Assignment 4 Report

20180396 오지오

A.

<Hyperplanes>

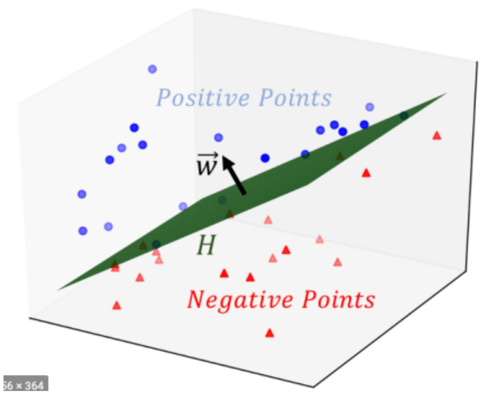
Before we handle the derivation of the SVM algorithm, we need to handle simply about the basic concepts of hyperplanes and margins. A hyperplane is an affine subspace of dimension D-1, when the corresponding vector space has the dimension of D. In this report we deal the hyperplane as a separator of the classes. Consider a function,

$$f: R^D \to R, \quad x \to < w.x > +b, \;\; w \in R^D, b \in R$$

Here, we can define a hyper plane as f(x)=0, where the w vector works as a normal vector to the hyperplane and b works as an intercept. We can prove that w is a normal vector to the hyperplane, by the following statements. Let, x_a, x_b arbitrary points on the hyperplane f. Then, $f(x_a) - f(x_b) = < w, x_a > +b - (< w, x_b > +b) = < w, x_a - x_b >$. Since we set both the points on the hyperplane, $f(x_a), f(x_b)$ all should be 0. Thus, w is orthogonal to any vector on the hyperplane.

Now that we defined hyperplanes, we can classify the data as positive or negative depending on the side of the hyperplane on which it occurs. For instance, in the below picture we might define points on the left of the hyperplane as positive and the points on the other side as negative. So, as y indicates true values (+1, -1), we can define the labels of the datasets as the following, if classified correctly.

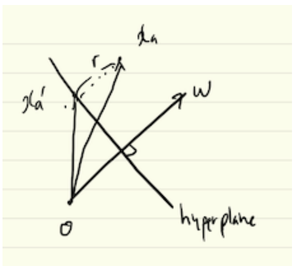
$$f(x_i) * y_i \geq 0$$



<Maximal Margin Classifier>

One idea to find the optimal classifier is to maximize the margin between the positive or negative samples. Margin is simply defined as the distance of the separating hyperplane to the closest examples in the dataset. For an arbitrary point, which is not on the hyperplane, we could express the vector as the following, where we should normalize the w vector by dividing it by its norm.

$$x_a = x'_a + r\frac{w}{||w||}$$



Now that we would like to make the examples be further than the hyperplane with the magnitude of r, thus we could think of one of the constraints as

$$f(x_i) * y_i = (< w, x_n > +b)y_i \geq r.$$

Here, f(x) indicates the distance between the point and the hyperplane, thus, the w vector should only indicate the direction here. Plus, should give no effect to the magnitude of the length, so we should have the restriction that the norm of the w vector should have unit length, in which we need a new restriction, where ||w|| = 1. Then, this we could be formulated as

$$\max_{w,b,r} r$$

s.t. $(< w, x_n > +b)y_i \geq r, ||w|| = 1, r > 0.$

<Derivation of the Hard Margin>

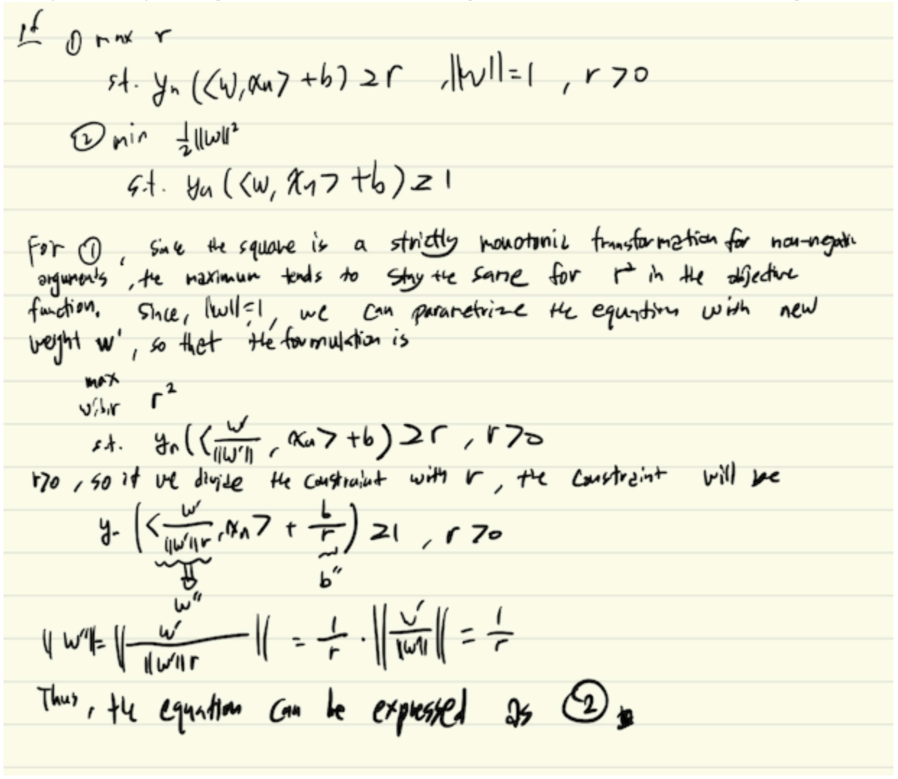
In the Maximal Margin Classifiers, we solve the problem based on the assumption that ||w||=1, here instead of choosing that the parameter should be normalized, we tend to choose a scale for the data. We choose this such that the predictor value $< w, x_n > +b$ is 1 at the closet example. We denote this closet point as, x_a and this point's orthogonal projection to the hyperplane can be denoted as x_a'. Using this information, we obtain $< w, x_a - r\frac{w}{||w||} > +b = 0$ and by bilinearity, this can be transformed as $< w, x_a > +b - r\frac{<w,w>}{||w||} = 0$. Since, $< w, x_n > +b$ is 1, we get to derive that r = 1/||w||. We considered the closest point to the hyperplane to have the distance 1, thus now we have the constraint $(< w, x_n > +b)y_i \geq 1$. Hence, the formulation will be as the following.

