the right extreme, are so close, that it is hard to tell which one is more likely.

To strictly define a confusing situation, we need a threshold. If the gap between the lowest tail and the second lowest one is beyond the threshold, the unique generator is easy to decide; otherwise, the situation is confusing. A difficulty arises in that there does not exist a fixed threshold applicable to any domain. For different domains, $\overline{-\text{lik}(S_p)}$ are of differing scales, resulting in different thresholds. Therefore, we resort to the statistical *two sample hypothesis testing* method [Devore, 91]. For two candidate systems S_{p1} and S_{p2}:

1. We calculate the *Z*-test value from statistics,[1]

$$Z = \frac{(\overline{-\text{lik}(S_{p1})}) - (\overline{-\text{lik}(S_{p2})})}{\sqrt{\sigma_{p1}^2 / N_{p1} + \sigma_{p2}^2 / N_{p2}}}, \tag{2-4}$$

where $\sigma_{p1}{}^2$ and $\sigma_{p2}{}^2$ are the sample variance of *-lik(S_{p1})* and *-lik(S_{p2})* respectively, defined as,

$$\sigma_p^2 = \frac{1}{N_p} \sum_{i=1}^{N_p} \{[-\log P(X_{qi} | S_p) - \log P(y_{qi} | S_p, X_{qi})] - [\overline{-\text{lik}(S_p)}]\}^2. \tag{2-5}$$

1. $P(y_{qi} \mid S_p, X_{qi})$ are independently identifically distributed (iid). Although theoretically $P(X_{qi} \mid S_p)$ is not iid, in practice, we roughly regard it as iid, and the hypothesis testing result is satisfactory.

N_{p1} and N_{p2} are the numbers of data points involved in the calculation of the likelihoods of system S_{p1} and S_{p2}. Sometimes N_{p1} and N_{p2} are equal. However, this is not a requirement. The bigger N_{p1} and/or N_{p2}, the larger the absolute value of the Z statistic tends to be.

2. The beauty of statistic Z is that its distribution is close to standard normal distribution if N_{p1} and N_{p2} are big enough, due to Central Limit Theorem. In this way, we can find a standard threshold for any domain and any observation sequence. We define this domain-independent threshold as Z_α. If $Z < -Z_\alpha$, S_{p1} has more potential than S_{p2} to be the generator of the unlabeled observation sequence $\boldsymbol{O}_q$. If $Z > Z_\alpha$, S_{p2} has more potential than S_{p1}. Otherwise, the observation sequence is confusing because S_{p1} and S_{p2} are closely likely to be its generator.

The value of Z_α depends on the significance level α. Referring to Figure 2-4, the smaller the significance level α, the remoter the threshold Z_α deviates from zero, then it is harder for Z to be bigger than Z_α or smaller than $-Z_\alpha$, so that maybe no candidate system is found to be more competitive than all others to be the underlying generator of $\boldsymbol{O}_q$. Therefore, the smaller α is, the "pickier" we are.

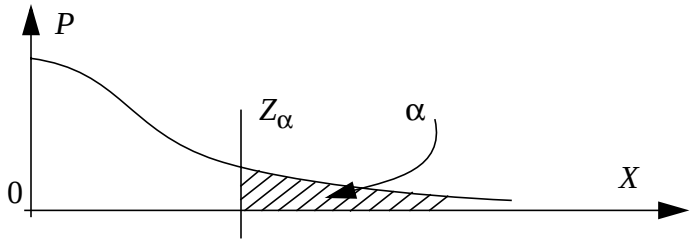

Figure 2-4: The physical meaning of Z_α.

In practice, the significance level α is pre-defined by the user of OMEGA, and Z_α can be found by consulting the standard normal distribution table.

3. With more data points, the absolute value of the Z statistic tends to be bigger, and it is eas-

ier to distinguish the competitiveness of the various candidate systems. Therefore, in Figure 2-2 and 2-3, with more data points, the five curves become more separated. But the redundant data points do not help to distinguish $\overline{-\text{lik}(S_p)}$, *p = 1, ..., 5,* any further; hence, the curves become smooth and consistent afterwards.

