CSCI 567 Homework 2

Furong (Flora) Jia & Qilin Ye

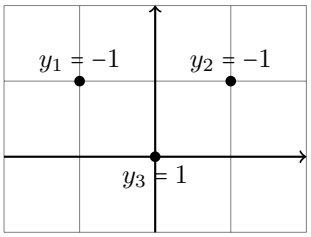
September 27, 2022

Problems available at `https://vatsalsharan.github.io/fall22/hw2.pdf`

1. SVMs

(1.1) No. A one-dimensional classifier partitions $\mathbb{R}$ into two connected subsets and a singleton containing the 0-dimensional hyperplane corresponding to the threshold. Since 0 is in the segment formed by -1 and 1, no such classifier exists.

(1.2) Yes, obviously. For example $y = 0.5$ works (and we will show this is the best one). For the plot below, each grid is 1 unit.



(1.3) We first note $\varphi(x_1) = (-1, 1)^T$, $\varphi(x_2) = (1, 1)^T$, and $\varphi(x_3) = (0, 0)$. The Gram matrix K is therefore

$$\{\varphi(x_i)^T \varphi(x_j)\}_{i,j} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

(1.4) The primal formulation is

$$\min_{w,b} \frac{\|w\|_2^2}{2} \qquad \text{subject to} \qquad \min_i y_i(w^T \varphi(x_i) + b) \geqslant 1$$

and the dual formulation is

$$\max_{\alpha_i, 1 \leqslant i \leqslant 3} \left[\sum \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \varphi(x_i)^T \varphi(x_j) \right] \qquad \text{subject to} \qquad \sum_i \alpha_i y_i = 0 \text{ and } \alpha_i \geqslant 0.$$

(1.5) In this very special case, for all (i, j) except $(1, 1)$ and $(2, 2)$, the terms in $\sum_{i,j}$ vanish. The dual then becomes

$$\max_{\alpha_i, 1 \leqslant i \leqslant 3} \alpha_1 + \alpha_2 + \alpha_3 - \alpha_1^2 - \alpha_2^2 \qquad \text{subject to } \alpha_1 + \alpha_2 = \alpha_3 \text{ and } \alpha_i \geqslant 0.$$

Completing the squares and using $\alpha_1 + \alpha_2 = \alpha_3$, we have

$$\alpha_1 + \alpha_2 + \alpha_3 - \alpha_1^2 - \alpha_2^2 = -\alpha_1^2 + 2\alpha_1 - \alpha_2^2 + 2\alpha_2 = -(\alpha_1 - 1)^2 - (\alpha_2 - 2)^2 + 2 \leqslant 2,$$

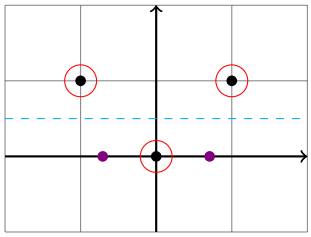
and it is obvious that $(\alpha_1^*, \alpha_2^*, \alpha_3^*) = (1, 1, 2)$ achieves this maximum. Using $w = \sum_i \alpha_i y_i \varphi(x_i)$, we have

$$w^* = -\begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 0 = \begin{bmatrix} 0 \\ -2 \end{bmatrix}.$$

The optimal b^* is given by (here we fix y_1)

$$b^* = y_1 - \sum_j \alpha_j y_j k(y_j, y_1) = y_1 - \alpha_1 y_1 k(x_1, x_1) = -1 + 2 = 1.$$

(1.6) In $\mathbb{R}^2$, the decision boundary is given by $\{(a, b) : -2b + 1 = 0\}$, namely the horizontal line $y = 0.5$, marked as the blue dashed line. All three vectors are support vectors (circled in red)! Here they are all equidistant to the decision boundary (or alternatively $y_i(w^{*T}\varphi(x_i) + b^*) = 1$ for all three points). In the one-dimensional space, the corresponding decision boundary is $\{x : -2x^2 + 1 = 0\}$, i.e., $\{\pm 1/\sqrt{2}\}$, represented by the purple dots on the x-axis.



