J Autom Reasoning (2014) 53:141–172
DOI 10.1007/s10817-014-9301-5

Machine Learning for First-Order Theorem Proving

Learning to Select a Good Heuristic

James P. Bridge · Sean B. Holden · Lawrence C. Paulson

Received: 13 September 2012 / Accepted: 27 January 2014 / Published online: 22 February 2014

Abstract We applied two state-of-the-art machine learning techniques to the problem of selecting a good heuristic in a first-order theorem prover. Our aim was to demonstrate that sufficient information is available from simple feature measurements of a conjecture and axioms to determine a good choice of heuristic, and that the choice process can be automatically learned. Selecting from a set of 5 heuristics, the learned results are better than any single heuristic. The same results are also comparable to the prover's own heuristic selection method, which has access to 82 heuristics including the 5 used by our method, and which required additional human expertise to guide its design. One version of our system is able to decline proof attempts. This achieves a significant reduction in total time required, while at the same time causing only a moderate reduction in the number of theorems proved. To our knowledge no earlier system has had this capability.

Keywords Automatic theorem proving · Machine learning · First-order logic with equality · Feature selection · Support vector machines · Gaussian processes

1 Introduction

Theorem provers for first-order logic (FOL) with equality have the potential to be largely automatic, but at present significant expert human input can be required to optimize fully the performance of a prover on a given problem. While the underlying approach to proof search employed by a given prover is based on one or more core algorithms, performance can be

J. P. Bridge · S. B. Holden (✉) · L. C. Paulson
Computer Laboratory, University of Cambridge, William Gates Building, 15 JJ Thomson Avenue, Cambridge CB3 0FD, UK
e-mail: sbh11@cl.cam.ac.uk

J. P. Bridge
e-mail: jpb65@cl.cam.ac.uk

L. C. Paulson
e-mail: lp15@cl.cam.ac.uk

affected by a number of parameter values and other decisions regarding the detail of the operation of these algorithms. Provers often incorporate one or more collections of standard settings for such parameters and other details; such a collection is known as a *heuristic*. The best heuristic to use will depend on the form of the conjecture to be proved and the accompanying axioms, however the relationship between these and the best heuristic is not obvious even to those with extensive experience, let alone to users who wish only to use the prover as a tool.

In this paper we apply two machine learning methods (see for example Bishop [3], Duda et al. [11] and Mitchell [32]) to the problem of heuristic selection. The approach taken is to use existing, well-established heuristics and to automatically learn to select a good heuristic using simple features of the conjecture to be proved and the associated axioms. This is related to the method available to the existing E theorem prover (Schulz [40]), which is able to select a heuristic using features but uses a less sophisticated approach: the features are used simply to divide problems into separate groups, each associated with a best heuristic selected using prior experimental results. The key differences between this approach and our work are as follows. First, in the former the division into groups restricts features to be binary or ternary. Second, it must be assumed rather than determined that problems within a group are best solved with the same heuristic. Our application of machine learning to the problem allows real-valued features to be used, and determines in a rigorous manner and with fewer preconceptions the possible connections between feature values and the best heuristic to use.

The task of heuristic selection lends itself well to machine learning. The connection between input feature values and the associated preferred heuristic is too complex to be derived directly; yet for any given sample problem the preferred heuristic may be found by running all heuristics. Obtaining labelled training data is thus straightforward given a good selection of trial problems. In the area of first-order logic with equality the *Thousands of Problems for Theorem Provers (TPTP)* library (Sutcliffe [43]) provides a readily available collection of trial problems from many different subject areas, and it is this library that we use to construct a training set from which to learn.

While attempts have been made by several researchers to incorporate machine learning methods into FOL theorem provers, we take a different approach to those attempted previously. Also, we apply learning methods known to be among the most powerful available, and which have not been applied in this context before: *support vector machines (SVMs)* (Shawe-Taylor and Cristianini [41]) and *Gaussian processes (GPs)* (Rasmussen and Williams [36]). A review of work prior to 1999 can be found in Denzinger et al. [9] and, for example, the *TEAMWORK* system of Denzinger et al. [8] employs case based reasoning, Fuchs [14] uses instance-based learning and Goller [16] folding architecture networks. The *MaLARea* system of Urban [45] makes use of a method based on naive Bayes learning (Mitchell [32]). The prior work most similar in its approach to our own has concentrated on learning novel heuristics; see for example Erkek [12], Fuchs and Fuchs [15] and Schulz [39]. Our work differs in that, rather than learning the form of the heuristic itself, we make use of known good heuristics and learn to select an appropriate one.

It is of note that to date simple machine learning methods have been employed with more success in SAT solvers. We provide a brief review of relevant work in Appendix A.

Section 2 explains in detail the problem addressed, the selection of a theorem-prover and heuristics, and the construction of our data set.[1] Section 3 introduces the performance measures used and explains in detail how our learners were trained. Section 4 explains the overall experimental method and how we applied our trained classifiers to the selection of heuristics within the theorem-prover. Section 5 discusses our results and Section 6 presents a comparison with the prover's own automatic heuristic selection method. Section 7 gives a brief discussion of possible further work, and Section 8 concludes the paper. Appendix A briefly reviews related work for SAT solvers, Appendix B provides a summary of the machine learning methods used and the notation used in the paper, and Appendix C presents some results not included in the main part of the paper.

2 The Problem

Initially we chose an extensive set of proof problems taken from the TPTP library[2] [43], and a suitable automated theorem prover with which to solve them. Sample problems were taken from the entire TPTP library at the time the work was done. The only filtering applied was to remove problems that were proved before the proof search had progressed to the point where dynamic features[3] could be measured, and to remove problems for which the prover reached saturation, as these were generally pathological cases. This gave us a collection of 6118 problems from which to produce training, validation and test data.

The theorem prover was capable of employing a number of different heuristics. For reasons discussed below we confined our experiments to a subset of these heuristics. The prover was run on every problem using each heuristic in this subset in turn and the time taken to find a proof recorded. A CPU time limit of 100 seconds was set during this process. This provided for each problem a measurement of how effective each possible heuristic was when applied to that problem. Throughout the following we denote by n the total number of problems employed.

Our aim was to find a method for automatically selecting the appropriate heuristic for a given problem. The data collected as described naturally lends itself to being used within a standard supervised learning framework, provided that a set of *features*—that is, a vector $\mathbf{x}$ in some sense characterising the problem—can be extracted from any given problem. Once this is achieved there are several potential ways in which each vector can be labelled with a classification y to produce the labelled data

$$\mathbf{s}^T = [(\mathbf{x}_1, y_1)(\mathbf{x}_2, y_2) \cdots (\mathbf{x}_n, y_n)]\,. \tag{1}$$

[1] The full sets of training, validation and test data used in our work can be obtained from the UCI Machine Learning Repository (Bache and Lichman [1]).

[2] We used version 3.2.0 (July 2006) of the TPTP library.

[3] Some of our features—referred to in what follows as *dynamic features*—were measured only after the proof search had run for some time. This is explained in greater detail in Subsection 2.2.

Table 1 The five heuristics used and their identifying labels within the E theorem prover

Heuristic	Labelling within E	Selections
1	G_E__021_K31_F1_PI_AE_S4_CS_SP_S2S	2442
2	H___081_B31_F1_PI_AE_S4_CS_SP_S0Y	437
3	H___047_K18_F1_PI_AE_R4_CS_SP_S2S	377
4	G_E__008_K18_F1_PI_AE_CS_SP_S0Y	329
5	G_E__008_K18_F1_PI_AE_R4_CS_SP_S2S	321

The final column states the number of times each was selected by E for the test set used in its design

The approach used in the present work is to produce a separate set $\mathbf{s}_h$ of labelled data for each heuristic h, such that each $\mathbf{x}$ has a binary label—+1 for class 1 or −1 for class 2—indicating whether or not h was the best heuristic, in the sense that it led to the fastest proof, for that problem.[4]

In the following subsections we provide some further detail regarding our experimental method.

2.1 Theorem Prover and Heuristics

Our work is based on the E theorem prover (version 0.99 Singtom, Schulz [40]). This prover is widely used and has been found in competition to have good performance. Importantly for the present work it is also open source and actively supported by its author; these were significant issues in choosing to employ it as, while many potential features and related measures were available in the code, it was necessary to modify the code in order to measure some of our features both before and during the proof search process.

The E theorem prover used has 82 built-in heuristics. This presents a problem in that to attempt to learn to select from among all possible heuristics would represent an insurmountable computational effort; we need at the very least to run the prover using each possible heuristic on a large number of problems from the TPTP library, in addition to then training classifiers for each heuristic. We therefore focused on a smaller set of heuristics; namely, the five heuristics most often selected by E for the TPTP problems being used.[5] Table 1 lists the five heuristics chosen using this approach according to their labelling within E. While this labelling is rather uninformative to the non-expert—for example, in the labelling for heuristic 1, `_PI` denotes a preference for initial clauses and `_SP` denotes simultaneous paramodulation—we refer the reader to Bridge [4] and Schulz [40] for a more detailed description of these heuristics.

Our aim was to predict automatically which of the five heuristics was best for any given problem, in the sense that it would lead to a proof most quickly. However for some problems none of the five heuristics could find a proof within the time limit allowed. We therefore

[4] An alternative way in which to construct a machine learning problem involves learning to predict the *actual time taken* by a given heuristic to produce a proof. We performed some initial experiments using GP regression, which suggested that this approach is unlikely to prove feasible for this data, although it has been used successfully in applying machine learning to SAT solvers (see Appendix A).

[5] An alternative, which we have not explored, might be to choose a subset of heuristics that as a collection solve the largest number of problems.

introduced a further heuristic—referred to as heuristic 0—to represent this case. The selection of heuristic 0 for a given problem is interpreted as an indication that the problem is too difficult for any of the heuristics available, and that we should therefore reject the problem without devoting time to a proof search. Heuristic 0 is thus the heuristic that immediately gives up, and is considered the best heuristic when none of the heuristics 1 to 5 are able to find a proof within the allowed time of 100 CPU seconds. In the following we refer to heuristics 0 to 5 using the notation H0, H1, . . . , H5.

2.2 Features of Conjectures and Axioms

A problem is initially presented to a theorem prover as a conjecture and a set of axioms. we need to extract a vector **x** of features from the problem description. In fact the problem is presented as a set of clauses derived from the axioms and the negation of the conjecture to be proved. There are many ways in which a set of features can be defined over a collection of clauses; for example, one feature might denote the proportion of clauses that are Horn clauses. Our first set of features is summarized in Table 2; these features were derived from the size and structure of the clauses, but no meaning was attached to elements such as the function or variable names. As these features are derived entirely from the description of the problem, prior to the start of any proof search, we refer to them as *static features*.

We also explored the possibility that running a proof search for a short period of time might yield new information that is relevant to choosing a good heuristic. The E theorem prover employs the *given clause algorithm* (see for example Denzinger et al. [10]). A search is conducted based on the negated conjecture and the axioms, the aim being to derive the empty clause and thus demonstrate inconsistency (Davis and Putnam [7], Huth and Ryan [24]). During the search for the empty clause, two sets of clauses are maintained. One set—the *processed clauses*—consists of clauses for which all possible inferences within the clause set have been tried. The second set is that of *unprocessed clauses*. The unprocessed clauses initially consist of the negated conjecture and the axioms. As the proof search progresses new clauses arise from inferences and these are placed in the unprocessed clause

Table 2 Description of the static features used

Feature number	Description
1	Fraction of clauses that are unit clauses.
2	Fraction of clauses that are Horn clauses.
3	Fraction of clauses that are ground Clauses.
4	Fraction of clauses that are demodulators.
5	Fraction of clauses that are rewrite rules (oriented demodulators).
6	Fraction of clauses that are purely positive.
7	Fraction of clauses that are purely negative.
8	Fraction of clauses that are mixed positive and negative.
9	Maximum clause length.
10	Average clause length.
11	Maximum clause depth.
12	Average clause depth.
13	Maximum clause weight.
14	Average clause weight.

set. In the given clause algorithm clauses are selected one at a time from the unprocessed set and then all inferences possible between the selected or given clause and the processed clause set are made.[6]

It seems reasonable to expect that features obtained based on the contents of the two sets of clauses after a certain degree of inference has been completed might yield useful information for selecting a heuristic.[7] We therefore also explored the use of *dynamic features*, which are measured using the proof state some time after the proof search has commenced. Dynamic features were measured using both the processed and unprocessed clause sets after a specified number (in this work 100) of given clauses had been selected and processed. Table 3 summarizes the dynamic features used. The fact that we have two distinct sets of clauses—processed and unprocessed—from which to draw provides some increase in flexibility in designing potentially useful features, and hence the set of dynamic features is somewhat larger than the set of static features.

