Exercise Session1: Classification

Support Vector Machines

Zirui Yan r0916941

1.1 A simple example: two Gaussians

randn(sz1,sz2) returns an sz1-by-sz2 array of random numbers where sz1, sz2 indicate the size of each dimension. Thus, **X1 = randn(50,2) + 1** generates 50 data points whose mean is (1,1) and **X2 = randn(51,2) - 1** generates 51 data points whose mean is (-1,-1) .

Since X1 and X2 have the same covariance matrices, using the hyperplane which has the same distances to the means of X1 and X2 can minimize the probability of misclassification. So perpendicular bisector of (1,1) and (-1,-1) is the optimal line to classify the data. The figure below shows the datasets X1, X2 and the optimal line to classify the data:

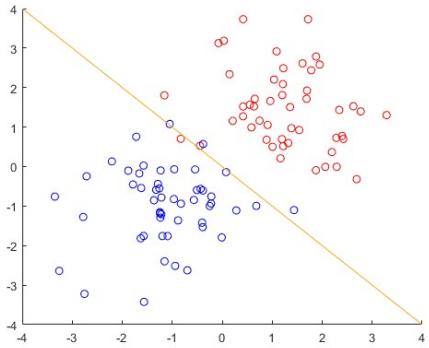

Figure 1: X1, X2 and the optimal line to classify the data

1.2 Support vector machine classifier

On https://cs.stanford.edu/people/karpathy/svmjs/demo , I do this experiment with both datasets having 11 data points. I assume the red data points should be inside quadrant IV and the green data points should be in the other quadrants. I tried linear and RBF kernels with different parameters. The figure below shows that SVM has the best performance when using RBF kernel with $\sigma = 2.5$:

It is not surprising that RBF kernel has better performance, since this problem has a nonlinear boundary.

(a) Support vectors are the vectors used for constructing the classifier. In Figure 2, support vectors are the data points which are not small. In up-left subfigures of Figure 2, RBF kernel uses all data points, which means every point is support vector. But for linear kernel, the support vectors are the data points which are on or beyond the corresponding boundary of the margin.