2. Kernel Composition

Proof. Let $k_1, k_2 : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be kernels and define $k(x, y) := k_1(x, y)k_2(x, y)$. Let $\varphi_1 : \mathbb{R}^d \to \mathbb{R}^m$ and $\varphi_2 : \mathbb{R}^d \to \mathbb{R}^n$ be the feature maps corresponding to k_1 and k_2. That is,

$$k_i(x, y) = \varphi_i(x)^T \varphi_i(y) \qquad \text{for } i = 1, 2.$$

Notation-wise, let $\varphi_i^{(j)}(x)$ denote the j^{th} component of $\varphi_i(x)$. Some computation yields

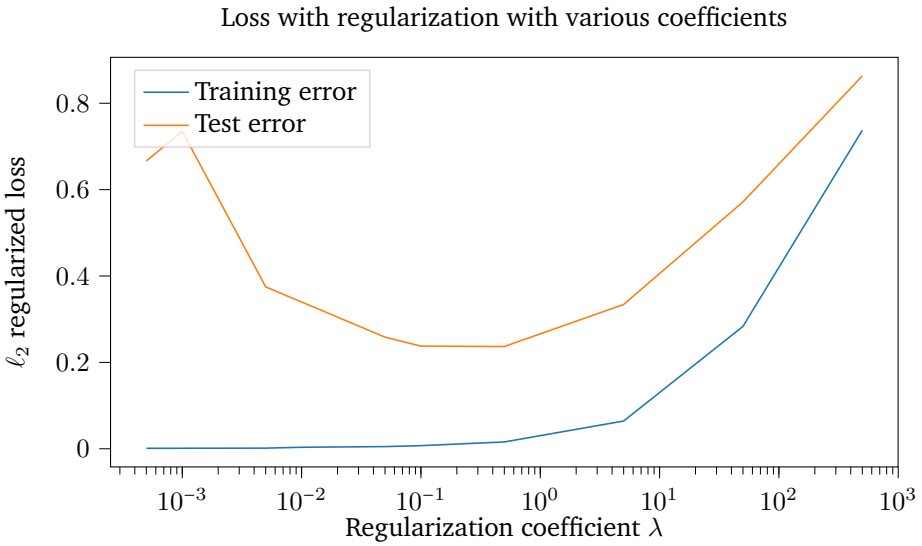
$$\begin{aligned} k(x, y) = k_1(x, y)k_2(x, y) &= [\varphi_1(x)^T \varphi_1(y)][\varphi_2(x)^T \varphi_2(y)] \\ &= \left(\sum_{i=1}^m \varphi_1^{(i)}(x)\varphi_1^{(i)}(y)\right)\left(\sum_{j=1}^n \varphi_2^{(j)}(x)\varphi_2^{(j)}(y)\right) \\ &= \sum_{i=1}^m \sum_{j=1}^n \varphi_1^{(i)}(x)\varphi_1^{(i)}(y)\varphi_2^{(j)}(x)\varphi_2^{(j)}(y) \\ &= \sum_{i=1}^m \sum_{j=1}^n [\varphi_1^{(i)}(x)\varphi_2^{(j)}(x)][\varphi_1^{(i)}(y)\varphi_2^{(j)}(y)] \\ &= \sum_{\substack{(i,j) \\ i \leqslant m \\ j \leqslant n}} \psi^{(i,j)}(x)\psi^{(i,j)}(y) = \psi(x)^T \psi(y) \end{aligned}$$

where $\psi : \mathbb{R}^d \to \mathbb{R}^{m \times n}$ with components $\psi^{(i,j)}(x) = \varphi_1^{(j)}(x)\varphi_2^{(j)}(x)$. Therefore $k = k_1 k_2$ is indeed induced by a feature map and is itself a kernel. More specifically, if K_1, K_2 are the kernel matrices for k_1 and k_2, then $K = K_1 \otimes K_2$ is the one for k. □

3. Regularization

(3.1) One particular run yields an average training error of $\approx 4.22 \cdot 10^{-12}$ and average test error of 0.735.

(3.2) The results are plotted below. The notable pattern is that training error increases as regularization coefficient increases, whereas test error first decreases before going back up again. There is no significant difference between these and the results in (3.1) in terms of magnitude. The training errors are all higher than that in (3.1) due to the extra $\lambda\|w\|_2^2$. For test errors, however, with appropriate choice of λ, we indeed achieve lower test error.



