Appears in the Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI 2007), Vancouver, British Columbia.

Refining Rules Incorporated into Knowledge-Based Support Vector Learners Via Successive Linear Programming

Richard Maclin[†], Edward Wild[‡], Jude Shavlik[‡], Lisa Torrey[‡], Trevor Walker[‡]

Computer Science Department[†]
University of Minnesota Duluth
1114 Kirby Drive
Duluth, MN 55812
rmaclin@d.umn.edu

Computer Sciences Department[‡]
University of Wisconsin Madison
1210 West Dayton Street
Madison, WI 53706
{wildt,shavlik,ltorrey,twalker}@cs.wisc.edu

Abstract

Knowledge-based classification and regression methods are especially powerful forms of learning. They allow a system to take advantage of prior domain knowledge supplied either by a human user or another algorithm, combining that knowledge with data to produce accurate models. A limitation of the use of prior knowledge occurs when the provided knowledge is incorrect. Such knowledge likely still contains useful information, but knowledge-based learners might not be able to fully exploit such information. In fact, incorrect knowledge can lead to poorer models than result from knowledge-free learners. We present a support-vector method for incorporating and refining domain knowledge that not only allows the learner to make use of that knowledge, but also suggests changes to the provided knowledge. Our approach is built on the knowledge-based classification and regression methods presented by Fung, Mangasarian, & Shavlik (2002; 2003) and by Mangasarian, Shavlik, & Wild (2004). Experiments on artificial data sets with known properties, as well as on a real-world data set, demonstrate that our method learns more accurate models while also adjusting the provided rules in intuitive ways. Our new algorithm provides an appealing extension to knowledge-based, support-vector learning that is not only able to combine knowledge from rules with data, but is also able to use the data to modify and change those rules to better fit the data.

Introduction

Support-vector methods that incorporate prior knowledge in the form of rules have become increasingly popular (Fung, Mangasarian, & Shavlik 2002; 2003; Mangasarian, Shavlik, & Wild 2004; Maclin *et al.* 2005; 2006; Le, Smola, & Gaertner 2006). One general assumption of these methods is that the advice provided is mostly accurate. This leaves open the question of whether advice that is significantly less accurate can be used effectively. Most of the knowledge-based support vector methods provide a mechanism based on slack variables to allow the data to overcome poor advice. But in general, this means that the learner does not adjust the rule, but rather uses the data to overcome the poor rule and develop a model comparable to one learned without using the prior knowledge. In this work we develop a method that can not only overcome poor rules, but can alter the rule to produce a more effective model when the rule does have some useful structure. As a result, our models can often repair a poor rule to allow the learner to outperform a learner that discards the rule. In addition, our method provides a mechanism for examining what changes were made to the rule, which can be made available to a user as method of analyzing what is wrong with the original rule.

Our method makes use of the Knowledge-Based Support Vector Machines (KBSVMs) presented by Fung et al. 2002; 2003 and Mangasarian et al. (Mangasarian, Shavlik, & Wild 2004). In our method, Rule-Refining Support Vector Machines (RRSVM), the learner corrects antecedents in provided rules as part of a numeric optimization problem. Our formulation cannot be solved by linear programming because the feasible region is nonlinear, due to the factors that we use to correct a rule. However, our mathematical program is bilinear in these factors, which allows for a solution via successive linear programming. The correction factors resulting from this process not only increase the accuracy of the model, but can be supplied to the human user as useful refinements of the rules.

Knowledge-Based Support Vector Methods

KBSVMs generally capture rules in the form IF *Antecedents* THEN *Consequent*. For example, in the promoter domain, which we examine later, we might say a gene sequence is a promoter if at least two or more DNA characters at positions (before the start of the gene) 36 (shortened to *p36*), 35 (*p35*), and 34 (*p34*) are 't', 't' and 'g'. This rule is:

IF *countOf(p36=t, p35=t, p34=g)* ≥ 2
THEN *Promoter=true*.

