Margin Driven Separate and Conquer by Assymmetric Loss Functions

Technical Report TUD–KE–2011–01

Lorenz Weizsäcker, Johannes Fürnkranz
Knowledge Engineering Group, Technische Universität Darmstadt

Abstract

Separate and Conquer training for binary classification greedily builds a linear decision tree from the root to the leaf. This training scheme stems from the context of rule learning where a single rule is not expressiveness enough to serve as full prediction model. When we apply kernel methods a single model usually suffices. Nonetheless, it might advantageous to use multiple models that apply to different parts of the input space in this context too. Single SVM-models have the short-coming that there one parameter only to trade-off regularization and error. This can be inappropriate if the classes are intermingled to different degree in different part of the input space. We try a margin-driven, kernel-based separate and conquer algorithm iteratively builds a list of partial models by means of a C-SVM-like objective with asymmetric losses. False-negatives can be undone by subsequent partial models, therefore we use a weaker loss on the class the model is to detect. Experimental results show that model lists obtained with this training scheme can indeed compensate weak kernels, though only for some of the probed data sets. They also document the main flaw of the scheme: it is likely to happen that taking out parts of the input space produce hard residual problems instead of alleviating them as intended.

Contents

1 Introduction

Separate and conquer learning (*seco*) [12], or *covering algorithm* [11], is a generic rule learning training scheme for classification. A seco model is a list of rules, or more generally, a list of partial models each covering another part of the input space. In terms of vector-like inputs a rule corresponds to a partially open, multi-dimensional interval in the original feature space. A single rule is usually not sufficiently expressive for successful discrimination. Therefor, one has to build a list of rules: a positive response of a rule on a test point is taken as predicting the class associated with that rule, while negative response means no commitment, and the test point is passed to the next rule. The last rule is obliged to make prediction.

With this work we want to investigate whether separate and conquer learning can be applied with kernel based partial models instead of rule-like partial models. The main issue in designing such a training algorithm is to generate hyperplanes that represent reasonable *cuts*, e.g. partial models. We do so by minimizing objectives with asymmetric loss functions that give little penalty to false-negatives because negatives does not mean a prediction. We apply that seco algorithm on common binary data-sets in order to compare its performance to plain SVM-training. Our pretense was that using multiple cuts should increase the performance at least if we use sufficiently weak kernels. Since the input model in terms of the applied kernel in most cases can easily be fitted to the prediction task, this would not mean an amendment in performance. But it would show that one can compensate to certain degree the weakness of a given input model by applying lists of partial models.

It turns out that performance comparison of the proposed seco algorithm to the baseline of plain SVM-training is inconclusive. For a fixed kernel, seco might yield an error-rate above or below the baseline depending on the data set. But it is clear that the seco scheme does not beat for instance Gaussian kernel that are jointly tuned with the error-vs-regularization trade-off-factor. We tried many variants of the seco scheme without finding a clear gain. Although there are many more untested variants which are potentially perform better, we finally came to the conclusion that seco scheme under study is not the wright way to go for potential benefits of problem decomposition along partial models.

Why should should we belief in benefits of partial models in context of kernel input representation anyway? With high capacity kernel like radial basis functions separability is not an issue. Rather, one has to find the suited degree of *inertia* in terms of a good combination of window size and weighting of regularization versus error penalty. This parameter selection is usually done through trial and error on sub-samples or, more elaborately, via bounds on the generalization error which are not tight but nonetheless predictive, see for instance [6]. Although the decision boundary is, at least for smaller window sizes, mainly influenced by neighboring training points, there also is distant effect through the global fitting of the inertia as part of training procedure. This distant effect is the greater the lower the capacity of the kernel function is. A potential advantage of partial model is that it allows individual adaption of inertia for different parts of the induced feature space.

However, there is no obvious way to find the parts of the feature space that should covered by separate models and we can only provide a simple and greedy scheme for detecting such. Furthermore, beside any positive effect it may have, feature space separation can also harm generalization. Errors in a subspace that already has been cut off are not relevant, but removed training points could also have positive influence on the subsequent decision border within the uncut area, if they were not removed from the training set.

There are many approaches to build partial models on binary classification. We only mention some of them and entirely omit methods that typically are referred in the context of ensemble methods. In [13] the authors apply an external hierarchical clustering algorithm that lays a partition tree over the input space in advance. The models for assigning the test instance to a leaf partition and subsequently to an output class are trained in a second step. A similar approach can be found in [8]. In contrast to such approaches and also to the one presented here, [3] jointly trains the partition and classification models by means of an objective on the global model. The penalty term of the objective has one variable factor for each level of the partition tree what prohibits standard dualization for more than one inner node. An instance based approach is presented in [9]. The k nearest neighboors of the test instance are selected as training set to which a linear SVM is applied. The distances are the euclidean distances in the input where the attributes are weighted by factors obtained from attribute-wise variance analysis. The training consists of joint tuning of the number k, the SVM-trade-off-parameter, as well as an parameter in the calculation of the distance weights.

In the first part of section 2 we give a formal introduction to separate and conquer training and fix the frame training scheme as used for this work. We then make an excursion to large margin framework that is supplemented by an subsection on some notions on kernels. In the subsections 2.5 and 2.6 track back to our seco scheme by fixing the way of supplying candidate submodels and selecting one them. Section 3 is reserved for the heuristics for optimizing the asymmetric objective given in subsection 2.5 and which we for searching candidate submodels. In section 4 we begin with another excursion on bounds for SVM-trade-off parameters. Subsequently we report on the experimental results for comparing the proposed algorithm with plain SVM-training and on additional experiments to light out the phenomenon of poor performance on residual problems. The final section contains a short conclusion that revisits the main characteristics of the training algorithm discussed in this work.

2 Partial Models

2.1 Seco Model: Sequence of One-sided Partial Models

We consider seco learning for binary classification. The training algorithm maps a training set $A \subset \mathscr{X} \times \mathscr{Y}$, with arbitrary input space $\mathscr{X}$ and outputs from $\mathscr{Y} = \{\pm 1\}$, to a *global model* ω that corresponds to a binary classifier $\omega \colon \mathscr{X} \to \mathscr{Y}$. For sake of lean notation we denote the model and the corresponding classifier by the same symbol and assume unique inputs such that the outputs are functional dependent on the inputs. ω is aimed to agree with the test set $E \subset \mathscr{X} \times \mathscr{Y}$ if A and E stems from the same source.

The global model is a list of *signed partial models* $\omega = \big((\alpha_1, s_1), \ldots, (\alpha_r, s_r), (\alpha_\dashv, s_\dashv)\big)$ for indefinite $r \geq 0$. A signed partial model (α, s) consists of a binary model α and a sign $s \in \{\pm 1\}$. The *sign* specifies the semantics of αs output. If $\alpha(x) = +1$, the true label $y(x)$ is predicted as $\hat{y}(x) = s$. If, on the other hand, $\alpha(x) = -1$, the partial model makes no commitment on the class of x. We refer to the class $+s$ as *target class* of (α, s) and to $-s$ as its *default class*. We say such partial model is *one-sided* opposed two a common *two-sided* binary model where both possible outputs represent a commitment on the estimated label.

To compute $\omega(x)$ on a test instance x we subsequently apply the partial models to it. The sign s of the first partial model with $\alpha(x) = +1$, thereby making a commitment, is taken as prediction of ω. If non of the first r partial models makes an commitment, the output equals the prediction of last partial model, $\omega(x) = s_\dashv \alpha_\dashv(x)$. That is, the last partial model is taken as two-sided model making a commitment in any case and we can fix its sign as $s_\dashv = +1$. Since the one-sided partial models somehow cut off a part of the feature space we also refer to them as *cuts*.

For training ω, we subsequently train its partial models. The *covered* points, e.g. those on that the current partial model makes a commitment, are removed from the training set before the next partial model is build, hence the name separate and conquer. The next model only refers to that part of the input space that has not been covered by the previous models, its output on the already covered part has no relevance.

2.2 Training Algorithm: Supply and Select

The seco training scheme below is generic as includes two subroutines, candidate supply and candidate assessment which we discuss later. But beside these two compounds this is the scheme as applied in the experiments. Here, you should read ω as the list of partial models build so far, and, correspondingly, A as the those training examples that have not yet been covered. For training the next partial model, we first build a set of candidate models and then select one based on an assessment function.

- build next cut $c = (\alpha, s)$
    - *candidate supply*: build a set C of candidates cuts
    - *candidate assessment*: select a cut c from C according to a real valued *selection criterion*
    - assess *stopping criteria*:
        - low coverage: c covers less than $1/16$ of the previously uncovered points
        - beggarly leftovers: c lets less then 32 points uncovered for either of the classes
        - easy task: the problem A can be solved in a two-sided manner with an cv-error below $1/32$
- if neither stopping criterion is fulfilled append c to ω, remove covered points from A and repeat
- otherwise drop c and finalize ω by $(\alpha_\dashv, +1)$ with $\alpha_\dashv$ trained in a two-sided manner

The stopping criteria are arbitrary tightenings of two needed conditions: If we allowed no coverage, we would run in infinite loop and if a class is not present in the uncovered set, we cannot train the final model. Note that this finalization strategy is conservative in the sense that if the design of the cuts turns out to be too close or, the other way round, too greedy, the final model still can fill in.

By time of being, the distinction between *candidate supply* and *candidate assessment* is rather informal. The first breaks down the infinite set of possible partial models to a *small* set of candidates and the later makes a final choice therefrom.

The candidates are build by heuristic optimization on different objectives each. If we were certain about the objective, we would assume the optimizer to be the best candidate. But we do not know what the right objective is and therefor try many of them and outsource the final decision to candidate assessment. The set of candidates is spanned by four dimension. First, there are two possible target classes. Second, the generic objective has an open parameter. Third, we additionally try different relatives weights on the different penalties for margin violations by target and default training points. Fourth, the objectives have local optima. We account for with an heuristics optimization but also by applying a small number of different starting points.

Once candidate supply has provided the pre-selection, candidate assessment calculates a selection value on each candidate and we take the minimizer as next partial model.

2.3 Large Margin Framework

The general model type considered in this work are linear combinations on pair-wise similarities likewise kernel-based SVM-training. A candidate partial models minimizes a target function Q which is a weighted sum of a quadratic regularization term and the average loss on the training points according to margin-based loss functions. The one-sided character of partial models is reflected by loss-asymmetry: the loss on default class L^- is some standard loss such as Hinge-loss while the loss on the target class L^+ is constant for negative margin values.