$\max_{w,b} \frac{1}{||w||}$
s.t. $(< w, x_n > +b)y_i \geq 1$ for all n.

Instead of maximizing the reciprocal of the norm, we tend to minimize the norm squared, and often include the constant 1/2, which doesn't affect the optimal w, in order to get a tidier form when computing the gradient. Hence, our formulation for the hard margin SVM will be as the following. We say that this is a hard margin, since it doesn't allow any violations of the margin condition, which is a bit different from the soft margins, which will be indicated afterwards.

$\min_{w,b} \frac{1}{2}||w||^2$
s.t. $(< w, x_n > +b)y_i \geq 1$ for all n.

The proof of proving that the maximal margin classifier and the hard margin SVM is equivalent is given below.



pf ① $\max\ r$
s.t. $y_n(\langle w, x_n\rangle + b) \geq r$, $||w|| = 1$, $r > 0$

② $\min\ \frac{1}{2}||w||^2$
s.t. $y_n(\langle w, x_n\rangle + b) \geq 1$

For ①, since the square is a strictly monotonic transformation for non-negative arguments, the maximum tends to stay the same for r^2 in the objective function. Since, $||w|| = 1$, we can parametrize the equation with new weight w', so that the formulation is

$\max_{w',b,r}\ r^2$
s.t. $y_n(\langle \frac{w'}{||w'||}, x_n\rangle + b) \geq r$, $r > 0$

$r > 0$, so if we divide the constraint with r, the constraint will be

$y_n(\langle \underbrace{\frac{w'}{||w'||r}}_{w''}, x_n\rangle + \underbrace{\frac{b}{r}}_{b''}) \geq 1$, $r > 0$

$||w''|| = ||\frac{w'}{||w'||r}|| = \frac{1}{r} \cdot ||\frac{w'}{||w'||}|| = \frac{1}{r}$

Thus, the equation can be expressed as ② ∎

<Derivation of Soft Margin (Geometrically)>
There might be some cases, where the two classes cannot be linearly separable and we may want some examples to fall in some particular regions. This soft margin models allow some classification errors. When interpretating this in a geometrical way, we add a slack variable, e_n, that corresponds to each example pair (x_n, y_n). This allows that particular point to be within the margin, even when in the wrong side of the hyperplane. This, then can be denoted as the following. (Slightly different form from the text, but indicating the same formulation)

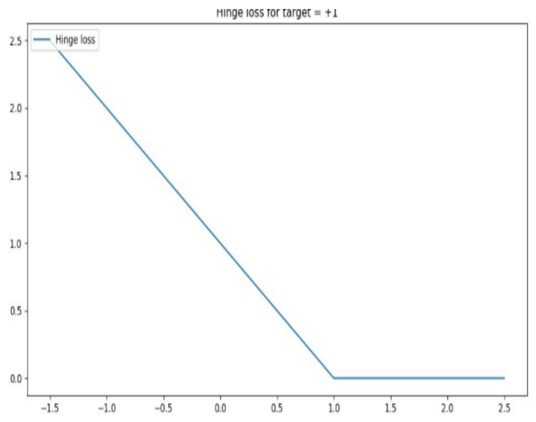
$\min_{w,b,e} \frac{1}{2}||w||^2 + C\sum_{n=1}^{N} e_n$
s.t. $(< w, x_n > +b)y_i \geq 1 - e_n,\ e_n \geq 0$ for all n.

Here, C is the regularization parameter where it trades off the size of the margin and the total amount of slack that we have.

