# Exploiting Sampling and Meta-learning for Parameter Setting for Support Vector Machines

Petr Kuba[1], Pavel Brazdil[2,3], Carlos Soares[2,3], Adam Woznica[2]

[1] Masaryk University, Brno, Czech Republic
[2] LIACC, University of Porto, R. Campo Alegre 823, 4150-180, Porto, Portugal
[3] Faculty of Economics, University of Porto, Portugal

**Abstract.** It is a known fact that good parameter settings affect the performance of many machine learning algorithms. Support Vector Machines (SVM) and Neural Networks are particularly affected. In this paper, we concentrate on SVM and discuss some ways to set its parameters. The first approach uses small samples, while the second one exploits meta-learning and past results. Both methods have been thoroughly evaluated. We show that both approaches enable us to obtain quite good results with significant savings in experimentation time.

## 1 Introduction

Support Vector Machines (SVM) [5] have become quite widespread in the last decade or so. Good results have been reported in various domains, but it is well known that these depend very much on good parameter settings. So using this algorithm is nontrivial and causes problems to inexperienced users, as they cannot rely on deeper knowledge of the algorithm, nor on previous experience. These users tend to loose a lot of time experimenting before good results can be obtained.

Our goal was to identify most important parameters for this algorithm and to find a strategy for selecting the best (or at least good) values of these parameters. At first we carried out a preliminary study of SVM on different datasets to establish the parameters in which reasonable results could be expected. This enabled us to formulate a principled strategy for experimenting with different parameter values. As an alternative, we used meta-learning methods that help to establish relationships between certain datasets characteristics and good parameter settings.

The rest of this paper is organized as follows. The next section discusses the strategy that can be followed for experimenting with different parameter values. Section 3 describes the approach based on meta-learning that is used to suggest parameter values to the user. This approach has been evaluated and this section described the basic results. Finally, Section 4 contains conclusions and suggestions for future work.

## 2 Searching through the Space of Parameter Settings

Support Vector Machines (SVM) have been well documented in literature (e.g. [5]) and therefore we will not describe the algorithm here. Although they can be used for both classification and regression task and here we concentrate on applications to regression problems only.

### 2.1 Systematic Parameter Variation

#### Determining the Parameter Ranges

In general, SVM has three parameters that need to be determined. The first one is the width of error tube, which is often referred to as $\varepsilon$. Then it is necessary to determine the type of the kernel function and its parameters. Generally it is expected that a good parameter setting of the kernel function is more important than the choice of function itself. So we decided to use only gaussian kernel and vary its parameter, which is standard deviation (*std*).

First, we have conducted a preliminary set of experiments whose aim was to determine the range of possible parameter settings in which good results would be obtained. All these experiments were performed on 42 different datasets, whose size varied and the largest ones contained up to 10,000 cases. All these datasets are shown in Table 2.1.

As the version of SVM used, SVMTorch [4], does not handle discrete attributes, the datasets were modified by binarizing all continuous attributes.

We were surprised to find that this range was exceedingly high. For instance, as far as the parameter *kernel std* is concerned, we could conclude that for some datasets good result was obtained if its value was set to 1, while for others it was necessary to set it to values greater than 10,000.

Another important observation is that the higher the $\varepsilon$ value, the lower the number of support vectors. When the error tube is too wide the number of vectors is zero (all data points fit into the error tube) and predictions of all values are 0. So if zero support vectors are found there is no reason to continue experiments for higher values of $\varepsilon$. It is very useful information for reducing the number of experiments and, thus saving time. On the other hand, if the $\varepsilon$ value is too low then the number of support vectors is very high and results are usually quite bad.

We also noticed that if the values of target attribute are normalized, the range of $\varepsilon$ values that need to be explored can be substantially decreased (the largest value than needs to be tried is around 1). So we followed this strategy and normalized the target attribute for all datasets before trying to learn the SVM model. Then of course the values need to be denormalized when the prediction is generated in the testing phase.

After this preliminary study we were able to determine the interval both parameters for which SVM gave good results. In addition, we have also defined a set of intermediate values to be considered in all future searches for good parameter setting.