The losses are non-increasing functions of the *functional margin* $m(x) = y(x)sf(x)$ where s is the given partial model and $f(x)$ is a real-valued *score* on x that we aim to be ≥ 0 iff the test input x belongs to the target class such we can use $h(x) = \text{sign}(sf(x))$ as predictor. We fix f as

$$f(x) = g(x) - \tau \qquad g(x) = \beta^\mathsf{T} K(A, x). \tag{2.1}$$

$g(x)$ and τ are called *relative score* and *threshold* respectively. The values $K(A, x) = (\kappa(\bar{x}, x))_{\bar{x} \in A}$ compare x to the training points with respect to a *similarity function* $\kappa\colon \mathscr{X} \times \mathscr{X} \to [0, 1]$ where we take greater value as greater similarity. For given κ, sign s, and trade-off parameter $C > 0$, a candidate partial model (α, s), $\alpha = (\beta, \tau)$, is obtained by minimizing an objective of the form

$$Q(\alpha) = R(\beta) + C \sum_{t \in \{-,+\}} \frac{1}{|A^t|} \sum_{x \in A^t} L^t(m(x)), \qquad m(x) = s\ y(x) f(x). \tag{2.2}$$

A^- and A^+ hold the training inputs that belong to the default and the target class respectively. $R(\beta)$ is called *regulizer*. It grows in the scale of β such that we cannot obtain smaller Q simply by increasing that scale and thereby decreasing $L^\pm$.

Independent of the concrete κ, $L^\cdot$ and R one can consider this as *large margin training* since the error term with decreasing loss functions pushes towards large values for functional margins $m(x)$. But the training scheme gets additional large-margin character and also becomes more well-rounded if we apply further restrictions on these three ingredients κ, $L^\pm$ and R. For more detailed information on the following we refer to section 2.2 and 4.2. in [14].

Consider the set of feasible relative score functions that are based on finite training sets:

$$G = \bigcup_{X \subset \mathscr{X}, |X| < \infty} G(X) \tag{2.3}$$

with

$$G(X) = \{g \mid g(x) = \beta^\mathsf{T} K(X, x),\ \beta \in \mathbb{R}^{|X|}\}. \tag{2.4}$$

In extension of the notation $K(\cdot,\cdot)$ we write $K(X, \bar{X})$ for the real matrix $(\kappa(x, \bar{x}))_{x \in X, \bar{x} \in \bar{X}}$ as well as $K(X)$ for the vector of functions $(\kappa(x, \cdot))_{x \in X}$. If we demand κ is to be a *kernel* in the sense that, for any finite X, the *kernel matrix* $K(X, X)$ is positive semi-definite, we can define a dot product on G. For $g, \bar{g} \in G$ with $g = \beta^\mathsf{T} K(X)$ and $\bar{g} = \bar{\beta}^\mathsf{T} K(\bar{X})$ it is given as

$$\langle g, \bar{g} \rangle = \bar{\beta}^\mathsf{T} K(X, \bar{X}) \bar{\beta}. \tag{2.5}$$

The relative score function g is now an element of a well-rounded dot-product space induced by the kernel κ. For any $x \in \mathscr{X}$, the relative score on x becomes the product of the model and xs representer $k(x, \cdot)$ in G: $g(x) = \langle g, k(x, \cdot) \rangle$. Also, this give us a metric $||g|| = \left(\langle g, g \rangle\right)^{1/2}$ and thereby a *scale* $\sigma = ||g||$ of the model. The inverse $\gamma = 1/||g||$ can be seen as *model margin* in the following sense. For the normalized model $\bar{g} = g/\sigma$, $\bar{\tau} = \tau/\sigma$ we have

$$m(x) = 1 \Leftrightarrow \langle \bar{g}, k(x, \cdot) \rangle = \bar{\tau} \pm \gamma. \tag{2.6}$$

In other words, x has functional margin 1 if the projection of the its representer on g has distance γ to the normalized threshold and the distance $\delta(x) = \langle \bar{g}, k(x, \cdot) \rangle - \bar{\tau}$ has the geometric interpretation as distance to the decision boundary. If we set $L^\cdot$ to vanish for $m \geq 1$, a point x makes no loss contribution if $k(x, \cdot)$ is on the wright side of boundary and $\delta(x)$ exceeds the model margin γ.

Representer Theorem

If we set $R = \langle g, g \rangle$, it can be seen that we wont obtain smaller Q by considering any relative score in G instead of restricting ourself to $G(A)$ (see Theorem 4.3 in [14]). In other words, we build the minimizer over $F(A) = \{f = g - \tau \mid g \in G(A), \tau \in \mathbb{R}\}$ and get the minimizer of the larger and more well-rounded $F = \{f = g - \tau \mid g \in G, \tau \in \mathbb{R}\}$. The first formulation of this so-called *representer theorem* was given in [10]. In order to obtain this result, you decompose the optimum f into the threshold τ, the projection g_G of $g = f - \tau$ on $G(A)$ and the rest $g_\perp = f - g_G - \tau$. For the loss terms, the rest is irrelevant, and by Pythagoras' theorem the quadratic regulizer can only decrease by dropping the orthogonal rest. Note that, opposed to common usage, we do not need to build the closure of G for this result since $G(A)$ has as a finite number of dimensions.

Kernel Transform

For our experiments we use an variant of the objective (2.2) that can somehow facilitate the optimization. Let M be a root of $K = K(A, A)$ and B the pseudo-inverse of M. They can be obtained from *sparse eigenvalue decomposition* $U^\top D U$ of K where the diagonal matrix D holds only the non-zeros eigenvalues and U the corresponding normalized eigenvectors as rows. Then, $M = D^{1/2} U^\top$ and $B = D^{-1/2} U$. We get an equivalent objective over $(\bar\beta, \tau)$ by setting $g(x) = \bar\beta^\top B K(A, x)$ and $R = \bar\beta^\top \bar\beta$. The regulizer becomes a bit simpler and number of parameter to be optimized may be smaller, if the rank q of K is smaller than $|A|$. As in principal component analysis we can also deliberately drop rows of U that correspond to smaller eigenvalues.

ν-SVM

The objective (2.2) is a asymmetric variant of the commonly C-SVM. The C-SVM has a sister called ν-SVM [14]. Although both are equivalent in terms of expressiveness and will under normal conditions yield same error rates, the latter is much more recommendable from conceptional point of view: it clearly separates the rule of regularization in term of small model margin, and regularization in terms of balanced model entries. Unfortunately, we cannot use ν-SVM-style objective here because we need the margin to be linear in the arguments of Q, otherwise we cannot apply the smoothing technique we need to achieve an heuristic minimization, sf. section 3.

2.4 Base-, Unit-, and Outer-Kernel

The kernels we apply in this work have two factors. First, we apply a parameter-free *unit kernel* $\kappa_U \colon \mathscr{X} \times \mathscr{X} \to [0, 1]$. Then, we map that value in unit interval again by a unary *outer mapping* $\rho_a \colon [0, 1] \to [0, 1]$ that depend on a parameter a.

$$\kappa(x, \bar x) = \rho_a(\kappa_U(x, \bar x)) \tag{2.7}$$

To obtain the unit kernel we start from *base kernel* κ_0 that defines the data or is provided along with the data. In case of vector data, $\mathscr{X} = \mathbb{R}^d$, the base kernel is κ_S that applies the canonical dot product on the entry-wise standardized inputs.

$$\kappa_0(x, \bar x) = \kappa_S(x, \bar x) = \langle x', \bar x' \rangle \qquad x' = (x - \mu)/\sigma \tag{2.8}$$

μ and σ are both in $\mathbb{R}^d$ and hold the entry-wise mean and standard deviation over the training set respectively. The division by σ is understood entry-wise too and in case the an entry has zero standard deviation the normalized entry is set to zero itself.

We then normalize the base kernel to the *normalized kernel* κ_N

$$\kappa_N(x, \bar x) = \kappa_0(x, \bar x) / \left(\kappa_0(x, x)\kappa_0(\bar x, \bar x)\right)^{1/2} \tag{2.9}$$

with values in $[-1, 1]$. For vector data we can instead normalize the standardized inputs them-self, $x' \mapsto x'/||x'||$. Finally, we apply a linear transform obtaining κ_U that has values in the unit-interval.

$$\kappa_U(x, \bar x) = \left(\kappa_N(x, \bar x) + 1\right)/2 \tag{2.10}$$

This κ_U fulfills condition for being a kernel given in section 2.3. The final transform does not affect definiteness as a matrix with identical, non-negative entries always has a real root. It might, however, change the rank of the kernel matrix.

We say that the down-stream ρ is a *outer kernel* iff $\kappa = \rho \circ \bar\kappa$ is kernel whenever $\bar\kappa$ is. Not every mapping on the unit interval has this property but we do not have a characterization of outer kernels. The two following functions are outer kernels on the unit interval.

$$\rho_a^{\text{pol}}(t) = t^a \qquad \rho_a^{\text{exp}}(t) = (a^t - 1)/(a - 1) \tag{2.11}$$

In case of the *polynomial outer kernel* ρ^{pol} this is clear because the set of kernels is closed under multiplication. For the *exponential outer kernel* ρ^{exp} you can apply theorem 13.4 in [14] along with the series expansion of ρ^{exp}.

See figure 4.2 for plots of both outer kernels for different parameters a. Both are convex bijections on the unit interval starting at $(0,0)$ and ending in $(1,1)$. In both cases a greater parameter a means that the graph make a deeper and sharper curve, e.g. gets closer to the point $(1,0)$.

Note that *exponential kernel* $\kappa^{\text{exp}} = \rho^{\text{exp}} \circ \kappa_{\text{U}}$ does not equal the Gauss kernel $\kappa^{\text{Gauss}}(x,\bar{x}) = \exp(-c||x-\bar{x}||^2)$, $c > 0$, even for normalized $x, \bar{x}$. The main difference is that we want κ^{exp} to start at 0 while the value of the Gaussian kernel is at least $1/e$ for normalized inputs. We can obtain the Gauss kernel on standardized and normalized inputs by applying $\rho_c(t) = \exp(4c(t-1))$ on unit kernel values.

We conclude this subsection with an informal discussion on base vs outer kernel. The base kernel initiates the pairwise-similarity view on the data. It holds the gross of the domain knowledge. One can argue that domain knowledge might help choosing a well-performing outer kernel. But we promote the view that the base kernel covers the domain knowledge while the purpose of the outer kernel is to add a parameter to the similarity measure that gives the model additional capacity. The parameter a is tuned along with the trade-off parameter C and this joint tuning is an indispensable part of the training. However, only a is part of the mode in the sense that it is needed at test time.

