UNIVERSITAT POLITÈCNICA DE CATALUNYA

FACULTAT D'INFORMÀTICA DE BARCELONA

Extension Report: Kernels

Analysis of the SVM-RFE algorithm for feature selection

Author:
Robert PLANAS

Director:
Luis A. BELANCHE

Bachelor Degree in Informatics Engineering
Specialization: Computing

May 25, 2021

Contents

Chapter 1

Non-linear Kernels

This modification intends to apply the required modifications in the calculation of the ranking criteria so that non-linear kernels can be used in the SVM.

1.1 Description and reasoning

When a problem is not linearly separable, we know that a hard-margin SVM will not be able to correctly place a decision boundary. In this case a soft-margin SVM may be used, but it only works to some extent and if the underlying distribution is near linearly separable. If it is not the case, much better results can be achieved by using non-linear kernels.

To use this method within SVM-RFE we must first be able to compute the ranking coefficient from a non-linear kernel. In contrast with the linear kernel case, where the ranking coefficient can be simplified to $(w_i)^2$, for non-linear kernels, however, since it is a more general case, no simplification can be performed. Instead, we use the general ranking coefficient for SVM (Equation 1), which we restate here:

$$DJ(i) = (1/2)(\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}\boldsymbol{\alpha} - \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}(-i)\boldsymbol{\alpha})$$

Note that the *hessian* matrix $\mathbf{H_{i,j}} = y_i y_j k(\mathbf{x_i}, \mathbf{x_j})$ needs be computed each iteration (since the dimension of $\mathbf{x_i}$ and $\mathbf{x_j}$ will change), and also for each feature removed in each iteration. This is slow. However, various optimizations may exist, as discussed in Section [ref.].

1.2 Pseudocode formalization

Definitions:

- $X_0 = [\vec{x_0}, \vec{x_1}, \ldots, \vec{x_k}]^T$ list of observations.
- $\vec{y} = [y_1, y_2, \ldots, y_k]^T$ list of labels.

Algorithm 1: SVM-RFE with general Kernel

Input: t, k // t = step, k = kernel function
Output: $\vec{r}$
Data: $X_0, \vec{y}$
$\vec{s} = [1, 2, \ldots, n]$ // subset of surviving features
$\vec{r} = []$ // feature ranked list
while $|\vec{s}| > 0$ **do**
  /* Restrict training examples to good feature indices */
  $X = X_0(:, \vec{s})$
  /* Precompute hessian matrix */
  $\mathbf{H_{i,j}} = y_i y_j k(\mathbf{x_i}, \mathbf{x_j})$ for all $\mathbf{x_i}, \mathbf{x_j} \in X$
  /* Train the classifier */
  $\vec{\alpha} =$ SVM-train(X, y, k)
  /* Compute the ranking criteria */
  $\vec{c} = [c_1, c_2, \ldots, c_{|\vec{s}|}]$
  for $c_l \in \vec{c}$ **do**
    /* Compute new hessian with the feature l removed */
    $\mathbf{H_{i,j}}(-l) = y_i y_j k(\mathbf{x_i}, \mathbf{x_j})$ for all $\mathbf{x_i}, \mathbf{x_j} \in X(-l)$
    /* Calculate ranking coefficient */
    $c_l = (1/2)(\boldsymbol{\alpha}^{\mathrm{T}} \mathbf{H} \boldsymbol{\alpha} - \boldsymbol{\alpha}^{\mathrm{T}} \mathbf{H}(-l) \boldsymbol{\alpha})$
  end
  /* Find the t features with the smallest ranking criterion */
  $\vec{f} =$ argsort$(\vec{c})(:t)$
  /* Iterate over the feature subset */
  for $f_i \in \vec{f}$ **do**
    /* Update the feature ranking list */
    $\vec{r} = [\vec{s}(f_i), ...\vec{r}]$
    /* Eliminate the feature selected */
    $\vec{s} = [...\vec{s}(1 : f_i - 1), ...\vec{s}(f_i + 1 : |\vec{s}|)]$
  end
end

