The economics of classification: error vs. complexity

Dick de Ridder, Elżbieta Pękalska and Robert P.W. Duin
Pattern Recognition Group, Dept. of Applied Physics, Delft University of Technology
Lorentzweg 1, 2628 CJ Delft, The Netherlands
E-mail: dick@ph.tn.tudelft.nl

Abstract

Although usually classifier error is the main concern in publications, in real applications classifier evaluation complexity may play a large role as well. In this paper, a simple economic model is proposed with which a trade-off between classifier error and calculated evaluation complexity can be formulated. This trade-off can then be used to judge the necessity of increasing sample size or number of features to decrease classification error or, conversely, feature extraction or prototype selection to decrease evaluation complexity. The model is applied to the benchmark problem of handwritten digit recognition and is shown to lead to interesting conclusions, given certain assumptions.

1. Introduction

Although in pattern recognition literature often a number of classifiers are compared based on performance (in terms of classification error on an independent test set), there are hardly any reports on how classifiers compare in terms of the amount of computation needed in the application phase (see e.g. [3] for a discussion on computational complexity of classifiers). Still, in practice many well-performing classifiers cannot be used in real applications because their computational complexity is too high. For example, applying the k-nearest neighbour method to a face recognition problem might well give good results, but is not feasible in electronic access situations in which a certain maximum response time is a given. In other applications, there might be a trade-off between error and computational complexity instead of a hard limit, as computational complexity of a large number of classifiers can be attacked using parallel computation. An example is image database retrieval.

This paper tries to quantify computational complexity of various statistical classifiers (i.e. trained using a set $\mathcal{L}$ of examples $\vec{x}$) and to show how error-complexity curves can be made for any classification problem. These curves can be used in an economic model of classification. Based on such curves, the effects of feature extraction and prototype selection will be discussed.

2. Complexity

Statistical classifiers have different types of complexity. The first is *training complexity*. This can vary widely, even for identical classifiers on data sets with identical parameters; for example, the optimisation involved in training the support vector classifier will depend on the amount of overlap present between the classes. However, in practice training time is often not an issue when designing a pattern recognition system; even if training takes days, it will usually only have to be performed once.

The second type of complexity is *evaluation complexity*. This complexity is more of an issue when designing a pattern recognition system, as the throughput and cost of such a system critically depends on it. In general, it depends on the type of classifier used, its parameters, and some numbers defined by the problem to be solved: n, the number of samples $\vec{x} \in \mathcal{L}$ used for training; d, the number of features (dimensions) in each sample; and c, the number of classes to be distinguished.

2.1. Measures

In computer science, complexity of algorithms is often measured in orders, denoted by the Landau symbol O [3], e.g. $O(n^2)$. However, the order only specifies the true complexity up to multiplication by and addition of arbitrary constants. For practical applications, this is not precise enough.

Computational complexity can also be expressed in FLOPs, or floating point operations. Although this is not an ideal measure [5], it suffices for judging computational cost of classifiers, as their evaluation usually consists of a number of additions and multiplications. In modern computer systems with floating point units (FPUs), addition and multiplication are comparable in complexity. Experiments[1] show that a typical modern CPU performs $\approx 10^7$ additions

[1] Values were derived by programming a repeated addition of two non-cacheable blocks of random double-precision floating point values, optimised compiling with `-O9` using `egcs-2.91.66` and running on an Intel Pentium III clocked at 733 Mhz, in single user mode under Linux 2.4.16.

1051-4651/02 $17.00 (c) 2002 IEEE

or multiplications a second. Division has a similar complexity and function evaluations (e.g. $\exp(x), x^p$) have roughly 2.5 times the complexity of an addition.

2.2. Classifier complexity

A fact to consider when estimating evaluation complexity is that many classifiers are only defined for the two-class case. They are usually generalised for the multi-class case by training c individual classifiers each separating a single class from all others, thereby increasing evaluation complexity c-fold. In complexity calculations, an additional c FLOPs are included for the combination rule. For discussing the complexity of various classifiers, it is useful to distinguish between parametric and non-parametric classifiers.