The demand on ρ of being an outer kernel is motivated by the aim of the geometrical interpretation of the model as hyper-plan in a space induced by κ which give us, as among others, the notion of model margin. It seems rather unlikely, however, that we would harm the performance by choosing some reasonable family of outer mappings that are not outer kernels. In order to study this, one could probe peace-wise linear outer mapping with $\rho(1) = 1$ and one or two increasing but otherwise arbitrary support points. This would mean that a has two or four compounds which would dramatically increase training time but it still could be tackled with tuning by grid search. We do not follow this direction here.

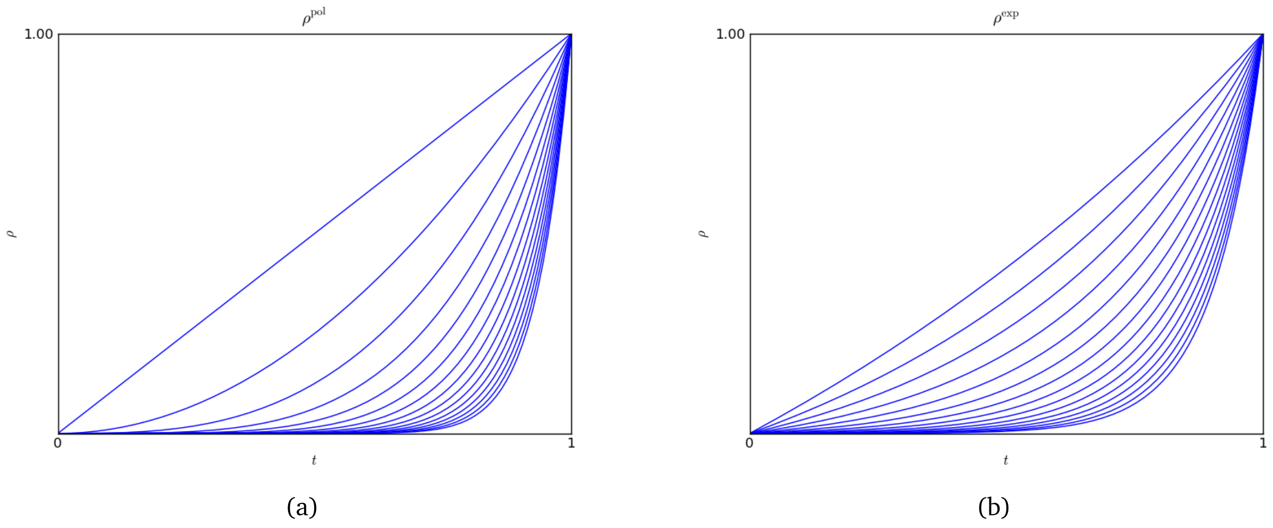


Figure 2.1: Plot (a) shows ρ^{pol}_a with $a = 1, \dots 16$, plot (b) shows ρ^{exp}_a with $a = 2^k, k = 1, \dots 16$. Both function types can be used as outer kernel. It would be very surprising, however, if not having this property affected the performance.

2.5 Candidate Supply: Asymmetric Loss Functions

As mentioned in in section 2.2 we seek for valuable candidates by heuristic optimization of different objectives. We fixed the generic form of the objectives in section 2.3 and further specified them section 2.4 by setting the type of kernels. Now we set the concrete margin based loss functions. We have to translate the concept one-sided partial models into concrete concrete asymmetric losses. There many ways to do this and we have not got good conclusions on this topic yet. In the following we present some simplistic thoughts. A rigorous analysis or empirical studies thereon is not part of this work.

False positive classification cannot be undone, in contrast uncovered target points which may be hit by subsequence partial model of same sign. Therefore, we want a loss design that defends negatives from margin violation more than the positives. A simple mean to achieve this is by bounding the loss on the target class by a constant. This can also be interpreted as a linear reward on coverage since the total target loss becomes multiple of the fraction of uncovered target points.

What should be the behavior in a imaginary *heads-up situation*? Think about a single negative that faces a single positive point, both lying on the correct side of the separating plane but still inside a given margin and the plane is orthogonal to the line connecting the two points. In such situation we might not want the plane not to be pushed in either direction which translates into equal derivations of L^- and L^+ for margin in the unit interval or at least within a part of it.

As additional directive we restrict to continuous loss functions in order to obtain surface of the objective not too ragged. We finally committed ourself to the loss pair shown in figure 4.1 that we considered as intelligible, and checked on development data sets that this choice can at least compete with other variants.

In the experiments we probe, however, different relative weights on the two loss function, e.g. the target loss is multiplied by different positive factors. This means that we introduce a parameter on the loss pair that controls certain aspect of the interplay of the two losses, such as the behavior in heads-up situations, and but that is not fixed in advance.

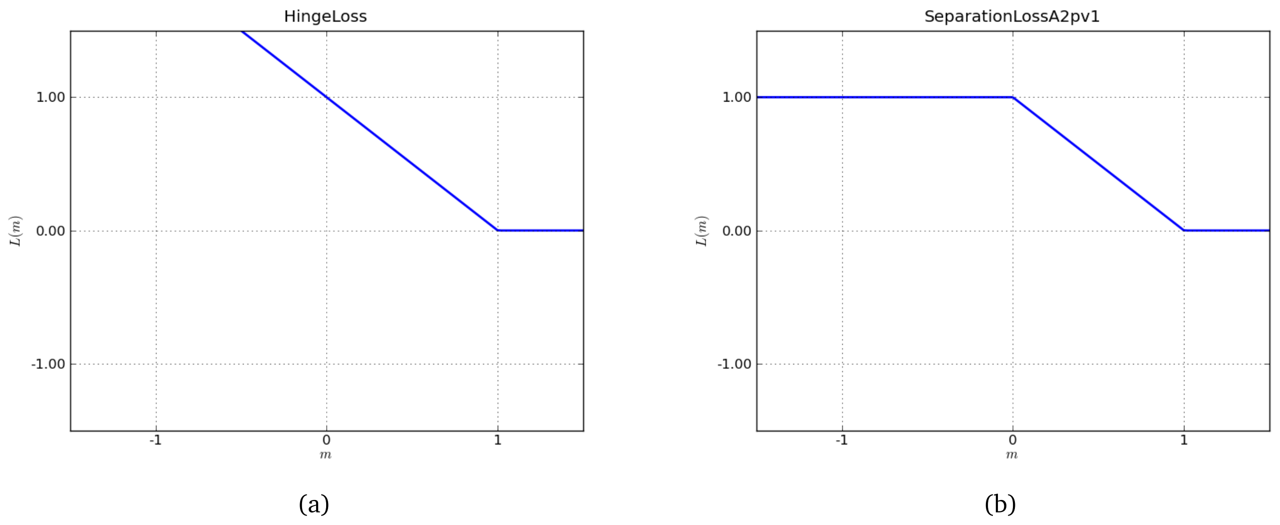
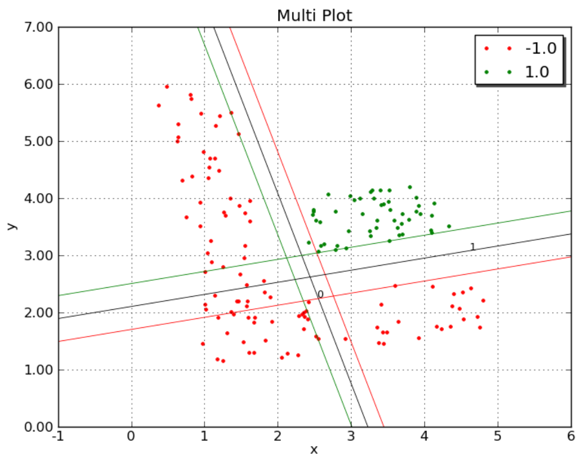
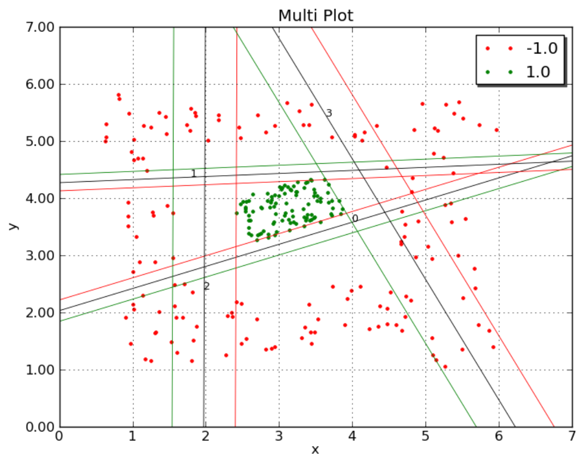


Figure 2.2: The loss function for the default class (a) and the target class (b) respectively. There are other ways to set the loss-pair but in the final experiments we used this combination.



(a)



(b)

Figure 2.3: 2-dimensional toy training data sets and the corresponding linear cuts (black lines). The cuts numbered according to their order in the sequence. The green and the red line indicate the functional unit margin. The green line is on the positive side relative two the sign of the cut. In (a) you can see how the cut (line 0) is too greedy: it tries to cover more red point at cost smaller margin and greater margin violations. This problem does not affect cut 2 and 3 in (b) because the cut 0 and 1 have already removed the points on top and bottom side.

2.6 Candidate Assessment: Last Cut Error Contribution

Within candidate supply we get around stating the quality of a cut to some extend, but when it comes to candidate assessment we have to define some explicit criterion to prefer one cut to another. It is not clear to us how such criterion should look like. A cut is a partial model that we have to judge by its contribution to the effectiveness of the global model. It is understood that a cut should not cover non-target points as far as possible. Cutting also aims to alleviate the job on the uncovered points, but whether a concrete cut does so depends the subsequent partial models which are not accessible at selection time. We therefore simplify reasoning in the spirit of the greediness of seco and judge any candidate as if was the last within the global model.

At first place, we define the *target criterion* that is based on test set. Since such value is unaccessible at training time, we also have to provide *viable criterion* that estimates the value of the target-criterion. One could also apply a criterion that is not associated to any target. The main advantage of an estimator-target break-down is that it supports separate evaluation of the selection scheme and that we can run experiments *relative* to this estimation subtask, see experimental section.