2.3 Efficiency Issues

The efficiency of OMEGA is important for two reasons: (1) OMEGA is an on-line technique. (2) Because OMEGA calculates $\overline{-\text{lik}(S_p)}$ for every possible candidate system, suppose there are one hundred candidate systems, S_1, S_2, ..., S_{100}, OMEGA will repeat the likelihood calculation for one hundred times to get $\overline{-\text{lik}(S_1)}$, ..., $\overline{-\text{lik}(S_{100})}$. When there are numerous candidate systems, the computational cost may be prohibitively high even if the task is off-line.

There are three ways to improve the efficiency,

1. Eliminate non-promising candidate systems from consideration:

Recall that the crucial steps of system classification are to calculate $\overline{-\text{lik}(S_p)}$, then compare the $\overline{-\text{lik}(S_p)}$ of the variant candidate systems to eliminate the non-promising candidates, and finally select the most likely one. The $\overline{-\text{lik}(S_p)}$ is calculated according to the following equation:

$$\overline{-\text{lik}(S_p)} = -\frac{1}{N_q}\sum_{i=1}^{N_q}\log P(X_{qi}|S_p) - \frac{1}{N_q}\sum_{i=1}^{N_q}\log P(y_{qi}|S_p, X_{qi})$$

In fact, it is unnecessary to consume all the N_q unlabeled observation data points to approximate $\overline{-\text{lik}(S_p)}$. With fewer data points, even only a single data point, we can still do it. The problem is that with fewer data points, it is more difficult to distinguish a candidate from the others, referring to Section 2.2. But if some systems are far less promising than the others, even with a limited number of data points, its $\overline{-\text{lik}(S_p)}$ value is significantly larger than the others', so that they can be neglected afterwards.

2. Speed up the calculation of the likelihoods:

Since $\overline{-\text{lik}(S_p)}$ is decided by $P(X_{qi} \mid S_p)$ and $P(y_{qi} \mid S_p, X_{qi})$, a quick calculation or approximation for $P(X_{qi} \mid S_p)$ and $P(y_{qi} \mid S_p, X_{qi})$ would improve the efficiency.

3. Focus on the promising candidates:

Even though we can eliminate unpromising candidate systems after a limited number of observations, at the early stage there may still be a large number of candidate systems involved in the processing. For example, if there are 10,000 candidate systems, perhaps after 100 observation data points, we can decide 9,999 candidates are irrelevant. Suppose that with fewer than 100 data points, no elimination can be performed and we have to calculate $\overline{-\text{lik}(S_p)}$ 10,000 times. To enhance the computational efficiency, it may be worthwhile to take a risk and focus on the more promising candidates from the beginning.

Compared with $P(y_{qi} \mid S_p, X_{qi})$, the computational cost of $P(X_{qi} \mid S_p)$ is much cheaper. Therefore, at the early stage with a limited number of (X_{qi}, y_{qi}), $i = 1, 2, ...$, we can eliminate those candidate systems whose $P(X_{qi} \mid S_p)$'s are far lower than the others'. Of course this selection may make mistakes, but in case there are too many candidate systems, the risk is worthy of taking.

To implement the second and the third solutions, we need the *kd-tree* technique, which will be described in Chapter 5 and 6. A kd-tree re-organizes the memory of the training data points in a tree structure and caches some useful information in the nodes. A kd-tree is useful in two respects: (1) We can implement alternative memory-based learning methods with dramatically less cost. Thus we can greatly enhance the efficiency of calculating $P(y_{qi} \mid S_p, X_{qi})$. (2) When a specific query X_{qi} is given, we can quickly retrieve all its neighboring training data points, so as to approximate $P(X_{qi} \mid S_p)$ rapidly. Based on these two aspects, we can improve the efficiency of approximating $\overline{-\text{lik}(S_p)}$, as well as focus on the promising candidates from the beginning.