<Derivation of the Soft Margin (View of Loss Function)>

We used loss functions in order to optimize various models, and we try to approach the SVM in this way. For the SVM the hyperplane is denoted as f(x) = <w,x> +b. With this in mind, let's try to think about the loss functions in SVM. Here, we tend to use the hindge loss, where the hindge loss is defined as

$$l(t) = \max\{0, 1-t\}, where\ t = y\, f(x).$$



When, f(x) is on the correct side and has a further distance than 1, the hindge loss returns the value of 0, while if it is within the margin (between 0 and 1), the hindge loss returns a positive value. When, in the wrong side of the plane, it returns even a bigger value. For the given training set, we tend to minimize the total loss, where the loss can be expressed as the sum of the regularizer and the error term as

$$\min \frac{1}{2}\left\|w\right\|^2 + C\sum_{n=1}^{N} \max\{0, 1 - y_n(< w, x_n > +b)$$

This form can be solved with gradients and we can see that this equation is equivalent to the above Soft Margin function by replacing the minimization of the hindge loss with the slack variable e, where we get $\min\max\{0, 1-t\}$ for the hindge loss, which is equivalent to minimizing e, with the constraints in which e>=0 and e>=1-t.

B.

$\min\limits_{w,b,e} \frac{1}{2}||w||^2 + C\sum_{n=1}^{N} e_n$

s.t. $(< w, x_n > +b)y_i \geq 1 - e_n,\ e_n \geq 0$ for all n.

Based on this formulation, we call w, b, and e as the primal variables and use the Lagrange multiplier $\alpha_n,\ \gamma_n$. Then the Lagrangian is given by

$$L(w,b,e,\alpha,\gamma) = \frac{1}{2}\left\|w\right\|^2 + C\sum_{n=1}^{N} e_n - \sum_{n=1}^{N} \alpha_n(y_n(< w, x_n > +b) - 1 + e_n) - \sum_{n=1}^{N} \gamma_n e_n.$$

If we differentiate this by the three primal variables, we get the following results.

$$\frac{\partial L}{\partial w} = w^T - \sum_{n=1}^{N} \alpha_n\, y_n x_n^T$$

$$\frac{\partial L}{\partial b} = \sum_{n=1}^{N} \alpha_n\, y_n$$

$$\frac{\partial L}{\partial e_n} = C - \alpha_n - \gamma_n$$

(Plus we set data, $\alpha_n > 0$ as support vectors). We can now find the maximum of the Lagrangian, by setting all the three equations above at zero, where we can get $w = \sum_{n=1}^{N} \alpha_n\, y_n x_n$. By replacing this to the expression for w into the Lagrangian, we obtain the dual as the following.

$$D(e,\alpha,\gamma) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle - \sum_{i=1}^{N} y_i \alpha_i \left\langle \sum_{j=1}^{N} y_j \alpha_j x_j , x_i \right\rangle + C\sum_{i=1}^{N} e_i - b\sum_{i=1}^{N} y_i \alpha_i + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i e_i - \sum_{i=1}^{N} \gamma_i \varepsilon_i$$

We also obtained that $\sum_{n=1}^{N} \alpha_n\, y_n = 0$, thus the term with which b is involved also disappears. Plus, inner products are bilinear as well as symmetric, therefore the first two terms are over the same objects. Thus, the dual can be expressed as the following.

$$D(e,\alpha,\gamma) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle + \sum_{i=1}^{N} \alpha_i + \sum_{i=1}^{N} (C - \alpha_i - \gamma_i)\varepsilon_i$$

The last term can be diminished since, we also obtained that $C - \alpha_n - \gamma_n = 0$. Therefore, we can determine the dual.

$$D(e,\alpha,\gamma) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle + \sum_{i=1}^{N} \alpha_i$$

We can recall that the Langrage multipliers are nonnegative, thus by $C - \gamma_n = \alpha_n$, we can derive that $\alpha_n \leq C$. In the Langrage duality, we tend to maximize the dual problem and this is no different from minimizing the the negative dual. Finally, we get the formulation as below.

$$\min_{\alpha} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle - \sum_{i=1}^{N} \alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{N} y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq C \quad \text{for all} \quad i = 1, \dots, N$$

C.

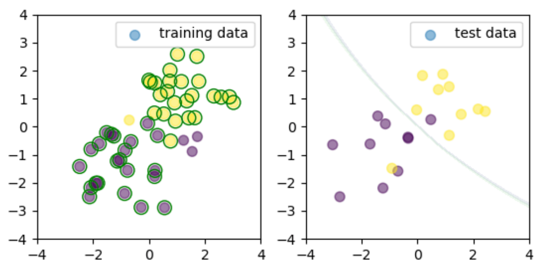
Before we evaluate the models, let's take a brief look at kernels in SVM.

In the formulation of the SVM, the inner product in the objective function occurs between only the examples. Thus, the set of features of g(x) to represent the x, the only change in the dual SVM, will be the inner product. This g(x) can be a nonlinear function, which helps the SVM to have nonlinear classification boundaries. This provides an additional option for the users to deal with noisy or linearly non-separable datasets. Instead of explicitly defining a non-linear map g(x) and compute the inner product, we rather define a similarity function k that gets the input of two examples, which we call it as a kernel. This has a feature map such that $k(x_i, x_j) = \langle g(x_i), g(x_j) \rangle$. This allows the model to have high-dimensional expressivity without overfitting, since we finally shrink the dimensions when actually computing the results. In this assignment, we deal with 3 kernels, (1) polynomial, (2) rbf, (3) Euclidean. The polynomial kernel has the form of $k(x,y) = (x*y+1)^d$. The rbf kernel has the form of $k(x,y) = \exp(-\gamma\,||x-y||^2)$, for $\gamma > 0$. (This has steep gradients and allows us to control some hyperparameters. In our project we control 3 hyperparameters to control the variance.) The Euclidean kernel has the form of d(x,y) where d indicates the Euclidean distance.