Assuming that the candidate cut is, in the sequence of partial models, followed only by the final symmetric model, a proximate target selection criterion is the contribution to the total error that is given by one-sided cut-error and the residual error of the subsequent model on the uncovered part. Both refer the current test set E that hold only those points from the original test set that are not covered by previous cuts.

- *cut error* e^{c}: fraction of test points that are covered but do not belong to the target class over all test points
- *residual error* e^{r}: error on uncovered test points of the residual model trained with uncovered training points

For the definition of the residual error we assume that the training of the final model is deterministic. Otherwise, we replace it by its expectation over the random bits.

The two error parts are weighted according to the fraction of points on which on which errors can be made. We denote these weights correspondingly to the error rates by p^{c} and p^{u}. They are the fraction of point in E that are covered, respectively uncovered by c. All together we take *last cut error contribution* of c as target criterion.

$$e^{\mathrm{t}} = p^{\mathrm{c}}e^{\mathrm{c}} + p^{\mathrm{r}}e^{\mathrm{r}}. \tag{2.12}$$

In order to obtain a simple viable criterion therefor we use estimators for the four ingredients as follows. The weights are estimate by corresponding weights on the training set. Similarly, we estimate e^{c} by training error. A better estimation based on the margin violations is set aside until we see how bad it is and whether a better estimation has the potential to provide gain to the overall performance. For the residual error, situation is a different. In any case we have to train the residual model to which the target criterion refers to. Since this training includes the adaption of the trade-off parameter via cross validation we already have reasonable cv-estimation of the residual error.

3 Heuristic Optimization through Iterative Smoothing

Our goal is to find a model (β^*, b^*) such that $Q(\beta^*, b^*)$ is local minimum on Q and the global minimum is *not much smaller*. To this end we follow the lines of [5] by applying a *continuation method* which we expect to provide a *sufficiently low* local minimum. We do not characterize here what we mean exactly by *much smaller* and *sufficiently low*.

The continuation method is simple and can be seen as a expectation driven simulated anealing, see for instance [16] for a brief explanation of the latter. The objective is iteratively optimized under decreasing level of smoothing and the solution is used as starting point for the subsequent iteration. In the following we the ansatz for smoothing and derive a close form smoothed objective as well as for its derivatives. The latter is important for efficient gradient descent.

At first place we incorporate the threshold b into β by adding constant compound -1 to the vectors $K(x)$ of transform inputs. This simplifies notation and unbinds the symbol b that we use otherwise later. We should keep in mind, however, that we still want to exclude the threshold from regularization and therefor denote by $\bar{\beta}$ the weights *without* the additional compound that corresponds to the threshold.

Let n be the number of dimensions of the transformed inputs and let t be a $(n+1)$-dimensional random variable that has normal distribution $\mathrm{N}(0,\sigma)$ with independent components of zero mean and standard deviation σ. The *smoothed objective* $\widetilde{Q}$ is the expectation of the objective Q when we add iid Gaussian *noise* to β.

$$\widetilde{Q}_\sigma(\beta) = \underset{t\sim\mathrm{N}(0,\sigma)}{\mathrm{E}} Q(\beta + t) \tag{3.1}$$

The greater the deviation σ is the more smoothed becomes the objective. Now assume that we further have given an appropriate decreasing sequence of standard deviation $\sigma_1 > \ldots > \sigma_r$. The heuristic to find a good model work as follows. We start with arbitrary vector β^0 and iteratively compute $\beta^i = \operatorname{argmin}_\beta^{\beta^{i-1}} \left(\widetilde{Q}_{\sigma_i}(\beta)\right)$ where the superscript indicates the starting point for local minimization. We finally chose $\beta^* = \beta^r$.

For efficient computation of the optimizer we want a closed forms for the smoothed objective $\widetilde{Q}$ and its first and second derivatives. On the other side, we also want to try different loss functions L within the objective since it is not obvious how they should be chosen. A solution to this conflict is to consider a view base functions that serve as piecewise loss-compounds and for which we can prepare the smoothing in advance. This way we obtain closed forms for a set loss functions that is sufficiently broad for the purpose of this work.

3.1 Generic Smoothed Objective

At first place, let us state the smoothed version of the regularization term $R(\beta)$. Correspondingly to $\bar{\beta}$ let $\bar{t}$ be the noise variable without the last compound. Since the noise components are independent the mixed terms of the smoothed regulizer vanish: $\widetilde{R}(\beta) = \mathrm{E}_t\left((\bar{\beta}+\bar{t})^\mathsf{T} K(\bar{\beta}+\bar{t})\right) = \bar{\beta}^\mathsf{T} K\bar{\beta} + \mathrm{E}_t\left(\bar{t}^\mathsf{T} K\bar{t}\right)$. By the decomposition $K = U^\mathsf{T} DU$ the latter addend simplifies to $\mathrm{E}_t\left(\bar{t}^\mathsf{T} D\bar{t}\right) = \sigma^2 \mathrm{tr}(K)$ where $\mathrm{tr}(K)$ the sum over the eigenvalues of K.

$$\widetilde{R}(\beta) = \bar{\beta}^\mathsf{T} K\bar{\beta} + \sigma^2\mathrm{tr}(K) \tag{3.2}$$

In order to tread the loss term we assume a given input x such that we can deal with a scaled kernel representation $\bar{x} = K(x)/(l\sigma)$, $l = ||K(x)||$, and a *relative sign* $\bar{s} = sy(x)$ that is positive iff x belongs to the target class. For an arbitrary margin based loss L^s the smoothed loss term is the expected loss on m_t that is the margin m under noise t.

$$\widetilde{L}^s(x) = \underset{t\sim\mathrm{N}(0,\sigma)}{\mathrm{E}} \left(L(m_t)\right) \qquad m_t = \bar{s}l(\beta+t)^\mathsf{T}\bar{x} = m + \bar{s}l\sigma\bar{x}^\mathsf{T} t \tag{3.3}$$

$\bar{x}$ has length $1/\sigma$ such that $\bar{x}^\mathsf{T} t$ follows N_1, the standard normal distribution. Hence, we can replace the randomized margin by $m_t = \bar{s}l\sigma t + m$ where t now is one-dimensional and denote by E the expectation with respect to $t \sim \mathrm{N}_1$.

$$\widetilde{L}^s(x) = \mathrm{E}\left(L^s(m_t)\right) \qquad m_t = \bar{s}l\sigma t + m \tag{3.4}$$

To solve the expectation (3.4) for a sufficiently broad set of loss functions we split the domain of the loss function into a sequence of consecutive parts $(I_0,\ldots,I_{k-1}) = \left((-\infty,e_1),[e_1,e_2),\ldots,[e_{k-1},\infty)\right)$ with $e_i < e_{i+1}$ and define k separate *loss parts* L_i^s that vanish outside the respective interval I_i such that the loss decomposed into the parts

$$\widetilde{L}^s(x) = \sum_i \widetilde{L}_i^s(x) \tag{3.5}$$

and we can compute the smoothing for the parts separately.

Assume the i-th part covers the interval $[q,r)$ and we want $h = L_i^s$ to have the shape on $[q,r)$ that some *base function* $\bar{h}$ reveals on the interval $[\bar{q},\bar{r})$ but with additional factor c and offset d.

$$h(m) = c\bar{h}(am+b)+d \tag{3.6}$$

where $a = (\bar{r}-\bar{q})/(r-q)$ and $b = -qa+\bar{q}$ are chosen correspondingly to the borders of the two intervals. The corresponding *smoothed loss part* is

$$\begin{aligned}
\widetilde{h}(m) &= \mathrm{E}\left(\llbracket q \le m_t < r \rrbracket\, h(m_t)\right) & (3.7)\\
&= \mathrm{E}\left(\llbracket \bar{q} \le am_t+b < \bar{r} \rrbracket\, c\bar{h}(am_t+b)+d\right) & (3.8)\\
&= \begin{cases} \mathrm{E}\left(\llbracket \check{q} \le t < \check{r} \rrbracket\, c\bar{h}(\bar{a}t+\bar{b})+d\right) & \bar{s}=+1 \\ \mathrm{E}\left(\llbracket \check{r} < t \le \check{q} \rrbracket\, c\bar{h}(\bar{a}t+\bar{b})+d\right) & \bar{s}=-1 \end{cases} & (3.9)\\
&= \bar{s}\int_{\check{q}}^{\check{r}} \left(c\bar{h}(\bar{a}t+\bar{b})+d\right)p(t)\mathrm{d}t & (3.10)
\end{aligned}$$

with

$$\check{q} = (\bar{q}-\bar{b})/\bar{a} \qquad \check{r} = (\bar{r}-\bar{b})/\bar{a} \qquad \bar{a} = a\bar{s}\sigma l \qquad \bar{b} = am+b \tag{3.11}$$

and p being the density of N_1. With F being distribution function of N_1 and defining H as the indefinite integral

$$H(t) = H(t,\bar{a},\bar{b}) = \int \bar{h}(\bar{a}t+\bar{b})p(t)\mathrm{d}t \tag{3.12}$$

we get the generic smoothed loss part

$$\widetilde{h} \;=\; s\left(c\left(H(\check{r},\bar{a},\bar{b})-H(\check{q},\bar{a},\bar{b})\right)+d\left(F(\check{r})-F(\check{q})\right)\right). \tag{3.13}$$

In the border cases of the intervals I_0 and I_{k-1} we apply improper values $\check{q}=-\infty$ and $\check{r}=\infty$ by evaluating the respective limits $\lim_{t\to-\infty} H(t)$ and $\lim_{t\to+\infty} H(t)$.