As for $\varepsilon$, we have decided to experiment with the following five values: 0.001, 0.008, 0.032, 0.128 and 1. Regards the *kernel std*, the following ten values were chosen: 1, 4, 16, 64, 256, 1000, 4000, 16000, 64000 and 128000. So to search through this space of values, we need to explore $5 \times 10 = 50$ pairs of values. This issue is addressed in the following subsection.

### Searching through the Parameter Space

Searching through the full parameter space requires quite a lot of computational effort. First it is necessary to test $5 \times 10 = 50$ pairs of values, as was pointed out in the previous section. We note that SVM takes quite a long time to run, particularly if the parameter settings are not right. So this phase took us several moths to finish. The results are shown in Table 2.1 (see column "Search on full data").

If our aim is to find a solution relatively quickly, the question is how to do this. This issue will be addressed in the next subsection.

### Using Samples to Search through the Parameter Space

The first thing that comes into mind is quite an obvious strategy: use a sample of the original dataset to determine the parameter setting and then use this setting to generate the model on the full dataset. But will this strategy lead to good results? It is conceivable that the parameter setting found to work well with a particular sample, may not be the right one, if we consider the full dataset. This is why we have conducted experiments to evaluate this strategy. The results are shown in Table 2.1, column "Search on sample data", which can be compared to the previous column (we have not considered the datasets, for which the sample size was below 1000 cases.)

The results in Table 2.1 indicate that the NMSE errors resulting from this search are quite similar to those obtained by searching in on the full dataset. It's noteworthy that for 26 datasets the result obtained on the full dataset, using the setting found on a small sample, was within 5% tolerance of the result that would be obtained by searching the space of parameter settings on full dataset. Given this, we can then conclude that as far as SVM is concerned, its parameters can be set on a sample. The values found will not be far off the ideal values.

### Using Samples and Reduced Search

To reduce the number of pairs of parameter settings that need to be searched through, we have considered the following strategy. We continue to use small samples to search for good parameter settings. First, we fix the value of $\varepsilon$ (say to 0.032) and vary value of *kernel std* and then search for the best value of $\varepsilon$. The method can be summarized as follows:

1. Randomly select a small sample S from given dataset D
2. Using sample S, do:

2.1. Fix value of $\varepsilon$ (e.g. 0.032)
2.2. Run experiments for all the values of *kernel std*
2.3. Set *STD_BEST* to the value of *kernel std* with the best result in the previous step
2.4. Run experiments for all the values of $\varepsilon$ and fix *kernel std* to *STD_BEST*
2.5. Set *EPS_BEST* to the value of with the best result in the previous step
3. Run SVM experiment on dataset D using values *STD_BEST* and *EPS_BEST* for the corresponding parameters.

The advantage of this method is that it reduces the number of experiments to be carried out. Instead of $5 \times 10 = 50$ experiments, we need only $10 + 5 - 1 = 14$ experiments. As the parameters are set using a limited amount of search, the risks are that the method could identify parameter settings, which are not quite right (i.e. rather far off the ideal values). This is why we have evaluated this alternative and the results are shown again in Table 2.1 (column "Reduced search on sample data").

It appears that the results are a bit worse, when compared to the previous alternative, but the differences are not very large. The loss of precision is justified, if we are prepared to save experimentation time. The saving in time is $50 - 14 = 36$ experiments.

If one experiment takes around 100 seconds (see the last column in Table 2.1), the saving is $36 \times 100$ seconds, that is, around 1 hour. If the experiment takes 10 times as much (around 1000 seconds), obviously the saving is proportionally larger, that is, much more significant from the user's point of view.

## 3 Parameter setting using meta-learning

### 3.1 Methodology for Metalearning

Meta-learning uses information about the past performance of algorithms to predict their performance on new datasets. It does that by inducing a mapping between data characteristics and algorithm performance. Here we consider an algorithm to be different versions of SVM algorithm, where each version uses a somewhat different parameter setting from the other ones.

Given that these parameters are continuous, we can address this as a regression problem. In principle, any regression algorithms could be used for metaleaning, but after some initial study, we have opted for regression trees (*rpart*) from the R package [7].

The input data for meta-learning contains, for each dataset, one row consisting of:

- Dataset characterization – one value for each characteristic (see below);
- Target value – the value of the parameter that gives the best result.