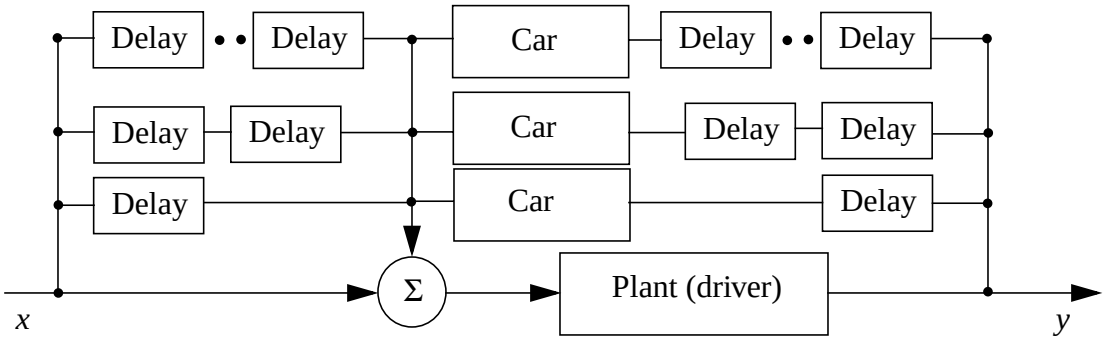

Figure 2-5: An one-input-one-output system with feedbacks and delays. The time order is important.

2.4 Pre-processing

In Subsection 2.1.2, we expand the input to include the delays and feedback so that the output at a certain moment is fully determined by the expanded input at that moment. More generally, for an one-input-one-output system illustrated in Figure 2-5, at time instant t, the output is simply y_t, while the input consists of x_t, x_{t-1}, ..., x_{t-p}, and y_{t-1}, ..., y_{t-q}. Thus, the input space dimensionality is $p + q + 1$.

If the time delays p and q are not known via prior knowledge; we have to figure them out based on empirical analysis of the observation data. Cross-validation, which is discussed in Chapter 7, is a useful technique to select the proper time delays.

It is straightforward to extend this method to transform time series with multiple input and/or output variables. It is not necessary for different input variables to have the same delay, nor likewise for the output variables. In the case where there are u input variables, whose time delays are p_1, ..., p_u, and there are v output variables with q_1, ..., q_v feedback variables, then the dimensionality of the transformed data point's input is $p_1 + ... + p_u + u + q_1 + ... + q_v$.

The transformation of the time series data is not always necessary. Imagining a set of data points { (x1, y1), (x2, y2), ..., (xT, yT) } are generated by a system which has no time delays and feedback, the output yt is fully determined by xt, and xt is independent from the previous ones, xt-1, xt-2, ... In this case, although the data points are collected as time passes, the order of time is not important and we can shuffle the data points randomly.

However, a high dimensionality of the data points is always a concern. Especially those transformed time series data points with expanded input tend to have a dimensionality which is prohibitively high for the further OMEGA steps. This motivates the pre-processing: decreasing the dimensionality of the input space.

Other alternatives may exist, but we choose two approaches: feature selection and Principal Component Analysis (PCA).

Feature selection

In the driving domain, many variables affect our driving performance. While the distance between our vehicle to the vehicle immediately in front of us is probably important, the distance from our vehicle to that one behind us may not be very important in most cases. Therefore, we should consider eliminating the latter distance from the input vector.

To perform feature selection, we follow *cross-validation* approach again. The biggest concern of cross-validation is the computational cost. Therefore, in Chapter 7, we explore ways to improve its efficiency. Feature selection may not be very crucial in the driving domain due to the large amount of prior knowledge. Feature selection is an important component of OMEGA, as a general purpose toolkit.

