Chapter 7

Feature Selection

Feature selection is not used in the system classification experiments, which will be discussed in Chapter 8 and 9. However, as an autonomous system, OMEGA includes feature selection as an important module.

7.1 Introduction

A fundamental problem of machine learning is to approximate the functional relationship *f( )* between an input $X = \{x_1, x_2, ..., x_M\}$ and an output *Y*, based on a memory of data points, $\{X_i, Y_i\}$, $i = 1, ..., N$, usually the X_i's are vectors of reals and the Y_i's are real numbers. Sometimes the output *Y* is not determined by the complete set of the input features $\{x_1, x_2, ..., x_M\}$, instead, it is decided only by a subset of them $\{x_{(1)}, x_{(2)}, ..., x_{(m)}\}$, where $m < M$. With sufficient data and time, it is fine to use all the input features, including those irrelevant features, to approximate the underlying function between the input and the output. But in practice, there are two problems which may be evoked by the irrelevant features involved in the learning process.

1.The irrelevant input features will induce greater computational cost. For example, using cached *k*d-trees as we discussed in last chapter, locally weighted linear regression's computational expense is $O(m^3 + m^2 \log N)$ for doing a single prediction, where *N* is the num-

ber of data points in memory and *m* is the number of features used. Apparently, with more features, the computational cost for predictions will increase polynomially; especially when there are a large number of such predictions, the computational cost will increase immensely.

2.The irrelevant input features may lead to overfitting. For example, in the domain of medical diagnosis, our purpose is to infer the relationship between the symptoms and their corresponding diagnosis. If by mistake we include the patient ID number as one input feature, an over-tuned machine learning process may come to the conclusion that the illness is determined by the ID number.

Another motivation for feature selection is that, since our goal is to approximate the underlying function between the input and the output, it is reasonable and important to ignore those input features with little effect on the output, so as to keep the size of the approximator model small. For example, [Akaike, 73] proposed several versions of model selection criteria, which basically are the trade-offs between high accuracy and small model size.

The feature selection problem has been studied by the statistics and machine learning communities for many years. It has received more attention recently because of enthusiastic research in data mining. According to [John et al., 94]'s definition, [Kira et al, 92] [Almuallim et al., 91] [Moore et al, 94] [Skalak, 94] [Koller et al, 96] can be labelled as "filter" models, while [Caruana et al., 94] [Langley et al, 94]'s research is classified as "wrapped around" methods. In the statistics community, feature selection is also known as "subset selection", which is surveyed thoroughly in [Miller, 90].

The brute-force feature selection method is to exhaustively evaluate all possible combinations of the input features, and then find the best subset. Obviously, the exhaustive search's computational cost is prohibitively high, with considerable danger of overfitting. Hence, people resort

1. Shuffle the data set and split into a training set of 70% of the data and a testset of the remaining 30%.
2. Let j vary among feature-set sizes: $j = (0 , 1 , 2 , ... , m)$
   a. Let fs_j = best feature set of size j, where "best" is measured as the minimizer of the leave-one-out cross-validation error over the training set.
   b. Let $Testscore_j$ = the RMS prediction error of feature set fs_j on the test set.

   End of loop of *(j)*.
3. Select the feature set fs_j for which the test-set score is minimized.

Figure 7-1: Cascaded cross-validation procedure for finding the best set of up to *m* features.

to greedy methods, such as forward selection. In this paper, we propose three greedier selection algorithms in order to further enhance the efficiency. We use real-world data sets from over ten different domains to compare the accuracy and efficiency of the various algorithms.

7.2 Cross Validation vs. Overfitting

The goal of feature selection is to choose a subset X_s of the complete set of input features $X = \{x_1, x_2, ... ,, x_M\}$ so that the subset X_s can predict the output Y with accuracy comparable to the performance of the complete input set X, and with great reduction of the computational cost.