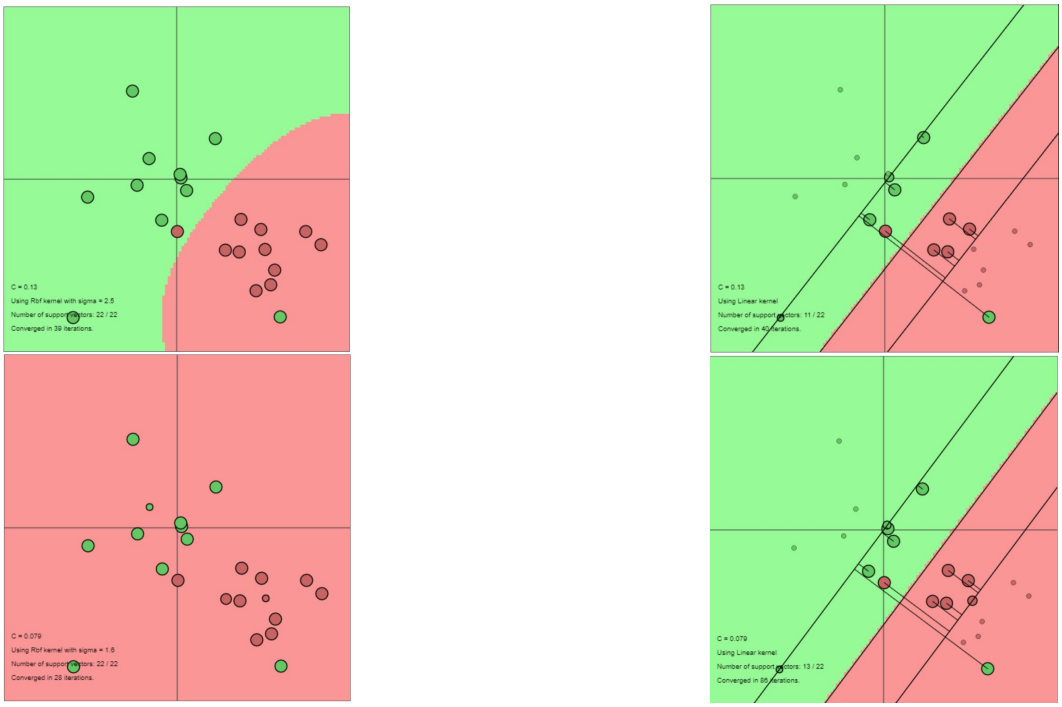

Figure 2: (up-left) SVM has the best performance when using RBF kernel with $\sigma = 2.5$; (up-right) using linear kernel with c=0.13; (down-left) using RBF kernel with $\sigma = 1.6$; (down-right) using linear kernel with c=0.079

In the case of RBF and MLP kernel, the number of hidden units corresponds to the number of support vectors. So every point is a support vector. From left 2 subfigures of Figure 2, it is hard to predict when the importance of the support vector change with RBF kernel. But the data point close to boundary is easier to be more important.

From right 2 subfigures of Figure 2, we find that a data point become a support vector when that data point is on or beyond the corresponding boundary of the margin. And also the importance of a support vector increase when it is beyond the boundary much more, which comes from the punishment of slack variables ξ:

$$minJ(w, \xi) = \frac{1}{2} w^T w + c \sum_{k=1}^{N} \xi_k$$

subject to:

$$y_k[w^T x_k = b \geq 1 - \xi_k]$$

(b) c decides the punishment of violation of original inequalities and σ decides the shape of the boundary.

Figure 3: (left) using RBF kernel with $\sigma = 79$; (right) using linear kernel with c=79

The Figure 3 shows that the margin could be very wide, since the punishment $c \sum_{k=1}^{N} \xi_k$ is small when c is very small.

(c) The Figure 3 also shows that RBF performs like linear kernel when σ is large, which can be proved by Taylor expansion:

$$exp(-\frac{(x-x_k)^T(x-x_k)}{\sigma}) = 1 - \frac{(x-x_k)^T(x-x_k)}{\sigma} + h.o.t$$

1.3.1 Influence of hyperparameters and kernel parameter

(a) When using polynomial kernel with degree=1 and t=1, the accuracy is 0.45 and classifier performs like linear kernel, which cannot solve this problem. From Figure 4, higher degree gives more flexible boundary for classifier. The accuracy for degree=2 is 0.95. The accuracy grows to 1 when degree is higher than 2. But it declines to 0.95 when degree=12, which means over-fitting happens.