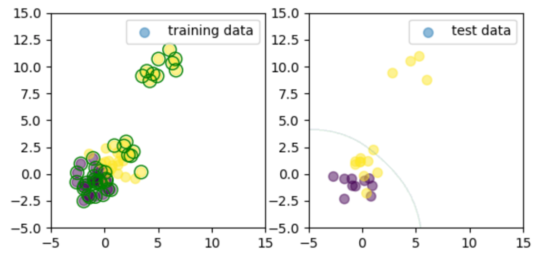
<Selecting the kernel>

Among the various kernels, that I could utilize, I had to choose one, that fit the dataset in this case the most. After lots of trials and experiments, I decided that the polynomial kernel with the degree of 1, will work the best. There were two reasons behind, this reasoning. First, in a straightforward way, experiments told me that the polynomial kernel with degree one, shows the best performance. The Euclidean kernels were excluded immediately, since it showed really poor performances, especially for those where outliers exist. It seemed to be so volatile to outliers. It seems that the kernel based on pure distances, caused this phenomenon. The following below, is the comparison table and the plots for each kernel. The C value is set to 1, in order to compare the function fairly.

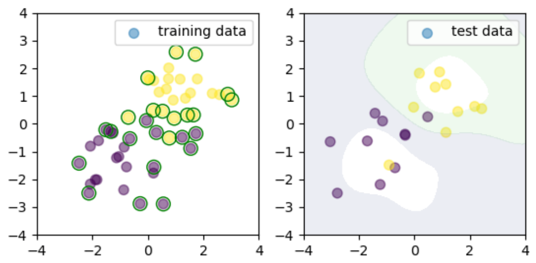
Synthetic data (old data)		WHOLE DATA		
	HYPERPARAM		Avg	STDEV
EUCLIDEAN		--------	22.42%	14.0012
POLYNOMIAL		1	89.25%	5.088
		2	45.75	10.759
		3	77.0833	13.2877
		4	52.1667	10.9557
		5	73	18.0247
RBF		1,1,1	87.5	6.1351
		0.5,1,1	89.0833	5.6452



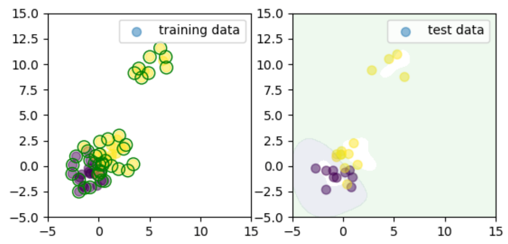
Euclidean kernel dataset 2



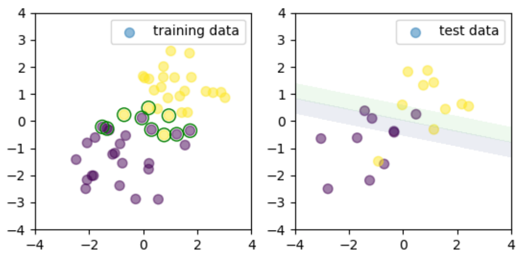
Euclidean kernel dataset 8



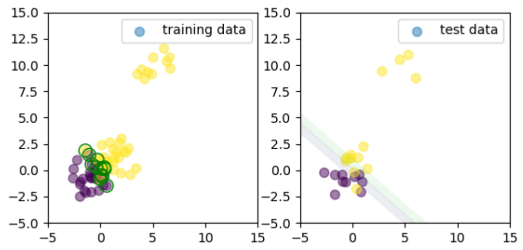
RBF kernel (0.5,1,1) dataset 2



RBF kernel (0.5,1,1) dataset 8



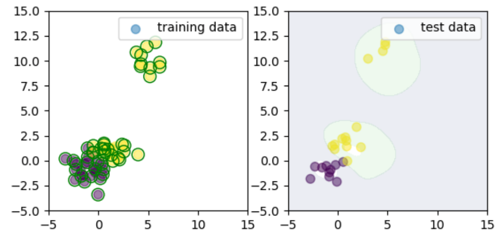
Polynomial kernel with degree 1 dataset 2



Polynomial kernel with degree 1 dataset 8

The most important reason, why I selected the polynomial kernel with degree one was because of the distribution of the data. If we look at the distribution of the data, we can see that the data seems to be well separated by a single line, since the data in different classes, tend to be separated vividly. One class is clustered in one section (usually at the south west parts in the plot), while the others are clustered in the other section (north east part relatively). The whole data can be classified easily divided into two parts with a single line, without any wiggly features. This is why the higher degrees of polynomial kernels tend to perform badly for this dataset. There is no need to use the RBF kernel, since we don't really need to boost their dimensions up and shrink it back again, since a line is enough to separate optimally.