Principal component analysis

In the driving domain, although we have eliminated those irrelevant input variables based on prior knowledge, the input dimension of the transformed data points may still be as high as 50, (referring to Chapter 8 and Chapter 9). To reduce the dimensionality, we resort to Principal Component Analysis (PCA) [Jolliffe, 86].

Each data point consists of two parts: input and output. Assume the input vector X is d-dimensional. Without loss of generality, X can be represented as a linear combination of a set of d orthonormal vectors $\boldsymbol{U}_k$,

$$X = \sum_{k=1}^{d} z_k \boldsymbol{U}_k$$

With fixed orthonormal vectors $\boldsymbol{U}_k$, $k = 1, ..., d$, different data points' inputs have differing coefficients z_k, $k = 1 ,..., d$. If we carefully choose $\boldsymbol{U}_k$, it is sometimes possible that the first M coefficients contains the most information, i.e.

$$X = \sum_{k=1}^{d} z_k \boldsymbol{U}_k = \sum_{k=1}^{M} z_k \boldsymbol{U}_k + \sum_{k=M+1}^{d} z_k \boldsymbol{U}_k \approx \sum_{k=1}^{M} z_k \boldsymbol{U}_k$$

If so, we can shrink the dimensionality of X from d down to M. Notice that only when all the data points satisfy the above equation, is PCA useful for compressing the dimensionality, as illustrated in Figure 2-6 (a). In the two cases illustrated in Figure 2-6 (b) and (c), PCA does not help.

In one of our experiments, PCA compressed the input dimensionality of the independent data points from 50 dimensions to 3; and in another case, it helped to reduce from 36 dimensions to 8.[2]

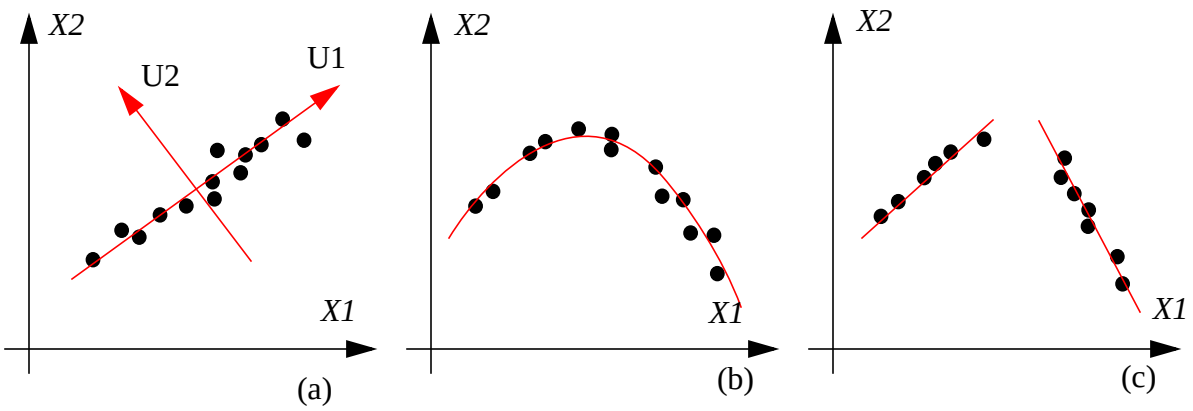

Figure 2-6: PCA can be used to compress the dimensionality of a set of data points. In (a), after the transformation of the coordinates, the information along U_2 axis is no more significant, so that the dimension is reduced from two to one. However, PCA may not be useful for all cases. Although there obviously exist submanifolds in (b) and (c), the conventional PCA does not help to reduce the dimensionality.

2.5 Memory-based learning

In this section, we discuss how to use memory-based learning methods to approximate $P(x_q \mid S_p)$ and $P(y_q \mid S_p, x_q)$. To do so, we need some knowledge of system S_p. Memory-based learning methods assume that the knowledge about a system S_p comes from a memory which consists of the observation data points of this system's previous behavior, $\{(x_{p1}, y_{p1}), (x_{p2}, y_{p2}), ..., \}$. Again, these data points have been pre-processed so that temporal order is no longer important.