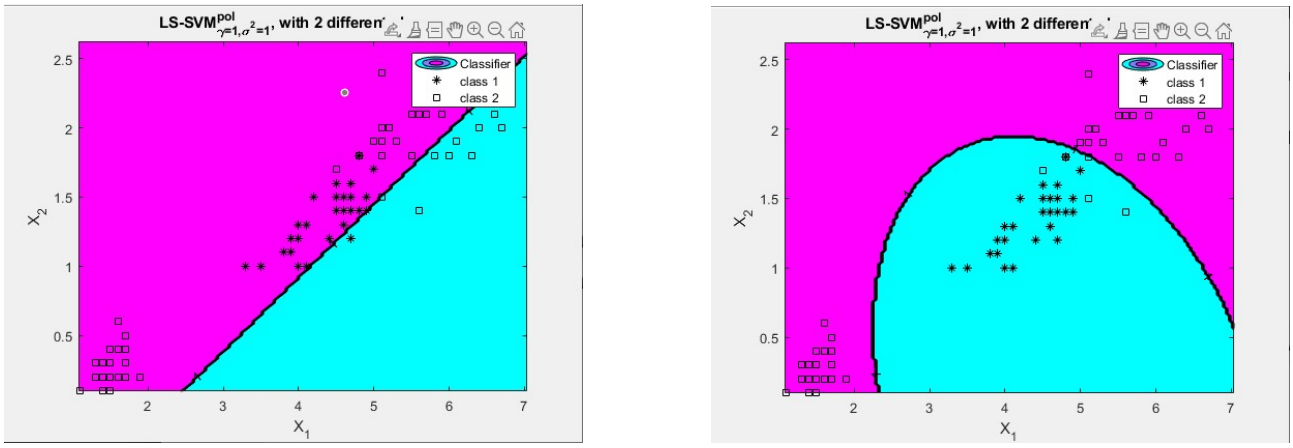

Figure 4: (from left to right) degree=1, gam=1; degree=2, gam=1

(b) The accuracy is always 1 when 0.04<sig2<12.04. Then I randomly generate 100 sig2's between 0.04 and 12.04 to check if all the value in this range is a good choice for sig2. And all the sig2$\in [0.04, 12.04]$ don't have misclassification on test set.

Figure 5 shows the hyperplane almost lies in the middle of two data sets,so I conclude sig2=3 is a very good value . And the accuracy is 1 when gam is in [0.1481, 26247].

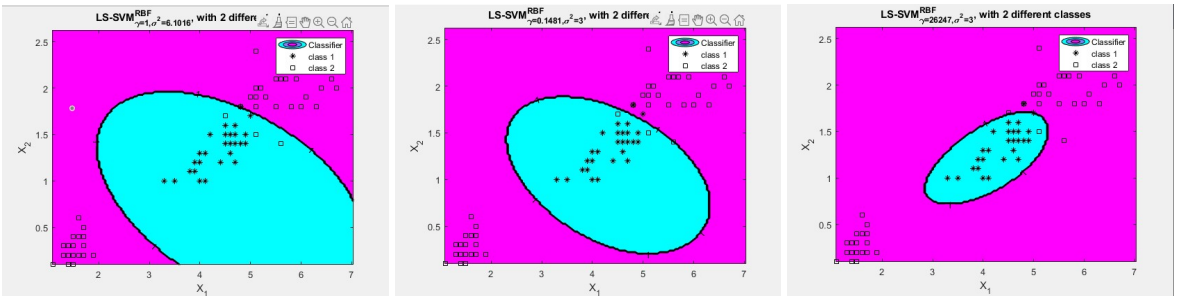

Figure 5: (from left to right)sig2=3, gam=1; sig2=3, gam=0.1482; sig2=3, gam=26247

When gam is too large, we set a large punishment for misclassification, so there is a higher possibility of overfitting. Figure 5 shows that the classifier wants to avoid misclassification, so the boundary shrinks a lot.

(c) SampleScript iris.m only use 6 points. And it gives the same good range as I gave in (b).

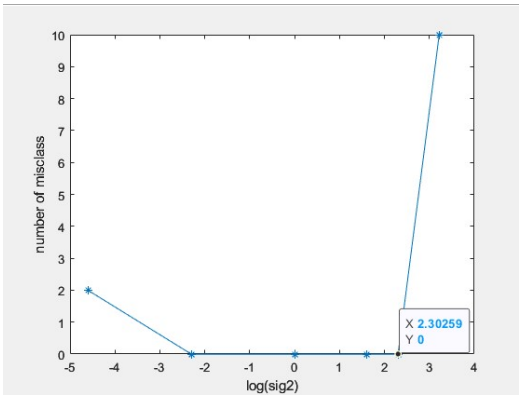

Figure 6: number of misclass with sig2=0.01, 0.1, 1, 5, 10, 25

1.3.2 Tuning parameters using validation

(b) Random split gives different performance every time running it. On the other hand, k-fold crossvalidation actually runs validation in every data point, so the result is more stable and accurate than random split.

When the size of data set is small, it is better to choose larger k in order to get a more reliable result. When the size of data set is large, it is better to use smaller k in order to reduce computation.

(a)Table 1 (on page 10) shows performances with 3 different method. Random split method gives different performances every time running the code, here the table only show the result of one time. Leave-one-out validation is a special case of k-fold crossvalidation. 10-fold crossvalidation and leave-one-out give similar results.