(3.3) Below are the results obtained from one run:

```
Results w.r.t. eta = 0.00005, 0.0005, and 0.05, with average over 10 runs each, and 10^6 iterations on SDG:
 ---------------- Results on training set ----------------
SGD training errors [0.01583857 0.00743117 0.00325771]
LS training errors  [8.33285852e-11 7.00683992e-12 2.65708366e-11]
Ridge training errors [0.00081002 0.00131496 0.00228058]
True training errors [0.05336222 0.05248965 0.05233532]
 ---------------- Results on test set ----------------
SGD test errors  [0.27368233 0.26432111 0.43295255]
LS test errors  [3.40637986 1.59478261 1.23070795]
Ridge test errors [0.7445873 0.8716912 0.43272321]
True test errors [0.05169631 0.05071892 0.05121475]
```

In these runs, we see that while SGD does not do as good as its two least squares counterparts on the training set, it performs consistently on the test set. This is because the training and test sets can be thought of as i.i.d. samples from the noisy distirbution, and our SGD model is trained to cope on it, whereas the two least squares methods, while minimizing the training error, took the noise into account, and this extra action did lowered their performance on the test set, where the noise was completely different. This is especially true on the unregularized least squares: it minimized the training error, but at what cost? — huge test errors.

In comparison, while SGD does better on the training data than the true model, it has a higher test error. The reason is identical to above — our true model ignores the noises completely, so the training error is higher,

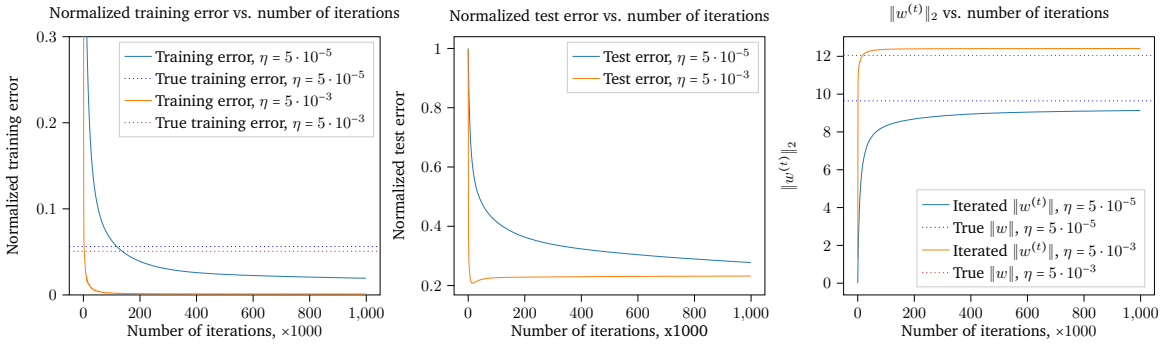
but moving on to the training set,this disadvantage is turned around, as the true model, unaffected by the training noises, naturally does a better job on the brand new test set.

(3.4)

- Normalized errors: since we are iterating SGD 1 million times, even a small step size like $5 \cdot 10^{-5}$ gets enough iterations to sufficiently minimize its objective function. We see that both $\eta = 5 \cdot 10^{-3}$ and $5 \cdot 10^{-5}$ have achieved significantly lower training errors. In particular, with $\eta = 5 \cdot 10^{-3}$, the final training error is only 2% of the true training error (computed using w_{true}).

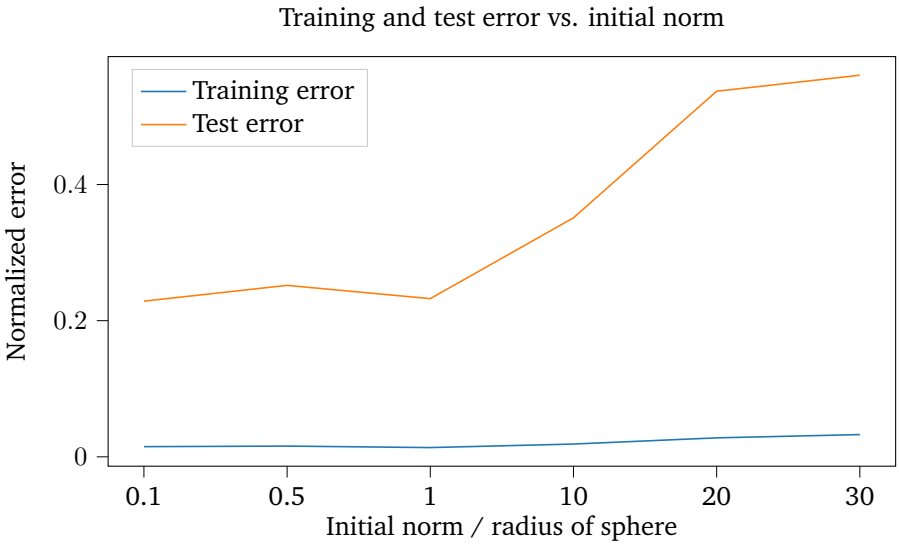
- The test error of $\eta = 5 \cdot 10^{-5}$ over iterations is behaving relatively normally (decreasing over time), but the test error of $\eta = 5 \cdot 10^{-3}$ bounces back after a few dozen thousand iterations. Connecting to the previous part, this is most likely because η is small but not too small so that it starts to overfit the training set by fitting the noises. While doing so guarantees small training error, it may be counterproductive in reducing the test error.