In generating the dynamic features we need to run the theorem prover, and hence we need to decide which heuristic to use. We used heuristic 1 in all cases to generate the dynamic features; this is the heuristic most often selected by E in auto mode. Using a fixed heuristic leads to consistency in the generation of features. While there is some possibility that this introduces a bias,[8] note that any system will need to face this problem. It is hoped that running the prover for only a short time to generate dynamic features will limit any adverse effects due to the use of a single heuristic.

After gathering the data it was found that static feature 5 and dynamic feature 21 were redundant, in the sense that they took the same value for all problems. These features were deleted from the data.[9]

2.3 Training, Optimization and Test Data

The prover was applied to each problem using each of the five fixed heuristics in turn, producing for each heuristic the time in seconds required to find a proof, or 100 if no proof was found. This data was then used to construct six further sets. For each fixed heuristic h, a corresponding set was derived with each feature vector labelled $+1$ if h found a proof for the corresponding problem and was the fastest to do so, or -1 if h failed to find a proof or was not the fastest. The sixth data set corresponded to heuristic 0; problems were labelled $+1$ if none of the five heuristics found a proof or -1 otherwise.[10]

Each data set of 6118 problems was then split into three subsets. The first half formed a *training set* of 3059 examples. The second half was split into two further sets in the usual manner: a *validation* set of 1529 examples used for parameter optimization, and a *test set* of 1530 examples used for computing performance. Note that the manner in which the data

[6] There is a variation of the given clause algorithm that was introduced by the Otter (now superseded by Prover9) theorem prover (McCune [30]). We only consider the version employed by E in this paper.

[7] We acknowledge Stephan Schulz (private email communication) who suggested to us that our dynamic feature number 2 might be relevant. This conjecture is discussed further in Section 7.

[8] The values obtained for dynamic features might depend strongly on the heuristic used to generate them. For example, say heuristic A generates 100 clauses and simplifies 1000 in the relevant time period, but heuristic B generates 1000 and simplifies 100. We might expect that heuristic A is the more likely to be successful.

[9] This has also been observed in related work on applying machine learning to SAT solvers (see Xu et al. [47]).

[10] An alternative method, which we have not explored, would be to label all successful heuristics positively.

Table 3 Description of the dynamic features used

Feature number	Description
1	Proportion of generated clauses kept. (Subsumed or trivial clauses are discarded.)
2	Sharing factor. (A measure of the number of shared terms.)
3	$\|P\|/\|P \cup U\|$
4	$\|U\|/\|A\|$
5	Ratio of longest clause lengths in P and A.
6	Ratio of average clause lengths in P and A.
7	Ratio of longest clause lengths in U and A.
8	Ratio of average clause lengths in U and A.
9	Ratio of maximum clause depths in P and A.
10	Ratio of average clause depths in P and A.
11	Ratio of maximum clause depths in U and A.
12	Ratio of average clause depths in U and A.
13	Ratio of maximum clause standard weights in P and A.
14	Ratio of average clause standard weights in P and A.
15	Ratio of maximum clause standard weights in U and A.
16	Ratio of average clause standard weights in U and A.
17	Ratio of the number of trivial clauses to $\|P\|$.
18	Ratio of the number of forward subsumed clauses to $\|P\|$.
19	Ratio of the number of non-trivial clauses to $\|P\|$.
20	Ratio of the number of other redundant clauses to $\|P\|$.
21	Ratio of the number of non-redundant deleted clauses to $\|P\|$.
22	Ratio of the number of backward subsumed clauses to $\|P\|$.
23	Ratio of the number of backward rewritten clauses to $\|P\|$.
24	Ratio of the number of backward rewritten literal clauses to $\|P\|$.
25	Ratio of the number of generated clauses to $\|P\|$.
26	Ratio of the number of generated literal clauses to $\|P\|$.
27	Ratio of the number of generated non-trivial clauses to $\|P\|$.
28	`context_sr_count`/$\|P\|$.
29	Ratio of paramodulations to $\|P\|$.
30	`factor_count`/$\|P\|$.
31	`resolv_count`/$\|P\|$.
32	Fraction of unit clauses in U.
33	Fraction of Horn clauses in U.
34	Fraction of ground clauses in U.
35	Fraction of demodulator clauses in U.
36	Fraction of rewrite rule clauses in $\|U\|$.
37	Fraction of clauses with only positive literals in U.
38	Fraction of clauses with only negative literals in U.
39	Fraction of clauses with positive and negative literals in U.

The set of processed clauses is denoted by P and the set of unprocessed clauses by U

The set of axioms is denoted by A. `context_sr_count`, `factor_count` and `resolv_count` are variables within E

Table 4 Imbalance in the training, validation and test sets

Heuristic	Training set	Validation set	Test set	All examples
0	0.420	0.421	0.408	0.417
1	0.182	0.170	0.178	0.178
2	0.075	0.087	0.081	0.079
3	0.122	0.122	0.123	0.122
4	0.099	0.095	0.110	0.101
5	0.102	0.104	0.100	0.102

This table shows the proportion of positive examples in each set of examples used

is generated, with one set for each heuristic, naturally leads to sets containing relatively few positive examples: with six possible outcomes represented by the six heuristics each individual classifier might be expected to have five times as many negative samples as positive ones. Table 4 summarizes the actual degree of imbalance in the data.

The data were normalized such that each feature in each training set had zero mean and unit variance across the set. Features in the validation and test sets were then normalized using the same offsets and scalings as applied to their corresponding training sets.

Table 5 provides basic performance data for the five fixed heuristics. This data denotes the number of theorems in the third subset of problems—that is, the subset reserved for computing performance—that each heuristic is able to prove, and the total time required to do so, including 100 seconds for each problem not solved within the time limit. It also contains equivalent data for the combination of the second (validation) and third sets.

For the purposes of placing our results in context it is of interest also to note the performance that would be obtained using a system capable of always choosing the best possible heuristic. For the test set this would result in 906 theorems proved in 65,593 seconds, and for the combined set 1,791 theorems proved in 133,464 seconds.

3 Machine Learning Methods

The machine learning algorithms used are complex and the details of their operation are extensively documented elsewhere. We will not describe them in great detail; Appendix B

Table 5 Performance of the individual heuristics in terms of the number of theorems proved within the time limit and the total time taken in seconds

Heuristic	Test set only		Combined set	
	Number proved	Time	Number proved	Time
1	774	79,336	1,514	162,029
2	695	87,005	1,352	177,530
3	722	83,409	1,424	168,593
4	726	83,438	1,421	169,598
5	673	88,130	1,339	176,959

Numbers are provided for the test set (1530 examples) and for the combination of validation and test sets (3059 examples)

provides a brief introduction to both SVMs and GP classifiers, and definitive references for the interested reader. We do however present in this section aspects of our experiments that are specific to our own work, particularly regarding the measurement of performance and the selection of algorithm-specific parameters.

In the following, we denote by $\mathbf{x}$ a vector of features corresponding to a single problem as described in Section 2.2, and by y the corresponding label. We denote a data set having n labelled examples by $\mathbf{s}$ as in (1). We will denote the heuristic to which a set corresponds by adding a numerical subscript, and the partition to which it corresponds by adding the superscript 'train', 'val' or 'test' so, for example, the validation set for heuristic 2 is denoted $\mathbf{s}_2^{\text{val}}$.

3.1 Measurement of Performance

We use three measures of performance in our experiments. Denote by P^+ the number of true positives obtained using a validation or test set $\mathbf{s}$ of size m. Similarly, denote by P^- the number of false positives, N^+ the number of true negatives, and N^- the number of false negatives. The first measure is the *accuracy*

$$\text{acc}(\mathbf{s}) = \frac{P^+ + N^+}{m}.$$

While this is a common performance measure, it is not the most informative when applied to problems having unbalanced classes, such as the problems considered here. We therefore also employ the *Matthews correlation coefficient* (Baldi et al. [2])

$$M(\mathbf{s}) = \frac{P^+N^+ - P^-N^-}{\sqrt{(P^+ + P^-)(P^+ + N^-)(N^+ + P^-)(N^+ + N^-)}}$$

where the denominator is set to 1 if any sum term is zero. This measure has the value 1 if perfect prediction is attained, 0 if the classifier is performing as a random classifier, and -1 if the classifier exactly disagrees with the data. Finally, we use the *F1 score* (He [22]). Define the *precision* $p = P^+/(P^+ + P^-)$ and the *recall* $r = P^+/(P^+ + N^-)$. The F1 score is

$$\text{F1}(\mathbf{s}) = \frac{2pr}{p + r}$$

and takes values between 0 (worst) and 1 (best).

3.2 Support Vector Machines

We used the software *SVMLight* (Joachims [25]) in our experiments. In order to apply an SVM, a specific kernel function must be chosen. (See Appendix B.1 for further information.) On the basis of preliminary experiments (see Bridge [4]) the *radial basis function (RBF)* kernel, defined as

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma ||\mathbf{x} - \mathbf{x}'||^2\right) \tag{2}$$

was selected. The RBF kernel function has a single parameter γ, and the SVM fitting process is controlled by two further parameters. The parameter C governs the trade-off between margin and training error, and thus the extent to which we are willing to tolerate misclassified training examples. (This is discussed further in Appendix B.1; it is an example of a *regularization* parameter.) The *cost factor* j sets the relative weights for positive and negative samples; setting j appropriately is important when dealing with imbalanced data as

mentioned above and illustrated in Table 4. In our experiments the cost factor was set using the method of Morik et al. [33] as

$$j = \frac{\text{Number of negative examples}}{\text{Number of positive examples}}.$$

A grid-search optimisation procedure was used to find values for γ and C; the procedure used is essentially a standard one for the experimental training of SVMs (Hsu et al. [23]) and involves searching over a range of (γ, C) values, selecting the pair leading to a maximum in an estimate of the corresponding performance; in this case the accuracy, F1 or Matthews score. In order to estimate performance we applied 10-fold stratified cross-validation (Kohavi [27]) to the combination of the training and validation sets as follows. The combined set $\mathbf{s}^{\text{CV}} = \mathbf{s}^{\text{train}} \cup \mathbf{s}^{\text{val}}$ was split into 10 subsets of approximately equal size, maintaining the relative numbers of positive and negative examples in each subset. Denote by $\mathbf{s}_i^{\text{CV}}$ the ith subset and by $\mathbf{s}_{-i}^{\text{CV}}$ the combination of the remaining subsets when the ith has been removed. Then, for each subset in turn, an SVM was trained using $\mathbf{s}_{-i}^{\text{CV}}$ and P^+, P^-, N^+ and N^- were found using $\mathbf{s}_i^{\text{CV}}$ as a test set. The latter values were accumulated over the 10 splits and the final values used to obtain a performance estimate for a pair (γ, C).

The final classifiers in each case were produced by setting C and γ to their optimal values, and then training a single SVM using the whole of $\mathbf{s}^{\text{CV}}$. Testing to establish a final performance value was performed using $\mathbf{s}^{\text{test}}$.