When there are n candidate systems, we will have at least n sets of observation data points. The memory contains all of them together. To distinguish the data points generated by different systems, each data is labeled by its generator. Suppose the p'th system has N_p memory data points and there are n candidate systems, the size of memory will be $N_1 + N_2 + ... + N_n$.

2. In first case, the loss of information is 14%. The second case loses 17%.

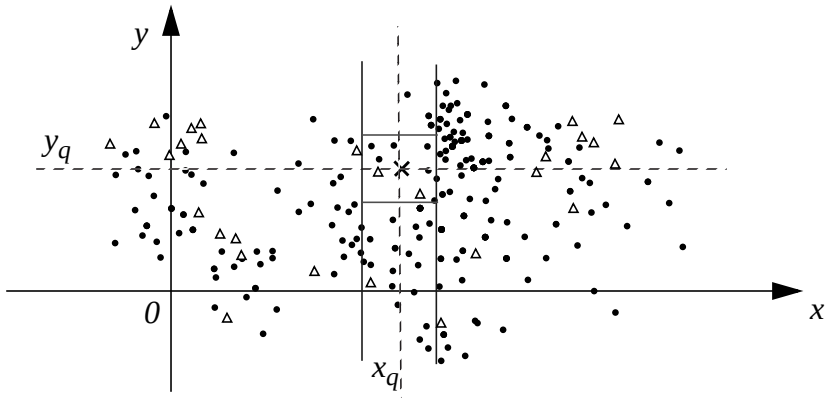

Figure 2-7: Memory-based learning methods to approximate $P(x_q \mid S_p)$ and $P(y_q \mid S_p, x_q)$

A naive method

In Figure 2-7 the *X*-axis is the input of a system, the *Y*-axis is the output. Each dot represents a data point of system S_p. There should exist data points generated by other systems in the memory, too. For example, the triangles are the data points of another system. The cross represents the unlabeled data point (x_q, y_q), which is a component of the observation sequence $\boldsymbol{O}_q$ whose underlying generator is unknown.

To approximate $P(x_q \mid S_p)$, we can simply count the number of the memory data points of S_p (the dots) in the stripe shown in Figure 2-7. The stripe defines the neighboring region of x_q. It is a big concern to decide the boundaries of the stripe, but let's temporarily assume that the boundaries can be easily decided. Suppose the number of dots in the stripe is N_q (N_q = 27 in this case), while the total number of dots in the whole memory space is N_p, then $P(x_q \mid S_p)$ can be approximated as N_q / N_p.

To approximate $P(y_q \mid S_p, x_q)$, we can simply count the number of dots in the square around the unlabeled data point (x_q, y_q); in this case, the number is 6. $P(y_q \mid S_p, x_q)$ can be approximated as the ratio of *6* to N_q, the number of dots in the stripe.

There is one question here: why do not we simply approximate $P((x_{q,} y_q) \mid S_p)$, instead of approximating two probabilities $P(x_q \mid S_p)$ and $P(y_q \mid S_p, x_q)$? In fact, $P((x_{q,} y_q) \mid S_p)$ can be approximated as the ratio of the number of dots in the square to the total number of dots in the whole memory space; in this case, the ratio is $6 / N_p$.

Recall Equation 2-2 and 2-3, which are

$$P(\boldsymbol{O}_q|S_p) \approx \prod_{i=1}^{N_q} P(y_{qi}|S_p, X_{qi})P(X_{qi}|S_p)$$

and

$$\overline{-\text{lik}(S_p)} = -\frac{1}{N_q}\sum_{i=1}^{N_q} \log P(x_{qi}|S_p) - \frac{1}{N_q}\sum_{i=1}^{N_q} \log P(y_{qi}|S_p, x_{qi}).$$