Rules in KBSVMs are generally represented in the form:

$$Bx \leq d \Longrightarrow x'w + b \geq x'h + \beta. \quad (1)$$

In this representation, the matrices B and d capture the antecedents of the rule. For example, if in our representation there is a 1 for feature 59 if *p36=t* (and 0 otherwise) and similarly a 1 at feature 63 for *p35=t* and a 1 at feature 66 for *p34=g*, then we could capture the rule above by having a row of B with zeros for all of the features except 59, 63, and 66. As the rule requires the count be ≥ 2 (while Equation 1 uses $\leq$) we would use negated antecedents for each of

these features, in this case we would set the row of B to have -1s for the features 59, 63, and 66. Then the corresponding value of d would be -2 to capture the threshold 2. This rule is met if at least two of the features have the value 1.

The right-hand side of the rule ($x'w+b \geq x'h+\beta$) captures what we want to say about the consequent using the terms h and β (the x represents the input features, and the w and b are the weights and threshold of the model to be learned). For a classification problem, if we want to say that under the conditions indicated by the B and d matrix the data point is a promoter (a positive example) we would simply say that we want $x'w+b \geq 1$ setting h to 0 and β to 1.

KBSVMs provide methods for incorporating rules into SVM formulations. One formulation of an optimization problem for a linear SVM (without domain knowledge) for classification is:

$$\begin{array}{rl} \min\limits_{w,b,s\geq 0} & \|w\|_1+\nu|b|+Ce's \\ \text{s.t.} & Y(Aw+eb)+s \geq e. \end{array} \tag{2}$$

Here, e denotes a vector of ones of appropriate dimension, and a prime symbol $'$ denotes the transpose of a vector or matrix. Thus for a vector s, $e's$ denotes the sum of the components of s. The absolute value of a scalar b is denoted $|b|$. The 1-norm of a vector x, denoted $\|x\|_1$, is defined as the sum of the absolute values of the components of x. That is if x has n components, $\|x\|_1 = \sum_{i=1}^{n} |x_i|$. The matrix A is the set of data, one row for each data point and one column for each feature. Y is a matrix whose diagonal elements are the class labels (1 or -1) corresponding to the points in A; all off-diagonal elements are 0. Parameters w and b represent the feature weights and threshold, s are the slack variables (terms used to allow the solution to deal with noisy data), and C and ν are penalties associated with the the slacks s and the threshold b. Note that although we are focusing on classification here and are using a linear rather than kernel model, our results also apply to regression and kernel methods as well.

In KBSVM the domain knowledge of Equation 1 is added as extra constraints to the optimization problem, causing the system to pick a model that takes into account the rules. So, the optimization problem in Equation 2 is extended to:

$$\begin{array}{rl} \min\limits_{w,b,s\geq 0,u\geq 0,z,\zeta\geq 0} & \|w\|_1+\nu|b|+Ce's+\mu_1 e'z+\mu_2\zeta \\ \text{s.t.} & Y(Aw+eb)+s \geq e \\ & -z \leq B'u+w-h \leq z \\ & -d'u+\zeta \geq \beta-b. \end{array} \tag{3}$$

As part of this process of adding the rules as constraints, the rules are softened by adding slack variables (in the above representation z and ζ) that allow the system to partially or completely discard a rule if it is contradicted by the data.

Knowledge Refinement

The slacks introduced for knowledge in KBSVMs allow for the refinement of the *implication* of a rule:

$$Bx \leq d \Longrightarrow x'w+b \geq x'h+\beta$$

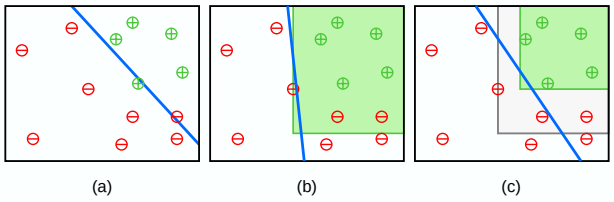

Figure 1: Surfaces that might be learned (a) without knowledge, (b) when knowledge (shown as a green (gray) box) cannot be refined, and (c) when knowledge (shown as smaller green (gray) box with the outer box showing the original rule) can be refined.

by in some sense allowing the system to alter the h and β terms. But it can be desirable to refine the term d in the left-hand side or antecedent of this implication, which changes the region $\{x|Bx \leq d\}$ over which the inequality in the right-hand side of the implication is enforced.