In the tables for 10-fold crossvalidation and leave-one-out, I use bold text to highlight the small performances(0.04, 0.05, 0.06), which indicate that the parameter combinations are good. The model with these parameter combinations have error rate less than 10%. But these is no sign to show the parameter combinations with 0.04 performance is better than the ones with 0.06 performance. According to this table, sig2=1, gam=100 is one of the good choice.

1.3.3 Automatic parameter tuning

(1) Simplex method: first, get the coupled Simulated Annealing result; then use the result as a starting value of simplex method and do the iteration until satisfying certain criterion.

Gridsearch: first, get the coupled Simulated Annealing result; then use the result to evaluate all possible hyper-parameters; try out all possible hyper-parameters.

Since the coupled Simulated Annealing results are different in every run, the starting values for simplex method are different and the grids for grid search are also different. So hyper-parameters differ a lot in different runs. Then we have different cost.

Simplex method is faster than grid search in most of time because grid search spend a lot of time on investigating different hyper-parameters. And simplex method does not need derivative information, but grid search needs it.

1.3.4 Using ROC curves

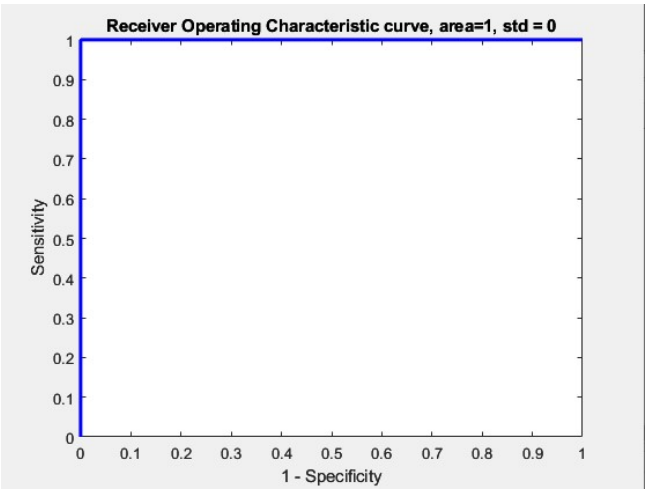

Figure 7: ROC curve for the iris.mat dataset

(a) The data in the test set are not used for training. Thus, the performance on the test set can help us estimate generalized performance. And if we judge the performance based on training data, it cannot avoid overfitting. So in practice, we compute the ROC curve on the test set, rather than on the training set.

(b) Figure 7 is the ROC curve for the iris.mat dataset. gam and sig2 are tuned by the method given in section 1.3.3. Sensitivity = TP/(TP + FN) and Specificity = TN/(FP + TN). So this figure means that every data point in the test set is classified correctly.

Although gam and sig2 differ a lot in different runs, the ROC curves are the same, which means all the combinations of gam and sig2 perform well on the test set.

1.3.5 Bayesian framework

(a) sig2=0.2140, gam=2.8045 is one result given by automatic parameter tuning. Figure 8 is the plots computed by Bayesian framework. More red the region is, it is more likely to belong to positive class. More blue the region is, it is more likely to belong to negative class.

Let's look at the upper plot of Figure 8 (sig2=0.2140, gam=2.8045). The data points of positive class are in the left-down and right-up corner, and the points of negative class are in the center. It is reasonable that the center has the most blue color and the corners have the most red color. The region which has transition color is a narrow circle, which means the performance of hyperparameters is good.

(b) Figure 8 shows the effects of changing the value of gam and sig2.

When gam becomes smaller, the model is underfitting and some positive data points are classified to negative class. When gam becomes bigger, although the classifier does well for the training data, the data points of negative class do not lie on the most blue region (100% belong to negative class), which means overfitting happens.

Changing sig2 changes the boundaries for blue and red regions. When sig2 becomes smaller, the boundaries are more complex. When sig2 becomes bigger, the boundaries look like lines. But the region where data points of negative class gather is still the most blue region when changing sig2. Same as positive.