When we decrease the value of C, it allows the outliers to exist, so we can see that as we decrease the value of C, the model seems to perform better. This might be good when we have a perfect assumption that outliers exist in the data. Plus, when we do this, we can see that the accuracy of the RBF kernel is boosted up by a big margin. However, this might not be an optimal kernel, since we can see in the plot that most of the areas are considered as the margin area and the actual classifier isn't really working well.



C=0.4, RBF kernel (0.5,1,1) dataset 5

So, the optimal average accuracy and its standard error was 89.25% with the standard error of 5.088 when C =1. Though, it showed slightly better results when C value was modified. The role of C will be explained later.

POLY	C=0.6	1	89.75	4.8883
POLY	C=0.4	1	89.8333	4.0961
POLY	C=0.1	1	89.75	5.6893
POLY	C=0.3	1	89.75	4.6667
RBF	C=0.4	0.5,1,1	90.3333	5.9184
EUCLIDEAN	C=0.4		21.5833	13.0461

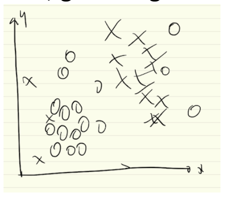
<Sensitivity Analysis for outliers>

After selecting the kernel, I tried to look at the effects of outliers in the data, in order to check the sensitivity of the SVM model. I thought that the effect of outliers would vary between the kernels used, so I tried to find the sensitivity for all 3 kernels and this as the result. Here, I just set the value of C to 1 for comparison purposes, though it might not give optimal results.

Synthetic data (old data)		DATA 0~4	C=1	
	HYPERPARAM		Avg	STDEV
EUCLIDEAN		--------	9	3.7417
POLYNOMIAL		1	86	3.7417
		2	44	11.1355
RBF		1,1,1	85	3.1623
		0.5,1,1	89	2
	DATA 5~9	C=1		
	HYPERPARAM		Avg	STDEV
EUCLIDEAN		--------	35.8333	4.2492
POLYNOMIAL		1	92.5	4.0825
		2	47.5	10.0692
RBF		1,1,1	92.5	4.0825
		0.5,1,1	92.5	4.0825

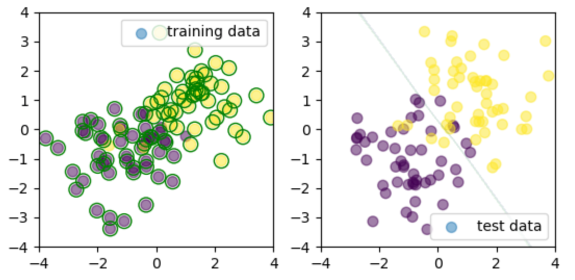
As we can see in the above data, Euclidean kernels give so much of a poor performance, so I tend to exclude that in the discussion. (It also seems to give an awkward result.) Surprisingly, in this dataset, the one with outliers showed much better performance than that of those without outliers. This is a strange phenomenon, since SVM is usually well-known as not robust to outliers. Then, why does this happen? After a long time of consideration, I derived the conclusion, that this was because of the special distribution that we had for our dataset. Let's say the data classified as purple as class 0, and yellow as class 1. In our datasets with outliers, all the outliers were datapoints from class1 and was actually in the region, where class1 existed. In SVM we control our classification boundary by the support vectors and those outlier points, which were far away didn't really distort the SVM classifier. It just affected the decision that determined the support vectors, which made the model to classify the points even better, since more and more points closer to the boundaries were taken into consideration. Thus, this boosted the performance of the classifier.

SVM is mostly robust to the outliers that are similar to above, however, in the worst cases and for some kernels, presence of a few outliers might distort the model, especially in cases shown below. The points on the wrong side, give a big value to the hindge loss function, completely changing the model in a bad way.

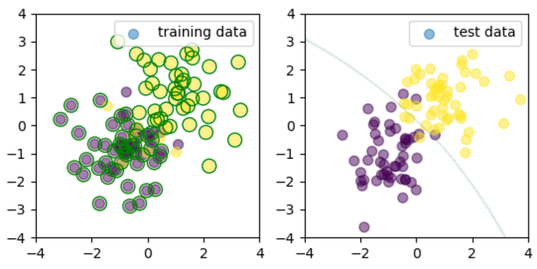


D.
<Selecting the kernel>
Among the three kernels, that were available for use, after numerous trials, the polynomial kernel with the degree of 1 and the RBF kernel showed the best results. The Euclidean kernel seemed to show good plots; however, the numbers were terrible and also showed high standard error. Since, the RBF kernels tend to gives the best accuracy and the polynomial kernel with degree 1 gives us the best standard error, it was hard for me to precisely define a better kernel, so I decided to see both cases, also by altering C, which will be elaborated afterwards. This model has lots of points that are cluttered and the data points aren't separated clearly as the above datasets, thus I thought that nonlinearity might enhance the performance.