Consider the simple problem shown in Figure 1. In Figure 1(a) we show a simple linear surface that might be produced based on that two-dimensional data. In Figure 1(b) we show the data and a region defining a rule in light green (gray) where the rule indicates that in that region the user thinks data should be positive (but the data seems to contradict some of this rule). If we assume that the learner wants to closely fit the rule we might get the decision surface shown in Figure 1(b). Finally, in Figure 1(c) we show what might happen using our approach. In it the learner shrinks the advice region to better fit the data, and then fits its learned surface to that region and the data. Note that although we get more training errors than from the surface learned from just the training data, we hope that the adjusted rule is useful and the resulting surface may be more accurate for the test data. Note also that our adjusted rule can be returned to the user to analyze changes. To do this we need to refine the d term of our rule.[1]

In our new approach we refine the given value of d by learning a vector δ such that

$$Bx \leq (d-\delta) \Longrightarrow x'w+b \geq x'h+\beta. \tag{4}$$

To learn δ, we add this implication to our optimization formulation in the same way as the original implication was added to Formulation 2 to produce Formulation 3. We also add a penalty on the 1-norm of δ to the objective, which allows us to trade off among reducing the complexity of the solution, fitting the data, making changes to the consequent of the rule represented by z and ζ, and making changes to the antecedent of the rule represented by δ.

We construct a new formulation for our method RRSVM by extending the formulation in Equation 3:

$$\begin{array}{rl} \min\limits_{w,b,s\geq 0,u\geq 0,z,\zeta\geq 0,\delta} & \|w\|_1+\nu|b|+Ce's+ \\ & \quad \mu_1 e'z+\mu_2\zeta+\rho\|\delta\|_1 \\ \text{s.t.} & Y(Aw+eb)+s \geq e \\ & -z \leq B'u+w-h \leq z \\ & (\delta-d)'u+\zeta \geq \beta-b. \end{array} \tag{5}$$

[1]Note that one might also wish to refine the B term. We leave that to future work, as this would only apply in complex rules containing individual antecedents that involve more than one feature.

Observe that this problem has nonlinear constraints because the term $\delta' u$ appears, both of which are free variables. However, the constraints are *bilinear* in δ and u, suggesting the method of solution shown in Algorithm 1. First, fix the value of δ and solve the optimization problem in (5) to set the value of u. Then, fix that value of u and solve for a new value of δ. This process is then iterated.

Algorithm 1 RRSVM via successive linear programming

$\delta^1 \leftarrow 0$
for $r = 1, \ldots$ **do**
  if $Bx \leq (d - \delta^r)$ has no solution x **then**
    return failure
  end if
  Let (w^r, b^r, u^r) solve

$$\begin{aligned} \min_{w,b,s\geq 0,u\geq 0,z,\zeta\geq 0} \quad & \|w\|_1 + \nu|b| + Ce's + \mu_1 e'z + \mu_2\zeta \\ \text{s.t.} \quad & Y(Aw + eb) + s \geq e \\ & -z \leq B'u + w - h \leq z \\ & (\delta^r - d)'u + \zeta \geq \beta - b \end{aligned} \tag{6}$$

  Let δ^{r+1} solve

$$\begin{aligned} \min_{w,b,s\geq 0,z,\zeta\geq 0,\delta} \quad & \|w\|_1 + \nu|b| + Ce's + \\ & \mu_1 e'z + \mu_2\zeta + \rho\|\delta\|_1 \\ \text{s.t.} \quad & Y(Aw + eb) + s \geq e \\ & -z \leq B'u^r + w - h \leq z \\ & (\delta - d)'u^r + \zeta \geq \beta - b \end{aligned} \tag{7}$$

  if $\|\delta^r - \delta^{r+1}\| \leq \varepsilon$ **then**
    return $(w^r, b^r, u^r, \delta^r)$
  end if
end for