The values of SVM parameters were rescaled logarithmically because range of values was too large. New ranges of $\varepsilon$ and *kernel std* parameters are $0 \ldots 5$ and

| Dataset | Search on full data NMSE | Search on sample data NMSE | Reduced search on sample data NMSE | Reduced search on sample data Train time (s) |
|---|---|---|---|---|
| abalone | 0.448 | 0.449 | 0.448 | 82.9 |
| ailerons | 0.248 | 0.248 | 0.248 | 8437.3 |
| bank32nh | 0.511 | 0.511 | 0.889 | 55.7 |
| bank8FM | 0.045 | 0.045 | 0.045 | 269.3 |
| cal_housing | 13977 | 14026 | 14097 | 132.4 |
| driving_noise | 0.470 | $<$ 1000 samples | 0.470 | 17.4 |
| fuel_cons_country | 0.191 | 0.229 | 0.235 | 34.8 |
| fuel_cons_total | 0.156 | 0.162 | 0.167 | 31 |
| fuel_cons_town | 0.164 | 0.165 | 0.180 | 33.5 |
| house_16H | 0.948 | 1.074 | 0.969 | 699 |
| house_8L | 0.901 | 2.653 | 0.989 | 383.7 |
| housing | 0.310 | $<$ 1000 samples | 0.352 | 13.5 |
| maint_interval | 0.724 | 0.727 | 1.166 | 29 |
| maximal_torque | 0.118 | 0.168 | 0.141 | 28.8 |
| steering_accel | 0.00000886 | 0.00000886 | 0.0000152 | 29.8 |
| top_speed | 0.104 | 0.106 | 0.124 | 36.9 |
| 1km | 0.260 | $<$ 1000 samples | 0.285 | 12.9 |
| CO2-emission | 0.207 | 0.212 | 0.293 | 27.4 |
| CW_drag | 0.191 | 0.192 | 0.360 | 14.4 |
| acceleration | 0.188 | 0.224 | 0.295 | 22.0 |
| available_power | 0.074 | 0.076 | 0.094 | 41.2 |
| cart | 0.053 | 0.053 | 0.053 | 963.2 |
| closedow | 0.274 | 0.274 | 0.276 | 70.2 |
| closenikkei | 1.001 | 1.001 | 1.003 | 33.8 |
| cpu_act | 0.224 | 0.224 | 0.223 | 304.6 |
| cpu_small | 0.286 | 0.315 | 0.326 | 301.6 |
| d2 | 0.523 | 0.682 | 0.571 | 272.6 |
| delta_ailerons | 0.407 | 0.777 | 0.790 | 22.8 |
| delta_elevators | 0.387 | 0.389 | 0.388 | 1408.3 |
| elevators | 0.932 | 0.932 | 0.943 | 4910.6 |
| fried | 0.064 | 0.080 | 0.079 | 5417.1 |
| fluid_discharge | 0.040 | $<$ 1000 samples | 0.046 | 13.1 |
| fluid_swirl | 0.420 | $<$ 1000 samples | 0.536 | 12.6 |
| heat | 0.004 | 0.060 | 0.056 | 46.9 |
| ind | 0.301 | 0.302 | 0.304 | 14.7 |
| kin8nm | 0.090 | 0.091 | 0.091 | 4127.3 |
| mv | 0.134 | 0.134 | 0.178 | 1341.2 |
| puma32H | 0.793 | 0.801 | 0.874 | 31.3 |
| puma8NH | 0.355 | 0.360 | 0.410 | 107.1 |
| steering_angle | 0.0000198 | 0.0000198 | 0.0000293 | 966.8 |
| steering_velocity | 0.0000140 | 0.0000154 | 0.0000140 | 263.4 |
| telecomm | 0.051 | 0.052 | 0.052 | 241.4 |

**Table 1.** Results of SVM on different regression datasets. The parameters were identified using different search methods through the space of different parameter settings. The NMSE and times were obtained on training the algorithm on full data and testing on a separate test set.

$0 \ldots 10$ respectively. As our aim is to predict good values of $\varepsilon$ and *kernel std* parameters, two separate experiments were performed. In one of them, we attempted to learn to predict $\varepsilon$ and in the other *kernel std*. Here we assume that both parameters are not significantly correlated. As we will see later, we have grounds to believe that this is so.