First, let us clarify how to evaluate the performance of a set of input features. In this chapter we use a very conservative form of feature set evaluation in order to avoid overfitting. This is important. Even if feature sets are evaluated by testset cross-validation or leave-one-out cross validation, an exhaustive search of possible feature-sets is likely to find a misleadingly well-scoring feature-set by chance. To prevent this, we use the *cascaded cross-validation* procedure in Figure 7-1, which selects from increasingly large sets of features (and thus from increasingly

large model classes). The score for the best feature set of a given size is computed by an independent cross-validation from the score for the best size of feature set.

Two notes about the procedure in Figure 7-1: First, the choice of 70/30 split for training and testing is somewhat arbitrary, but is empirically a good practical ratio according to more detailed experiments. Second, note that Figure 7-1 does not describe how we search for the best feature set of size *j* in Step 2a. This is the subject of Section 7-3.

To evaluate the performance a feature selection algorithm is more complicated than to evaluate a feature set. This is because in order to evaluate an algorithm, we must first ask the algorithm to find the best feature subset. Second, to give a fair estimate of how well the feature selection algorithm performs, we should try the first step on different datasets. Therefore, the full procedure of evaluating the performance of a feature selection algorithm, which is described in Figure 7-2, has two layers of loops. The inner loop is to use an algorithm to find the best subset of features. The outer loop is to evaluate the performance of the algorithm using different datasets.

7.3 Feature selection algorithms

In this section, we introduce the conventional feature selection algorithm: forward feature selection algorithm; then we explore three greedy variants of the forward algorithm, in order to improve the computational efficiency without sacrificing too much accuracy.

7.3.1 Forward feature selection

The forward feature selection procedure begins by evaluating all feature subsets which consist of only one input attribute. In other words, we start by measuring the Leave-One-Out Cross Validation (LOOCV) error of the one-component subsets, $\{X_1\}$, $\{X_2\}$, ..., $\{X_M\}$, where M is the input dimensionality; so that we can find the best individual feature, $X_{(1)}$.

1. Collect a training data set from the specific domain.
2. Shuffle the data set.
3. Break it into *P* partitions, (say *P = 20*)
4. For each partition (*i = 0, 1, ..., P-1*)
   a. Let *OuterTrainset(i)* = all partitions except *i*.
   b. Let *OuterTestset(i)* = the *i*'th partition
   c. Let *InnerTrain(i)* = randomly chosen 70% of the *OuterTrainset(i)*.
   d. Let *InnerTest(i)* = the remaining 30% of the *OuterTrainset(i)*.
   e. For *j = 0, 1, ..., m*
      Search for the best feature set with *j* components, fs_{ij}.using leave-one-out on *InnerTrain(i)*
      Let $InnerTestScore_{ij}$ = RMS score of fs_{ij} on *InnerTest(i)*.
      End loop of *(j)*.
   f. Select the fs_{ij} with the best inner test score.
   g. Let $OuterScore_i$ = RMS score of the selected feature set on *OuterTestset(i)*
   End of loop of *(i)*.
5. Return the mean Outer Score.

Figure 7-2: Full procedure for evaluating feature selection of up to *m* attributes.

Next, forward selection finds the best subset consisting of two components, $X_{(1)}$ and one other feature from the remaining *M - 1* input attributes. Hence, there are a total of *M - 1* pairs. Let's assume $X_{(2)}$ is the other attribute in the best pair besides $X_{(1)}$.

Afterwards, the input subsets with three, four, and more features are evaluated. According to forward selection, the best subset with *m* features is the *m*-tuple consisting of $X_{(1)}, X_{(2)}, ..., X_{(m)}$, while overall the best feature set is the winner out of all the *M* steps. Assuming the cost of a LOOCV evaluation with *i* features is *C(i)*, then the computational cost of forward selection searching for a feature subset of size *m* out of *M* total input attributes will be

$$MC(1) + (M-1)C(2) + ... + (M-m+1)C(m).$$

For example, the cost of one prediction with one-nearest-neighbor as the function approximator, using a kd-tree with *j* inputs, is *O(j log N)* where *N* is the number of datapoints. Thus, the

cost of computing the mean leave-one-out error, which involves N predictions, is $O(j\ N\ log\ N)$. And so the full cost of feature selection using the above formula is $O(m^2\ M\ N\ log\ N)$.