3.2 Derivatives

The derivatives can also be stated in a generic way. The dependence of the loss on β goes through $\check{q},\check{r}$ and $\bar{b}$. We can eliminate the latter by $\bar{b} = \bar{q}-\bar{a}\check{q} = \bar{r}-\bar{a}\check{r}$ since we evaluate H at $\check{q}$ and $\check{r}$ only. Let us look at the case $t=\check{q}$, the case $t=\check{r}$ can treaded the same way. We substitute $\bar{b}$ by $\bar{q}-\bar{a}\check{q}$ and obtain $\bar{H}(t) = H(t,\bar{a},\bar{q}-\bar{a}t)$ on which we can apply the chain rule.

$$\frac{\mathrm{d}H(t)}{\mathrm{d}\beta} = \frac{\mathrm{d}\bar{H}(t)}{\mathrm{d}\beta} = \frac{\mathrm{d}\bar{H}(t)}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}\beta} = -\frac{\mathrm{d}\bar{H}(t)}{\mathrm{d}t}\bar{x} \tag{3.14}$$

By doing so twice we also get a pre-evaluated second derivative.

$$\frac{\mathrm{d}^2H(t)}{\mathrm{d}^2\beta} = \frac{\mathrm{d}^2\bar{H}(t)}{\mathrm{d}^2\beta} = \frac{\mathrm{d}^2\bar{H}(t)}{\mathrm{d}^2t}\bar{x}^\mathsf{T}\bar{x} \tag{3.15}$$

For given H stating the derivatives is now quite comfortable. All we have to do is to state the one-dimensional derivatives of $\bar{H}$. Afterwards, we might back-substitute $\bar{b}$ where possible. Since the smoothing of a reasonable function will be continuous, we can combine this proceeding with evaluation of the limits.

$$\frac{\mathrm{d}\lim_{t\to\infty}H(t)}{\mathrm{d}\beta} = \lim_{t\to\infty}\frac{\mathrm{d}H(t)}{\mathrm{d}\beta} \tag{3.16}$$

To summarize we state the generic derivatives of a smoothed loss part $\widetilde{h}$ where we use the $\cdot'$-notation to refer to the derivatives. As pointed out above the first derivatives of H is given as $\mathrm{d}\bar{H}/\mathrm{d}t$ and second as by $\mathrm{d}^2\bar{H}/\mathrm{d}^2t$.

$$\begin{aligned}
\widetilde{h}' &= -s\left(c\left(H'(\check{r},\bar{a},\bar{b})-H'(\check{q},\bar{a},\bar{b})\right)+d\left(p(\check{r})-p(\check{q})\right)\right)\bar{x} \qquad \text{and} & (3.17)\\
\widetilde{h}'' &= s\left(c\left(H''(\check{r},\bar{a},\bar{b})-H''(\check{q},\bar{a},\bar{b})\right)+d\left(p'(\check{r})-p'(\check{q})\right)\right)\bar{x}\bar{x}^\mathsf{T} & (3.18)
\end{aligned}$$

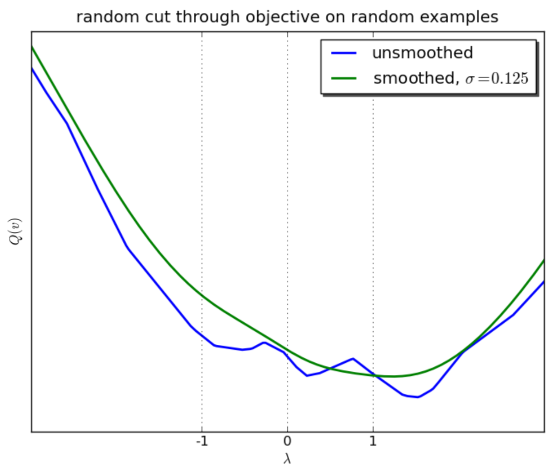
There are not very many base function for which we know how to get an analytical solution of the integral (3.12). Beside Gaussian density function itself we can only state H for monomials if we want to support arbitrary $\bar{a}$ and $\bar{b}$. Possibly, more base functions can be supplied with more mathematical knowledge but we should not forget that there are other server issues in with proposed training scheme. After all, we can combine the loss from the base function in a piecewise manner. In the following we give the solution of (3.12) for linear and quadratic $\bar{h}$.

For $\bar{h}(t) = t$ we obtain the first moment of the normal distribution by substitution. Let $c = (2\pi)^{-1/2}$.

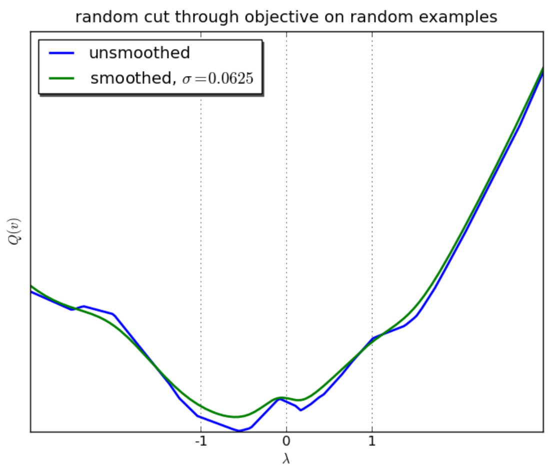
$$\begin{aligned} \int t\ p(t)\mathrm{d}t &= -c\int -t\exp(-t^2/2)\mathrm{d}t && (3.19)\\ &= -c\int \exp(s)\mathrm{d}s,\ s=-t^2/2 && (3.20)\\ &= -p(t) && (3.21)\end{aligned}$$

In order to solve the smoothed error on the base function $\bar{h}(t) = t^2$ we further need the second moment that can be obtained as follows.

$$\begin{aligned} \int t^2\ p(t)\mathrm{d}t &= -\int t\ p'(t)\mathrm{d}t && (3.22)\\ &= -tp(t) + F(t) && (3.23)\end{aligned}$$



(a)



(b)

Figure 3.1: The plots show the unsmoothed and the smoothed objective Q and $\widetilde{Q}$ with losses depicted in figure 4.1 and based on 32 random point from $\mathbb{R}^d$, $d = 32$. The objectives are evaluated at $v_0 + \lambda v_1$ with random $v_0, v_1 \in \mathbb{R}^d$ and λ along the x-axis. The two plots show the same objective based on different random points and different evaluation vectors v_0, v_1. As you can see from the right plot, one has to decrease the smoothing carefully in order to avoid bad local minima. In this work we use an ad-hoc scheme for setting the iterative smoothing scales.

4 Experiments

4.1 Bounds for Trade-Off Parameter

Choosing a good value for regularization-vs-error trade-off-factor C is an essential part of the C-SVM-training. In context of binary SVM with symmetric loss different methods have been developed to set that parameter guided by bounds on the generalization error. A common, though not very elegant method of setting C is to try different values and compare the performance of the resulting model on a *pre-test set* that is taken out of the training set. This is usually done in cross validation manner with a fold number of, say, 10 such that the trial training set is not much smaller than the entire training set on which the chosen C finally is applied.

In the following we discuss ways to set a reasonable search interval when the C is set by pre-testing. This way we may save computation cost and we get a better feeling as we do not totally surrender to trial and error. We derive two values c_0 and c_1 from training data. c_0 is a lower bound that holds under very weak assumptions. The second value, c_1, can be seen as upper bound for C but only under unrealistic assumptions. We rather consider c_1 as good first guess for C around which one should try different values. Though c_0 is a real bound, it is also less relevant in practice, since we observe that the best value with respect to the test set, C^*, can be expected to be within an interval of $[2^{-5}c_1, 2^{+5}c_1]$ and the bound c_0 is below that range.

For both values we consider the case of Hinge-loss $L(m) = \max(0, 1-m)$. The ansatz for the bound c_0 on C is that we assume the model margin not to be greater than $D/2$ with D being the maximal distance between two points from opposite classes. This give us a lower bound $\sigma_0 = 2/D$ on the model scale σ. For small C, the error is dominated by the regulization term in the sense that the model will always be pushed to smaller scale. We write $\mathrm{d}f/\mathrm{d}^- s$ and $\mathrm{d}f/\mathrm{d}^+ s$ for derivatives of f with respect to the scalar $s > 0$ in negative ($\rightarrow 0$) and positive ($\rightarrow \infty$) direction respectively. For sufficiently small C we have

$$-\frac{\mathrm{d}R}{\mathrm{d}^-\sigma} > C\frac{\mathrm{d}E}{\mathrm{d}^-\sigma}. \tag{4.1}$$

This means $\mathrm{d}Q/\mathrm{d}^-\sigma < 0$ and σ will shrink. The left hand side is $2\sigma \geq 2\sigma_0 = 4/D$ since we assume $\sigma \geq \sigma_0 = 2/D$. The right hand side is bound from above by $C\ \mathrm{d}L/\mathrm{d}^-\sigma \leq CZ$ with $Z = \max_{x \in A} ||K(x)||$ because $\langle \beta/\sigma, K(x)\rangle \leq ||K(x)||$. Hence, a sufficient condition for (4.1) is

$$C < c_0 = \frac{4}{ZD}. \tag{4.2}$$

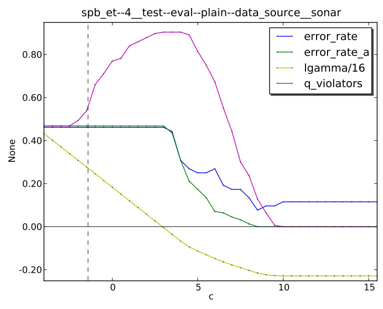
The corresponding proceeding for an upper bound on C would be to find a c_1 such that the absolute derivation of the error term always is greater than that of the regulization term if $C > c_1$. Unfortunately, there occur two problems. First, we do not get an upper bound for the scale as easily as before. We can get one if we assume linear separability and that the distance between the convex hulls of the two classes equals d. In such case, a geometric margin of $d/2$ is sufficient and $\sigma_1 = 2/d$ upper bounds the scale of the optimal model. The minimal distance of two points from different classes is not that bound since the distance of the convex hull can be smaller than that. Second, we need a lower bound on $z(x) = \langle \beta/\sigma, K(x)\rangle$ which we cannot state in general since the the example representation can even be orthogonal to β. If we also assume some lower bound $z \leq z(x)$ we correspondingly get

$$\frac{\mathrm{d}R}{\mathrm{d}^+\sigma} < -C\frac{1}{n}\frac{\mathrm{d}E}{\mathrm{d}^+\sigma}. \tag{4.3}$$

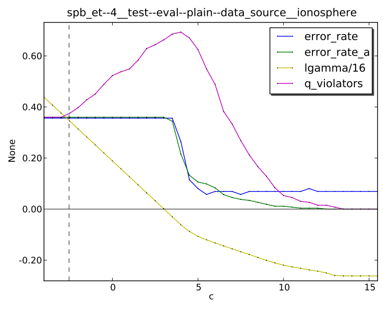
Thus, sufficiently large here means

$$\frac{4n}{zd} = c_1 < C. \tag{4.4}$$

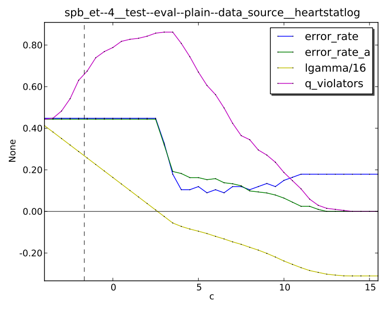
The practical relevance of this discussion is limited. We derived a lower bound on C that holds for any data set and kernel function. If we apply normalized a kernel, that bound will always be close to a fixed value since Z will be close 1 and D close to 2. For non-normalized kernels the bound may help. It should be noted, however, that the optimization for small values of C, and in particular for too small values, come and low computational cost. Concerning the upper bound, we have not derived true bound. It remains future work to define true upper bounds or deriving a good first guess on C based on the kernel values or a subset thereof.