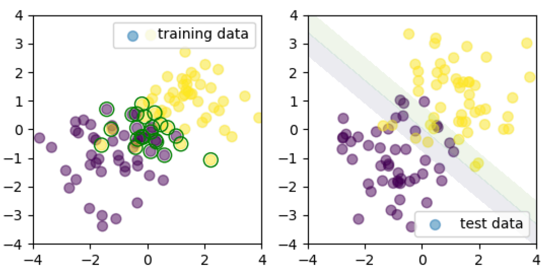
	HYPERPARAM		AVG	STDEV
EUCLIDEAN		--------	8.1	2.7731
POLYNOMIAL		1	91.9	1.8138
		2	48.3	4.5177
		3	89.6	2.6153
RBF		1,1,1	91.7	2.0025
		0.5,1,1	91.7	2.2825
		2,1,1	91.6	2.0591
		0.8,0.5,1	91.8	2.358
		1,0.5,1.2	92	2.3664
		1,0.5,1.5	91.9	2.3431



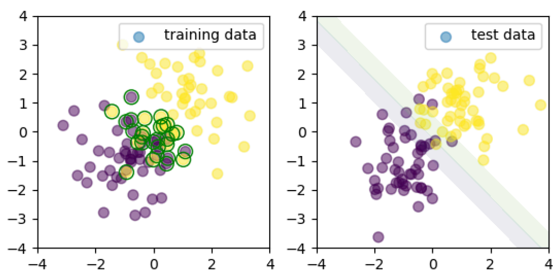
Euclidean kernel dataset 3



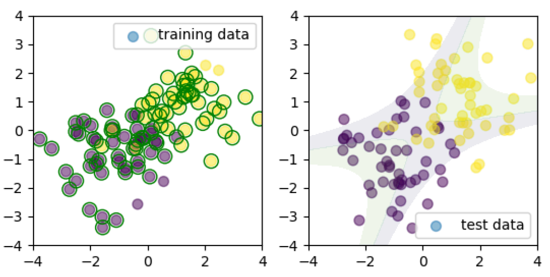
Euclidean kernel dataset 8



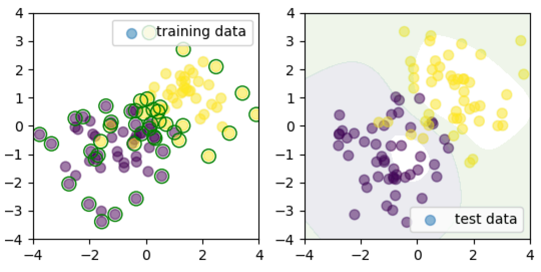
Polynomial kernel with degree 1 dataset 3



Polynomial kernel with degree 1 dataset 8



Polynomial kernel with degree 2 dataset 3



RBF kernel (0.5,1,1) dataset 3

For polynomial kernels, with C=1, it had 91.9 % accuracy with a good standard error of 1.8. This seems to happen, since the dataset here also can be well-divided with a single line. As I modified the values of C, it seemed that no real big changes were occurring to the prediction accuracy, obtaining similar or slightly worse results than when

C was 1.

POLYNOMIAL	C=0.4		91.9	1.8138
	C=0.7		91.8	1.8868
	C=1.3		91.9	1.8138

<RBF kernels>
In our model, we have 3 hyperparameters for the RBF kernels, where the RBF kernel is expressed as this form.

$$k(x_i, x_j) = \frac{1}{\sigma} \exp\left(\frac{\|x_i - x_j\|}{l}\right)$$

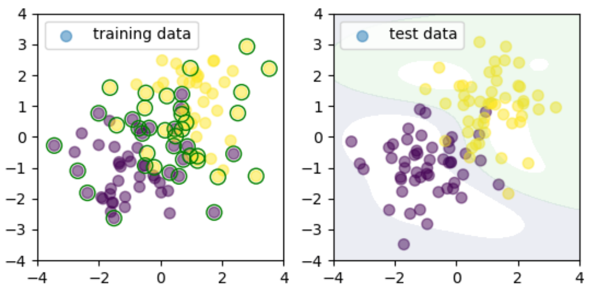
$$\text{where } \frac{\|x_i - x_j\|^2}{l} = \frac{x_{i1} - x_{j1}}{l_1} \cdot \frac{x_{i2} - x_{j2}}{l_2}$$

Here, the 3 parameters that we control is $1/\sigma$, l_1, l_2. The first parameter, $1/\sigma$ determines how far the influence of a single training example can reach. Low values of this parameter mean "far", while high values indicate "close". L_1, l_2 parameters work as a weighting term, telling us, where we will put more weight on. If we set both as 1, we think that the change in the axis of x_1 has the same weight as the change on the axis of x_2.
In all kernels, we get to control one common parameter, which is C, indicated numerous times above. This C, simply speaking, is the budget, for allowing errors. This trades off correct classification of training examples against maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all the training points correctly. A lower value of C, larger margins are allowed, at the cost of training accuracy. However, if the C value is too big, overfitting might occur.

<My selection of RBF Kernel Hyperparameters>
As we can see in the plots above, in my perspective I thought that though there are differences between the two classes in the axis of x_2, the major factor that distinguishes the two classes were the differences in the distance along the axis of x_1. This was because, the one class is clustered at the left, while the other is mostly clustered at the right. In order to prove this, I tried one case, where the three inputs were "1, 0.5, 1.2", which is shown as having the most optimal accuracy of 92 with the standard error of 2.37 and also tried "1,1.2,0.5". As expected, the accuracy slightly dropped by 0.4. Plus, the Gamma parameter was derived by lots of trials. In our dataset, the two classes, though some are cluttered are relatively well divided. Thus, the different gamma values really didn't affect the accuracy by a big margin. After various trials, C was determined to be 1.5. As I increased the C value from a very small value until 1.8, the biggest accuracy as long as the smallest standard error was obtained when C value was 1.5, which seemed to be the global maximum, when the hyperparameters were fixed as 1,1.2,0.5, which also seemed to be the global maximum. (After fixing the values of l1 and l2, I tried to vary $1/\sigma$, but 1 gave me the optimal result.)