- The vectors $w^{(t)}$ outputted by SDG roughly converges to something close to w (but not w due to the noises).



(3.5) The general trend is that both training and test errors increase as our starting point is drawn from a sphere with larger radius. This does share *some* similarity with the ℓ_2 regularization model, if we consider our initial starting point radius r as a regularization coefficient, i.e., SGD "punishes" starting points that are far away from the origin, in our case. However, the trends are not identical — here the test error barely decreases for increasing but small r.

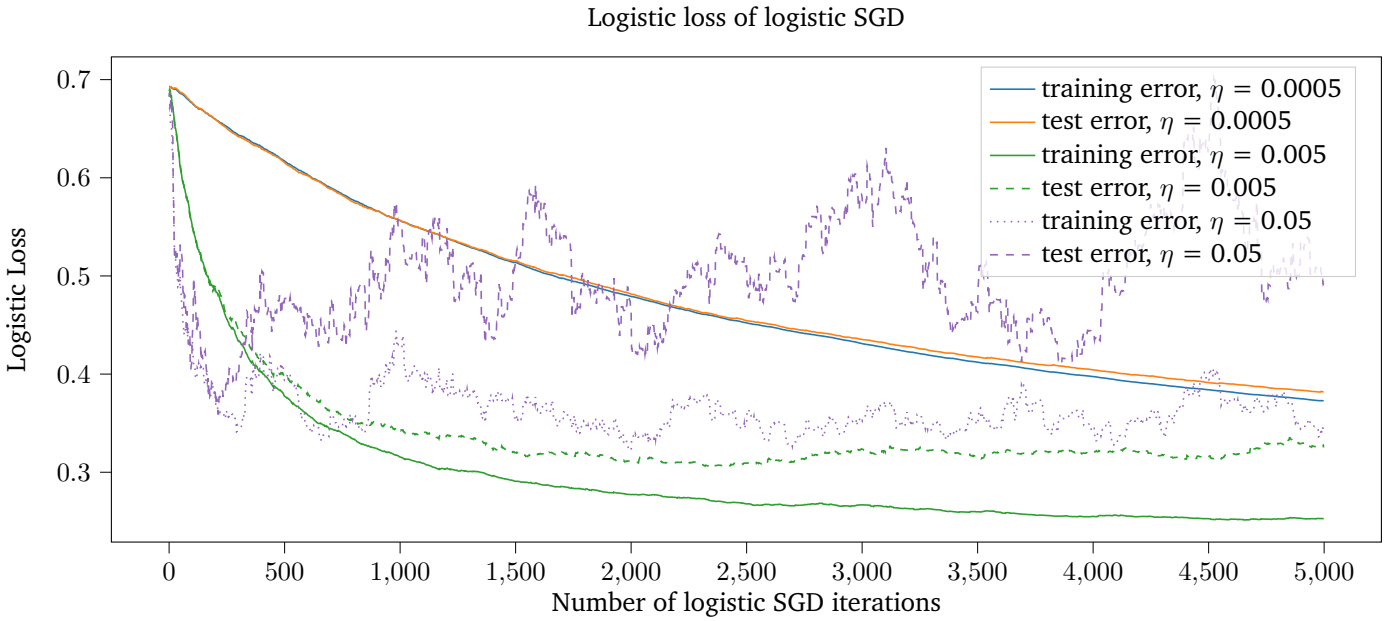
To see why the errors increase with larger r, we increased the magnitude of iterations (to 10^7) on $r = 30$ and found the errors decreasing dramatically as well. Hence, we conclude that, in this case, an SDG with a larger r is not unable to minimize the objective function; instead, it simply requires a larger amount of iteration to achieve the same result we easily get with an appropriate small r.