To find the overall best input feature set, we can also employ exhaustive search. Exhaustive search begins with searching the best one-component subset of the input features, which is the same in the forward selection algorithm; then it goes to find the best two-component feature subset which may consist of *any* pairs of the input features. Afterwards, it moves to find the best triple out of all the combinations of any three input features, etc. It is straightforward to see that the cost of exhaustive search is the following:

$$MC(1) + \binom{M}{2}C(2) + \ldots + \binom{M}{m}C(m)$$

Compared with the exhaustive search, forward selection is much cheaper.

However, forward selection may suffer because of its greediness. For example, if $X_{(1)}$ is the best individual feature, it does not guarantee that either $\{X_{(1)}, X_{(2)}\}$ or $\{X_{(1)}, X_{(3)}\}$ must be better than $\{X_{(2)}, X_{(3)}\}$. Therefore, a forward selection algorithm may select a feature set different from that selected by exhaustive searching. With a bad selection of the input features, the prediction $\hat{Y}_q$ of a query $X_q = \{x_1, x_2, \ldots, x_M\}$ may be significantly different from the true Y_q.

7.3.2 Three Variants of Forward Selection

In this subsection, we will investigate the following two questions based on empirical analysis using real world datasets mixed with artificially designed features.

1. How severely does the greediness of forward selection lead to a bad selection of the input features?

2. If the greediness of forward selection does not have a significantly negative effect on accuracy, how can we modify forward selection algorithm to be greedier in order to improve

the efficiency even further?

We postpone the first question until the next section. In this chapter, we propose three greedier feature selection algorithms whose goal is to select no more than *m* features from a total of *M* input attributes, and with tolerable loss of prediction accuracy.

Super Greedy Algorithm

Do all the 1-attribute LOOCV calculations, sort the individual features according to their LOOCV mean error, then take the *m* best features as the selected subset. We thus do *M* computations involving one feature and one computation involving *m* features. If nearest neighbor is the function approximator, the cost of super greedy algorithm is $O((M + m) N \log N)$.

Greedy Algorithm

Do all the 1-attribute LOOCVs and sort them, take the best two individual features and evaluate their LOOCV error, then take the best three individual features, and so on, until *m* features have been evaluated. Compared with the super greedy algorithm, this algorithm may conclude at a subset whose size is smaller than *m* but whose inner testset error is smaller than that of the *m*-component feature set. Hence, the greedy algorithm may end up with a better feature set than the super-greedy one does. The cost of the greedy algorithm for nearest neighbor is $O((M + m^2) N \log N)$.

Restricted Forward Selection (RFS)

1. Calculate all the 1-feature set LOOCV errors, and sort the features according to the corresponding LOOCV errors. Suppose the features ranking from the most important to the least important are $X_{(1)}, X_{(2)}, \ldots, X_{(M)}$.
2. Do the LOOCVs of 2-feature subsets which consist of the winner of the first round, $X_{(1)}$, along with another feature, either $X_{(2)}$, or $X_{(3)}$, or any other one until $X_{(M/2)}$. There are

$M/2$ of these pairs. The winner of this round will be the best 2-component feature subset chosen by RFS.

3.Calculate the LOOCV errors of $M/3$ subsets which consist of the winner of the second round, along with the other $M/3$ features at the top of the remaining rank. In this way, RFS will select its best feature triple.

4.Continue this procedure, until RFS has found the best *m*-component feature set.

5.From Step 1 to Step 4, RFS has found *m* feature sets whose sizes range from *1* to *m*. By comparing their LOOCV errors, RFS can find the best overall feature set.