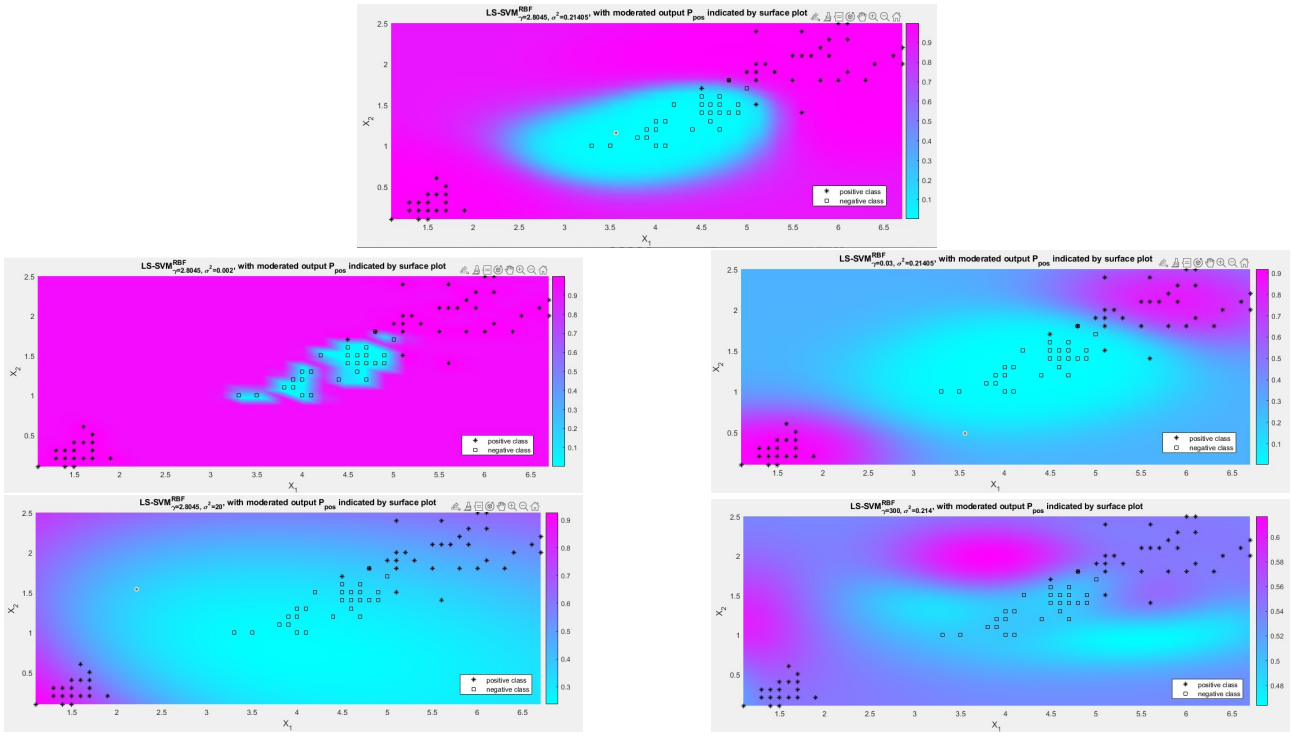

Figure 8: (up)sig2=0.2140, gam=2.8045; (middle-left) sig2=0.002, gam=2.8045; (down-left) sig2=20, gam=2.8045;(middle-right)sig2=0.2140, gam=0.03;(down-right)sig2=0.2140, gam=300;

Ripley dataset

(a) Figure 9 shows ach class of Ripley dataset gathers around two centers, which comes from that each class have been generated by a mixture of two Gaussian distributions. It is obvious that the second property is more important for classifying the two class.