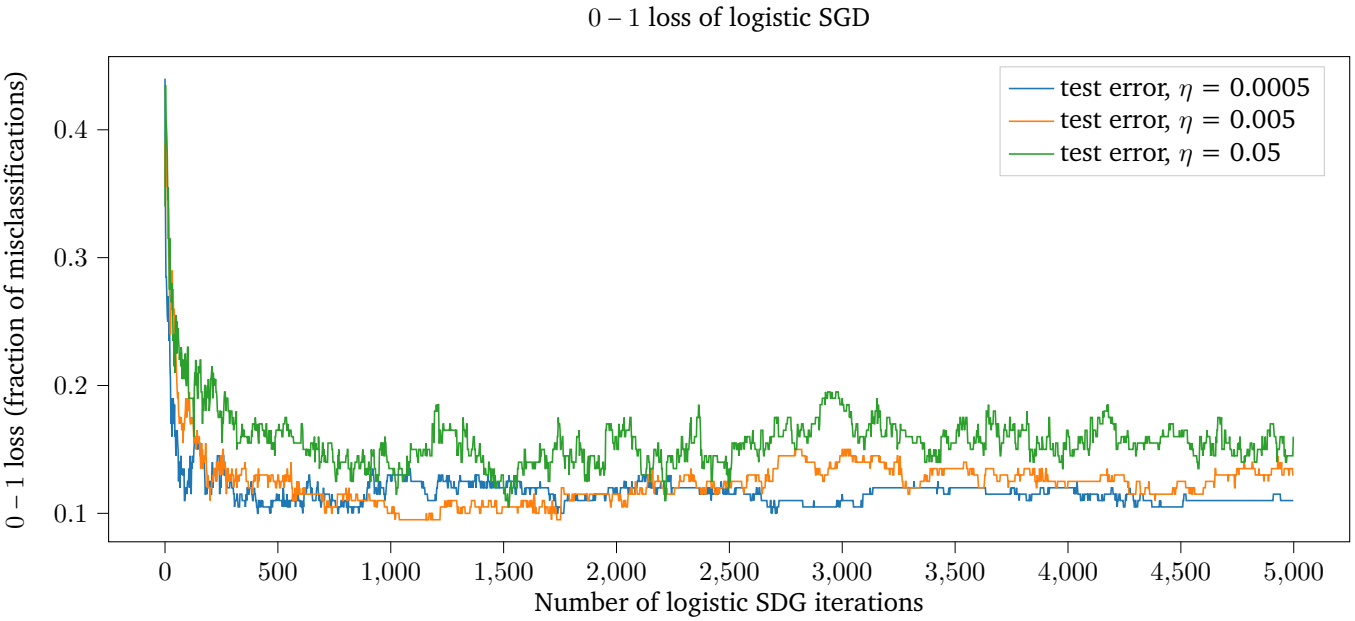
4. Logistic Regression

(4.1)

- $\eta = 0.005$ is the best here — we all love it. It has nicely decreasing training and test errors. (It seems like $\eta = 0.0025$ would be even better; here, with $\eta = 0.005$, the performance on the test set seems slightly random. Sometimes it bounces back, signaling that η is slightly too big.)
- $\eta = 0.0005$ behaves nicely, but unfortunately it is too small and 5000 iterations is far from enough for it to sufficiently approximate objective's minimizer.
- $\eta = 0.05$ is way too big — both training and test errors behave randomly, implying that we constantly overshoots the sweet spot when updating by η times our gradient.



(4.2) In this particular run, the average mean $0-1$ loss of iterations 4700 to 5000 are ≈ 0.11057 for $\eta = 0.0005$, ≈ 0.13283 for $\eta = 0.005$, and 0.151 for $\eta = 0.05$. The clear winner here is $\eta = 0.0005$.