The difference between RFS and conventional Forward Selection (FS) is that at each step to insert an additional feature into the subset, FS considers all the remaining features, while RFS only tries a part of them which seem more promising. The cost of RFS for nearest neighbor is *O(M m N log N)*.

For all these varieties of forward selection, we want to know how cheap and how accurate they are compared with the conventional forward selection method. To answer these questions, we resort to experiments using real world datasets.

7.4 Experiments

In this section, we compare the greedy algorithms with the conventional methods empirically. We run ten experiments; for each experiment, we try two datasets with different input dimensionalities; and for each dataset, we use three different function approximators.

To evaluate the influence of the greediness on the accuracy and efficiency of the feature selection process, we use twelve real world datasets from StatLib/CMU and UCI's machine learning data repository. These datasets come from different domains, such as biology, sociology, robotics, etc. The datasets each contain 62 to 1601 points, and each point consists of an input vector

and a scalar output. The dimensionality of the input varies from 3 to 13. In all of these examples we set *m* (the maximum feature set size) to be 10.

Table 7-1: Preliminary comparison of ES vs. FS

Domain (dim)	*20Fold Mean Errors*			*Time Cost*			*Selected Features*	
	ES	*FS*	*ES / FS*	*ES*	*FS*	*ES / FS*	*ES*	*FS*
Crab (7)	0.415	0.469	0.885	35644	522	68.28	A,F,G	A,E
Halibut (7)	57.972	52.267	1.109	61759	713	86.62	B,C,G	A,D,E,G
Irish (5)	0.863	0.905	0.954	138088	1142	120.91	A,C,E	A,D
Litter (3)	0.780	0.868	0.899	4982	117	42.58	A,B,C	A,B,C

Our first experiment demonstrates that Exhaustive Search (ES) is prohibitively time-consuming. We choose four domains with not-too-large datasets and limited input dimensionality for this test. Referring to Table 7-1, even for these easy cases, ES is far more expensive than the Forward Selection algorithm (FS), while it is not significantly more accurate than FS. However, the features selected by FS may differ from the result of ES. That is because some of the input features are not mutually independent.

Our second experiment investigates the influence of greediness. We compare the three greedier algorithms, Super Greedy, Greedy and Restricted Forward Selection (RFS), with the conventional FS in three aspects:(1) The probabilities for these algorithms to select any useless features, (2) The prediction errors using the feature set selected by these algorithms, and (3) The time cost for these algorithms to find their feature sets.

For example, if a raw data file consists of three input attributes, *U, V, W* and an output *Y,* we generate a new dataset consisting of more input features, *U, V, W, cU, cV, cW,* R_1, R_2,..., R_{10}, and the output *Y,* in which *cU, cV* and *cW* are copies of *U, V* and *W* but corrupted with 20%

noise, while R_1 to R_{10} are independent random numbers. The chance that any of these useless features is selected can be treated as an estimation of the probability for the certain feature selection algorithm to make a mistake.

Table 7-2: Greediness comparison

Domain (dim)	*Funct. Apprx.*	*# Corrupt / Total Corrupts*				*# Noise / Total Noise*			
		Super	*Greedy*	*RFS*	*FS*	*Super*	*Greedy*	*RFS*	*FS*
Bodyfat (13)	Nearest	0.23	0.12	0.10	0.12	0.10	0.05	0.05	0.06
	LocLin	0.31	0.08	0.17	0.18	0.00	0.00	0.05	0.20
	GlbLin	0.31	0.23	0.15	0.00	0.00	0.00	0.00	0.40
Boston (13)	Nearest	0.23	0.19	0.21	0.17	0.20	0.20	0.23	0.35
	LocLin	0.15	0.15	0.12	0.15	0.30	0.30	0.30	0.33
	GlbLin	0.15	0.12	0.15	0.23	0.40	0.30	0.30	0.40
Crab (7)	Nearest	0.29	0.29	0.29	0.29	0.30	0.13	0.17	0.20
	LocLin	0.29	0.14‘	0.21	0.21	0.40	0.40	0.20	0.15
	GlbLin	0.29	0.14	0.29	0.24	0.40	0.30	0.15	0.17
Halibut (7)	Nearest	0.57	0.57	0.14	0.43	0.10	0.10	0.10	0.10
	LocLin	0.43	0.21	0.04	0.24	0.20	0.10	0.10	0.20
	GlbLin	0.36	0.29	0.00	0.14	0.25	0.10	0.20	0.10
Irish (5)	Nearest	0.60	0.60	0.00	0.00	0.20	0.20	0.10	0.10
	LocLin	0.40	0.40	0.38	0.38	0.30	0.30	0.15	0.25
	GlbLin	0.60	0.60	0.30	0.40	0.30	0.30	0.40	0.25
Litter (3)	Nearest	0.67	0.33	0.33	0.33	0.30	0.00	0.05	0.07
	LocLin	0.67	0.33	0.33	0.33	0.30	0.00	0.05	0.07
	GlbLin	0.33	0.33	0.00	0.43	0.50	0.20	0.35	0.50