C modify	HYPERPARAM		AVG	STDEV
RBF	C=0.6	1,0.5,1.2	91.8	2.3152
	C=0.8	1,0.5,1.2	91.9	2.3854
	C=1.3	1,0.5,1.2	92.1	2.2561
	C=1.5	1,0.5,1.2	92.2	2.0881
	C=1.8	1,0.5,1.2	92	2.3238

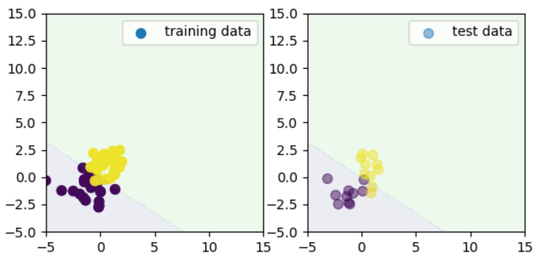
The final accuracy and its standard error resulted as 92.2 with the standard error of 2.088.



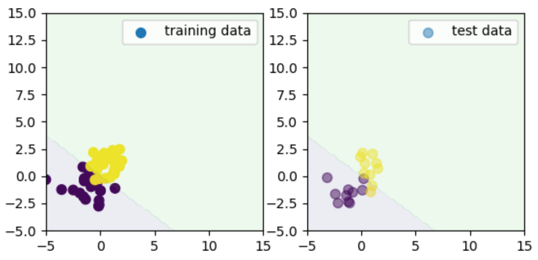
RBF kernel /1,0.5,1.2 /C=1.5/Dataset 4

For the new data, the points are a bit cluttered than the synthetic dataset that we dealt in C. Although each point for each class seems to be separated at each area, points from different classes are clustered around the boundaries and some points seem to be in the wrong area. Here, the hard margin SVM, where it requires the classes to be classified nearly perfectly, doesn't work well. However, the soft margin SVM, that we introduced in this model, where we allow some of misclassification along the boundaries, works well although datapoints are clustered along the boundaries. We try to modify the hyperparameter for the SVM, C, in order to control this, where higher values of C indicate smaller margin, having low bias though high variance. On the other hand, smaller values of C indicate bigger margins, having low variances, but might have high biases. By controlling this C value as done above, we can handle these cluttered datasets and find the optimal boundary. This feature of the SVM allows, better performances for cluttered points.

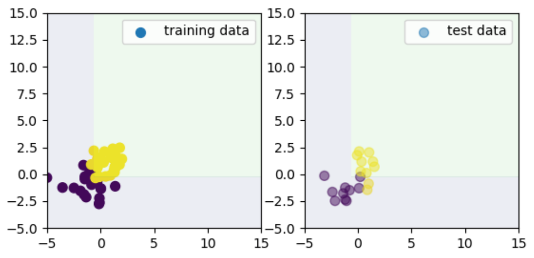
E.