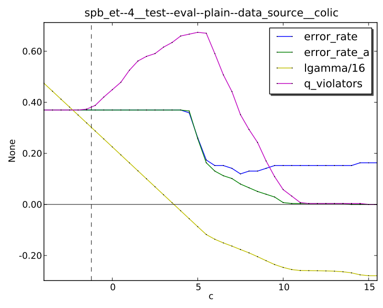
(a)



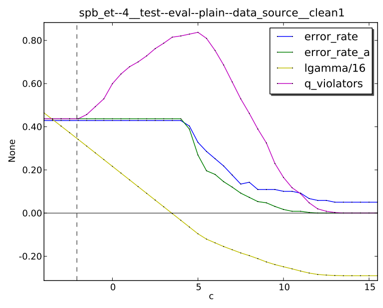
(b)



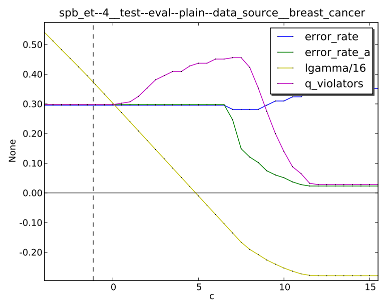
(c)



(d)



(e)



(f)

Figure 4.1: Each plots represents one data set through cubic kernel with a given training-test-set partition. The x-axis shows different values of $\log_2 C$. The green and the blue line indicate the training and test error respectively. The yellow line is the logarithm of model margin divided by 16 for better fitting the plot. The pink line shows the fraction of the margin violators within the training set. The vertical dashed line indicate the lower bounds c_0 on C. The training was done with LibSVM which puts an additional factor $1/2$ in R and we have correspondingly adapted the bound.

4.2 Deep Access to Experimental Code

We use a new experimentation tool [7] that supports access to the experimental code that been used for producing the experimental result. In the following plots and tables you find in the bottom-right corner hexadecimal string of length ten, the so-called *run-number*. A run-number allows you to access the code as it was when the experiment producing the respective result was run. We give a brief instruction how you execute such *original code inspection*. Unfortunately, the tool is linux-only.

- Download and unpack the peewit from its homepage [7].
- In order to make it run you have to install pyhton, numpy, matplotlib, and git.
- Download and unpack the project-export that contains the experimental code for this work from [1].
- Create a file `base_dir` in the directory `your_base_dir/projects/mdsc_sgo/.peewit/` that contains a single line with the path to `your_base_dir` relative to your home-directory. This overwrites the *base-dir* that was used by those who created the project.
- In order to have acces to the version history of peewit itself, you need to pull the peewit-repository from its host at sourceforge.net. To this end, open `mdsc_sgo/package/project_conf.py` and add the call `AR.pull_peewit()` at the end. This will turn your peewit-directory into local copy of the repo such that you can access the version of peewit that was used for running the respective experiment.
- Now, go to the project-code in `projects/mdsc_sgo/package` and call the function `restore` on the archivist instance in a position where it is sufficiently initialized. For instance you can do this at the very end of `project_conf.py`. As argument take of the run-number that you can find in this section.

In principal, the project-export is extensive enough that you can run the experiments again. However, for the main experiments we used a remote cluster and you should have access to such in order to run them in a reasonable time. If you want to do so, you further need to install ssh and unison for communication with the cluster head. Also, you have to adapt the file `project_conf.py` to the settings of your cluster. Note, whatever changes you make in the project while you are in a past state accessed by calling the restore-function will permanently vanish as soon as you make a valid call of that function again.

4.3 Baseline

We start with standard binary SVM-experiments that we use as baseline. They are accomplished with LibSVM [4] using the precomputed kernel values all based on the unit kernel as given in section 2.4. This allows a more direct comparison to cut-based results where we can use identical kernels. For the asymmetric training we cannot afford tuning of kernel parameters but we want to compare cut-based results to standard SVM-training with tuned parameters as well.

Table 4.3 shows the results for eight binary sets from [2]. We apply three polynomial outer kernel with fixed degree of 1, 2, and 3 and as polynomial and exponential outer kernel with tuned degree and window size respectively. It is noticeable that only in one case the kernel tuning clearly outperforms the polynomial kernel of degree 3.

The trade-off parameter C is probed on the values $C = 2^c$, $c = 4,\dots,16$. The kernel parameters are jointly tuned with C over the values $a = 1,\dots,16$, and $a = 2^b$, $b = -4,\dots,8$, for the polynomial and the exponential outer kernel respectively.

	poly_1	poly_2	poly_3	poly_g	gauss_s
sonar	0.245	0.130	0.130	0.115	0.125
ionosphere	0.097	0.091	0.100	0.105	0.088
colic	0.174	0.179	0.163	0.166	0.174
heartstatlog	0.163	0.174	0.178	0.185	0.193
breast_cancer	0.304	0.287	0.273	0.273	0.273
clean1	0.145	0.067	0.057	0.059	0.051
diabetes	0.233	0.229	0.251	0.229	0.233
parkinsons	0.133	0.112	0.107	0.056	0.056

svm_et--9--leaf_final--main 9cd754dede

Table 4.1: The baseline results obtained from symmetric SVM-training using LibSVM. The left three columns refer to polynomial outer kernel of degree 1, 2, and 3. The remaining two columns refer to training with polynomial and exponential outer kernel where the kernel parameter is tuned jointly with the trade-off parameter C.

4.4 Runs with the Proposed Scheme

We run an experiment with the datasets used for the baseline experiment, see table 4.3, using the separate and conquer scheme given in section 2.2. In the following we summarize details on scheme that have not been mentioned in previous sections and also give additional information on the experimental settings.

For the candidate supply we probed both signs $s = \pm 1$, the weights $C = 2^k, k = 8,\dots,15$ along with an additional factor $C^+ = 2^k$, $k = -1, 0, 1$ with which we scaled the L^+ on the target class relative to the loss L^- on the default class. That is the first term was multiplied by C and latter by CC^+. Further, we computed for each of these parameter setting four candidates stemming from independent random initializations of (β, τ). In total we therefor selected from $168 = 2 \cdot 7 \cdot 3 \cdot 4$ candidates. Many of the model are *run-away-models* with great τ and small entries of β. They have an objective value near zero as they have no coverage along with small regularization term.

In order to accomplish the optimization we chose a simple way with four iteration and a fixed sequence of window sizes $\sigma = (1/8)^k$, $k = 0,\dots,3$. We used the eigenvalue decomposition of the kernel matrix as described at the end off section 2.3 and dropped eigenvector that correspond to a eigenvalue smaller that t with

$$t = 2^{\mu+b} \qquad \mu = \operatorname*{avg}_{d\in D} \log_2 d, \tag{4.5}$$

D being the set of eigenvalues and $b = -4$. Such cut-off is necessary because the computed eigenvalue are always non-zero for numerical reasons and we cannot distinguish those numerical non-zeros from true small eigenvalues.

		error_plain	error_total	fraction_cut	error_cut	fraction_res	error_res	n_cuts
poly_1	sonar	0.255	0.139	0.399	0.058	0.601	0.219	2.625
	ionosphere	0.091	0.097	0.402	0.026	0.598	0.141	1.875
	colic	0.160	0.185	0.793	0.136	0.207	0.411	3.250
	heartstatlog	0.182	0.180	0.648	0.082	0.352	0.349	4.125
	breast_cancer	0.328	0.283	0.696	0.220	0.304	0.403	4.625
	clean1	0.137	0.111	0.809	0.084	0.191	0.223	3.625
	diabetes	0.236	0.217	0.474	0.074	0.526	0.352	6.250
	parkinsons	0.138	0.161	0.476	0.031	0.524	0.283	2.500
poly_2	sonar	0.139	0.117	0.337	0.038	0.663	0.182	2.875
	ionosphere	0.080	0.081	0.527	0.037	0.473	0.129	2.250
	colic	0.174	0.184	0.601	0.098	0.399	0.322	2.875
	heartstatlog	0.160	0.184	0.653	0.089	0.347	0.365	3.000
	breast_cancer	0.276	0.258	0.686	0.168	0.314	0.448	3.625
	clean1	0.061	0.053	0.546	0.030	0.454	0.101	2.500
	diabetes	0.228	0.244	0.457	0.122	0.543	0.364	3.625
	parkinsons	0.112	0.080	0.527	0.021	0.472	0.151	3.000
poly_3	sonar	0.125	0.091	0.404	0.043	0.596	0.114	3.875
	ionosphere	0.094	0.073	0.524	0.028	0.476	0.121	2.250
	colic	0.185	0.178	0.579	0.090	0.421	0.297	3.125
	heartstatlog	0.167	0.176	0.618	0.115	0.382	0.288	4.125
	breast_cancer	0.308	0.290	0.637	0.157	0.363	0.505	3.625
	clean1	0.059	0.048	0.364	0.025	0.636	0.060	2.000
	diabetes	0.237	0.258	0.547	0.150	0.453	0.434	3.375
	parkinsons	0.081	0.087	0.351	0.015	0.649	0.144	2.750

`etree_sgo--9cfsl1--eval_sequence--del_mins 08007735d9`

Table 4.2: The first column *error_plain* is the baseline. The total error of the seco algorithm is given in the second column *error_total*. The following columns show the decomposition of the error into cut error and error on the residual problem. The *fraction*-columns indicate what fraction of the test set was cut and what fraction was not. The last column gives the number of cuts. It is not integer because, as all column, it is averaged over the 8 cv-folds.

For the initial (β, τ) we have independently drawn the entries from the interval $[-8, 8]$. For gaining computation time we passed an increased stopping threshold ϵ to the optimization engine for the greater smoothing window sizes . The engine stops as soon as the l_2-norm of gradient drops below that value.