Parametric classifiers are based on a (strong) model of the data or discriminant and estimate parameters of a distribution or discriminant function on the training set. Therefore, they summarise the data; the number of samples in the training set, n, has no influence on evaluation complexity. The complexity of evaluating a new sample $\vec{z}$ can be computed in detail for these classifiers. In the equations below, $\vec{w}$ is a $d \times 1$ weight vector and b is an offset term; $\vec{\mu}$ is a $d \times 1$ mean vector and $\vec{G}$ is a $d \times d$ covariance matrix. Note that multiplication of two d-dimensional vectors will cost d multiplications and d additions, and so has a complexity of $2d$. Multiplying a $1 \times d$ vector by a $d \times m$ matrix has a complexity of $2dm$. The parametric classifiers $S(\vec{z})$ considered here are[2]:

- `ldc`, the Bayes plug-in, linear classifier and `fisherc`, Fisher's linear discriminant. These have the same computational complexity, since they can both be expressed as $S(\vec{z}) = \vec{w}^T\vec{z} + b$ (for each class). Their complexity thus is $c(2d+1)+c = 2c(d+1)$.
- `qdc`, the Bayes plug-in, quadratic classifier. This classifier calculates, for each class, $S(\vec{z}) = (\vec{z}^T\vec{G}_1^{-1}\vec{z} - \vec{w}_1^T\vec{z} + b_1) - (\vec{z}^T\vec{G}_2^{-1}\vec{z} - \vec{w}_2^T\vec{z} + b_2)$ where $\vec{w}_i^T = 2\vec{\mu}_i^T\vec{G}_i^{-1}$ and $b_i = \vec{\mu}_i^T\vec{G}_i^{-1}\vec{\mu}_1$ can be pre-calculated. The complexity of a single classifier is $4d^2+8d+2$ and the complexity of the c combined classifiers becomes $c(4d^2+8d+3)$.
- a feed-forward neural network (considered here to be parametric as its complexity is independent of n). Evaluating a neural network with h hidden units (including output units) and q connections between units (excluding unit biases) consists of q multiplications and additions, followed by h additions of bias terms and h evaluations of the unit transfer function, e.g. the sigmoid; the total complexity therefore is $2q+3.5h$. The neural network used in this paper is the LeCun shared weight network [2] ($q = 63,660$; $h = 1000$).

[2]The abbreviations of the classifiers correspond to the function names in the PRTOOLS pattern recognition toolbox for MATLAB [4].

Non-parametric classifiers base classification on the training set itself, and therefore preserve the training set (at least partly). Classifiers considered here are the k-nearest neighbour classifier and an approximative version of it, k-AESA [6], support vector classifiers, and representation set-based classifiers [8]. The latter train a linear or quadratic discriminant in the space spanned by the distances between samples in the training set and will be discussed in section 4.

A number of these classifiers depend on distance or kernel evaluations. The complexity of a Euclidian distance calculation $D(\vec{z},\vec{x})$, where $\vec{z}$ and $\vec{x}$ are d-dimensional vectors, is $3d$. A polynomial kernel (the only kernel considered here) $K_P(\vec{z},\vec{x}) = \sum_{i=1}^{d}(z_i x_i + 1)^p$ has complexity $5.5d$. The non-parametric classifiers used here are:

- `knnc`, the k-nearest neighbour classifier. Distances to all prototypes have to be calculated ($2nd$) and the minimum will have to be stored in a sorted list of k nearest prototypes ($n\log_2 k$). The total complexity therefore is $n(3d+\log_2 k)$. In experiments below, k was always 1 (which was optimal for this particular data set).
- `kaesa` speeds up evaluation of the k-nearest neighbour classifier. However, evaluation complexity cannot be predicted as it depends on the data set at hand. In the experiments it was therefore measured.
- `svc`, the support vector classifier (trained using *SVM-Torch* [1]). For each class, kernel functions between the new sample $\vec{z}$ and all support vectors for that classifier are calculated. However, as samples may be support vectors for more than one classifier, only the number of *unique support vectors* n^s determines the complexity. For the polynomial kernel, this amounts to $n^s(5.5d)$. After calculating the kernels, the results are weighted and summed for each of the c classifiers, at a cost of $(1+2n^j)$, where n^j is the number of support vectors for the classifier describing class j. Overall, the complexity is $2c+2n^{s'}+5.5dn^s$, where $n^{s'} = \sum_{j=1}^{c} n^j$.

2.3. Error-complexity curves

The classifiers above can be trained on any problem. In such experiments, it is good practice to vary the number of samples in the training set to investigate the dependency of performance on n. However, n also directly influences computational cost. The data set we use here as an example is a pre-processed subset of the NIST handwritten digit database [9, 2], with $c = 10$, $d = 256$ (16×16 pixels) and $n \in c \cdot \{10, 50, 250, 1000\}$. Error e (in %, measured on an independent test set of 1,000 samples/class) vs. computational complexity f (in FLOPs) is shown in figure 1(a), for various classifiers. There clearly is a wide range of complexity, between 10^3 and 10^7 FLOPs. Furthermore, it is obvious that in different complexity ranges different classifiers dominate.

Given figure 1(a), we can draw a curve indicating what the best possible performance is given a certain complexity over all training sample sizes used. All classifiers above this curve are irrelevant, as a cheaper solution giving at most the same error can be found to the left. Note that in practice such a curve can easily be used to find the classifier (1) giving lowest error, at the point of contact of a line of constant error e to the curve; or (2) having a maximum complexity: for applications in which there is a fixed upper bound to the computational complexity (i.e. when reaction time instead of throughput is the issue), one can simply intersect a line of constant f with the curve and find the classifier giving lowest error e to the left of it.

For the handwritten digit recognition problem, the error-complexity curve is shown in figure 1(b). Note how only `ldc`, `svc`, `kaesa` and the neural network are relevant.

3. Economics

To compare classification error to computational complexity for a given classifier, both will have to be specified in a common unit. The easiest unit to work in for real applications is cost. Below, all cost is expressed in euro's.

To calculate the cost of classification errors, we can multiply the cost of one error ($\in c_o$) by the probability of misclassification (e): $c_e = e \cdot c_o$. The cost of complexity is slightly more involved. Say a CPU is capable of v FLOPs per second and costs $\in c_c$ per year including secondary devices, depreciation and maintenance (the "total cost of ownership"). There are 3.15×10^7 seconds in each year. Furthermore, let f be the number of FLOPs needed to classify a single sample $\vec{z}$ and s the percentage of CPU time alotted to classification. Assuming classification can be spread over multiple CPUs without overhead[3], the cost per evaluation can be expressed as: $c_p = \frac{c_c \cdot f}{3.15 \times 10^7 \cdot v \cdot s}$.

This leaves several parameters which depend on currently available hardware. Experimentally (see section 2.1), a reasonable value for v was found to be 10^7. For high-availability platforms (e.g. servers made by IBM, Sun or HP/Compaq), c_c currently amounts to roughly $\in 10^4$ per year. Finally, s of course depends on the problem at hand. However, in our experience in many projects where signal processing (audio or video) necessarily preceeded classification, 25% seems reasonable.