The value of γ was varied between 2^{-15} and 2^5 in logarithmic steps, the value doubling at each step. Similarly, the value of C was varied between 2^{-5} and 2^{15}. The process was not repeated with finer steps in the values of C and γ as the results indicated that the peaks in estimated performance were not sharp. Additionally, due to the use of a finite number of samples for both learning and validation the variation of the performance measure on a small scale is not smooth, and so it is not possible to continuously refine the values of C and γ in the same manner as might be used to find the maximum of a smooth mathematical function.

This procedure was repeated for the full feature set, the static feature set alone and for the dynamic feature set alone.

3.3 Gaussian Process Classifiers

We used the *GPML* library (Rasmussen and Williams [36]) in our experiments. Some initial experiments suggested that results were rather insensitive to the mean and covariance functions used. We therefore used the zero mean function in conjunction with the *squared exponential* function

$$\text{cov}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2l^2}\right)$$

as it appears to be a common and effective choice, and as its similarity to the RBF kernel provides for some degree of comparability with the SVMs used. It has already been noted that we conduct our experiments using the full set of features, and also the static or dynamic features only. With the selection of good features in mind, we in fact employed the squared exponential covariance function with *automatic relevance determination (ARD)*

$$\text{cov}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \mathbf{H}^{-2}(\mathbf{x} - \mathbf{x}')\right)$$

where $\mathbf{H} = \text{diag}[h_1\ h_2 \ldots h_m]$ and m is the number of features. The ARD principle involves estimating the m hyperparameters h_i (see Appendix B), which then provide an indication

of the importance of the corresponding features. If a large value is obtained for h_i, this is an indication that the similarity measure between a pair of feature vectors $\mathbf{x}$ and $\mathbf{x}'$ is insensitive to differences in the ith feature. The feature is therefore of limited relevance, and the value of $1/h_i$ provides a direct measure of the relevance of feature i.

We used the Laplace inference method as this proved more stable than the Expectation Propagation alternative and moderately better performing than Variational Bayes. We used the logistic likelihood function. Hyperparameters $[\sigma^2 \ h_1 \ldots h_m]$ were initialized to 0 for the purposes of optimization, and optimization was limited to 100 function evaluations; increasing this limit offered little improvement.

The model selection and hyperparameter estimation tasks for a GP classifier are typically addressed by maximizing the marginal likelihood. This is the approach taken here, and it does not require us to use a validation set as is the case when constructing an SVM. We did however find that a validation set was useful for the following reason. As a GP classifier produces an output indicating the probability that some $\mathbf{x}$ should be classified $+1$ it is usual to choose a threshold t for the output and to assign $\mathbf{x}$ to $+1$ when the output is greater than t or -1 otherwise. Often t is set to 0.5, however we found that this is often a suboptimal value and therefore used the validation set to optimize it. Specifically, for a given performance measure the value of t was varied between 0 and 1 and the corresponding measure computed for the validation set. The value for t maximizing this performance was then used as a final value for testing. It is unusual to see this method applied in the design of SVMs, where outputs produced are arbitrary reals and a threshold of 0 is used as standard. Consequently optimization of the threshold was not applied to the SVMs. Appendix C provides some further discussion of this method, demonstrating that: the uniform use of $t = 0.5$ leads to reduced performance; optimal values of t for measures $M(\mathbf{s})$ and $\mathrm{F1}(\mathbf{s})$ tend to coincide; and accuracy is, as expected, an inappropriate performance measure for this data.

4 Experiments

4.1 Experiment 1: Classifier Performance

Our first experiment was a direct application of the classifiers to the data, resulting in a set of six classifiers, each predicting whether or not the corresponding heuristic should be selected for a given problem. At this stage the aim is merely to show that the learned predictions can be better than if a random classifier were used; that is, that something has been gained from the learning process.

In addition to training the classifiers using all features we trained classifiers using only the static features, and using only the dynamic features in order to establish whether one set appeared more useful than the other.

4.2 Experiment 2: Combining Classifiers for Heuristic Selection

In order to meaningfully be applied to general theorem proving problems, the separate classifiers need to be combined into a heuristic selection process. If the classifiers were perfect then for any problem one, and only one, classifier would return a positive result, with heuristic 0 indicating a problem that cannot be proved within the time limit by any of heuristics 1 to 5. With perfect classifiers, selecting a heuristic is just a case of observing which heuristic classifier gives a positive result.

In practice the classifiers are very unlikely to be perfect. For some problems more than one classifier will return a positive result, while for others no classifiers may return a positive result. A mechanism is needed to deal with such cases. With both SVM and GP classifiers there is a natural way to achieve this. The output $f(\mathbf{x})$ is in both cases a real value; arbitrary in the case of SVMs, and in the interval [0, 1] for GPs. In both cases we choose a threshold t, typically 0 for SVMs or 0.5 for GPs, and assign an input $\mathbf{x}$ to $+1$ if $f(\mathbf{x}) > t$ or to -1 otherwise. The distance

$$d_t(\mathbf{x}) = f(\mathbf{x}) - t$$

provides a natural measure of the commitment a classifier makes to labelling $\mathbf{x}$ as $+1$, and thus in choosing which of heuristics 1 to 5 to employ we simply select the one for which $d_t(\mathbf{x})$ is largest. In our experiments this includes cases where $d_t(\mathbf{x})$ is negative for all five classifiers.

Incorporating heuristic 0 introduces some further complication. Preliminary experiments showed that treating it in common with the other heuristics, and thus selecting it when it had the largest $d_t(\mathbf{x})$ even when that value was negative, led to far too many problems being rejected. Heuristic 0 was therefore only selected if $d_t(\mathbf{x})$ was positive as well as being greater than that for all other classifiers. Experiments were run both with and without heuristic 0 being a candidate for selection.

5 Results

5.1 Classifier Performance

Table 6 shows the best performance measures obtained when training SVMs and GPs as described above. Here, we assessed the performance of the classifiers on the test set, employing values for (γ, C) and thresholds chosen using the validation set.

Two conclusions can be drawn from these results. First, there is little difference in performance between SVMs and GPs, although we might argue that SVMs appear slightly preferable for F1 and Matthews scores while GPs appear preferable in terms of accuracy. Second, the performance changes only negligibly according to which of the three sets of features is used, although we might argue that there is some degradation when using dynamic features only, particularly in the Matthews score.

We chose not to analyse these results further, for example by examining the statistical significance of the apparent changes according to which set of features is used, as it is not our aim here to focus on classifier performance, either relative (SVM versus GP) or absolute; rather, we want to know how the choices made by the classifiers affect theorem proving, and we address this in the next subsection.

Two final points are perhaps noteworthy. First, while optimization of the threshold for GP classifiers is rarely undertaken in the literature, it turns out to be a critical step for this problem. We will not elaborate further on this point here, however Appendix C presents a full discussion. Second, while it appears that little is to be gained by the inclusion of dynamic features, this is perhaps to be expected given our earlier discussion in Section 2.2 regarding the need to use a single heuristic to generate them.

5.2 Combining Classifiers for Heuristic Selection

Table 7 shows how the classifiers perform when used to select heuristics. Once again we provide results for classifiers trained using all features, static features only, and dynamic

Table 6 Performance measured using accuracy, F1 score and Matthews coefficient of individual SVM and GP classifiers, using optimized (γ, C) for SVMs and optimized thresholds for GPs

Support vector machines

	All features			Static			Dynamic		
Heuristic	Acc	F1	Matt	Acc	F1	Matt	Acc	F1	Matt
0	0.81	0.77	0.61	0.80	0.75	0.59	0.79	0.74	0.57
1	0.76	0.48	0.35	0.77	0.45	0.31	0.81	0.36	0.26
2	0.92	0.30	0.23	0.92	0.29	0.22	0.92	0.29	0.22
3	0.86	0.47	0.39	0.82	0.43	0.35	0.87	0.42	0.33
4	0.89	0.40	0.32	0.87	0.42	0.34	0.89	0.42	0.34
5	0.88	0.39	0.32	0.85	0.40	0.34	0.88	0.38	0.30

Gaussian process classifiers

	All features			Static			Dynamic		
Heuristic	Acc	F1	Matt	Acc	F1	Matt	Acc	F1	Matt
0	0.80	0.73	0.57	0.79	0.72	0.56	0.71	0.71	0.47
1	**0.84**	0.45	0.34	**0.83**	0.44	0.32	**0.83**	**0.45**	**0.29**
2	*0.92*	0.23	0.19	*0.92*	0.23	0.15	*0.92*	0.21	0.12
3	**0.88**	0.44	0.35	**0.88**	0.39	0.28	**0.88**	0.40	0.32
4	**0.90**	0.37	0.31	**0.89**	0.37	0.33	*0.89*	0.38	0.31
5	**0.90**	0.33	0.27	**0.90**	0.32	0.26	**0.90**	0.24	0.17

Results are shown using all features, and the static and dynamic features individually. In the GP results, italic text indicates that the result is equal to the SVM result, and bold text indicates that the result improves on the SVM result

features only. We also give results when heuristic 0 is not used, when it is treated in common with heuristics 1 to 5, and when it is only selected if in addition it has positive $d_t(\mathbf{x})$. The time in seconds includes 100 seconds for each failed proof. For the SVM the results are

Table 7 Performance of classifiers when used for selection of heuristics

SVM with γ and C optimized on $\mathbf{s}_i^{\text{val}} \cup \mathbf{s}_i^{\text{train}}$ and performance assessed using $\mathbf{s}_i^{\text{test}}$ only.

	No H0		With H0		H0, positive margin	
	Number	Time	Number	Time	Number	Time
All	827/827	73,549/73,549	700/709	22,003/23,005	716/741	23,178/25,593
Static	833/822	72,845/74,350	726/718	26,784/28,093	739/728	28,608/29,116
Dynamic	810/809	75,268/75,286	667/666	23,152/23,075	702/703	27,946/27,815

GP with thresholds optimized using $\mathbf{s}_i^{\text{val}}$ and performance assessed using $\mathbf{s}_i^{\text{test}}$ only.

	No H0		With H0		H0, positive margin	
	Number	Time	Number	Time	Number	Time
All	816	74,973	720	35,264	724	36,467
Static	804	75,590	698	34,937	702	35,339
Dynamic	812	75,551	536	18,679	592	21,622

Numbers are explained within the text

shown in pairs. In each pair the first result was obtained with (γ, C) optimized using F1, and the second with (γ, C) optimized using Matthews.

Comparing first with the performance of individual, fixed heuristics as shown in the relevant (test set only) columns of Table 5, we see that when our learners are forced to make a choice of heuristic—that is, H0 is not available—they outperform any fixed heuristic in terms of both the number of theorems proved and the total time taken. The SVM trained using static features is the best-performing combination here, proving 833 theorems in 72,845 seconds.

Overall, the dynamic features once again appear to perform worse than the static or combined sets, the static features being preferable with the SVM and all features preferable with the GP. We might argue that the SVM performs slightly better than the GP, although there is no clear distinction.

When H0 is available, such that our system can decline to attempt a proof, we see only a moderate reduction in the number of theorems proved, but a much larger reduction in the time spent. Our classifiers are therefore effectively identifying problems for which the available heuristics are likely to be of limited effectiveness; to our knowledge, this ability has not previously been demonstrated. Selecting H0 only when it has positive margin leads, as expected, to more theorems being proved with a corresponding increase in total time taken.

6 A Further Experiment: Comparison with E's Auto Mode

It seems reasonable to ask how our approach compares with E's automatic selection of heuristics. This is complicated somewhat by the fact that E chooses from 82 possible heuristics whereas we have limited our work to 5. Any comparison is further complicated by the fact that we wished to use the same training, validation and test data as in the experiments already described. Recall that these data were generated using the times taken for heuristics H1 to H5 to prove each theorem, within an upper limit of 100 seconds. The data are therefore dependent on both the machine and the version of E used. The comparison we now describe was performed at a much later date than the construction of the three sets of data; consequently we no longer had use of the original hardware, and what follows has been constructed with this in mind in order to obtain a meaningful comparison.