Table 4.2 shows the test set error of the baseline along with the error of the proposed scheme that further is decomposed into cut error and residual error. The total error of the proposed scheme is sometimes below sometimes above the baseline depending on the data set and the kernel. We cannot take a judgment from these result which training yield better performance. When you compare the baseline result from table 4.3 with those given in table 4.2, you find that they greatly diverge although they supposed to be identical. The explanation is that they were computed on machines with different architecture (32 vs 64 bit) what, for reason we do not understand, has the same affect as using different random seeds and hence different sub-sample selection in cross validation. It is displeasing to see that this make a great difference in performance. We have not further inspected this phenomenon but it lets us take the results with a pinch of salt.

4.5 Run Relative to Cut Error Estimation

The selection criterion by which we chose a cut from the set of candidate cuts takes two factors into account: the estimated false-positive rate the cut makes itself and an estimation of the residual error obtained from cross-validation. For the first factor we use the empirical error-rate on the training set as estimation. Since we not only have the counts from the contingency table but also know functional and geometric margin, we might try to obtain better estimates by taking those into account. For example we can use one-sided version of the error estimate given in [15].

But before we consider adaptions of those to be applied in our training scheme, we wanted to find what total performance gain can we obtain if we found good estimates. To this end we run the seco-experiment again, but cheated in estimating the cut error. We let the selection criterion take the test set cut error instead of its estimator obtained from the training into account. Table 4.3 shows the result. We see that performance gain is rather modest and then dropped the attempts to ameliorate estimation of the cut error.

		error_plain	error_total	fraction_cut	error_cut	fraction_res	error_res	n_cuts
poly_1	sonar	0.255	0.163	0.471	0.048	0.529	0.257	2.875
	ionosphere	0.091	0.095	0.507	0.026	0.493	0.164	2.375
	colic	0.160	0.187	0.704	0.073	0.296	0.443	3.625
	heartstatlog	0.182	0.134	0.697	0.060	0.303	0.312	4.375
	breast_cancer	0.328	0.243	0.707	0.126	0.293	0.548	5.625
	clean1	0.137	0.080	0.656	0.025	0.344	0.191	3.625
	diabetes	0.236	0.231	0.513	0.057	0.487	0.410	6.500
	parkinsons	0.138	0.102	0.517	0.020	0.483	0.192	2.625
poly_2	sonar	0.139	0.138	0.394	0.029	0.606	0.207	2.875
	ionosphere	0.080	0.078	0.464	0.020	0.536	0.128	2.375
	colic	0.174	0.158	0.660	0.090	0.340	0.412	3.375
	heartstatlog	0.160	0.191	0.700	0.097	0.300	0.410	3.750
	breast_cancer	0.276	0.261	0.672	0.098	0.328	0.601	4.875
	clean1	0.061	0.055	0.494	0.013	0.506	0.093	2.375
	diabetes	0.228	0.217	0.419	0.073	0.581	0.368	4.500
	parkinsons	0.112	0.103	0.580	0.016	0.420	0.198	2.875
poly_3	sonar	0.125	0.116	0.322	0.014	0.678	0.179	3.000
	ionosphere	0.094	0.082	0.510	0.031	0.490	0.135	2.625
	colic	0.185	0.164	0.402	0.035	0.598	0.266	2.375
	heartstatlog	0.167	0.165	0.600	0.063	0.400	0.296	4.125
	breast_cancer	0.308	0.264	0.731	0.140	0.269	0.594	5.000
	clean1	0.059	0.046	0.393	0.004	0.607	0.072	2.125
	diabetes	0.237	0.245	0.531	0.113	0.469	0.488	5.000
	parkinsons	0.081	0.072	0.481	0.000	0.519	0.133	2.750

`etree_sgo--10_cheat_ce--eval_sequence--del_mins_p2 61f8ce7248`

Table 4.3: Likewise table 4.2. Here, we cheated by replacing the emprirical cut error of candidate cuts by the corresponding test error. With this relativized run we probed the potential of better cut error estimation. The performance gain of this cheat is rather modest.

4.6 Hard Residual Problems

There is a constant in all 24 data-kernel-combinations: whenever the error of the seco-scheme is greater than that of the baseline, the residual error exceeds the baseline-error by at least 40 per cent. The residual problem is restricted to the uncut area of the input space but otherwise is treaded in the very same manner as the original problem with respect to training and testing. In this work we cannot give a final explanation for this observation but we in the following we want do provide some discussion thereon.

We consider three explanations for the high error rate on the residual problem.

- small training set
- restriction to a part of the input space
- the difficult areas are left to the residual

Obviously, there are less training examples to build a model on the residual problem than for the original problem. However, this explanation does not hold as can be seen from the results of another experiment that are given in table 4.4. In this experiment we created random cuts until we found on that covers at least 2/3 of the training points but lets 32 point uncovered for both classes. We then applied this cut obtaining a smaller residual problem. We then draw a random sub-sample form the original problem with the same number of negatives and positives as the residual problem. The performance of the model training on that small random sub-sample is given in the column *error_sub* in table 4.4. The error rate is much closer to the baseline than the residual error in table 4.2 although the fractions of uncut training points are similar.

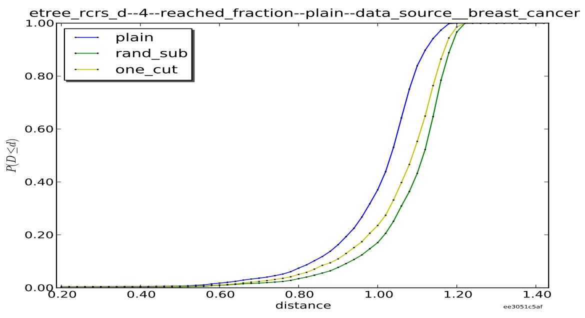
The second explanation is a bit daring. It says that the cut nature of the residual problem make it harder in general. A reason could be that for given number of training examples, extra examples outside the uncut area provide more valuable training information for residual test points than additional example in the uncut area. Unfortunately, this explanation is not only daring but also difficult to check. We can probe the performance with and without additional training point in- and out-side the residual area. But by doing so we would change another parameter that we belief to have strong influence to test error rate, namely the average distance of test point to, say the $k = 10$ nearest training points. We call this measure *example-distance* (at k). In table 4.4 you find the columns `error_res` and `error_sub` holding the error rate of residual model and a model from training on a random subset of the original problem and. The number of training points for the latter equals the number of training points for the residual problem but the example-distance is greater since the training example now are spread over the entire input space. We take this explanation why the majority of random subset models perform worse than the residual models. See figure 4.2 for distributions example distance at $k = \inf$. In any case we take from comparing those two error-rate columns to the residual errors of table 4.2 that the restriction to cut subspace is not an explanation of the great residual errors that the proposed scheme reveals.

Since the first two explanation for the bad residual performance are not sufficient we favor the last, most unpleasant explanation. This basically means that at least in those e cases where the residual error high relative to the baseline the cuts produce hard residual problems instead of alleviating them.

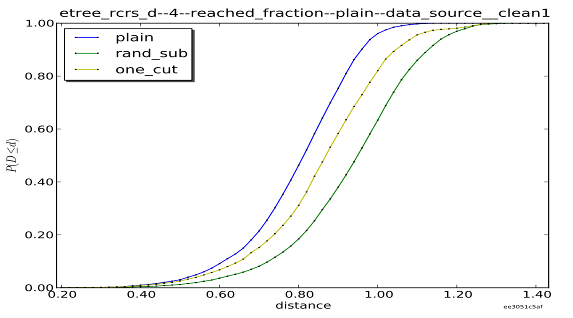
		error_plain	error_res	error_sub	fraction_res
poly_1	sonar	0.255	0.229	0.274	0.548
	ionosphere	0.091	0.108	0.108	0.422
	colic	0.160	0.207	0.201	0.484
	heartstatlog	0.182	0.161	0.204	0.611
	breast_cancer	0.328	0.333	0.329	0.545
	clean1	0.137	0.142	0.181	0.507
	diabetes	0.236	0.292	0.253	0.346
	parkinsons	0.138	0.216	0.240	0.590
poly_2	sonar	0.139	0.159	0.221	0.591
	ionosphere	0.080	0.124	0.117	0.407
	colic	0.174	0.149	0.168	0.451
	heartstatlog	0.160	0.148	0.185	0.574
	breast_cancer	0.276	0.264	0.290	0.510
	clean1	0.061	0.104	0.143	0.424
	diabetes	0.228	0.215	0.267	0.501
	parkinsons	0.112	0.180	0.137	0.560
poly_3	sonar	0.125	0.132	0.154	0.548
	ionosphere	0.094	0.103	0.074	0.490
	colic	0.185	0.151	0.174	0.438
	heartstatlog	0.167	0.142	0.178	0.504
	breast_cancer	0.308	0.295	0.287	0.509
	clean1	0.059	0.090	0.122	0.506
	diabetes	0.237	0.301	0.302	0.303
	parkinsons	0.081	0.120	0.112	0.584

`etree_rcrs--11--eval_rcrs--main 7831d8e5a4`

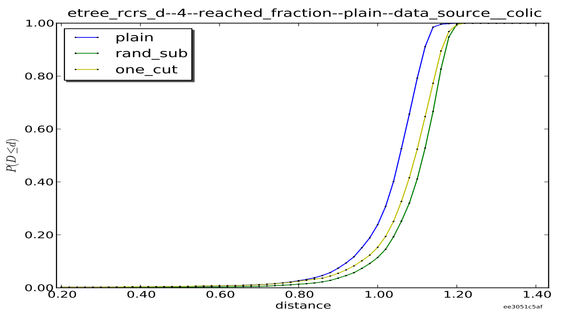
Table 4.4: Comparing the performance of residual problem of a random cut with the an equal-sized random sub-sample of the original problem. *error_plain* is the base line where the entire training set is used. *error_res* is the error of model that was trained on examples that lie in half space defined by a random plane (fulfilling certain constrains). *error_sub* refers to training with same number of example randomly drawn from the original training set and *fraction_res* is the fraction of examples used for the latter two measures.