These estimates allow us to equate cost of error and cost of complexity:

$$c_e = c_p \quad (1)$$

$$e \cdot c_o = \frac{c_c \cdot f}{3.15 \times 10^7 \cdot v \cdot s} = 1.27 \times 10^{-10} \cdot f \quad (2)$$

The constant on the right of the equation may seem very small. However, consider the handwritten digit recognition problem discussed earlier, which classifies sub-sampled images directly: $n = 10{,}000$, $d = 256$ and $c = 10$. The error e is 0.0138 for the polynomial support vector classifier trained on 1000 samples per class with $p = 3$, and 0.0132 with $p = 4$. The difference in error is 6×10^{-4}; the difference in complexity is 857,356 FLOPs. This indicates what the cost of a single error, c_o, must be to justify the added expense:

$$c_o = \frac{1.27 \times 10^{-10} \cdot 8.57 \times 10^5}{6 \times 10^{-4}} = \in 0.18 \quad (3)$$

While not a large amount, it is not negligible. If the true cost of a single error is smaller than this amount, it will pay off to make a slightly larger error (in this case by using the support vector classifier with $p = 3$). In general, situations such as these are likely to occur in, for example, inspection of mass-produced simple objects such as screws and nails, toys, foodstuffs or even medicine. Of course, the number derived above depends linearly on the current cost of CPU power and relative CPU usage for classification. In each individual application at each time, the resulting number will be different. However, the model shows that simple assumptions can lead to real trade-offs.

The model above can be used in a number of other ways. For a solution costing $\in c$ per object, it should hold that $c = c_e + c_p$. This equation can be used to draw iso-cost lines. Given the parameters above and, say, a cost of $c_o = \in 0.01$, the iso-cost lines will be $e = c - 1.27 \times 10^{-8} f$. The point of contact between the iso-cost line and the error-complexity curves with minimum c then specifies the classifier to be chosen. Alternatively, c can be fixed (to give a maximum total cost per object); in this case, any classifier on the error-complexity curve below the iso-cost line is sufficient.

4. Feature extraction and prototype selection

Computational complexity can, of course, be lowered by performing feature extraction, for example using PCA. This will lower d, the number of dimensions, to m; the added complexity is only the mapping of an incoming vector to the feature subspace, $2md$ FLOPs. In an experiment, PCA was applied to lower the dimensionality of the digit data to 16, 32 and 64 dimensions. Figure 1(c) shows the results; classifiers trained on PCA-projected data are denoted by a preceding `p-`. Clearly, feature extraction helps to lower the error-complexity curve over a broad range.

A classification method recently proposed [7] is to train ordinary classifiers (`ldc` and `qdc`) in the space spanned by the distances of samples in the training set to a set of prototypes. These classifiers, `r-ldc` and `r-qdc`, work in the same space as the k-nearest neighbour method, but impose a stronger model and are therefore expected to exhibit better performance for small sample sizes. Their complexities

[3] Or, equivalently, that various samples can be evaluated in parallel.

can be found simply by replacing d by m in the original expressions. An additional step, calculating the distances to the prototypes, takes $2md$ FLOPs, where m is the number of prototypes. In experiments, we selected $m \in c \cdot \{2,4,6,8\}$ random prototypes out of the n in the current training set, calculated the m distances to these prototypes for each training sample and used these as its features. The results are shown in figure 1(d). It shows that the application of prototype selection in this problem is limited to a range of medium complexity. However, in this region it performs comparable to PCA. An open question, currently under research, is whether performance can be improved using a more principled method than random prototype selection.

5. Conclusions

This paper discussed how evaluation complexity of classifiers can be calculated and error-complexity curves can be drawn. A simple economic model showed that even with current technology, in real applications high evaluation complexity can become too expensive. Finally, error-complexity curves were shown to be useful in judging the effect of feature extraction and prototype selection.

References

[1] R. Collobert and S. Bengio. SVMTorch: support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001.

[2] D. de Ridder. *Adaptive methods of image processing*. PhD thesis, Delft University of Technology, Delft, 2001.

[3] R. Duda, P. Hart, and D. Stork. *Pattern classification*. John Wiley & Sons, New York, NY, 2nd edition, 2001.

[4] R. Duin. PRTOOLS, a MATLAB toolbox for pattern recognition, 2000. version 3.0.

[5] J. Hennessy and D. Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann, San Mateo, CA, 1990.