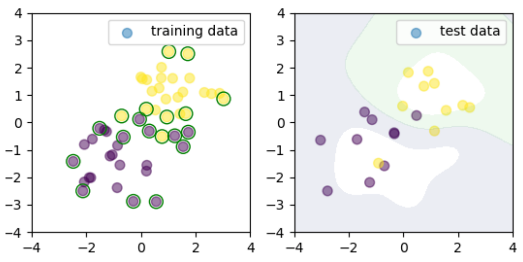
LDA dataset2



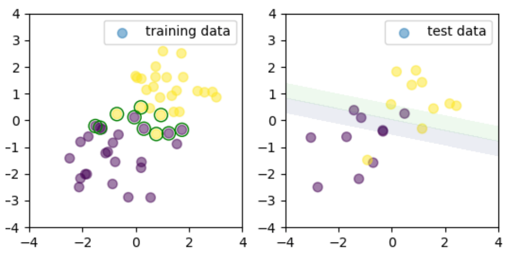
Logistic Regression dataset2



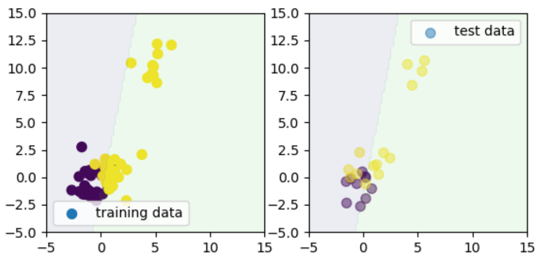
CART Tree Dataset2



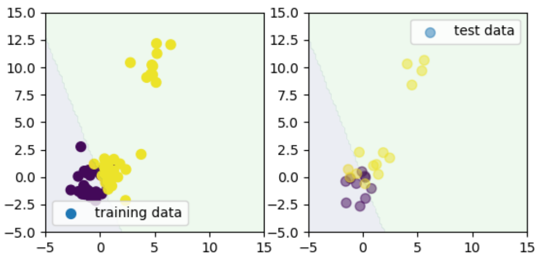
RBF kernel SVM dataset2



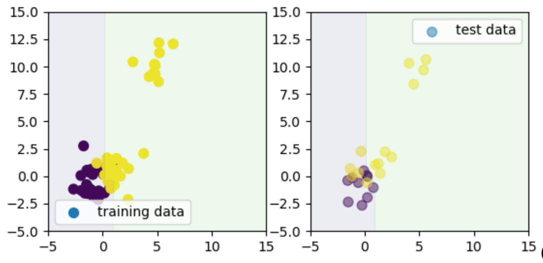
Degree 1 Polynomial kernel SVM dataset2



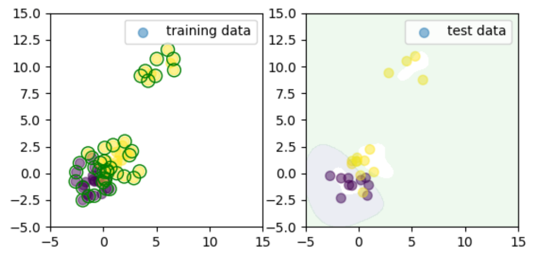
LDA dataset8



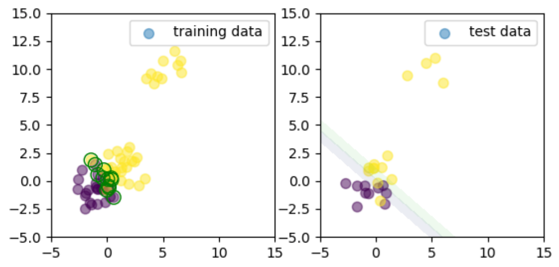
Logistic Regression Dataset8



CART tree Dataset8



RBF kernel SVM dataset8



Degree 1 Polynomial kernel SVM dataset8

From the plots, we can see some basic differences from LDA, LR, and decision trees. First, let's deal with the the differences between the logistic regression and the LDA. Though it is possible to make nonlinear boundaries with feature engineering in logistic regression, in most cases, LR supports linear boundaries. Plus, it allows us to estimate probabilities unlike the SVM method. However, SVM allows both linear and nonlinear boundaries using the trick of kernels and usually, except for some, handle outliers slightly better. Moreover, LR performs poorly when the classes are separable, so SVM has a slight advantage over LR in these cases since hindge loss is also evaluated slightly better than the log loss in logistic regression. LDA also has a big difference from the SVM, so that it only can deal with linear boundaries and requires the normal distribution. SVM seems to better in this point of view, but the LDA has the closed form, which makes computation a bit simpler than the SVM, especially for the complicated or high-computational kernels. Plus, this also stands for the LR, but SVM doesn't allow feature selection, which is a big difference, since by LDA we can reduce the features to make the model a bit simpler. SVM uses all parameters to predict the model, which might give a slight advantage in training error, but as a tradeoff, the interpretability decreases. The details of each methods are mentioned below. Lastly, the decision tree results really seem to be different, since the boundaries of decision trees are somewhat unique, since they are all parallel to the axes. It cannot have any wiggly lines on the decision boundaries, it could only have straight lines parallel to the axes. Unlike SVM, this can be used for regression tasks and does feature selection internally.

Now let's compare detailed properties of LDA/LR/DT/SVM. Logistic regression has good probabilistic interpretation and the algorithm can be regularized to prevent overfitting. Plus, it can be updated easily by the SGD, however, they ought to underperform when multiple or nonlinear decision boundaries are required. Plus, due to low flexibility it is not optimal to capture some complex relationships. LDA provides a low-dimensional view for the data, which might be useful when there are numerous response classes. Also, it is easy to compute, since there is an explicit solution for the process. But it requires the normal distribution for inputs. On the other hand, trees are the most interpretative and is very good for visualization. It also performs feature selection implicitly without any external efforts. Plus, it allows to perform on both regression and classification tasks. However, it is very easily overfitted, since it is prone to create over-complex trees. Also, if two variables have similar expressiveness in the model, a slight change in the selection of the variable, might lead to a substantial change in the model. Also, a slight change in the dataset might also give a big change in the model, leading to high variance and low stability. Moreover, as mentioned above when the dataset cannot be divided well by horizontal or vertical ways, the performance will be very poor. Lastly, it is likely to create biased trees if some classes dominate. SVM use a mechanism called a kernel, in which it calculates the distances between observations. This finds the decision boundary, hyperplane, that maximizes the distance between the closet members of the separated classes. By the help of kernels SVM allows to obtain non-linear decision boundaries, and we have numerous kernels that we could choose from in order to optimize the classification. Also, it is robust against overfitting, especially in high dimensional spaces, and no pre-distributions or distributions are required. Plus, it doesn't suffer from any multicollinearity issues. Plus, SVM uses hinge loss, where it utilizes data that are the most difficult to discriminate. However, SVM are trickier to tune due to the importance of picking the correct kernel, which takes time and effort. Plus, it doesn't scale well to huge datasets. Moreover, some kernels aren't that robust to outliers.