Proposition 1. *For $\varepsilon = 0$, the sequence of objective values converges to the value $\|\bar{w}\|_1 + \nu|\bar{b}| + Ce'\bar{s} + \mu_1 e'\bar{z} + \mu_2\bar{\zeta} + \rho\|\bar{\delta}\|_1$ where $\bar{s}$, $\bar{z}$, and $\bar{\zeta}$ are computed from any accumulation point $(\bar{w}, b, \bar{u}, \bar{\delta})$ of the sequence of iterates $\{(w^r, b^r, u^r, \delta^r)\}_{r=1}^{\infty}$ generated by Algorithm 1. Such an accumulation point satisfies the following local minimum property:*

$$\begin{aligned} (\bar{w}, \bar{b}) \in \quad \operatorname*{argmin}_{w,b,s\geq 0,u\geq 0,z,\zeta\geq 0} \quad & \|w\|_1 + \nu|b| + Ce's + \\ & \mu_1 e'z + \mu_2\zeta \\ \text{s.t.} \quad & Y(Aw + eb) + s \geq e \\ & -z \leq B'u + w - h \leq z \\ & (\bar{\delta} - d)'u + \zeta \geq \beta - b. \end{aligned} \tag{8}$$

Proof. The sequence of objective values is bounded below by 0, and the solution of each linear program generates a feasible point for the succeeding linear program. The sequence is therefore bounded and nonincreasing, and hence converges. That each accumulation point of the sequence satisfies the local minimum property (8) can be seen by noting that each point of the sequence $(w^r, b^r, u^r, \delta^r)$ satisfies (8) with $(\bar{w}, \bar{b}, \bar{u}, \bar{\delta})$ replaced by $(w^r, b^r, u^r, \delta^r)$ due to the first linear program (6) in Algorithm 1. Hence, any accumulation point $(w^r, b^r, u^r, \delta^r)$ of the sequence $\{(w^r, b^r, u^r, \delta^r)\}_{r=1}^{\infty}$ satisfies the local minimum property (8). □

We can use (6) from Algorithm 1 in conjunction with Mangasarian, Shavlik, & Wild (2004)[Proposition 2.1] to immediately prove the following result, which shows the refined prior knowledge imposed as a result of Algorithm 1.

Proposition 2. *Let $(\bar{w}, \bar{b}, \bar{u}, \bar{\delta})$ be the result of Algorithm 1, with associated slack variables $(\bar{z}, \bar{\zeta})$. Then the function $f(x) = x'\bar{w} + \bar{b}$ satisfies:*

$$Bx \leq (d - \bar{\delta}) \Longrightarrow x'\bar{w} + \bar{b} \geq (h - \hat{z})'x + \beta - \bar{\zeta} \tag{9}$$

with $-\bar{z} \leq \hat{z} \leq \bar{z}$ such that $B'\bar{u} + w - (h - \hat{z}) = 0$.

Note our method necessitates an additional parameter (ρ) for the optimization problem, which must be learned as well as parameter(s) for controlling the termination of the bilinear process (in terms of a maximum steps and the tolerance to use in measuring whether or not the process should be terminated). We will discuss this aspect in our experiments.

Note also that our bilinear process returns the learned δ terms, making it possible for the user to examine the resulting value to see how the optimization altered the rules.

Experiments

We tested our method RRSVM on artificial data sets with known properties. This testing allowed us to determine that RRSVM works as suggested under a variety of conditions. We also performed experiments on the promoters-106 data set (Towell, Shavlik, & Noordewier 1990), an early domain theory that has become outdated (knowledge of promoters has significantly advanced), but nevertheless has been the subject of significant experimentation in theory refinement. Our results demonstrate that RRSVM works well, especially when the user provides rules that are not correct, but which have structure that can be exploited.

Methodology

One critical aspect of our work that must be addressed is the selection of parameter values. In the original (no knowledge) representation of the SVM formulation, there are two parameters C and ν that penalize the size of the slacks (the misfit for each data point) and the threshold, respectively. The weights w are implicitly penalized with value 1. The KBSVM approach, another of our experimental controls, is from Fung, Mangasarian, and Shavlik (2002; 2003) and is captured in Formulation 3; this approach has C and ν and in addition the parameters μ_1 and μ_2 that are used to slack the knowledge terms as described previously. Finally, our approach RRSVM includes the terms C, ν, μ_1, and μ_2 as well as a fifth term ρ which is used to penalize the δ terms.