1.3 Results

Notes on the implementation

Using non-linear kernels requires a change in the underlying implementation we've been using for SVM. Until now, we've been relying on the `LinearSVC` model provided by `Sklearn`, this implementation is in turn based on the `LIBLINEAR` implementation for SVM written by the *National Taiwan University*. The authors state that this solver is much faster than the more general version, thus the reason we've been using it, but it can only handle linear kernels. The general version, `LIBSVM`, created by the same team, is the option we're going to use instead. Because now we need to switch, it is interesting to see what will be the increase in computational cost, shown in the table 1.1.

For this first test we've generated artificial datasets with 100 features each, 20 which are informative. Because we want to test the differences with the linear kernel we've used 6 Gaussian clusters per class, this makes the problem more difficult for lineal separators.

To obtain similar results to what we would get with SVM-RFE, we've tested under the random feature selection model (which is the same as RFE with random criteria) and made a comparison using different amount of samples.

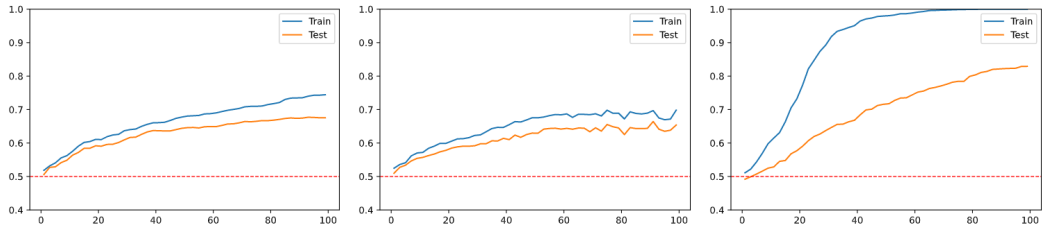
Obs.	`LIBLINEAR`	`LIBSVM` *Linear*	`LIBSVM` *Precomputed*
500	02.67s	08.17s — x0.32	12.04s — x0.22
1000	02.94s	17.30s — x0.17	23.84s — x0.12
2000	04.28s	43.96s — x0.10	50.36s — x0.08

TABLE 1.1: Cost in time and speedup of a random selection under different implementations and sample sizes.

Note that the sample size increases the cost more than linearly, this may suggest that methods such as dynamic sampling can perform even better than with the `LIBLINEAR` implementation.

We also note that using a precomputed kernel matrix (*Gramm* matrix) does affect computational cost. This is likely an implementation detail and it is not significant enough to require further investigation.

We can also plot some examples to compare differences in accuracy, in Figure 1.1 note that the precomputed kernel, which is a degree-3 polynomial kernel, performs much better at the start, thus we can expect it to also perform significantly better with SVM-RFE. At the same time, we can appreciate some differences between the two linear kernels, but they're still clearly following the same curve, which indicates that both implementations are equivalent.

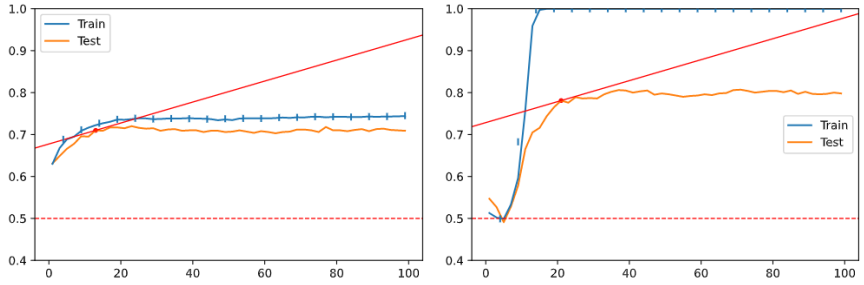


`LIBLINEAR` `LIBSVM` *Linear* `LIBSVM` *Precomputed*

FIGURE 1.1: Comparison of random selection with 2000 observations.