[6] A. Juan and E. Vidal. On the use of edit distances and an efficient k-NN search technique (k-AESA) for fast and accurate string classification. In *Proc. ICPR 2000*, volume 2, pages 680–683, Los Alamitos, CA, 2000. IAPR, IEEE Computer Society Press.

[7] E. Pekalska and R. Duin. Automatic pattern recognition by similarity representations. *Electronics Letters*, 37(3):159–160, 2001.

[8] E. Pekalska and R. Duin. Dissimilarity representations allow for building good classifiers. *Pattern Recognition Letters*, 23(8):943–956, 2002.

[9] C. L. Wilson and M. D. Garris. Handprinted character database 3, february 1992. National Institute of Standards and Technology; Advanced Systems division.

This work was partly supported by the Dutch Foundation for Applied Sciences (STW).

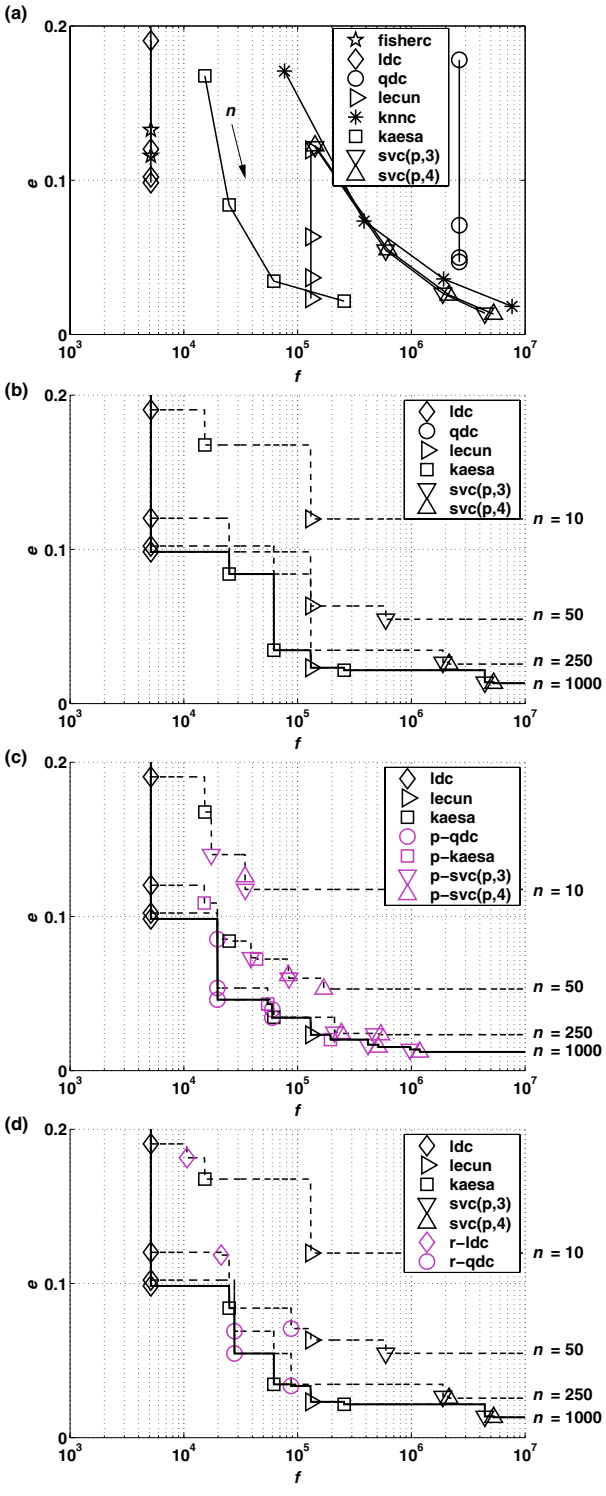

Figure 1. (a) Classifier complexity f (in FLOPs) vs. error e (in %), for $n = 10$, 50, 250 and 1000. (b) Error-complexity curves (only relevant classifiers shown). (c) Same, with PCA-trained classifiers. (d) Same, with representation set-based classifiers.