For $n = 1, \ldots, 5$ let SVM_n be the SVM trained to predict whether heuristic n is the best to use and let $\mathrm{SVM}_n(\mathbf{x})$ be its output before thresholding when applied to a feature vector $\mathbf{x}$. Also, define

$$C_n = \{\mathbf{x} \in \mathbf{s}^{\mathrm{test}} | \mathrm{SVM}_n(\mathbf{x}) > \mathrm{SVM}_m(\mathbf{x}) \text{ for } m \neq n\},$$

the set of theorems in the test set that should, according to the SVMs, be addressed using the nth of the heuristics H1 to H5. We partition C_n into two further sets

$$H_n = \{\mathbf{x} \in C_n | \text{at least one of H1 to H5 proves } \mathbf{x}\}$$

and

$$\overline{H}_n = C_n \backslash H_n = \{\mathbf{x} \in C_n | \text{none of H1 to H5 proves } \mathbf{x}\}$$

denoting theorems provable by at least one of the SVM-selected heuristics, and theorems not provable by any of them, respectively when the data set was originally constructed.

Table 8 compares heuristic selection using SVMs trained with all features and limited to heuristics H1 to H5, versus E's auto mode using all 82 heuristics. Let S be the set corresponding to each row in the table; that is, $S = H_1$ for the first row and so on. Also, let H

Table 8 Comparison of heuristic selection using SVMs trained with all features and limited to heuristics H1 to H5, versus E's auto mode using all 82 heuristics

	Number proved		Time taken	
	E auto 82 heuristics	SVM 5 heuristics	E auto 82 heuristics	SVM 5 heuristics
H_1 (281)	270	259	1,670	3,084
H_2 (124)	114	107	1,127	1,883
H_3 (200)	198	195	497	799
H_4 (135)	126	123	1,346	1,650
H_5 (166)	152	146	1,969	2,410
$\overline{H}_1$ (122)	12	1	11,123	12,157
$\overline{H}_2$ (105)	3	1	10,258	10,501
$\overline{H}_3$ (105)	4	3	10,213	10,418
$\overline{H}_4$ (86)	3	0	8,477	8,600
$\overline{H}_5$ (206)	0	1	20,600	20,593
Totals	882	836	67,278	72,094

Numbers in brackets show the size of the corresponding set. See the text for a detailed explanation of the remaining numbers

denote the SVM-selected hypothesis corresponding to the row, so for example on the row for $S = \overline{H}_3$ we have $H =$ H3. The columns in Table 8 are interpreted as follows:

- For the E in auto mode columns, we show the number of theorems in S that E can prove, and the time taken by E to prove these theorems, including 100 seconds each for theorems in S that it fails to prove.
- For the SVM columns, we show the number of theorems in S that the H can prove, and the corresponding time taken, again including 100 seconds for each unproved theorem.

Note that while the data used to define the sets H_n and $\overline{H}_n$ was the original data, as described above, the entries on the table were generated using the currently available machine. For this reason, the number of theorems proved by the SVMs for the $\overline{H}_n$ sets can be non-zero—in a small number of cases the faster machine allows a theorem to be proved within 100 seconds using one of heuristics H1 to H5, which previously could not.

It is immediately apparent that, while E outperforms our method in this comparison, the difference is in fact rather slight; in particular, it should be noted that E has at least three significant advantages in this experiment:

- E's auto mode has access to 82 heuristics, and our method has reduced this to 5. Some of the 82 will have been indispensable for proving certain theorems, and it is likely that many of E's full complement are there to address rare special cases within the TPTP library.
- Our approach is fully automatic: there is no human intervention in the tuning process, whereas such intervention was required in constructing E's method for selecting heuristics.
- E was optimized using an earlier version of the entire TPTP library, potentially including problems in $\mathbf{s}^{\text{test}}$, whereas our method learns without access to any of $\mathbf{s}^{\text{test}}$.

The last of these points merits further explanation. In effect, the test set used by E in this case is not independent; E has a built in advantage, having had access to at least some of the test set during its design, whereas our method is selecting heuristics for TPTP problems of which it has no prior knowledge. (We are assuming here that our split of the data did not result in a test set containing no problems from the version of the TPTP library used in optimizing E; this would seem to be extremely unlikely.) In machine learning terms, it is likely that some of E's performance lead is the result of *overfitting* (see Bishop [3]).

7 Discussion and Further Work

7.1 Finding Optimal Feature Sets

The work presented in this paper was originally motivated by a discussion regarding applications of machine learning to theorem proving that might fruitfully be explored (Schulz, private email communication). Specifically:

> "Use meta-learning on the proofstate to recognize ... strategies over time ... there is a tantalizing result that proof states leading to a proof have, after a few seconds, a much lower sharing factor (on average) than proof searches that fail."

We have demonstrated that machine learning can effectively be applied to theorem proving. While we have not studied the sharing factor (dynamic feature number 2) specifically in terms of its effectiveness in identifying a heuristic, we have found that dynamic features in general have provided little or no increase in performance when used in addition to static features. However, it would be interesting to conduct more general further work on the selection of good features.

Modern machine learning methods can be tolerant to the use of large numbers of features, even if some are redundant. Even so, results may be improved by selecting an optimal set of features to use. Determining which features are significant may also provide useful information as to which aspects of a problem are important in heuristic development. From the results above it is clear that the full set of features can be reduced without making a major negative impact on performance, either in terms of classification or theorem-proving. It is also clear that the possible reduction in the number of features may be considerable—51 features for the full set versus 13 for the static features only.

It is infeasible to consider all possible subsets of features in a systematic way. However there are a number of approaches that may be taken to feature selection (Guyon and Elisseeff [18]). With the SVM machine learning approach, the selection of optimal features was investigated in our early work by removing features one at a time, observing the effect this had on the whole machine learning and heuristic selection process, and then determining which feature to permanently remove. One conclusion was that only a few features appear necessary, and it thus became feasible to do an exhaustive test of all subsets of up to 3 features. The results of this investigation can be found in [4]. We do not reproduce them here as they were obtained using an incompatible experimental setup.

It was with these early results in mind that the GP classifiers in our experiments were trained using the squared exponential ARD kernel—this kernel can provide an automatic assessment of the significance of each feature. However, in the results obtained to date there is little clear indication that any fixed small, subset of good features is identified by this approach. Nonetheless, feature selection in this problem remains an interesting area for future work, with both SVMs and GPs. We would particularly like to apply the more robust

approach of Chu et al. [5] in the context of GPs, and the *Multiple Kernel Learning (MKL)* technique (Lanckriet et al. [28]) as a means of identifying good related sets of features, rather than individual features, as in the work of Pilkington et al. [35].

In addition to determining the best subset of features from our existing set it would be interesting to consider adding further features to those under consideration; for example, properties of the signature such as maximum and average arity or number of symbols, or the use of absolute values rather than ratios in defining dynamic features.

7.2 Alternative Class Labels

The learning methods used in our work were designed from the outset to perform binary classification using two specified labels to denote the classes. It is of interest to ask whether improved results might be obtained if a more subtle labelling of classes were used, either by modifying the SVM and GP algorithms, or by applying algorithms designed specifically with such labellings in mind. For example, rather than labelling an example as $+1$ if a heuristic is fastest and -1 otherwise, we could perhaps explore a real-valued labelling that is more informative in the case where two heuristics solve a problem in very similar times. Alternatively, as this problem can also be seen as a multi-class classification problem it might be of interest to explore methods such as the multi-class GP classifier of Williams and Barber [46].

7.3 Further Comparison with E's Auto Mode

In Section 6 we compared our learned selection of a heuristic with E's auto mode. It would perhaps be interesting to extend this by examining how our learner compares with E on the set of examples for which E itself only selects from our set of heuristics. This would provide an indication of whether the problems for which E selects a heuristic outside of our set are also problems for which our heuristics are ineffective.

7.4 Application of Heuristic H0

It would be interesting to explore whether our approach to learning heuristic H0—indicating that a problem is too difficult to be attempted—might be used in a manner similar to the way in which machine learning has been used by portfolio SAT solvers. (Portfolio SAT solvers are discussed in Appendix A.) Specifically, by learning to select such a heuristic for each of a collection of different FOL provers, we might use the resulting classifiers to select automatically a suitable solver for a given problem.

8 Conclusions

We applied two powerful machine learning techniques to the task of heuristic selection within a theorem-prover. We find that our learners perform better than any single heuristic to which they have access, in terms of both the number of theorems proved and the overall time taken. In addition, we find that their performance remains comparable to that of the prover's own selection method despite the fact that the latter has several advantages: significant human expertise was required in its design whereas our learners require no intervention; it has access to 82 heuristics whereas we use a subset of only 5 of these; and it is likely that it had access during the tuning process to at least some of the problems on which it was tested,

whereas the learners are assessed only on problems not seen during training. If we allow our system to decline to attempt a proof, we see only a moderate reduction in the number of theorems proved but a much larger reduction in the time required; to our knowledge this is an ability that has not previously been demonstrated. Finally, evidence is found to suggest that smaller subsets of features might provide comparable performance; this is a subject for future work.

Acknowledgments James Bridge acknowledges the support of the Engineering and Physical Sciences Research Council (EPSRC) under a Doctoral Training Account Studentship EP/P502365/1. We acknowledge the UCI Machine Learning Repository [1] for their efforts in making available such a valuable resource. We thank two anonymous reviewers for their careful reading and constructive criticism.

Appendix A: Machine Learning for SAT Solvers

Several *portfolio solvers* have used simple methods—typically some form of linear regression (see for example Bishop [3] and Hastie et al. [21])—in order to select a solver from a portfolio. Many use features ultimately derived from those suggested by Nudelman et al. [34].

Haim and Walsh [19] use a simple approach based on linear ridge regression to predict the cost of solving an instance. They divide their features into two kinds. First, features related to the structure of an instance and based on measures such as the average size of a clause or the fraction of binary clauses. Second, features related to the behaviour of the search process, and based on measures such as the size of the backjumps or the fraction of the variables unassigned when backtracking occurs.

SATzilla2007 (Xu et al. [47]) is a portfolio solver using ridge regression to predict the running time of a given algorithm. (This is often referred to as an *empirical hardness* approach.) It uses 48 features derived from those presented in [34] and a hierarchical learning method combining ridge regression with sparse multinomial logistic regression, the latter for predicting whether or not an instance is satisfiable. (A similar method is used by Haim and Walsh [20] who, instead of learning to select from a portfolio of solvers, learn to select from a portfolio of 9 restart strategies.) Further developments of SATzilla2007 employ a more sophisticated performance measure as an alternative to running time, and a more complex hierarchical classifier. SATzilla2012 (Xu et al. [49]) introduces further features (Xu et al. [48]); there are 138 features in total derived from the preprocessed CNF formulae, and including measurements related to size, graph structure, balance, presence of Horn formulae, DPLL probing, clause learning, survey propagation and other relevant properties. It again departs from the empirical hardness approach by employing multiple instances of the cost-sensitive classification model of Ting [44] to predict which of a pair of solvers is preferred.

Kadioglu et al. [26] describe a portfolio method that uses the same 48 basic features as SATzilla2007 in conjunction with the k-nearest neighbour (k-NN) learning algorithm [21]. They also explore the use of distance weighting in the k-NN approach, and a further method by which examples are clustered and a different value of k assigned to each cluster.

Finally, Samulowitz et al. [38] explore the use of multinomial logistic regression for solving *quantified Boolean formulae (QBFs)*. They address the problem of choosing variable selection heuristics while executing a modified version of the Davis-Putman-Logemann-Loveland algorithm (Davis et al. [6]). They employ 78 features including many that are also appropriate for SAT problems [34] but also some more specific to QBFs.