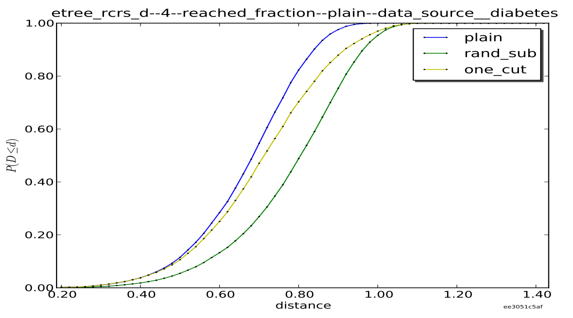
(a)



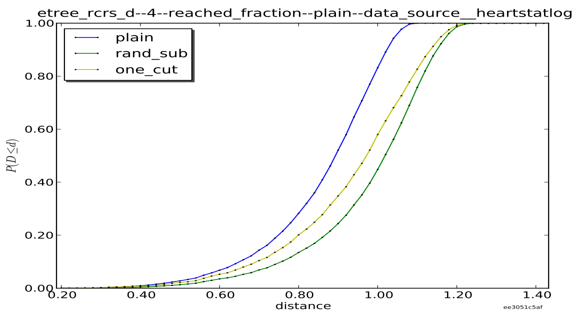
(b)



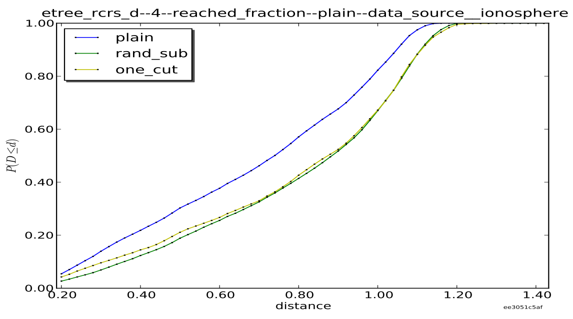
(c)



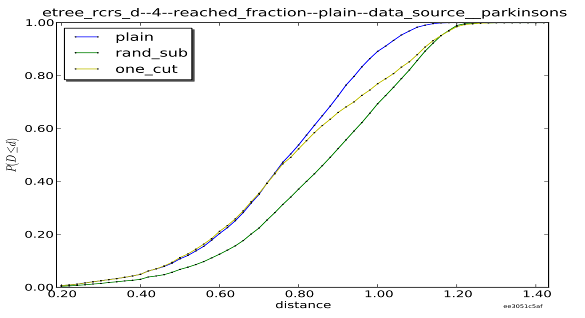
(d)



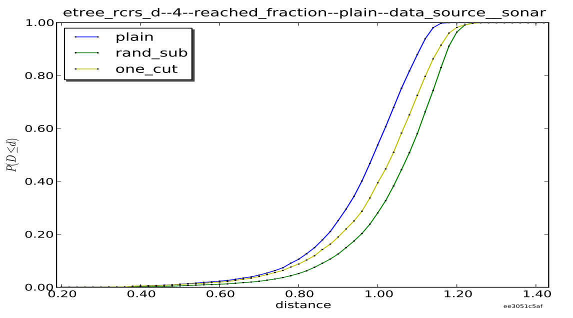
(e)



(f)



(g)



(h)

Figure 4.2: The plots show distributions of distances between test points and training points for different data sets under cubic outer kernel and averaged over the test points and samples. The blue lines represent the original problems, the yellow lines the residual problems obtained from a random cut, and the green lines a random subsample of the original problem with the same size as the residual problem. The broadminded reader may consider the following interpretation: for lower distance the distance-distribution on the residual is more similar to that of the original problem as the the cut is unlikely to remove small distances. For greater distances on the other hand the restriction of measuring in one half space only loses influence if the number test and training points is fixed.

5 Conclusion

5.1 Motivations

In this work we tread a training scheme that builds a list of partial models that refer to the same kernel but cover distinct parts of the input space. We see two motivations for such ensembles:

- *non-expressive kernel*: The used kernel might be *discriminative* for parts of the input space but not on the entire space. The reason to chose such non-expressive kernel can either be a particular input spaces $\mathcal{X}$ for which definition of suited base kernel is not evident, or, the observation that kernels with greater capacity do not increase performance.

- *heterogeneous inertia*: Different parts of the input space may refer to *subproblems* of different difficulty in the sense that different model margins can be obtained for given number of examples and given error. In such a case the parameter that controls the trade-off between regularity and empirical error of the model should be set for those subproblems individually. Otherwise the parts have to figure out a harmful compromise in setting the trade-off parameter.

These two motivations remain a bit vague by the end of this project. We do not provide examples of inputs spaces for which we are bound to non-expressive kernels. And we do not define what we mean by *subproblem*, the clarity of this notion is somehow bounded by the clarity of the notion *cluster*. The motivation heterogeneous inertia assumes that this notion of subproblems makes sense and that such subproblems exist in real data. One can look at the proposed cutting scheme can method for supervised clustering but we believe that there are better means to detect subproblem closer to classical clustering algorithms. If we further investigate the potential problem of single trade-off-parameters, we approach it from a clustering perspective.

5.2 Algorithm and Code

We designed and implemented a margin-driven, kernel-based separate and conquer algorithm. The algorithm has a subroutine *candidate supply* that shortlists the set of candidate cuts from uncountable infinite to a view hundred. Each candidate is an optimum of an SVM-like but asymmetric objective that puts weak penalty on false-negatives as those can be corrected by subsequent models. The non-monotone objective is minimized by an heuristics taken from [5].

Given those candidates cut another subroutine selects one from the candidate set according to a selection criterion that tries to take potential alleviation on the residual problem into account by evaluation the hardness of the residual as minimal cross-validation over different trade-off parameters.

The result show that odds are not bad that you can decrease the test error by applying the proposed scheme if you bound to some given kernel. When the scheme uses fixed kernel is does not, however, outperform standard SVM-training with joint trade-off-parameter and kernel tuning.

At the beginning of this project we did not know what would be suitable design for the pair of loss-functions. For this reason we build a tool that allows the heuristic optimization with generic patch-work losses. This means that we can freely define custom loss functions that divide the domain into an arbitrary number of segment on each of which the loss can take the linear transform of the shape some base function take on some segment that again can be freely chosen. We plan to clean up the code of this tool and make in on-line available, please ask for the code via email up to then, or take out the code from the project export that is available available by now [1]. SVM-like training with custom loss functions can be used for instance to study variation of symmetric SVM-training, semi-supervised training as done in [5], or multiclass training with class-dependent losses.

5.3 Important Deficiencies

The analysis of the studied training becomes rather complex by the fact that scheme has different compounds each of which comes with proper deficiencies. It is not clear which of them or what combination of them is most harmful. In the following we list some important problems.

Selection Criterion

The applied selection criterion is an approximation of an target-criterion. The latter is the one we aim to apply but since it is unaccessible we fall back to an approximation. The approximation error might be an issue or not, but in any case the target criterion is questionable itself. It assumes that the selected cut will be the last one. A more convincing variant would be to estimate the residual error by cross validation based very seco scheme itself, what we considered as unfeasible.

Coverage Driven Objective

The loss on the target class basically is the fraction of uncovered points. It has the role of an opponent to the Hinge-loss on the default class. If we applied zero loss on the target class we would obtain no-coverage cuts only. We do not consider great coverage as value by itself. Still, if we assume fixed given cut error and a fixed residual error greater the former, a greater coverage means lower total error. But an incentive on coverage can also be problematic. Considers a simple situation where the examples are all inside three spheres, one positive, two negatives, and the three touching each other or have little overlap. In this situation it would be smarter to first separate one negative sphere from positive one and then the remaining one. The proposed coverage driven objective, however, would find cuts that take large parts of both negatives spheres at once.

Optimization Heuristics

The heuristics that we use to find good local minima of the candidate supply objective can easily fail. This happens if the *valley* around the corresponding local minimum is rather small such that the minimum has little weight in smoothing and is average out. Another issue of the heuristics is that by using random initialization we prefer minima close to the origin because on average the origin is closest to a random vector. Intuitively however, the local minima that represent cuts in the outer regions of the example set are the ones we should prefer.

5.4 Acknowledgement

We thank Sang-Hyeun Park for helpful discussions.

Bibliography

[1] Peewit-project-export for margin driven separate and conquer training. `http://www.ke.tu-darmstadt.de/resources/peewit/projects/project_export_mdsc_sgo/at_download/file`, 2011. 16, 22

[2] A. Asuncion and D. Newman. UCI machine learning repository, 2007. 17

[3] K. Bennet and J. Blue. A support vector machine approach to decision trees. In *IEEE International Conference on Neural Networks (IJCNN'98)*, volume III, pages III–2396–III–2401, Anchorage, AK, July 1998. IEEE. RPI. 3

[4] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`. 17

[5] O. Chapelle, M. Chi, and A. Zien. A continuation method for semi-supervised SVMs. In W. W. Cohen and A. Moore, editors, *ICML*, volume 148 of *ACM International Conference Proceeding Series*, pages 185–192. ACM, 2006. 12, 22

[6] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1/3):131, 2002. 3

[7] K. E. Group. peewit 0.6. `http://www.ke.tu-darmstadt.de/resources/peewit`, 2010. 16

[8] R. Jin and H. Liu. SWITCH: A novel approach to ensemble learning for heterogeneous data. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *ECML*, volume 3201 of *Lecture Notes in Computer Science*, pages 560–562. Springer, 2004. 3

[9] V. Kecman and J. P. Brooks. Locally linear support vector machines and other local models. In *IJCNN*, pages 1–6, 2010. 3

[10] G. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions and stochastic processes. *J. Math. Anal. Appl.*, 33:82–95, 1971. 7

[11] R. S. Michalski. Theory of plausible reasoning - foundations and methodology. In G. Epstein, editor, *Proceedings of the 20th International Symposium on Multiple-Valued Logic*, pages 325–327, Charlotte, NC, May 1990. IEEE Computer Society Press. 3

[12] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990. 3

[13] S. Pang, I. Havukkala, and N. Kasabov. Two-class SVM trees (2-SVMT) for biomarker data analysis. In J. Wang, Z. Yi, J. M. Zurada, B.-L. Lu, and H. Yin, editors, *Advances in Neural Networks - ISNN 2006, Third International Symposium on Neural Networks, Chengdu, China, May 28 - June 1, 2006, Proceedings, Part III*, volume 3973 of *Lecture Notes in Computer Science*, pages 629–634. Springer, 2006. 3

[14] B. Schoelkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, MA, 2002. 6, 7, 8

[15] J. Shawe-Taylor and N. Cristianini. On the generalization of soft margin algorithms. *IEEE Transactions on Information Theory*, 48(10):2721–2735, 2002. 18

[16] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. The MIT Press, Cambridge, Massachusetts, 2 edition, 2000. 12