### 3.2 Data characteristics used in Metalearning

Many measures have been developed to characterize data for meta-learning. Earlier work has focused on general, statistical and information-theoretic measures [2], which still seem to stand quite well against other approaches suggested later [6, 1]. Recent results in the setting of meta-learning for classification algorithm selection, show that it is possible to identify a small subset of this type of features that is quite predictive [3]. Here we follow the same approach and use the following set of features, representing certain properties identified beforehand:

- *Number of examples*: Scalability with respect to this measure.
- *Number of attributes (binarized)*: Scalability with respect to this measure. Given that we know beforehand that attributes are binarized, we count each symbolic attribute $i$ as $k_i - 1$ attributes, where $k_i$ is the number of values of attribute $i$.
- *Proportion of symbolic attributes*: Preference for symbolic or numeric attributes.
- *Ratio of number of examples to the number of attributes (binarized)*: Potential for overfitting. If this value is small, learning algorithms may find a model that adapts too well to the specificities of the training data, which may be caused by noisy or irrelevant attributes, and thus, have poor generalization performance.
- *Proportion of numeric attributes with outliers*: Robustness with respect to outlying values, which are possibly due to noise. An attribute is considered to have outliers if the ratio of the variances of mean value and the $\alpha$-trimmed mean is smaller than 0.7. We have used $\alpha = 0.05$.
- *Coefficient of variation of the target*: Robustness to sparsity of the target, represented by the ratio of the standard deviation and the average of target values.
- *Sparsity of the target*: Robustness to sparsity of the target, obtained by discretizing the coefficient of variation into three bins: less than 20%, between 20% and 50%, and larger than 50%.
- *Are there outliers in the target?* Robustness with respect to this property of the target.
- *Stationarity of the target*: Robustness with respect to this property of the target.
- *Coefficient of linear regression, $R^2$*: Linearity of the data, which is indicative of the amount of useful information in numeric attributes.
- *Coefficient of linear regression, $R^2$*, using binarized symbolic attributes: Linearity of the data, which is indicative of the amount of useful information in

the attributes. Symbolic attributes are binarized for the reasons explained above.
- *Average absolute correlation between numeric attributes*: Robustness to irrelevant attributes.
- *Average absolute correlation of numeric attributes to the target*: Average amount of information in the numeric attributes, which is indicative of the amount of useful information in numeric attributes (individually) but may also be regarded as an indication of the number of irrelevant attributes.
- *Average dispersion gain*: Average gain in dispersion obtained with the best split for each symbolic attribute, which serves a similar purpose as the previous meta-feature, only for symbolic attributes.

After choosing the set of measures, we decided to carry out visual feature selection as a simple method for checking the quality and usefulness of the measures for our data. Correlation graph for each couple of measures and for parameters $\varepsilon$ and *kernel std* were generated.

It appeared that there was little or no correlation between $\varepsilon$ and *kernel std* parameters. This was confirmed by our experiments where $\varepsilon$ was added to the data and used to predict *kernel std* and vice versa and the results didn't improve. Moreover it appeared that there is no correlation between the *number of examples* and parameters $\varepsilon$ or *std*, and so this measure was removed.

### 3.3 Experiments and Evaluation of the Method

To evaluate whether meta-learning brings some benefits, we have carried out leave-one-out evaluation. With this method, $42 - 1 = 41$ datasets were used as a learning set for the regression algorithm (*rpart*) and then the model generated was tested on the remaining case. This was repeated 42 times. One typical regression tree obtained for predicting $\varepsilon$ is shown below (it has been slightly modified for legibility):

```
[node split                                n     dev.      yval ]
1) root                                    41    91.83     2.87
    2) n.examples.rel.n.attrs >= 32.47     31    72.08     2.55
         4) r.squared.bin >= 0.71          15    27.93     1.77 *
         5) r.squared.bin < 0.71           16    26.41     3.28 *
    3) n.examples.rel.n.attrs < 32.47      10    6.97      3.85 *
```

The branch identified by "4" can be interpreted as follows: If the *ratio of number of examples to the number of attributes* (n.examples.rel.n.attrs) is higher or equal than 32.47 and the $R^2$ *binarized* (r.squared.bin) is higher or equal to 0.71, then the predicted value of $\varepsilon$ (identified by yval) for a particular dataset is 1.77. If we de-normalize the value of 1.77 we get a value somewhere between the first value of $\varepsilon$=0.001 and the second one, which is $\varepsilon$=0.008. The symbol "*" identifies the terminal node in the tree. This rule covers n=15 cases and the deviance is 27.93, meaning that the standard deviation is sqrt(27.93) = 5.28.