In order to set these parameters we supply each system with a range of possible values and let it select, for each data set, which combination of parameters to use. The set of possible values we tried are: $C \in \{10, 100, 10^3\}$, $\nu \in \{1, 100\}$, $\mu_1 \in \{0, 10^3, 10^6\}$, and $\rho \in \{100, 10^4, 10^6\}$. In addition μ_2 values were set to either $\mu_2 = \mu_1$ or $\mu_2 = 10\mu_1$. Our ranges of values were based on suggestions from previously KBSVM papers and we believe they represent a reasonable range of values. Note, in some experiments we restricted μ_1 values to only $\{10^3, 10^6\}$, which we discuss in those experiments.

Table 1: Methodology for our artificial data set experiment.

```
TrainingSet = 200 random data points
TestSet = 1000 random data points
for TrainSetSize = 10, 25, 50, 75, 100, 125, 150, 200 do
   TrainSet = first TrainSetSize points of TrainingSet
   BestScore=inf
   for Each Set of Parameters P do
      Score = 10-fold cross-val. err. on TrainSet using P
      if Score < BestScore then
         BestScore=Score, BestParams=P
      end if
   end for
   Using BestParams train model on TrainSet
   Score that model on TestSet
end for
```

In order to determine which set of parameters to use each learner performs a ten-fold cross validation on just its training set with each set of parameter values to estimate the effectiveness of that set. The learner picks the parameters that produced the best results across the ten folds of the training set, and uses those parameters to learn a model for the entire training set. The learned model is then tested on data not seen during parameter selection or subsequent training.

Note that one effect of this approach is that our advice methods have a larger variety of parameters to try and therefore might find a better model, but as will be seen in our experiments below, we concentrate on cases where the overall amount of data is small (since knowledge should most affect those cases). When the amount of data is small, overfitting becomes a significant possibility so the larger variety of parameters is likely balanced by this potential problem.

Artificial Data

To test RRSVM in a controlled situation where the true function is known, we generated data with the rule:

IF $(3f_1 - 4f_2 + 0.5) > 0$ THEN *class=pos* ELSE *class=neg*.

For these experiments we did not include noise, though the results with Gaussian noise are similar (with slower convergence) in other experiments we performed on artificial data. The data sets were generated with 10 input features (2 meaningful and 8 distractor features) with values chosen randomly between 0 and 1 and labeled using the above rule.

Using the above rule we performed 20 repeats using the methodology shown in Table 1. Using this method we performed experiments using two different rules as knowledge, one rule based on the correct domain theory:

IF $(3f_1 - 4f_2 + 0.5) \geq 0$ THEN *class=pos*

which we will call "Good" advice and one where we altered the threshold by the value -1.0 to get

IF $(3f_1 - 4f_2 + 0.5) \geq -1$ THEN *class=pos*

which we call "Bad" advice, though it has useful structure.

We tested as learners the system without advice, the KBSVM method using first Good and then Bad advice and RRSVM using Good and Bad advice. In addition, we tested the KBSVM method where we supplied the set of parameters indicated in the previous section, but where we eliminated the possibility of $\mu_1 = 0$. Setting $\mu_1 = 0$ basically allows the learner to ignore the advice. We wanted to see results in cases where the learner has to make use of the advice as a test. We also include as a baseline a learner without advice that is given only the relevant features (another form of advice). These results are presented in Figure 2.

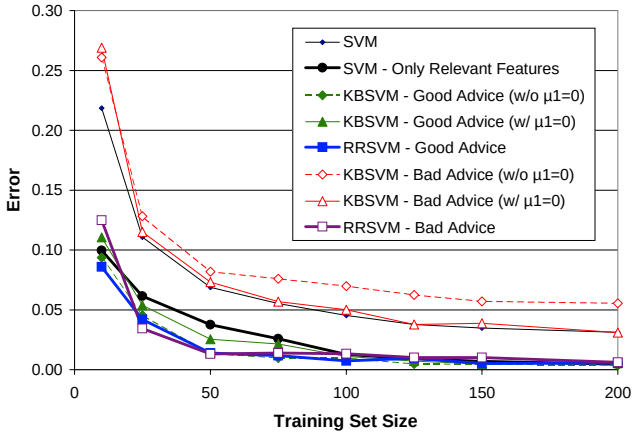

Figure 2: Results for the artificial data set and rule using the methodology in the text. Results are averaged over 20 runs with each method receiving the same set of data in each run.