Appendix B: Machine Learning Methods

We use a standard notation when describing machine learning techniques, corresponding essentially to that of Bishop [3]. Scalars $x \in \mathbb{R}$ are denoted using lower case, vectors $\mathbf{x} \in \mathbb{R}^n$ using bold lower case and matrices $\mathsf{X} \in \mathbb{R}^{n \times m}$ using upper case and an alternative font. The transpose of $\mathbf{x}$ is denoted $\mathbf{x}^T$ and vectors are column vectors by default.

We denote a data set[11] having n labelled examples by

$$\mathbf{s}^T = [(\mathbf{x}_1, y_1)(\mathbf{x}_2, y_2) \ldots (\mathbf{x}_n, y_n)]$$

and we define the corresponding

$$\mathsf{X} = [\mathbf{x}_1\ \mathbf{x}_2 \ldots \mathbf{x}_n]$$

and

$$\mathbf{y}^T = [y_1 y_2 \ldots y_n]\,.$$

B.1 Support Vector Machines

Support vector machines (SVMs) are kernel-based sparse classifiers, and are described in detail in [41]. In this appendix we give a brief introduction. We try to keep the technical requirements to a minimum, assuming that the interested reader will refer to the suggested sources; in particular we do not describe in detail the theory of constrained optimization (see for example Luenberger [29]) required to fully understand the training algorithm.

The SVM method is based on transforming the feature space containing the feature vectors $\mathbf{x}$ to a new space, typically of much higher dimension, using a mapping Φ. The underlying idea is that by making a good choice of Φ we make it easier to separate the two classes using a hyperplane in the larger space. Further, we select the hyperplane that, in addition to separating the two classes, is as far away as possible from any $\mathbf{x}_i$ in the training set $\mathbf{s}$.

The general expression for the distance of a point $\Phi(\mathbf{x})$ to a hyperplane $f(\mathbf{x}) = 0$ in the transformed space is $|f(\mathbf{x})|$ where

$$f(\mathbf{x}) = \mathbf{w}^T \Phi(x) + b. \tag{3}$$

Here, $\mathbf{w}$ is normal to the hyperplane, and when $||\mathbf{w}|| = 1$ the offset b is the distance from the origin to the hyperplane. An SVM classifies a new input $\mathbf{x}$ by computing

$$y = \mathrm{sgn}(f(\mathbf{x})) \tag{4}$$

where the sgn function takes the value $+1$ for arguments greater than 0, and -1 otherwise. We can alternatively define the *margin* of a training example $(\mathbf{x}_i, y_i)$ as

$$M(\mathbf{x}_i) = y_i f(\mathbf{x}_i).$$

This quantity is positive when $\mathrm{sgn}(f(\mathbf{x}_i)) = y_i$ and negative otherwise; its magnitude corresponds to the distance of $\mathbf{x}_i$ from the hyperplane. The training process therefore involves choosing appropriate values for $\mathbf{w}$ and b, such that the corresponding hyperplane separates the classes and is as far away as possible from any training feature vector $\mathbf{x}_i$. This

[11] It is common in the machine learning literature to refer to a training *set* when the object in question is more correctly a *sequence*. This should not however be the cause of any confusion in what follows.

corresponds to maximizing the smallest margin, and thus can be described as an optimization procedure

$$\arg\max_{\mathbf{w},b}\left[\min_i M(\mathbf{x}_i)\right].$$

While this optimization problem is not in a form that is convenient to solve, we can rewrite it as

$$\arg\min_{\mathbf{w},b}\left[\frac{1}{2}||\mathbf{w}||^2\right] \tag{5}$$

subject to the constraints

$$M(\mathbf{x}_i) \geq 1 \text{ for } i = 1, \ldots, n. \tag{6}$$

This is a quadratic optimization with linear inequality constraints and can be solved by the standard method (Luenberger [29]) of introducing Lagrange multipliers $\alpha_i \geq 0$ and forming the Lagrangian

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n}\alpha_i(M(\mathbf{x}_i) - 1).$$

Differentiating with respect to $\mathbf{w}$ and b and setting to zero yields the dual problem

$$\arg\max_{\alpha}\left[\sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}_j)\right]$$

subject to the constraints

$$\alpha_i \geq 0 \text{ for } i = 1, \ldots, n$$

and

$$\sum_{i=1}^{n}\alpha_i y_i = 0.$$

This dual problem can be solved for α using any suitable numerical solver. One consequence of setting $\partial L(\mathbf{w}, b, \alpha)/\partial\mathbf{w} = 0$ is that we obtain the condition

$$\mathbf{w} = \sum_{i=1}^{n} y_i\alpha_i\Phi(\mathbf{x}_i). \tag{7}$$

It should in fact not be surprising that $\mathbf{w}$ admits such a representation. Informally, assume that $\mathbf{s}$ contains samples in both the positive and negative classes. Label the transformed vectors $\Phi(\mathbf{x})$ for the positive class $\mathbf{z}_i^+$ and those for the negative class $\mathbf{z}_j^-$. Any vector joining a point $\mathbf{z}_i^+$ to a point $\mathbf{z}_j^-$ must pass through the dividing hyperplane, so there must be a point lying in the dividing hyperplane given by

$$\mathbf{z}_k = \mathbf{z}_i^+ + \beta_k(\mathbf{z}_j^- - \mathbf{z}_i^+)$$

where β_k lies between 0 and 1. Taking all pairs $\mathbf{z}_i^+$ and $\mathbf{z}_j^-$ we can find a set of values for β_k such that the resulting points are on the dividing hyperplane. We can then represent any point on the dividing hyperplane as a linear combination of the $\mathbf{z}_k$, and this includes the vector $\mathbf{w}$ normal to and lying on the dividing hyperplane. A linear combination of the $\mathbf{z}_k$ is also a linear combination of the transformed vectors $\Phi(\mathbf{x}_i)$, and thus the α_i in (7) exist as claimed. We could of course restrict this argument to using only examples near the dividing hyperplane, in contrast to methods such as Rosenblatt's perceptron [37] where all examples are used, and this idea is made rigorous below.

Substituting (7) into (3) we see that the trained SVM can be expressed entirely in terms of α as

$$f(\mathbf{x}) = \sum_{i=1}^{n} y_i \alpha_i \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}) + b. \tag{8}$$

It is possible to show using the *Karush-Kuhn-Tucker* conditions [29] for the optimization problem that its solution has the property

$$\alpha_i(M(\mathbf{x}_i) - 1) = 0 \text{ for } i = 1, \ldots, n.$$

Comparing with the constraint (6) this implies that we have $\alpha_i \neq 0$ only for the examples—known as *support vectors*—for which $\Phi(\mathbf{x}_i)$ is closest to the hyperplane. Consequently the SVM is in practice a *sparse* technique.

Equation (8) hides a potential problem: values $\Phi(\mathbf{x})$ can exist in a space of high, perhaps even infinite dimension, and consequently the inner products $\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x})$ might be computationally intractable. We therefore avoid the direct computation of such inner products by introducing a *kernel function*

$$K_{\mathbf{p}}(\mathbf{x}, \mathbf{x}') = \Phi^T(\mathbf{x})\Phi(\mathbf{x}')$$

with associated parameter values $\mathbf{p}$. While it might be supposed that this offers little or no benefit, a remarkable theorem of Mercer (Mercer [31]) characterizes the functions Φ for which a corresponding K exists, and the kernel is usually much easier to compute than the explicit inner product. (We can in fact proceed without knowing what function Φ corresponds to a given K.) In addition, there are well-defined transformations allowing new kernels to be defined from known ones (see Shawe-Taylor and Cristianini [42]). Table 9 shows four of the kernel functions most commonly employed in applying SVMs. The linear kernel corresponds to an untransformed space such as that used by a linear perceptron. It has the advantage that no parameters need be set, and the corresponding limitation of inflexibility. The polynomial kernel generalizes the linear kernel but is still often insufficiently flexible for complex data. The sigmoid tanh and radial basis kernels provide better flexibility; in preliminary experiments (Bridge [4]) the best results for our problem were obtained with the radial basis kernel. Any parameters $\mathbf{p}$ associated with a kernel are usually learned by optimization using a validation set of training data (Hsu et al. [23]) as explained in Section 3.2.

Our final expression for a trained SVM is

$$y = \text{sgn}\left[\sum_{i=1}^{n} y_i \alpha_i K_{\mathbf{p}}(\mathbf{x}_i, \mathbf{x}) + b\right].$$

We have until now assumed that it is possible to find a hyperplane in the extended space that separates the positive from the negative examples. This is not always possible, and the

Table 9 Some common SVM kernel functions

Linear	Polynomial
$\mathbf{x}^T\mathbf{x}'$	$\left(s\mathbf{x}^T\mathbf{x}' + c\right)^d$
Sigmoid tanh	Radial Basis
$\tanh\left(s\mathbf{x}^T\mathbf{x}' + c\right)$	$\exp\left(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2\right)$

Table 10 Some common Gaussian process covariance functions

Squared exponential	γ-exponential
$\exp\left(-\frac{\|\|\mathbf{x}-\mathbf{x}'\|\|^2}{2l^2}\right)$	$\exp\left(-\left(\frac{\|\|\mathbf{x}-\mathbf{x}'\|\|}{l}\right)^{\gamma}\right)$
Rational quadratic	Neural network
$\left(1+\frac{\|\|\mathbf{x}-\mathbf{x}'\|\|^2}{2\alpha l^2}\right)^{-\alpha}$	$\frac{2}{\pi}\sin^{-1}\left(\frac{2\mathbf{x}^T\mathsf{S}\mathbf{x}'}{\sqrt{(1+2\mathbf{x}^T\mathsf{S}\mathbf{x})(1+2\mathbf{x}'^T\mathsf{S}\mathbf{x}')}}\right)$

SVM algorithm is therefore modified by the introduction of *slack variables* ϵ_i. Rewriting the basic optimization problem (5) and (6) as

$$\arg\min_{\mathbf{w},b}\left[\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\epsilon_i\right]$$

subject to the constraints

$$M(\mathbf{x}_i) \geq 1-\epsilon_i \text{ and } \epsilon_i \geq 0 \text{ for } i = 1,\ldots,n$$

allows misclassifications, with C setting the balance between maximizing margin and minimizing misclassifications. The parameter C is typically learned along with any kernel parameters $\mathbf{p}$ using the search procedure described above.