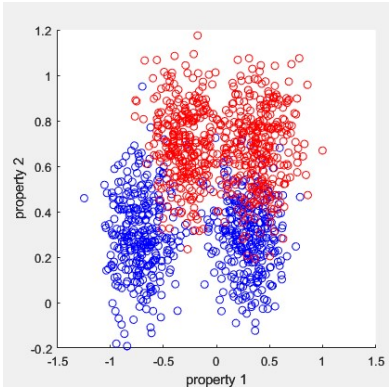

Figure 9: Visualization of Ripley dataset: (red) positive class; (blue) negative class

Three models all can do the classification since this dataset is simple. But a curve is better to classify the two classes, so polynomial kernel and RBF will perform a little bit better to solve this problem.

(b) Figure 10 shows the ROC of the three models and caption gives the error rate. Every model can handle this problem, but RBF kernel performs the best. I would like to choose linear kernel because this problem is simple and performance of linear kernel is good enough.

(c) I satisfy. Since there's some big overlap of the Gaussian distributions, it is impossible to find a classifier whose

error rate is extremely low.

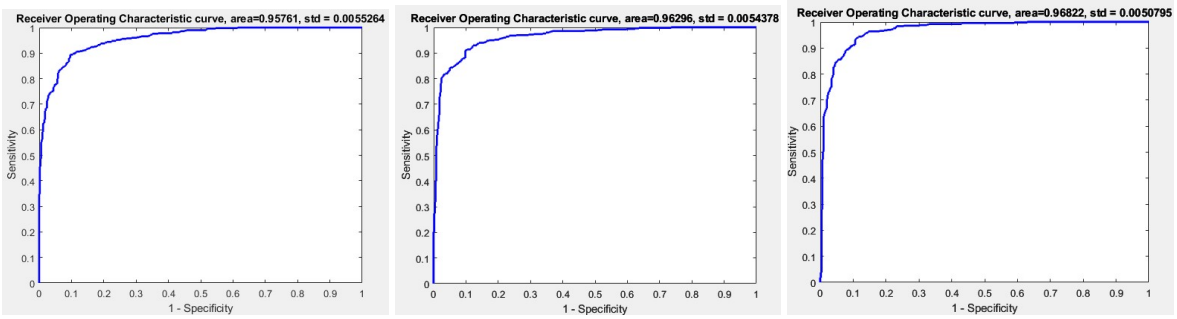

Figure 10: (left) linear kernel with gam=0.0075935, and area=0.95761 error rate = 10.80%; (middle) polynomial kernel with gam=75.7345 t=9.27152 degree=5, and area=0.96296 error rate = 10.10%; (right) RBF kernel with gam=3.2423 sig2=0.49486, and area=0.96822 error rate = 9.50%

Wisconsin Breast Cancer dataset

(a) This data set has 30 properties, so I choose to use command **parallelcoords** to visualize the dataset. As shown in Figure 11, there is no difference between the two class in some properties. Thus, we can remove these properties out off input space and remain important properties 1-4,13-14,21-24.

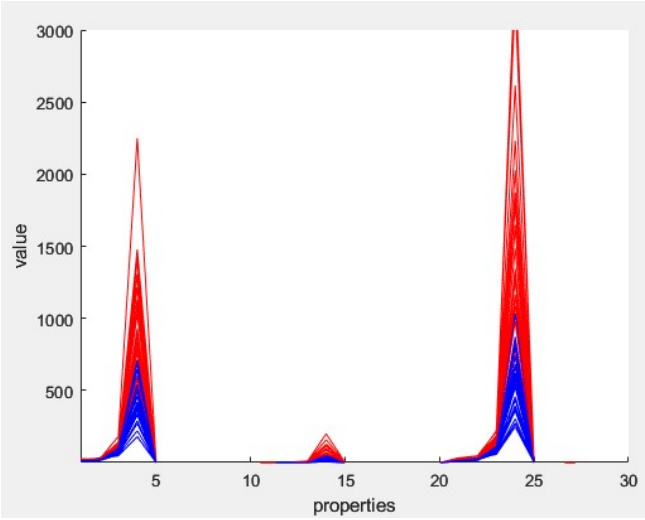

Figure 11: Visualization of Wisconsin Breast Cancer dataset: (red lines) positive class; (blue lines) negative class

It is easy to find hyperplane to classify for this data since the two classes perform very differently in the important properties. Thus, the three models still enough to solve this problem.