There are three advantages of decomposing $P((x_{q,} y_q) \mid S_p)$ into $P(x_q \mid S_p)$ and $P(y_q \mid S_p, x_q)$. First of all, we can try any machine learning methods to approximate $P(y_q \mid S_p, x_q)$, for example neural networks and Bayes networks. Hence, $P(y_q \mid S_p, x_q)$ is a socket for alternative methods to plug in. Second, the approximation of $P((x_{q,} y_q) \mid S_p)$ is an interpolation problem, but the approximation of $P(y_q \mid S_p, x_q)$ can be extrapolation as well. Finally, the probability $P(y_q \mid S_p, x_q)$ is about the function relationship between the system input and output. If we have some domain knowledge of the system S_p, we can use them to improve the approximation of $P(y_q \mid S_p, x_q)$.

The goodness of the naive method is its simplicity. However, it is difficult to define the boundaries of the stripe and the square. If the stripe is too narrow and the square is too small, the approximation of the probabilities will be too sensitive to the noise of the limited number of the memory data points in the stripe and the square. Otherwise, if the stripe is too wide and the square is too big, it is hard to tell the difference between $P(x_q \mid S_p)$ and $P((x_q + \delta) \mid S_p)$, as well

as the difference between $P(y_q \mid S_p, x_q)$ and $P((y_q+\xi) \mid S_p, (x_q+\delta))$. Besides, the inconsistency of the distribution of the memory data points brings more troubles. In the case of Figure 2-7, if we expand the stripe to be wider, the value of $P(y_q \mid S_p, x_q)$ will chance greatly. In fact, it will become larger, because there are numerous memory data points residing just outside the boundaries.

Therefore, we consider Kernel density estimation, because it does not require any boundaries.

Kernel density estimation

Kernel density estimation does not neglect any data points in memory, so that every memory data point is involved in the approximation of $P(x_q \mid S_p)$ and $P(y_q \mid S_p, x_q)$. However, higher weights are assigned to those memory data points neighboring to the unlabeled data point (x_q, y_q), so that the neighboring memory data points have stronger impact on the approximation of $P(x_q \mid S_p)$ and $P(y_q \mid S_p, x_q)$. Conversely, remote memory data points have smaller weights, therefore any single remote data points hardly has any influence on the approximation, but if many remote memory data points express the same preference, the approximation will be biased in their favor.

Using Kernel density estimation, $P(x_q \mid S_p)$ can be approximated as,

$$P(x_q|S_p) = \sum_{i=1}^{N_p} w(x_i, x_q)/N_q \tag{2-6}$$

in which N_p is the total number of data points in memory generated by S_p. w_i is the weight associated with the i'th one among them, usually defined as a Gaussian function of the Euclidean distance from x_q to the concerned memory data point,

$$w(x_i, x_q) = \text{Const} \times \exp\left(-\frac{\|x_i - x_q\|^2}{2K_w^2}\right). \tag{2-7}$$

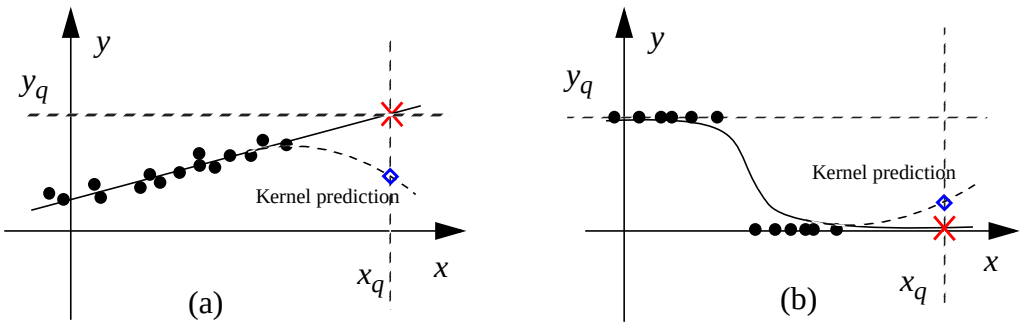

Figure 2-8: Kernel regression does not extrapolate.