A first thing to note from the results is that both KBSVM and RRSVM, with Good advice, perform very well at a fairly small amount of data (50 data points) and statistically significantly (paired t-test, $p < 0.05$) outperform the learner without advice all along the learning curve. Note that the learners with Good advice sometimes achieve lower error than a learner given just the relevant features, though the difference is not statistically significant.

A second observation from these results is that RRSVM, with its ability to refine advice, quickly performs as well using Bad advice as the other KBSVM systems (and our system) when using Good advice. We have observed in our other experiments that how quickly this occurs generally depends on how noisy the data is (as one would expect). This indicates RRSVM is able to make use of advice that is bad as long as it still has some useful structure.

A third thing to note is that KBSVM with Bad advice and the ability to ignore advice performs only as well as the learner without advice, and KBSVM when forced to use advice ($\mu_1 = 0$ is not allowed) perform worse than even the learner without advice. And this difference is statistically significant from 100 training examples on.

Note also that RRSVM is able to recover how the rules have been altered as part of the learning process. In our experiments we altered d by 1.0 and would expect the learner would recover a similar value. Figure 3 shows the resulting learned δ values which quickly become close to 1 with little deviation as the number of examples grows.

Overall, our experiments on artificial data seem to indicate that our method for refining advice, under certain conditions, makes use of advice that is not accurate, but that has useful structure. In our next experiments we look at a real-world data set that has been used for theory refinement.

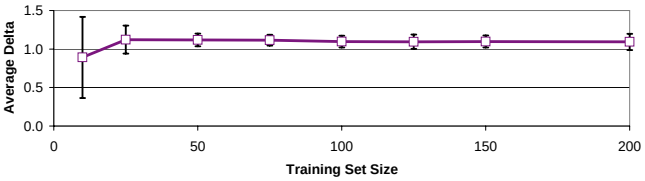

Figure 3: δ values averaged over the 20 runs for refining the "Bad" advice discussed in the text, with the standard deviation shown for each value.

Promoters

The promoters-106 data set is a set of DNA sequences originally presented by Towell, Shavlik, & Noordewier (1990) that has been used extensively in theory refinement. The data set consists of a set of 53 examples of DNA sequences that contain gene promoters and 53 examples that do not. Each strand consists of 57 DNA characters starting from 50 before the promoter is expected to start to 7 after the promoter starts on the chromosome. In our representation we use a simple 1,0 representation for each feature for each of the four possible DNA characters: a, g, t, and c. Basically each feature represents one Boolean test of whether a character occurs at a position (such as p37=g which is 1 if the character at position 37 before the start is g and 0 otherwise).

The domain theory for promoters-106 has two major parts, one of which, the conformation portion of the theory has been repeatedly rejected by other researchers. In the other part of the theory there are four rules for each of two regions of the DNA sequence corresponding to sets of characters that would occur in these regions. This would lead to 16 rules (four combinations for each of the two regions), which would make the learning process fairly unwieldly and make the issue of "Bad" rules more complex. Instead, following the observations of Ortega (1995), we combined the four rules for each of the two regions into a single antecedent. The four rules for the first region (called minus35) of the promoter tested the following conditions

minus35 :- p37=c, p36=t, p35=t, p34=g, p33=a, p32=c.
minus35 :- p36=t, p35=t, p34=g, p32=c, p31=a.
minus35 :- p36=t, p35=t, p34=g, p33=a, p32=c, p31=a.
minus35 :- p36=t, p35=t, p34=g, p33=a, p32=c.

Note that there is significant shared structure to these rules (all ask that *p36=t, p35=t,* etc.). Similarly the second (minus10) region was defined as follows:

minus10 :- p14=t, p13=a, p12=t, p11=a, p10=a, p9=t.
minus10 :- p13=t, p12=a, p10=a, p8=t.
minus10 :- p13=t, p12=a, p11=t, p10=a, p9=a, p8=t.
minus10 :- p12=t, p11=a, p7=t.

Our combined rule takes the following form:

IF *countOf(p37=c, p36=t,p35=t, p34=g, p33=a, p32=c, p31=a)* $\geq$ *T35* AND
countOf(p14=t, p13=(a or t), p12=(t or a), p11=(a or t), p10=a, p9=(t or a), p8=t, p7=t) $\geq$ *T10*
THEN Promoter=true.