Table 7-2: Greediness comparison

Domain (dim)	*Funct. Apprx.*	*# Corrupt / Total Corrupts*				*# Noise / Total Noise*			
		Super	*Greedy*	*RFS*	*FS*	*Super*	*Greedy*	*RFS*	*FS*
Mpg (9)	Nearest	0.44	0.44	0.41	0.44	0.00	0.00	0.07	0.05
	LocLin	0.44	0.33	0.22	0.30	0.00	0.00	0.10	0.23
	GlbLin	0.33	0.28	0.22	0.17	0.00	0.00	0.20	0.20
Nursing (6)	Nearest	0.33	0.00	0.25	0.25	0.30	0.10	0.15	0.15
	LocLin	0.33	0.08	0.33	0.22	0.40	0.25	0.20	0.20
	GlbLin	0.33	0.25	0.33	0.25	0.40	0.35	0.20	0.30
Places (8)	Nearest	0.31	0.00	0.00	0.00	0.15	0.00	0.00	0.00
	LocLin	0.38	0.24	0.16	0.40	0.20	0.10	0.00	0.10
	GlbLin	0.25	0.25	0.23	0.31	0.35	0.15	0.15	0.25
Sleep (7)	Nearest	0.29	0.00	0.04	0.04	0.25	0.10	0.13	0.17
	LocLin	0.43	0.11	0.03	0.00	0.20	0.03	0.08	0.10
	GlbLin	0.26	0.21	0.26	0.29	0.40	0.15	0.18	0.40
Strike (6)	Nearest	0.33	0.17	0.17	0.17	0.30	0.00	0.03	0.03
	LocLin	0.58	0.00	0.00	0.00	0.15	0.00	0.00	0.05
	GlbLin	0.50	0.33	0.22	0.33	0.15	0.00	0.08	0.18
White-cell (13)	Nearest	0.15	0.15	0.08	0.23	0.40	0.20	0.15	0.25
	LocLin	0.15	0.04	0.02	0.02	0.04	0.10	0.27	0.27
	GlbLin	0.12	0.14	0.08	0.04	0.40	0.35	0.25	0.25
Mean over all twelve datasets	Nearest	0.37	0.27	0.17	0.21	0.23	0.10	0.11	0.13
	LocLin	0.38	0.18	0.17	0.20	0.24	0.13	0.13	0.18
	GlbLin	0.30	0.26	0.19	0.23	0.29	0.18	0.21	0.28
TOTAL	-	0.35	0.24	0.18	0.21	0.25	0.14	0.15	0.20

As we observe in Table 7-2, FS does not eliminate more useless features than the greedier competitors except the Super Greedy one. However, the greedier an algorithm is, the more easily it is confused by the relevant but corrupted features.