Therefore, with respect to different x_q's, the weights associated with an identical memory data point may be different. The higher the Euclidean distance $||x_i - x_q||$, the smaller the weight. K_w is the *kernel width*. The higher the kernel width, the less the weights change with respect to different distances. There are many other possible definitions of the weight [Atkeson et al., 97].

$P(y_q \mid S_p, x_q)$ can be approximated as,

$$P(y_q|S_p, x_q) = \left(\sum_{i=1}^{N_p} w(x_i, x_q)v(y_i, y_q)\right) \Big/ \left(\sum_{i=1}^{N_q} w(x_i, x_q)\right). \quad (2\text{-}8)$$

$v(y_i, y_q)$ is also a weighting function but with respect to the Euclidean distance of $|| y_i - y_q||$. If y's value is continuous, it is fine to define $v(y_i, y_q)$ as a Gaussian function in a way similar to Equation 2-7. However, when y is discrete or categorical, we should be more careful. For example, when y is boolean, the weighting function $v(y_i, y_q)$ can be defined as,

$$v(y_i, y_q) = \begin{cases} 1 & \text{When } y_i = y_q \\ 0 & \text{Otherwise} \end{cases}.$$

Kernel density estimation is useful in many cases, its drawback is that it is only good for interpolation, it does not extrapolate. This is not desirable for the approximation of $P(y_q \mid S_p, x_q)$.

Referring to Figure 2-8(a), suppose we want to approximate $P(y_q \mid S_p, x_q)$, while (x_q, y_q) locates at the position of the cross, intuitively it should be fairly large because it is on the "trend" of the memory data points. However, Kernel density estimation's results will be smaller than they should be. Kernel density estimation does not extrapolate in both continuous case and categorical one. Figure 2-8(b) shows the similar problem in a categorical case.

Locally weighted linear and logistic regressions

Locally weighted linear regression is applicable for both interpolation and extrapolation. Although in many cases, the relationship between the input and the output is more complicated than linear, in any local region, sometimes the relationship can still be approximated as a linear one, illustrated in Figure 2-9. Locally weighted linear regression is a popular memory-based learning method. But it works only when the output *y* is continuous.

The counterpart of locally weighted linear regression for cases when the output *y* is discrete or categorical is locally weighted *logistic* regression. Logistic regression has been explored by the statistical community since 1970's. We improve this technique by following a locally weighted paradigm, so that in the toolkit of memory-based learning method, we have a more reliable classifier.

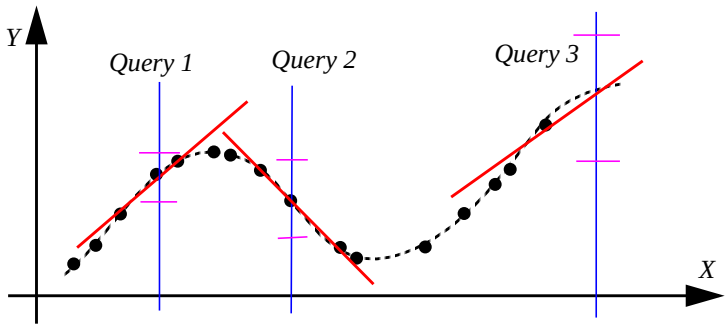

Figure 2-9: Locally weighted linear regression can approximate non-linear functional relationship. It works for both interpolation and extrapolation. The pairs of horizontal bars indicate the variance.

Similar to the principle of locally weighted linear regression, locally weighted *logistic* regression assumes the relationship between the input and output in any local region can be approximated in a form of a simple function. But unlike locally weighted linear regression, which assumes the local relationship is *linear*, locally weighted logistic regression approximates the local relationship in the form of a *logistic* function of a linear combination of inputs. Logistic functions are also referred to as *sigmoid* functions, which are monotonic continuous functions between zero and one. The details will be discussed in Chapter 4.