In experiments with the parameter $\varepsilon$, the mean error of predicted value using leave-one-out evaluation method was 1.43. It means that the distance between predicted and real value of parameter is less than 1.43 tested values, on average, on a scale between 1 and 5.

In the experiments with parameter *kernel std*, the mean error of predicted value using leave-one-out evaluation method was 2.14. As there 10 possible values of *kernel std* (there were 5 values for $\varepsilon$), this could be considered quite good result.

It is also interesting to note which dataset measures cropped out most often in our leave-one-out study. In the test with $\varepsilon$ these were: *Ratio of number of examples to the number of attributes, Coefficient of linear regression ($R^2$, using binarized symbolic attributes) and Stationarity of the target.* In the test with *kernel std* these were: *Number of attributes (binarized), Proportion of numeric attributes with outliers* and *Coefficient of linear regression ($R^2$, using binarized symbolic attributes).*

### 3.4 Future Work

It would be interesting to see whether this strategy is competitive when compared to the other ones described earlier. To verify that we could carry out the following study:

- suggest the initial parameter values with the help of the meta-learning approach,
- define a small local neighborhood (exploiting possibly again the meta-learning approach);
- carry out a restricted local search for better parameter settings;
- verify whether better results than for the approaches described earlier (e.g. using the Reduced search on sample data);

We plan to carry out further work along these lines in future.

## 4 Conclusions

In this paper we have explored various approaches for setting parameters of SVM. These can be divided into two groups. The first one relies on our prior knowledge of useful parameter ranges. We have shown that the search for good parameter setting can be carried out in a principled way and can exploit data samples, without significant loss of predictive accuracy (i.e. being able to predict the correct parameter value).

The second group uses meta-learning to solve the same task. The advantage of this method is that there is no need to run any experiments. It's only necessary to generate characteristics of the given dataset.

The advantage of the method relying on sampling, on the other hand, is that search using samples does not need any knowledge about past performance of SVM on different datasets.

One way to improve results of presented task that was not explored in this paper is to combine both methods. It is possible to use meta-learning to predict the range of good values of parameters and then use search using data samples to find the best setting in this range. The advantage of this approach is that reduced search space for sampling would allow us to test denser set of values with the same effort.

**Acknowledgements**

We would like to thank the LIACC members for useful discussions. This work has been supported by ESPRIT METAL Project.

## References

1. H. Bensusan and A. Kalousis, Estimating the Predictive Accuracy of a Classifier, In Proc. Proceedings of the 12th European Conference on Machine Learning, pages 25-36, 2001.
2. P. Brazdil, J. Gama and B. Henery, Characterizing the Applicability of Classification Algorithms Using Meta-Level Learning, Proc. of the European Conference on Machine Learning (ECML-94), pages 83-102, 1994.
3. P.Brazdil, C.Soares and J.Costa: Ranking Learning Algorithms, To appear in Machine Learning Journal, Kluwer Academic Publishers, 2003.
4. R. Collobert and S. Bengio, SVMTorch: Support Vector Machines for Large-Scale Regression Problme, Journal of Machine Learning Research, vol 1, pages 143-160, 2001.
5. Nello Cristianini and John Shawe-Taylor. An Introduction to Support Vector Machines. Cambridge University Press, Cambridge, UK, 2000.
6. B. Pfahringer, H. Bensusan and C. Giraud-Carrier, Tell Me Who Can Learn You and I Can Tell You Who You are: Landmarking Various Learning Algorithms, In Proc. of the Seventeenth International Conference on Machine Learning (ICML2000), pages 743-750, 2000.
7. The R Project for Statistical Computing, http://www.r-project.org/.