The question then becomes how to set the thresholds *T35* and *T10*. Based on our examination of the rules we set *T35* to 5 and *T10* to 4 for our useful advice, which comes close to capturing the domain theory. We will call this our Original advice. To make the advice work less well we significantly lowered the thresholds to 2 and 2 for each, which made the rules match far too often. We will call this our Poor advice.

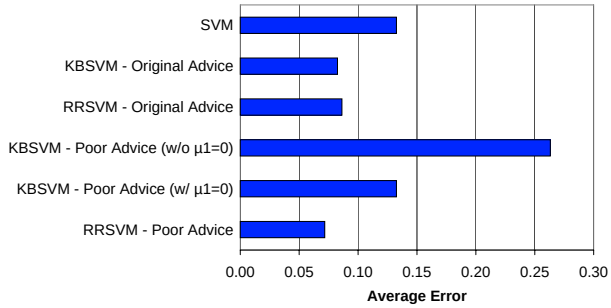

Figure 4: Average error rates for 20 ten-fold cross-validation experiments run on the promoters-106 data set with the rule discussed in the text used for advice.

We then tested this by running 20 repeats of ten-fold cross validation on the promoters-106 data set using the methodology described above to select parameters. Figure 4 shows the results of these experiments.

First, note that our linear model with no knowledge performed about as well as other models have on this data. Next, note that our advice models perform well compared to other learners, though not as well as KBANN (Towell & Shavlik 1994), which used ensembles of neural networks.

In terms of performance with the advice, note when KBSVM is not able to ignore the Poor advice, it performs very poorly (as expected), while the KBSVM system that is able to ignore the advice performs as well as the learner without advice. Finally note that learners with the Original advice perform very well, and that RRSVM that refined the Poor advice performs best of all (though the difference between RRSVM and learners with the Original advice is not statistically significant). These results indicate that RRSVM can, on a real-world data set, make useful changes to a piece of advice even if that advice is not particularly accurate.

In terms of how the advice was adjusted, for the Poor Advice in about 60% of the cases both of the thresholds *T35* and *T10* were adjusted upwards between 2 and 3. In the remaining cases the advice threshold was adjusted so that it only applied to a small number of cases while the other threshold was left alone (generally *T35* would be shifted to 5 or 6 or *T10* would be shifted to 6 or 7). This suggests that good rules might be made by strongly focusing on one region and then allowing a much looser fit for the other region.

Overall, the promoter results suggest RRSVM *can* refine advice for a real-world problem and give some insight into our data. Our method's ability to compile advice into a complex learning method and to indicate how the advice has changed makes it a novel knowledge refinement method.

The two main drawbacks of our approach are that the iterative procedure of RRSVM takes more time than standard KBSVM approaches and that there are more parameters to be set. In our experiments we found that the time taken by RRSVM did grow linearly in the number of iterations with respect to KBSVM, but our algorithm did not take advantage of the fact that a good solution had been found on the previ-

ous iteration. It may be possible to use the previous solution as a "hot start" for the optimization to significantly reduce the time. Regarding the issue of setting parameters, we believe that approaches such as those presented by Bennett et al. (2006) or Zhu et al. (2003) could be used to automatically determine effective parameter values.

Related Work

Our work relates most closely to Knowledge-Based Support Vector methods (Fung, Mangasarian, & Shavlik 2002; 2003; Mangasarian, Shavlik, & Wild 2004; Maclin *et al.* 2005; 2006; Le, Smola, & Gaertner 2006). Our approach differs from these methods in that we able to refine the antecedents of the knowledge and to examine the resulting changes to the knowledge. KBANN (Towell & Shavlik 1994) is another example of a system that in a sense "compiles" symbolic knowledge to bias a numeric learner, and there has been significant work in attempting to extract learned knowledge from such networks (Fu 1991; Fung, Sandilya, & Rao 2005; Thrun 1995; Towell & Shavlik 1993). Our work differs from most of these methods in that we are inserting our rules into SVMs rather than neural networks (though the method of Fung, Sandilya, & Rao 2005 does apply to linear support vector methods as well), and our process of determining how the rules have been changed is much simpler, and does not require complex processes to extract the refined rule.