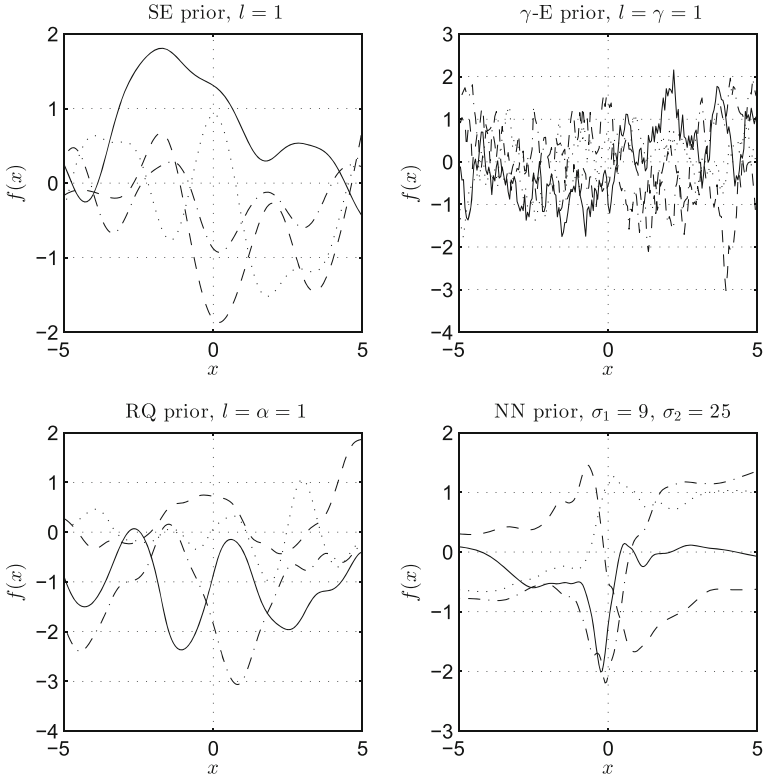

Fig. 1 Examples of functions drawn at random from Gaussian process priors using the covariance functions in Table 10. In the case of the NN prior the parameter matrix is $\mathsf{S} = \text{diag}(\sigma_1, \sigma_2)$

B.2 Gaussian Process Classifiers

Gaussian process classifiers (Rasmussen and Williams [36]) provide an alternative kernel-based approach to supervised learning forming part of the more general Bayesian framework (Bishop [3]). In this section we give a brief introduction to Bayesian supervised learning and Gaussian process classifiers. The presentation is deliberately brief, and the interested reader can find the details in [3, 36]; the relevant material on random processes and on probability can be found in Grimmett and Stirzaker [17].

Many supervised learning techniques are primarily a means of choosing a vector $\mathbf{z}$ of parameters associated with some function $f(\mathbf{x}; \mathbf{z})$; in the case of an SVM, the parameters are

$$\mathbf{z}^T = [\alpha\ b\ \mathbf{p}]$$

and the associated function is

$$f(\mathbf{x}; \mathbf{z}) = \operatorname{sgn}\left(\sum_{i=1}^{n} y_i \alpha_i K_{\mathbf{p}}(\mathbf{x}_i, \mathbf{x}) + b\right).$$

A training algorithm can be regarded as a function $T : \mathbf{s} \mapsto \mathbf{z}$ mapping the training examples to the parameters. This is however by no means the only way in which the problem can be cast.

The Bayesian framework takes a fundamentally probabilistic approach to learning. It can be shown (Duda et al. [11]) that to obtain the best possible *generalisation error*, defined as

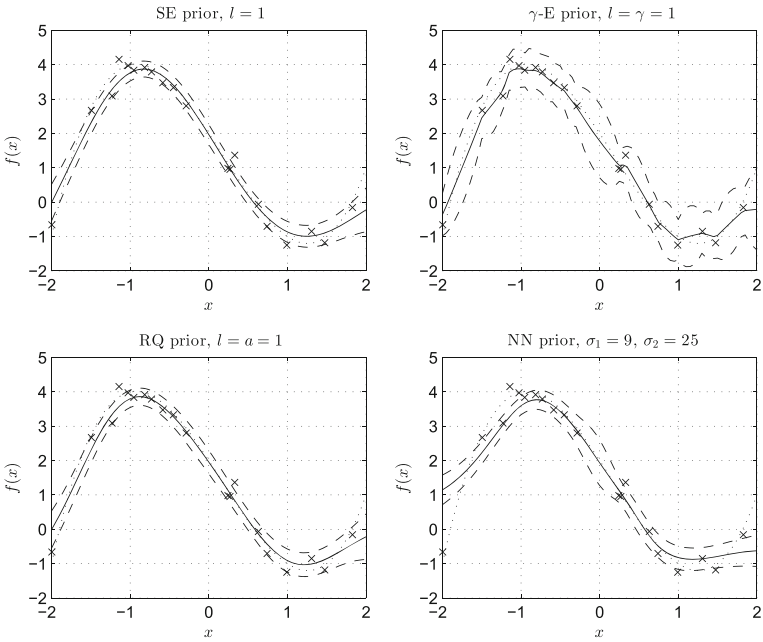

Fig. 2 The mean and variance of the predictive distribution for Gaussian process regression using the covariance functions in Table 10. The *dotted line* shows the polynomial $p(x)$ and the crosses show the training examples. See the text for an explanation of the *solid* and *dashed lines*

the probability of misclassifying a new example $(\mathbf{x}, y)$, we should use the *Bayes-optimal classifier*

$$f_{\text{Bayes}}(\mathbf{x}) = \begin{cases} +1 \text{ if } \Pr(y = +1|\mathbf{x}, \mathbf{s}) \geq 1/2 \\ -1 \text{ otherwise} \end{cases}.$$

A simplification is usually made at this point whereby we consider the feature vectors $\mathbf{x}$ to be *fixed*, rather than random variables. If we now wish to classify a new point $\mathbf{x}$ then the expression of interest becomes $\Pr(y = +1|\mathbf{y})$; however we will commit a slight abuse of notation in the hope of increasing the clarity of what follows for the non-specialist. We will write $\Pr(y = +1|\mathbf{y}; \mathbf{x}, \mathsf{X})$—the semicolon indicating that what follows are not to be considered random variables.

If we have a model such as an SVM or a neural network, having parameters $\mathbf{z}$, then we can compute the desired expression by noting that by the usual operation of computing a marginal distribution

$$\Pr(y|\mathbf{y}; \mathbf{x}, \mathsf{X}) = \int p(y, \mathbf{z}|\mathbf{y}; \mathbf{x}, \mathsf{X})\, d\mathbf{z}.$$

Using the definition of conditional probability we can split the integrand to obtain

$$\begin{aligned} \Pr(y|\mathbf{y}; \mathbf{x}, \mathsf{X}) &= \int \Pr(y|\mathbf{z}, \mathbf{y}; \mathbf{x}, \mathsf{X}) p(\mathbf{z}|\mathbf{y}; \mathbf{x}, \mathsf{X})\, d\mathbf{z} \\ &= \int \Pr(y|\mathbf{z}; \mathbf{x}) p(\mathbf{z}|\mathbf{y}; \mathsf{X})\, d\mathbf{z} \end{aligned} \tag{9}$$

where the simplification in the second line follows because $\mathbf{s}$ provides no additional information about y if we know $\mathbf{x}$ and $\mathbf{z}$, and $\mathbf{x}$ provides no additional information about $\mathbf{z}$ if we know $\mathbf{s}$. We know from Bayes' theorem that

$$p(\mathbf{z}|\mathbf{y}; \mathsf{X}) = \frac{1}{Z} \Pr(\mathbf{y}|\mathbf{z}; \mathsf{X}) p(\mathbf{z}; \mathsf{X}) \tag{10}$$

$$Z = \int \Pr(\mathbf{y}|\mathbf{z}; \mathsf{X}) p(\mathbf{z}; \mathsf{X})\, d\mathbf{z}. \tag{11}$$

It is usually assumed that examples are independent and identically distributed (i.i.d.) and hence

$$\Pr(\mathbf{y}|\mathbf{z}; \mathsf{X}) = \prod_{i=1}^{n} \Pr(y_i|\mathbf{z}; \mathbf{x}_i). \tag{12}$$

Table 11 Performance measured using accuracy, F1 score and Matthews coefficient of individual GP classifiers using a fixed threshold of $t = 0.5$

Heuristic	All features			Static			Dynamic		
	Acc	F1	Matt	Acc	F1	Matt	Acc	F1	Matt
0	0.75	0.71	0.50	0.77	0.73	0.53	0.72	0.70	0.45
1	0.84	0.28	0.27	0.83	0.16	0.18	0.83	0.31	0.27
2	0.91	0.16	0.19	0.92	0.04	0.09	0.92	0.14	0.17
3	0.88	0.28	0.28	0.88	0.22	0.23	0.88	0.29	0.28
4	0.90	0.04	0.08	0.90	0.10	0.16	0.90	0.04	0.09
5	0.90	0.04	0.07	0.90	0.01	0.01	0.90	0.06	0.11

Results are shown using all features, and the static and dynamic features individually

Table 12 Performance of GP classifiers with fixed $t = 0.5$ when used for selection of heuristics

GP with fixed $t = 0.5$ with performance assessed using $\mathbf{s}_i^{\text{val}}$ and $\mathbf{s}_i^{\text{test}}$ combined

	No H0		With H0		H0, positive margin	
	Number	Time	Number	Time	Number	Time
All	1582	153,434	958	20,882	1235	50,075
Static	1571	154,429	884	19,264	1257	49,345
Dynamic	1586	154,853	948	22,324	1139	41,821

GP with fixed $t = 0.5$ with performance assessed using $\mathbf{s}_i^{\text{test}}$ only

	No H0		With H0		H0, positive margin	
	Number	Time	Number	Time	Number	Time
All	810	74,742	490	10,289	631	22,794
Static	806	75,129	457	9,241	634	23,661
Dynamic	807	75,976	491	10,194	583	20,241

Comparing (9), (10) and (12) we see that (dropping the abusive notation) two fundamental quantities are required: the *prior* $p(\mathbf{z})$ and the *likelihood* $\Pr(y|\mathbf{z})$. The prior quantifies our uncertainty about what the parameter vector might be in the absence of any data, and generally corresponds to a *regularisation* term in the non-Bayesian approach. The likelihood quantifies our uncertainty about how labels might appear, and generally models *noise* in the data. Having specified these two quantities a classifier is constructed by evaluating the integral in (9). This is in general a non-trivial process. The details of how the prior and likelihood can be chosen, and how the task of integration can be achieved, can be found in [3].

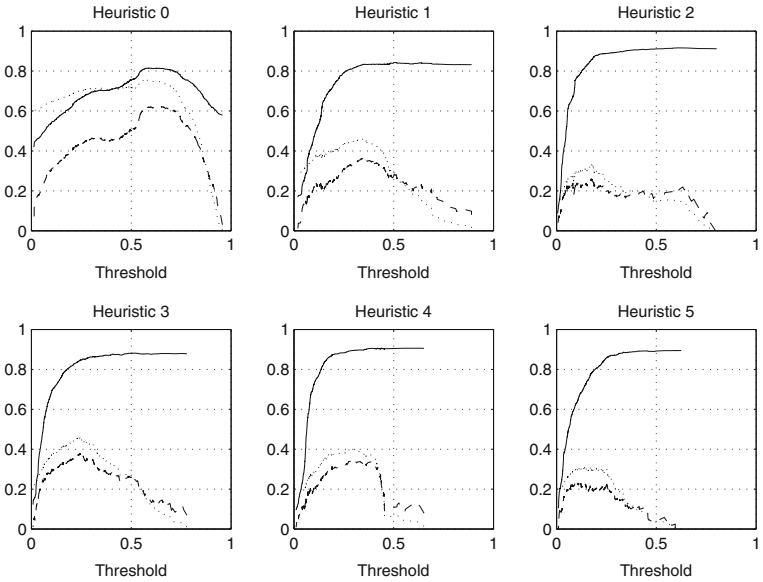

Fig. 3 Variation of performance measures with threshold t, evaluated using the validation sets, for classifiers trained using all features. The *solid line* shows accuracy, the *dotted line* F1 and the *dashed line* Matthews

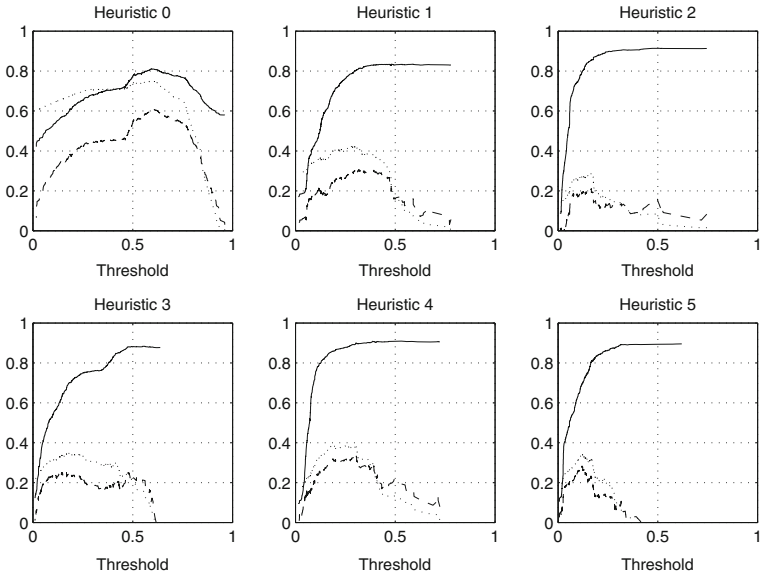

Fig. 4 Variation of performance measures with threshold t, evaluated using the validation sets, for classifiers trained using only static features. The *solid line* shows accuracy, the *dotted line* F1 and the *dashed line* Matthews

Gaussian process classifiers begin with the following observation: given that any parameter vector $\mathbf{z}$ specifies a function $f(\mathbf{x}; \mathbf{z})$, and we need to specify a prior $p(\mathbf{z})$ and a likelihood $\Pr(y|\mathbf{z})$, why not simply circumvent the need for parameters and work in terms of a prior $p(f)$ and a likelihood $\Pr(y|f)$ *defined directly in terms of functions*?