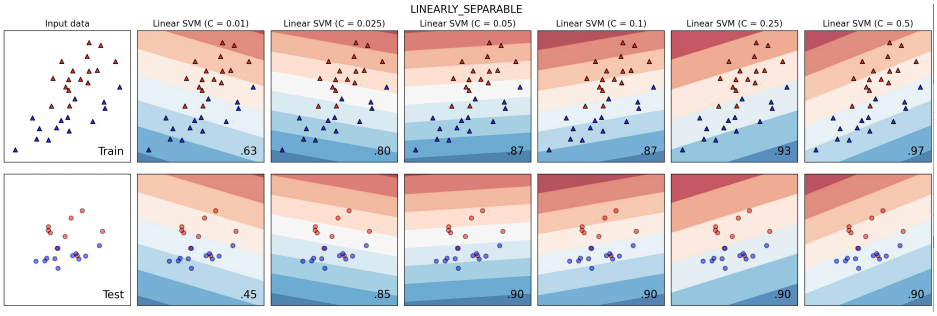


(4.3) Overall, the logistic loss is a successful surrogate for the $0-1$ loss. The reason why $\eta = 0.0005$ in (4.1) is outperformed by $\eta = 0.005$ is because we only iterated 5000 times, a number far from enough for a small learning rate. If we were able to increase the magnitude like in the previous problem, the results will agree (checked) with what is depicted in (4.2).

5. Classifier Comparison

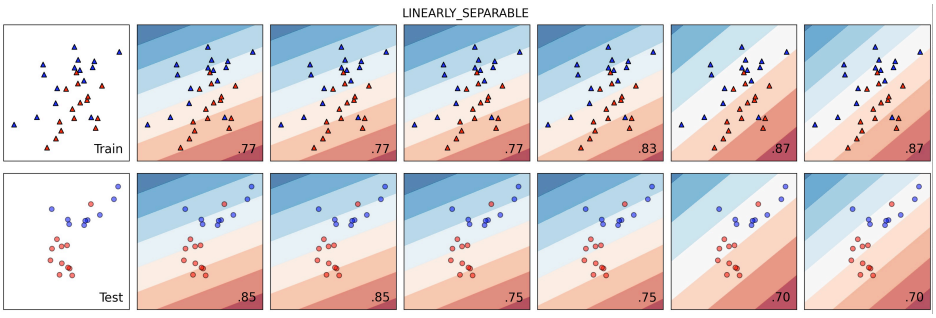
(5.1) Linear classifiers don't do well on MOON and CIRCLES because they are not linearly separable. In particular, linear classifiers fail miserable on CIRCLES because of its circular distribution. SVM with KBF kernel performs much better because it can generate nonlinear decision boundaries that predict labels with high accuracy.

(5.2) Linear SVM: most of the time, as C increases, both training and test set accuracy increase, but there does not seem to be a fixed pattern on which set has a lower error. Also, the orientation of the decision boundaries tends to rotate counterclockwise as C increases.



Training and test errors both decrease as C increases.

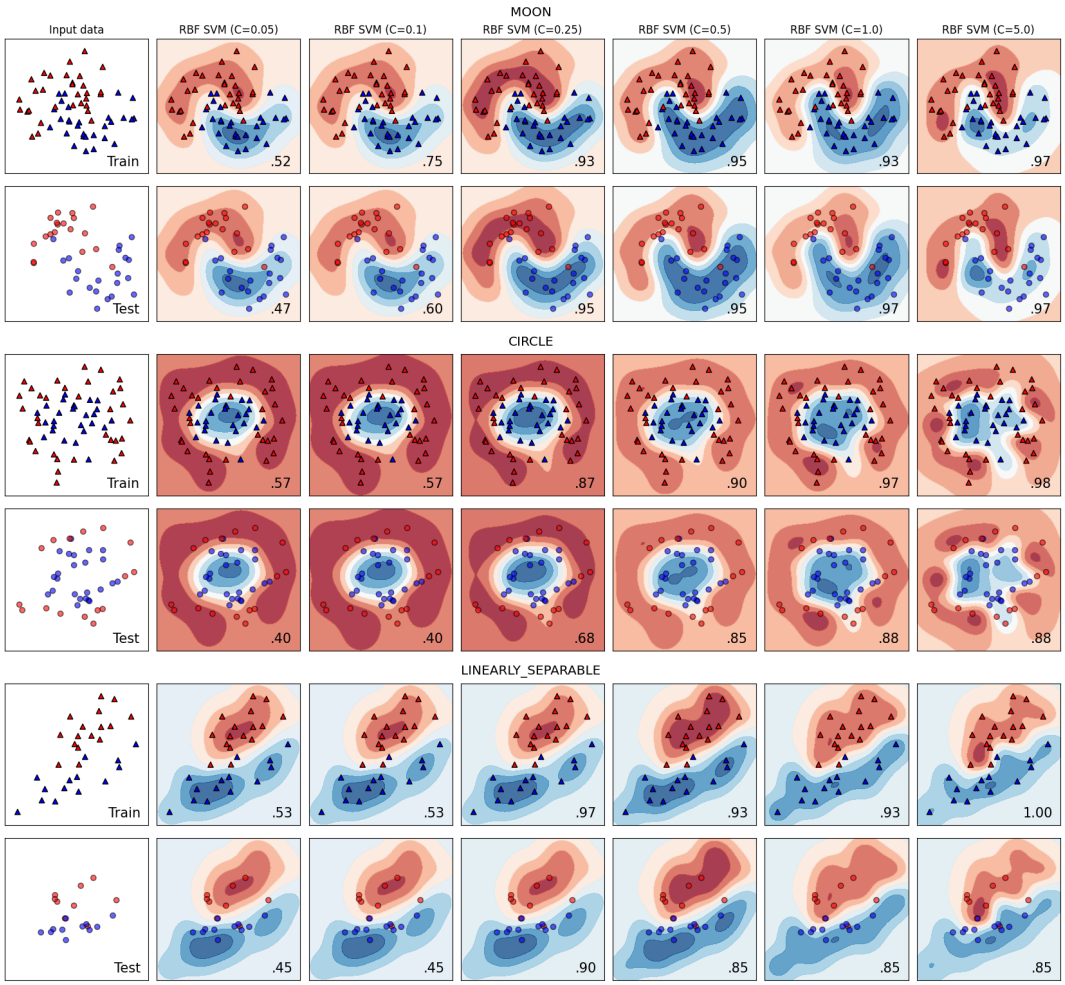
However there does appear to be counterexamples, where the test error significantly increases. The following is obtained by using `n_samples=50, n_features=2, n_redundant=0, n_informative=2, flip_y=0.1, n_clusters_per_class=1`:



An example where the test error increases.

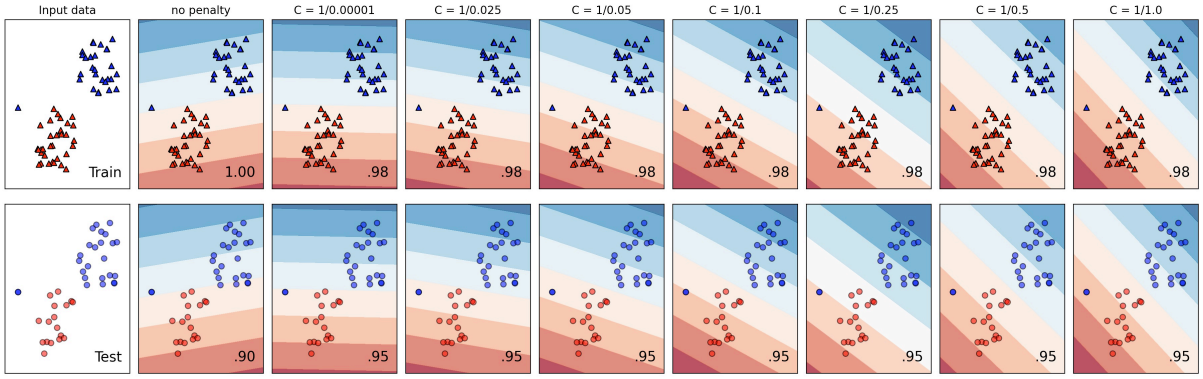
(5.3) For this part we investigated the outputs for $C = 0.5, 0.1, 0.25, 0.5, 1.0$, and 5.0. (Large C tend to have identical effects, as even stronger regularization on these few points will yield little difference.)

- Training error on all three: larger C, stronger regularization, lower training error.
- Overall, when C is smooth, the decision boundary is more smooth, whereas when C is large, the decision boundary is more fragmented.
- Test error happens to follow the same rule. For MOON and LINEARLY_SEPARABLE, the test error dramatically drops as C increases from 0.1 to 0.25. For CIRCLE, this leap takes place between $C = 0.25$ and 0.5.



SVM with RBF kernel on different data sets

(5.4) There is no difference when changing C on the three data sets given. However, the following does illustrate the effect of regularization:



Logistic regression with various penalties on a special data set

Without penalization, the model attempts to fit all data (which is indeed doable, since the training set is linearly separable). The stronger the regularization, the more it realizes the blue triangle on the far left is an outlier and disregards it.