Since the input features may be mutually dependent, the different algorithms may find different feature sets. To measure the goodness of these selected feature sets, we calculate the mean 20-fold score. As described in Section 7-2, our scoring is carefully designed to avoid overfitting, so that the smaller the score, the better the corresponding feature set is. To confirm the consistency, we test the four algorithms in all the twelve domains from StatLib and UCI. For each domain, we apply the algorithms to two datasets. Both of the datasets are generated based on the same raw data file, but with different numbers of corrupted features and independent noise. And for each dataset, we try three function approximators, nearest neighbor (Nearest), locally weighted linear regression (LocLin) and global linear regression (GlbLin). For the sake of conciseness, we only list the ratios. If a ratio is close to 1.0, the corresponding algorithm's performance is not significantly different from that of FS. The experimental results are shown in Table 7-3. In addition, we also list the ratios of the number of seconds consumed by the greedier algorithms to that of FS.

First, we observe in Table 7-3 that the three greedier feature selection algorithms do not suffer great loss in accuracy, since the average ratios of the 20-fold scores to those of FS are very close to 1.0. In fact, RFS performs almost as well as FS. Second, as we expected, the greedier algorithms improve the efficiency. Super greedy algorithm (Super) is ten times faster than forward selection (FS), while greedy algorithm (Greedy) seven times, and the restricted forward selection (RFS) three times. Finally, restricted forward selection (RFS) performs better than the conventional FS in all aspects.

To further confirm our conclusion, we do the third experiment. This time, we insert more independent random noise and corrupted features to the datasets. For example, if the original data

Table 7-3: Greediness comparison

Domain (dim)	*Funct. Apprx.*	*20Fold() / 20Fold(FS)*			*Cost() / Cost(FS)*		
		Super	*Greedy*	*RFS*	*Super*	*Greedy*	*RFS*
Bodyfat (13)	Nearest	0.975	0.969	0.915	0.095	0.126	0.330
	LocLin	1.080	1.015	0.973	0.062	0.092	0.287
	GlbLin	0.984	0.981	0.966	0.084	0.109	0.247
Boston (13)	Nearest	0.876	0.872	0.881	0.105	0.145	0.389
	LocLin	1.091	1.091	0.969	0.058	0.080	0.270
	GlbLin	1.059	1.052	1.068	0.084	0.127	0.287
Crab (7)	Nearest	1.107	1.039	0.973	0.123	0.149	0.358
	LocLin	1.121	1.093	1.024	0.095	0.128	0.349
	GlbLin	1.123	1.101	0.957	0.079	0.116	0.319
Halibut (7)	Nearest	1.089	1.108	1.051	0.133	0.163	0.376
	LocLin	1.395	1.322	1.198	0.079	0.130	0.312
	GlbLin	1.073	1.018	1.022	0.079	0.137	0.273
Irish (5)	Nearest	1.132	1.072	0.954	0.127	0.171	0.343
	LocLin	1.039	0.979	0.984	0.086	0.137	0.316
	GlbLin	0.981	0.981	0.992	0.096	0.180	0.373
Litter (3)	Nearest	1.370	1.014	1.000	0.145	0.222	0.419
	LocLin	1.301	0.960	0.989	0.099	0.179	0.361
	GlbLin	0.886	0.902	0.930	0.111	0.179	0.410
Mpg (9)	Nearest	1.384	1.250	1.084	0.112	0.165	0.398
	LocLin	1.550	1.524	1.081	0.074	0.093	0.271
	GlbLin	1.295	1.317	1.014	0.086	0.142	0.298
Nursing (6)	Nearest	1.315	1.128	0.998	0.102	0.172	0.327
	LocLin	1.171	1.106	1.063	0.072	0.121	0.260
	GlbLin	1.044	1.043	1.002	0.092	0.137	0.267