(b) As discussed in (a), linear kernel is enough. But it is uncommon that linear kernel has the best performance. I choose linear kernel because of its simplicity and lowest error rate

(c) I'm satisfied with the performances and my prediction is correct. But my input selection depends on my observation. Perhaps automatic relevance determination can have better performance.

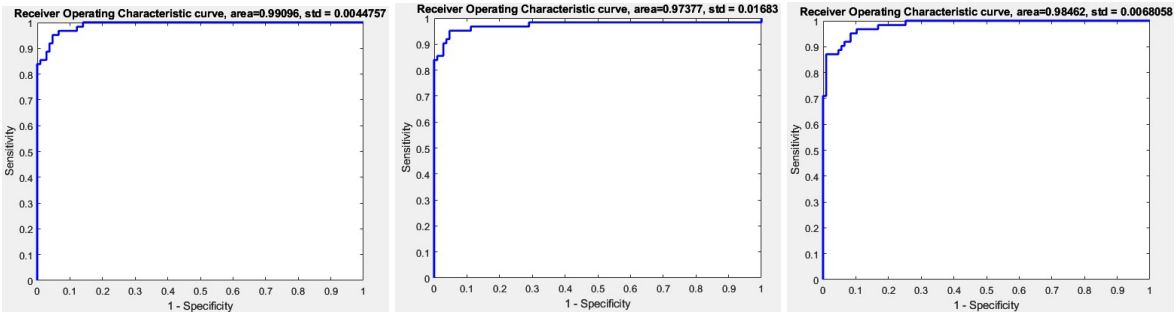

Figure 12: (left) linear kernel with gam=31.0427, and area=0.95761 error rate = 5.33%; (middle) polynomial kernel with gam=1.6487 t=1.6487 degree=3, and area=0.96296 error rate = 7.10%; (right) RBF kernel with gam=26.8765 sig2=1.36886, and area=0.96822 error rate = 7.69%

Diabetes dataset

(a) Diabetes dataset is time-series. So it is hard to say which single property is important. I visualize the data by **parallelcoords**.

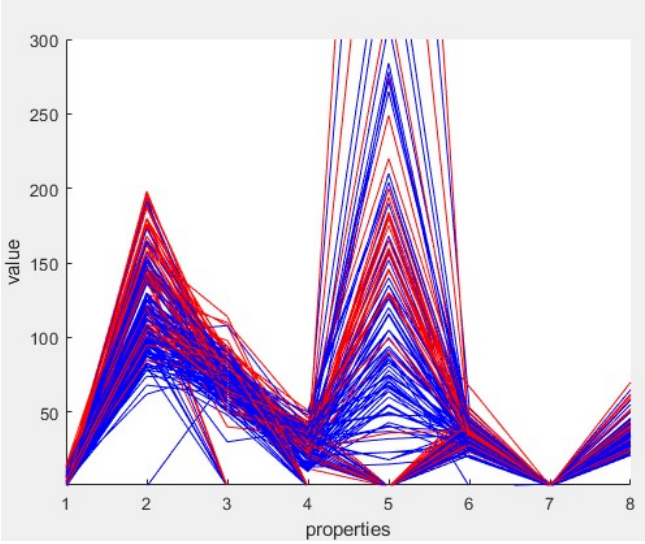

Figure 13: Visualization of Ripley dataset: (red) positive class; (blue) negative class

It is important to use the kernel which can capture the time-series features. There are some kernels are specially designed for time series. But in our case, I predict RBF has the best performance since it can capture non-linear properties.

(b) Out of my expectation, even RBF kernel performs badly. Figure 14 shows the ROC of the three models for one run and caption gives the corresponding error rate.

However, the area of ROC and error rate differ in every single run. For example, the error rate of polynomial kernel is around 27%, but sometimes the error rate goes to 46%. Generally, linear kernel and RBF have better performance (error rates are around 22%), but still not good enough.