There has also been significant work on directly manipulating rules in their symbolic form rather than "compiling" them into another learner. Pazzani and Kibler (1992) developed mechanisms for refining rules represented in first order logic using inductive logic programming methods. Ourston and Mooney (1994) created a method for analyzing first order logical proofs generated from sets of rules and used data to correct and complete the proofs to learn new knowledge. Our work differs from these and related methods in that we are focusing on extending support vector methods, which have proven very effective in a wide range of problems, to be able to handle additional sources of training information, namely general rules in addition to labeled examples, while allowing both types of information to be noisy.

Conclusions and Future Work

We have presented a novel method for incorporating domain knowledge provided by a user into a support vector learning method, and we have shown how that domain knowledge can be refined by the learning process. Our method, Rule-Refining Support Vector Machines (RRSVM) allows a learner to make use of domain knowledge, even if it is not very accurate, as long as it has some useful structure. RRSVM also reports the refinements to the rules so that they can be examined and perhaps analyzed by the human user. RRSVM extends previous work in knowledge-based support vector methods. Experiments with RRSVM demonstrate that we can effectively refine knowledge provided by a human user, which we demonstrate on both artificial data sets and on the real-world promoters-106 data set.

For future work, we plan to apply RRSVM to larger problems and to test on real-world regression problems in addition to classification. We plan to look at RoboCup advice as in Maclin et al. (2006). We also plan to examine methods to allow RRSVM to expand on advice. We will examine methods to add new variables to rules that were not previously mentioned. We also plan to look at ways to extend methods for extracting rules from SVMs similar to Fung, Sandilya & Rao (2005) to allow for additional adjustment of rules. Finally, we would like to look at methods for refining nonlinear knowledge rules and develop methods for extracting useful changes to such rules.

References

Bennett, K.; Hu, J.; Kunapuli, G.; and Pang, J.-S. 2006. Model selection via bilevel optimization. In *IJCNN*.

Fu, L. 1991. Rule learning by searching on adapted nets. In *AAAI*.

Fung, G.; Mangasarian, O.; and Shavlik, J. 2002. Knowledge-based support vector machine classifiers. In *NIPS*.

Fung, G.; Mangasarian, O.; and Shavlik, J. 2003. Knowledge-based nonlinear kernel classifiers. In *COLT*.

Fung, G.; Sandilya, S.; and Rao, R. 2005. Rule extraction from linear support vector machines. In *KDD*.

Le, Q.; Smola, A.; and Gaertner, T. 2006. Simpler knowledge-based support vector machines. In *ICML*.

Maclin, R.; Shavlik, J.; Torrey, L.; Walker, T.; and Wild, E. 2005. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *AAAI*.

Maclin, R.; Shavlik, J.; Walker, T.; and Torrey, L. 2006. A simple and effective method for incorporating advice into kernel methods. In *AAAI*.

Mangasarian, O.; Shavlik, J.; and Wild, E. 2004. Knowledge-based kernel approximation. *JMLR* 5:1127–1141.

Ortega, J. 1995. On the informativeness of the DNA Promoter sequences domain theory. *JAIR* 2:361–367.

Ourston, D., and Mooney, R. 1994. Theory refinement combining analytical and empirical methods. *Artificial Intelligence* 66:273–309.

Pazzani, M., and Kibler, D. 1992. The utility of knowledge in inductive learning. *Machine Learning* 9:57–94.

Thrun, S. 1995. Extracting rules from artificial neural networks with distributed representations. In *NIPS*.

Towell, G., and Shavlik, J. 1993. Extracting refined rules from knowledge-based neural networks. *Machine Learning* 13:71–101.

Towell, G., and Shavlik, J. 1994. Knowledge-based artificial neural networks. *Artificial Intelligence* 70:119–165.

Towell, G.; Shavlik, J.; and Noordewier, M. 1990. Refinement of approximate domain theories by knowledge-based ne ural networks. In *AAAI*, 861–866.

Zhu, J.; Rosset, S.; Hastie, T.; and Tibshirani, R. 2003. 1-norm support vector machines. In *NIPS*.