Definition 1 Let $\text{cov}(\mathbf{x}, \mathbf{x}')$ denote a covariance function[12] and let $\mu(\mathbf{x})$ denote a *mean function*. A random function f is called a *Gaussian process* if for any fixed, finite sequence $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ the vector

$$\mathbf{f}^T = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)]$$

has Gaussian density

$$p(\mathbf{f}) = (2\pi)^{-n/2}|\mathsf{C}|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{f}-\mu)^T \mathsf{C}^{-1}(\mathbf{f}-\mu)\right)$$

where C is the *covariance matrix* $C_{ij} = \text{cov}(\mathbf{x}_i, \mathbf{x}_j)$ and

$$\mu^T = [\mu(\mathbf{x}_1), \mu(\mathbf{x}_2), \ldots, \mu(\mathbf{x}_n)]$$

is the *mean vector*. We write $f \sim N(\mu, \text{cov})$ to denote that f is a Gaussian process.

Table 10 specifies four commonly-encountered covariance functions, and Fig. 1 shows, for each of these covariance functions, four samples from the corresponding GP with mean $\mu(\mathbf{x}) = 0$. These now correspond to the prior $p(f)$ introduced above.

[12] The covariance function takes the place of the kernel function used by an SVM. There are certain conditions that the covariance function must posses; see [36] for details.

For the case of regression (rather than classification) the process of inference is now quite straightforward. Assume that examples are modified by additive i.i.d. Gaussian noise ϵ with mean 0 and variance σ_{n}^2, so

$$y = f(\mathbf{x}) + \epsilon.$$

If f is a Gaussian process with mean μ and covariance cov then any y value generated in this way must have mean $\mu(\mathbf{x})$ and variance $\mathrm{cov}(\mathbf{x}, \mathbf{x}) + \sigma_{\mathrm{n}}^2$. Similarly, starting with any finite sequence of feature vectors $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ and generating the corresponding y values leads to a vector $\mathbf{y}$ having a Gaussian density with mean μ where $\mu_i = \mu(\mathbf{x}_i)$ and covariance matrix C where $C_{ij} = \mathrm{cov}(\mathbf{x}_i, \mathbf{x}_j) + \sigma_{\mathrm{n}}^2 \delta_{i,j}$.

Assume now we have a training sequence $\mathbf{s}$ and we wish to predict the output required for a new input $\mathbf{x}_{\mathrm{new}}$. By the same argument as in the previous paragraph, the joint density $p(y_{\mathrm{test}}, \mathbf{y})$ is Gaussian with mean $(\mu(\mathbf{x}_{\mathrm{test}}), \mu)$ and covariance

$$\mathsf{C}' = \begin{pmatrix} \mathrm{cov}(\mathbf{x}_{\mathrm{new}}, \mathbf{x}_{\mathrm{new}}) & \mathbf{c}^T \\ \mathbf{c} & \mathsf{C} \end{pmatrix}$$

where $\mathbf{c} = (\mathrm{cov}(\mathbf{x}_{\mathrm{test}}, \mathbf{x}_1), \ldots, \mathrm{cov}(\mathbf{x}_{\mathrm{test}}, \mathbf{x}_n))$. Referring back to (9), we want to compute the conditional distribution $p(y_{\mathrm{test}}|\mathbf{y})$. However, as we have just shown that the joint $p(y_{\mathrm{test}}, \mathbf{y})$ is Gaussian, *the conditional distribution is Gaussian also*. The identities required to compute the relevant mean and covariance are as follows. Let $\mathbf{x}$ be a vector of n jointly Gaussian-distributed random variables, having mean μ and covariance matrix Σ such that

$$p(\mathbf{x}) = (2\pi)^{-n/2} |\Sigma|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)\right).$$

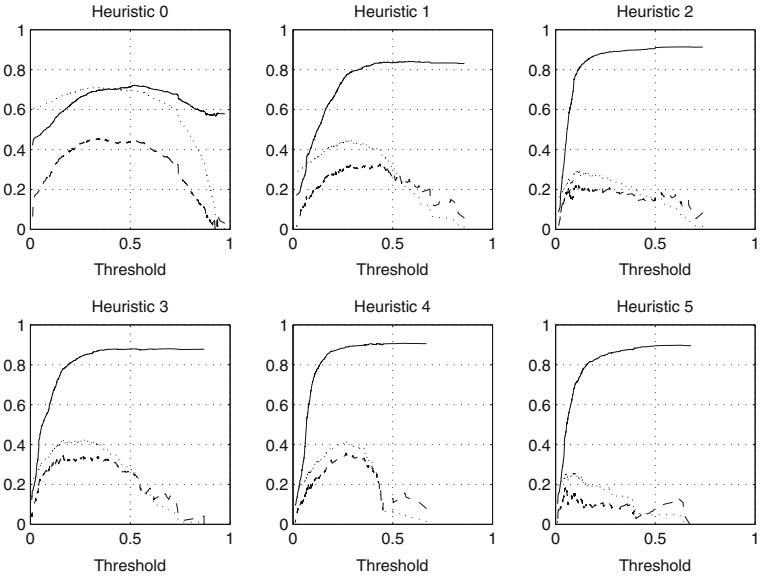

Fig. 5 Variation of performance measures with threshold t, evaluated using the validation sets, for classifiers trained using only dynamic features. The *solid line* shows accuracy, the *dotted line* F1 and the *dashed line* Matthews

If we partition $\mathbf{x}$ such that $\mathbf{x} = (\mathbf{x}_1\ \mathbf{x}_2)$ and write $\mu = (\mu_1\ \mu_2)$ and

$$\Sigma = \begin{pmatrix} \Sigma_1 & \Sigma_2 \\ \Sigma_2^T & \Sigma_3 \end{pmatrix}$$

in the corresponding partitioned form, then both the marginal density $p(\mathbf{x}_1)$ and the conditional density $p(\mathbf{x}_1|\mathbf{x}_2)$ are also Gaussian. Specifically, $p(\mathbf{x}_1)$ has mean μ_1 and covariance Σ_1, and $p(\mathbf{x}_1|\mathbf{x}_2)$ has mean

$$\mu' = \mu_1 + \Sigma_2 \Sigma_3^{-1}(\mathbf{x}_2 - \mu_2)$$

and covariance

$$\Sigma' = \Sigma_1 - \Sigma_2 \Sigma_3^{-1} \Sigma_2^T .$$

Applying these results gives the mean and variance for y_{test}, conditioned on the training examples, as

$$\begin{aligned} \mu_{\text{test}} &= \mu(\mathbf{x}_{\text{test}}) + \mathbf{c}^T \mathsf{C}^{-1}(\mathbf{y} - \mu) \\ \sigma^2_{\text{test}} &= \text{cov}(\mathbf{x}_{\text{test}}, \mathbf{x}_{\text{test}}) - \mathbf{c}^T \mathsf{C}^{-1} \mathbf{c}. \end{aligned}$$

Figure 2 shows a simple example of GP regression in one dimension, using the same covariance functions as illustrated in Fig. 1. In this figure the training data were generated by selecting x values uniformly at random in the interval $[-2, 2]$. The corresponding y values were then obtained by evaluating the polynomial

$$p(x) = x^3 - \frac{1}{2}x^2 - \frac{7}{2}x + 2$$

and adding Gaussian noise ϵ of zero mean and variance 0.1 such that $y = p(x) + \epsilon$. Each graph shows the same 20 training examples. The solid line shows the resulting interpolator μ_{test} for values of x in the relevant range, and the dashed lines show an interval of $2\sigma_{\text{test}}$

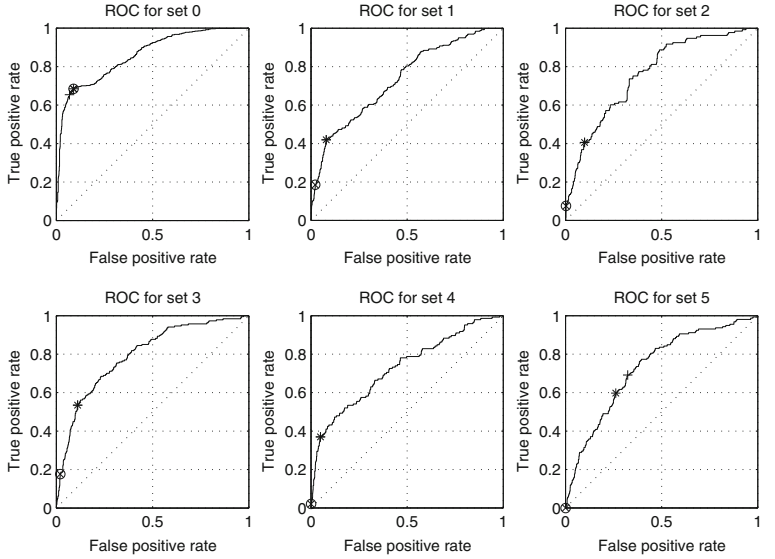

Fig. 6 ROC curves for the individual heuristic classifiers, evaluated using the validation sets, for classifiers trained using all features. In each case, a *circle* indicates the optimal ROC operating point, a *cross* the point corresponding to maximum accuracy, a *star* maximum F1, and a *plus* maximum Matthews

above and below μ_{test}. Note that the latter widen in areas where there is little data, and thus give an indication of the confidence of our prediction.

Just as for SVM kernels, GP covariance functions may posses one or more parameters $\mathbf{p}$. These are typically set by maximizing the *marginal likelihood* or *evidence*

$$E(\mathbf{p}) = \log p(\mathbf{y}|\mathbf{p}; \mathsf{X})$$

using some variant of conjugate gradient search; details can be found in [36].

In order to extend the method to classification problems we begin with a Gaussian process f and define

$$\Pr(y = +1|f; \mathbf{x}) = \sigma(f(\mathbf{x}))$$

where σ can be any function that sensibly maps the range of f to the interval [0, 1]. This new process is then used much as above, although we have the added complication that as f itself is not directly observed it must be integrated out of the equations. Unfortunately the resulting computations are rarely analytically tractable and we therefore need to resort to approximation techniques; again we invite the interested reader to consult [36] for the details.

Appendix C: The effect of optimizing the GP threshold

Tables 11 and 12 show results corresponding to Tables 6 and 7 for GP classifiers when a fixed threshold of $t = 0.5$ is used, as is generally the case in the literature. Note that classification performance is reduced, particularly when measured by the F1 and Matthews scores, which in this work are more relevant than accuracy as we have imbalanced data. Comparing also with Table 5 we see that while we still outperform any fixed heuristic if H0 is not included as an option, performance is reduced when H0 is included.