(c) Not satisfy with the result. Maybe we can use some other kernels, such as DTW kernels (dynamic time warping). Or we can use AR or ARX model to model the time series, then apply SVM to the parameters of AR or ARX model.

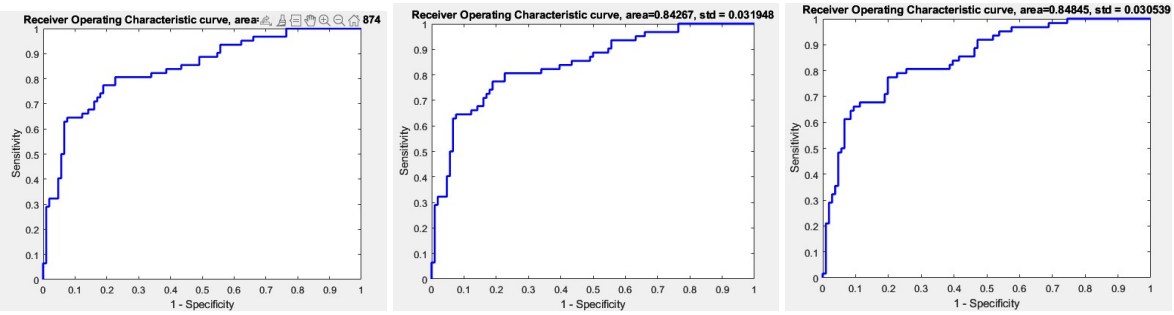

Figure 14: (left) linear kernel with gam=8.5955, and area=0.84312 error rate = 21.43%; (middle) polynomial kernel with gam=3.5531128890923e-17 t=1650996705.7644 degree=3, and area=0.84267 error rate = 21.43%; (right) RBF kernel with gam=1662.0864 sig2=3573.6198, and area=0.84845 error rate = 19.64%

perf	sig2=0.001	sig2=0.01	sig2=0.1	sig2=1	sig2=10	sig2=100
gam=0.01	0.4000	0.2000	0.4000	0.4000	0.3000	0.3500
gam=0.1	0.3500	0.1000	0	0	0.4000	0.4000
gam=1	0.1500	0.1000	0.0500	0.1000	0	0.4000
gam=10	0.2000	0	0.1000	0.1500	0.1500	0.4000
gam=100	0.0500	0.2000	0.0500	0	0	0.1500
gam=1000	0.1500	0.1000	0.0500	0.0500	0.0500	0.0500
perf	sig2=0.001	sig2=0.01	sig2=0.1	sig2=1	sig2=10	sig2=100
gam=0.01	0.3300	0.3300	0.3300	0.3300	0.3300	0.3300
gam=0.1	0.3300	0.2100	**0.0400**	**0.0500**	0.3300	0.3300
gam=1	0.2200	**0.0400**	**0.0400**	**0.0500**	**0.0500**	0.3300
gam=10	0.1800	**0.0500**	**0.0400**	**0.0500**	**0.0500**	0.3700
gam=100	0.1800	**0.0500**	**0.0600**	**0.0400**	**0.0600**	**0.0500**
gam=1000	0.2200	**0.0500**	**0.0500**	**0.0400**	**0.0500**	**0.0600**
perf	sig2=0.001	sig2=0.01	sig2=0.1	sig2=1	sig2=10	sig2=100
gam=0.01	0.3300	0.3300	0.3300	0.3300	0.3300	0.3300
gam=0.1	0.3300	0.2000	**0.0400**	**0.0500**	0.3300	0.3300
gam=1	0.1800	**0.0500**	**0.0400**	**0.0500**	**0.0500**	0.3300
gam=10	0.1800	**0.0500**	**0.0400**	**0.0500**	**0.0600**	0.3400
gam=100	0.1800	**0.0500**	**0.0400**	**0.0400**	**0.0400**	**0.0500**
gam=1000	0.1800	**0.0500**	**0.0500**	**0.0400**	**0.0500**	**0.0500**

Table 1: performances with (up) random split, (middle) 10-fold crossvalidation, (down) leave-one-out validation