Performance evaluation

We observe experimentally in figure 1.2 that, for some datasets, using a non-lineal kernel improves the accuracy of SVM-RFE.



Linear Poly $D = 3$

FIGURE 1.2: Comparison of SVM-RFE on linear and non-linear kernel with 1000 observations and step of 5.

Non-lineal kernels also involve more hyper-parameters, thus increasing the importance of model selection. The choice of kernel is in itself a hyper-parameter. If a polynomial kernel is chosen then the *degree* and *coefficient* are also hyper-parameters, if a Gaussian kernel is chosen then *gamma*.

MADELON

For this experiment we use the MADELON dataset with a sample of 1200 observations after normalization. We're using a trade-off linear scalarization of 0.8 to determine the optimum, and a constant step of 20 and 10 for the RFE and validation phases respectively. Also, for numerical stability, all results are an average of a 6-fold cross-validation.

Polynomial

deg.	**1**			**2**			**3**		
C	Feat.	Acc.	Cost	Feat.	Acc.	Cost	Feat.	Acc.	Cost
0.1	11	58.92%	0.333	21	69.08%	0.256	21	74.25%	0.214
0.2	1	61.33%	0.310	21	69.17%	0.255	21	76.58%	0.196
0.3	1	61.42%	0.309	21	70.17%	0.247	21	77.08%	0.191
0.4	1	61.33%	0.310	21	70.92%	0.241	21	77.42%	0.189
0.5	11	61.42%	0.313	21	70.83%	0.242	21	78.33%	0.182
0.6	1	61.25%	0.310	21	71.00%	0.240	21	78.00%	0.184
0.7	1	61.08%	0.312	21	70.58%	0.244	21	77.50%	0.188
0.8	1	61.25%	0.310	21	70.00%	0.248	21	76.92%	0.193
0.9	1	61.08%	0.312	21	70.00%	0.248	21	78.42%	0.181
deg.	**4**			**5**			**6**		
C	Feat.	Acc.	Cost	Feat.	Acc.	Cost	Feat.	Acc.	Cost
0.1	21	72.83%	0.226	21	70.58%	0.244	21	67.17%	0.271
0.2	21	80.92%	0.161	21	78.08%	0.184	21	77.75%	0.186
0.3	21	81.92%	0.153	21	83.92%	0.137	21	81.67%	0.155
0.4	21	82.75%	0.146	21	84.75%	0.130	21	83.58%	0.140
0.5	21	83.08%	0.144	21	84.75%	0.130	21	85.84%	0.122
0.6	21	84.00%	0.136	21	85.75%	0.122	21	84.16%	0.135
0.7	21	82.92%	0.145	21	85.50%	0.124	21	84.08%	0.132
0.8	21	83.75%	0.138	21	85.42%	0.125	21	86.16%	0.115
0.9	21	84.33%	0.138	21	86.83%	0.114	21	84.08%	0.132
deg.	**7**			**8**			**9**		
C	Feat.	Acc.	Cost	Feat.	Acc.	Cost	Feat.	Acc.	Cost
0.1	11	66.33%	0.274	11	64.92%	0.285	11	63.25%	0.298
0.2	11	69.17%	0.251	11	57.33%	0.345	11	56.67%	0.351
0.3	11	70.75%	0.238	11	55.17%	0.363	11	51.50%	0.392
0.4	11	83.00%	0.140	11	58.17%	0.339	21	53.50%	0.380
0.5	21	82.25%	0.150	11	80.25%	0.162	11	58.75%	0.334
0.6	11	84.42%	0.129	11	85.33%	0.122	11	79.83%	0.166
0.7	11	85.58%	0.120	11	80.41%	0.161	21	80.10%	0.160
0.8	11	84.08%	0.132	11	82.50%	0.144	11	82.75%	0.142
0.9	11	83.42%	0.137	11	85.75%	0.118	11	82.00%	0.148

TABLE 1.2: Perfomrance.