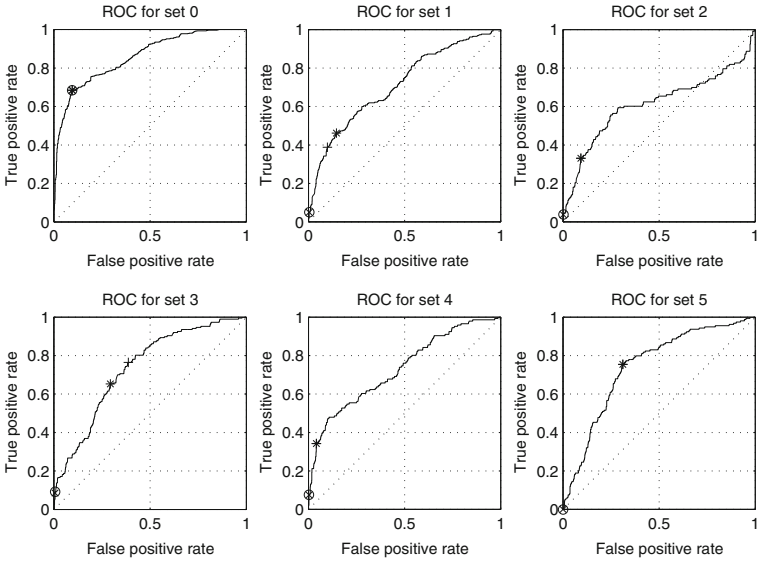

Fig. 7 ROC curves for the individual heuristic classifiers, evaluated using the validation sets, for classifiers trained using only static features. In each case, a *circle* indicates the optimal ROC operating point, a *cross* the point corresponding to maximum accuracy, a *star* maximum F1, and a *plus* maximum Matthews

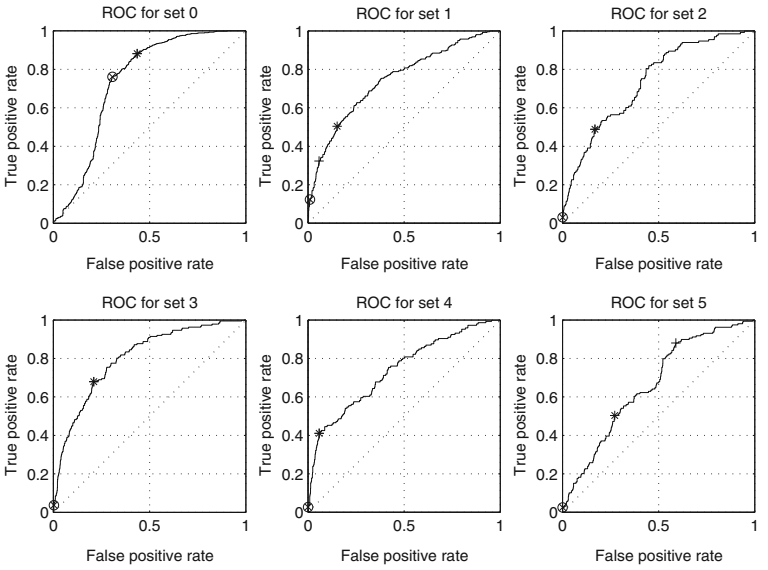

Fig. 8 ROC curves for the individual heuristic classifiers, evaluated using the validation sets, for classifiers trained using only dynamic features. In each case, a *circle* indicates the optimal ROC operating point, a *cross* the point corresponding to maximum accuracy, a *star* maximum F1, and a *plus* maximum Matthews

Figures 3, 4 and 5 show how the accuracy, F1 and Matthews measures vary with the threshold t when measured using the validation sets, for the six individual classifiers, and for the three sets of features. Figures 6, 7 and 8 show the ROC curves (Fawcett [13]) for the classifiers, again evaluated using the validation sets, and for all three sets of features. Clearly the imbalance in the data leads to accuracy being a somewhat uninformative performance measure; however it is also clear that the peaks in F1 and Matthews scores tend to correspond, and thus both measures lead to the same choice of threshold.

References

1. Bache, K., Lichman, M.: UCI Machine Learning Repository (2013).http://archive.ics.uci.edu/ml
2. Baldi, P., Brunak, S., Chauvin, Y., Anderson, C.A.F., Nielsen, H.: Assessing the accuracy of prediction algorithms for classification: an overview. Bioinformatics **16**(5), 412–424 (2000)
3. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer-Verlag (2006)
4. Bridge, J.P.: Machine Learning and Automated Theorem Proving. Tech. Rep. UCAM-CL-TR-792, University of Cambridge, Computer Laboratory (2010). http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-792.pdf
5. Chu, W., Ghahramani, Z., Falciani, F., Wild, D.L.: Biomarker discovery in microarray gene expression data with Gaussian processes. Bioinformatics **21**(16), 3385–3393 (2005)
6. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962). doi:10.1145/368273.368557
7. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960). doi:10.1145/321033.321034
8. Denzinger, J., Fuchs, M., Fuchs, M.: High performance ATP systems by combining several AI methods. In: Proceedings Fifteenth International Joint Conference on Artificial Intelligence (IJCAI) 1997, pp. 102–107. Morgan Kaufmann (1997)
9. Denzinger, J., Fuchs, M., Goller, C., Schulz, S.: Learning from Previous Proof Experience. Technical Report AR99-4, Institut für Informatik, Technische Universität München (1999)

10. Denzinger, J., Kronenburg, M., Schulz, S.: Discount - a distributed and learning equational prover. J. Autom. Reason. **18**, 189–198 (1997). doi:10.1023/A:1005879229581
11. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. Wiley (2000)
12. Erkek, C.A.: Mixture of Experts Learning in Automated Theorem Proving. Master's thesis, Bogazici University (2010)
13. Fawcett, T.: An introduction to ROC analysis. Pattern Recogn. Lett. **27**, 861–874 (2006)
14. Fuchs, M.: Automatic selection of search-guiding heuristics for theorem proving. In: Proceedings of the 10th FLAIRS, pp. 1–5. Florida AI Research Society, Daytona Beach (1998)
15. Fuchs, M., Fuchs, M.: Feature-based learning of search-guiding heuristics for theorem proving. AI Commun. **11**(3–4), 175–189 (1998)
16. Goller, C.: Learning search-control heuristics for automated deduction systems with folding architecture networks. In: Proceedings European Symposium on Artificial Neural Networks. D-Facto publications (1999)
17. Grimmett, G., Stirzaker, D.: Probability and Random Processes. Oxford University Press (2001)
18. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
19. Haim, S., Walsh, T.: Online estimation of SAT solving runtime. In: Kleine Büning, H., Zhao, X. (eds.) Theory and Applications of Satisfiability Testing – SAT 2008, Lecture Notes in Computer Science, vol. 4996, pp. 133–138. Springer, Berlin (2008). doi:10.1007/978-3-540-79719-7_12
20. Haim, S., Walsh, T.: Restart strategy selection using machine learning techniques. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing - SAT 2009, Lecture Notes in Computer Science, vol. 5584, pp. 312–325. Springer, Berlin (2009). doi:10.1007/978-3-642-02777-2_30
21. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer Series in Statistics, 2nd edn. Springer (2009)
22. He, H.: Learning from imbalanced data. IEEE Trans. Knowl. Data Eng. **21**(9), 1263–1284 (2009)
23. Hsu, C.W., Chang, C.C., Lin, C.J., et al.: A practical guide to support vector classification. Tech. rep., Department of Computer Science, National Taiwan University (2003)
24. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems, 2nd edn. Cambridge University Press (2004)
25. Joachims, T.: Making large-scale SVM learning practical. In: Schölkopf, B., Burges, C., Smola, A. (eds.) Advances in Kernel Methods - Support Vector Learning, chap. 11, pp. 169–184. MIT Press, Cambridge, MA (1999)
26. Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm selection and scheduling. In: Lee, J. (ed.) Principles and Practice of Constraint Programming – CP 2011, Lecture Notes in Computer Science, vol. 6876, pp. 454–469. Springer, Berlin (2011). doi:10.1007/978-3-642-23786-7_35
27. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95), vol. 2, pp. 1137–1143. Morgan Kaufmann (1995)
28. Lanckriet, G.R.G., Bie, T.D., Cristianini, N., Jordan, M.I., Noble, W.S.: A statistical framework for genomic data fusion. Bioinformatics **20**(16), 2626–2635 (2004)
29. Luenberger, D.G.: Linear and Nonlinear Programming. Kluwer (2003)
30. McCune, W.: Prover9 and Mace4 (2005–2010). http://www.cs.unm.edu/~mccune/prover9/
31. Mercer, J.: Functions of positive and negative type and their connection with the theory of integral equations. Philos. Trans. R. Soc. Lond. **209**, 415–446 (1909)
32. Mitchell, T.: Machine Learning. McGraw Hill (1997)
33. Morik, K., Brockhausen, P., Joachims, T.: Combining statistical learning with a knowledge-based approach – a case study in intensive care monitoring. In: International Conference on Machine Learning (ICML), pp. 268–277. Bled, Slowenien (1999)
34. Nudelman, E., Leyton-Brown, K., Hoos, H., Devkar, A., Shoham, Y.: Understanding random SAT: Beyond the clauses-to-variables ratio. In: Wallace, M. (ed.) Principles and Practice of Constraint Programming – CP 2004, Lecture Notes in Computer Science, vol. 3258, pp. 438–452. Springer, Berlin (2004). doi:10.1007/978-3-540-30201-8_33
35. Pilkington, N.C.V., Trotter, M.W.B., Holden, S.B.: Multiple kernel learning for drug discovery. Mol. Inform. **31**(3–4), 313–322 (2012)
36. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA (2006)
37. Rosenblatt, F.: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books (1962)

38. Samulowitz, H., Memisevic, R.: Learning to solve QBF. In: Proceedings of the 22nd National Conference on Artificial Intelligence - AAAI'07, vol. 1, pp. 255–260. AAAI Press (2007). http://dl.acm.org/citation.cfm?id=1619645.1619686
39. Schulz, S.: Learning Search Control Knowledge for Equational Deduction. No. 230 in DISKI. Akademische Verlagsgesellschaft Aka GmbH Berlin (2000)
40. Schulz, S.: E – a brainiac theorem prover. AI Commun. **15**(2/3), 111–126 (2002)
41. Shawe-Taylor, J., Cristianini, N.: Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge University Press, Cambridge (2000)
42. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
43. Sutcliffe, G.: The TPTP problem library and associated infrastructure: the FOF and CNF parts, v3.5.0. J. Autom. Reason. **43**(4), 337–362 (2009)
44. Ting, K.M.: An instance-weighted method to induce cost-sensitive trees. IEEE Trans. Knowl. Data Eng. **14**(3), 659–665 (2002)
45. Urban, J.: MaLARea: a metasystem for automated reasoning in large theories. In: Urban, J., Sutcliffe, G., Schulz, S. (eds.) Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories, no. 257 in CEUR Workshop Proceedings, pp. 45–58 (2007)
46. Williams, C.K.I., Barber, D.: Bayesian classification with Gaussian processes. IEEE Trans Pattern. Anal. Mach. Intell. **20**(12), 1342–1351 (1998)
47. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. **32**, 565–606 (2008)
48. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: Features for SAT (2012). Available at www.cs.ubc.ca/labs/beta/Projects/SATzilla/
49. Xu, L., Hutter, F., Shen, J., Hoos, H., Leyton-Brown, K.: Satzilla2012: improved algorithm slection based on cost-sensitive classification models. In: Balint, A., Belov, A., Diepold, D., Gerber, S., Järvisalo, M., Sinz, C. (eds.) Proceedings of SAT Challange 2012: Solver and Benchmark Descriptions, Department of Computer Science Series of Publications B, vol. B-2012-2, pp. 57–58. University of Helsinki (2012)