Table 7-3: Greediness comparison

Domain (dim)	*Funct. Apprx.*	*20Fold() / 20Fold(FS)*			*Cost() / Cost(FS)*		
		Super	*Greedy*	*RFS*	*Super*	*Greedy*	*RFS*
Places (8)	Nearest	1.367	1.000	1.000	0.118	0.154	0.364
	LocLin	0.998	1.017	0.993	0.071	0.112	0.316
	GlbLin	1.041	1.044	1.064	0.091	0.130	0.265
Sleep (7)	Nearest	1.098	0.883	0.981	0.143	0.165	0.361
	LocLin	1.170	0.852	0.922	0.090	0.113	0.273
	GlbLin	0.918	0.925	1.026	0.096	0.122	0.276
Strike (6)	Nearest	1.142	0.952	1.000	0.161	0.178	0.424
	LocLin	1.172	0.987	1.003	0.068	0.108	0.293
	GlbLin	1.004	0.992	0.993	0.093	0.166	0.310
White-cell (13)	Nearest	0.854	0.718	0.906	0.100	0.138	0.288
	LocLin	1.259	0.821	0.931	0.077	0.088	0.254
	GlbLin	0.940	0.942	0.910	0.098	0.109	0.291
Mean over all twelve datasets	Nearest	1.142	1.001	0.978	0.122	0.163	0.365
	LocLin	1.196	1.064	1.011	0.077	0.115	0.296
	GlbLin	1.029	1.025	0.995	0.091	0.138	0.301
TOTAL	-	1.122	1.030	0.995	0.097	0.138	0.321

set consists of three input features, *{U,V,W}*, the new artificial data file contains *{U, cU, V, cV, cU * cV, W, cW, cV * cW, R_1,..., R_{40}}*. The results are listed in Table 7-4 and Table 7-5.

Comparing Table 7-2 with Table 7-4, we notice that with more input features, the probability for any corrupted feature to be selected remains almost the same, while that of independent noise reduces greatly. Comparing Table 7-3 with Table 7-5, with more input features, (1) the prediction accuracies of the feature sets selected by the variety of the algorithms are roughly

Table 7-4: Greediness comparison with more inputs

	Funct. Apprx.	*# Corrupt / Total Corrupts*				*# Noise / Total Noise*			
		Super	*Greedy*	*RFS*	*FS*	*Super*	*Greedy*	*RFS*	*FS*
Mean Values	Nearest	0.29	0.33	0.30	0.38	0.04	0.04	0.03	0.04
	LocLin	0.38	0.38	0.25	0.41	0.05	0.03	0.02	0.03
	GlbLin	0.38	0.25	0.29	0.16	0.05	0.05	0.08	0.07
TOTAL	-	0.35	0.32	0.28	0.32	0.05	0.04	0.04	0.05

Table 7-5: Greediness comparison with more inputs

	Funct. Apprx.	*20Fold( ) / 20Fold(FS)*			*Cost( ) / Cost(FS)*		
		Super	*Greedy*	*RFS*	*Super*	*Greedy*	*RFS*
Mean Values	Nearest	1.197	1.056	1.001	0.080	0.080	0.282
	LocLin	1.202	1.059	1.040	0.071	0.084	0.281
	GlbLin	1.032	1.026	0.998	0.079	0.104	0.294
TOTAL	-	1.144	1.047	1.013	0.077	0.088	0.286

consistent, because the 20fold scores in the two tables are almost the same; (2) the efficiency ratio of the greedier alternatives to FS is a little higher.

In summary, in theory the greediness of feature selection algorithms may lead to great reduction in the accuracy of function approximating, but in practice it does not happen quite often. The three greedier algorithms we propose in this paper improve the efficiency of the forward selection algorithm, especially for larger datasets with high input dimensionalities, without significant loss in accuracy. Even in the case the accuracy is more crucial than the efficiency, restricted forward selection is more competitive than the conventional forward selection.

7.5 Summary

In this chapter, we explore three greedier variants of the forward selection method. Our investigation shows that the greediness of the feature selection algorithms greatly improves the efficiency, while does not corrupt the correctness of the selected feature set so that the prediction accuracy using the selected features remains satisfactory. As an application, we apply feature selection to a prototype system of Chinese and Japanese handwriting recognition.