Approximate $P(y_q \mid S_p, x_q)$ using regression methods

Kernel regression is good enough to approximate $P(x_q \mid S_p)$. In this subsection, we focus on how to use the regression methods to approximate $P(y_q \mid S_p, x_q)$. We discuss this issue in three cases according to the different distribution types of y_q.

1. Suppose the conditional distribution of y_q given a specific x_q is Gaussian, i.e.,

$$P(y_q|S_p, x_q) = \frac{1}{\sqrt{2\pi}\sigma_q}\exp\left(-\frac{(y_q - E(y_q|S_p, x_q))^2}{2\sigma_q^2}\right),$$

   in which $E(y_q \mid S_p, x_q)$ can be predicted using locally weighted linear regression technique, the variance σ_q^2 can be approximated as,

$$\sigma_q^2 = \text{Var}(y_q|S_p, x_q) = E(y_q^2|S_p, x_q) - E^2(y_q|S_p, x_q).$$

   When the conditional distribution of y_q is continuous and *uni-modal*, we will always treat it as a Gaussian distribution. Therefore, the above method is applicable for many cases.

2. When output y_q is discrete or categorical, we can approximate $P(y_q \mid S_p, x_q)$ using locally weighted logistic regression.

3. When output y_q is continuous, but with multiple modes, there are two approaches. First, we can use the techniques like [King et al., 96] to perform the distribution approximation. But this approach still relies on some prior knowledge of the distribution. Second, as a general purpose approach, we can discretizes the output so as to employ the logistic regression approach described in last paragraph.

   For example, suppose the output y_q is continuous within [0, 10). Regardless of whether y's distribution is uni-modal or multi-modal, we discretize it into five equal-sized bins; so that when y_q's value is between [0, 2), we transform it into a categorical value, $(1, 0, 0, 0, 0)^T$. While y is between [2, 4), the corresponding categorical value is $(0, 1, 0, 0, 0)^T$. Now we can use locally weighted logistic regression to approximate $P(y_q \mid S_p, x_q)$.

   However, the discretization approach has two problems. First, in the example above, $P(y_q = 2.5 \mid S_p, x_q)$ and $P(y_q = 3.5 \mid S_p, x_q)$ will be identical, because $y_q = 2.5$ and $y_q = 3.5$ are in the same bin. Therefore, the variance of $P(y_q \mid S_p, x_q)$ increases with fewer bins.

   Second, increasing the discretization resolution causes increased loss of information. For example, as categorical values, both (0, 1, 0, 0, 0) and (0, 0, 1, 0, 0) are differing from (1, 0, 0, 0, 0), but one cannot tell that (0, 1, 0, 0, 0) is closer to (1, 0, 0, 0, 0) than (0, 0, 1, 0, 0). Thus, we retain the information that $P(y_q = 1.0 \mid S_p, x_q)$ and $P(y_q = 3.9 \mid S_p, x_q)$ are both different from $P(y_q = 4.0 \mid S_p, x_q)$, but lose the information that $P(y_q = 3.9 \mid S_p, x_q)$ and $P(y_q = 4.0 \mid S_p, x_q)$ are closely related to each other.

Overall, we still suggest the discretizing method as a general purpose approach. In our experiments in Chapters 3, 8 and 9, we discretized the outputs into 8 or 10 categories, and found the results to be satisfactory.

2.6 Summary

In this chapter, we introduce the main steps for system classification: pre-processing, prediction, likelihood calculation, and hypothesis testing. In addition, we discuss three ways to improve the efficiency.

This chapter is the framework of OMEGA technique, although we mention other relevant topics, i.e. feature selection, logistic regression-based classifier and kd-tree technique. We will discuss these topics in depth in later chapters.

The next chapter discusses an experiment, demonstrating the usefulness of OMEGA system. More complicated experiments will be discussed in Chapter 8 and 9, after we have finished the discussion on feature selection, logistic regression, and kd-tree.