Analysis Report

The requirement is to classify Data Set usind Perceptron/Linear SVM and Kernel SVM.

Task Performed:

Data Set Generation and Classifying into Classes Regions, splitting training/testing, Executing the Perceptron/Linear SVM/ Kernel SVM, fine tuning "C" Parameters and"Gamma" values to improve the accuracy between the predicted and calculated test pattern.**Experimented with different values of “C” Parameter and “G” Parameter to improve the accuracy of the algorithms.**

Data Set Details

Each Region data set is divided into train and test data sets using different ratios as recommended in the question and collated in a single Train Data Set and Test Data set.

These patterns are then passed to Perceptron/SVM Linear/SVM Kernel Algorithm.

Weights/Intercept/Accuracy is calculated and tabled below:

Graph/Tables are prepared and shared below. Complete details are provided along with the code in the pdf attached.

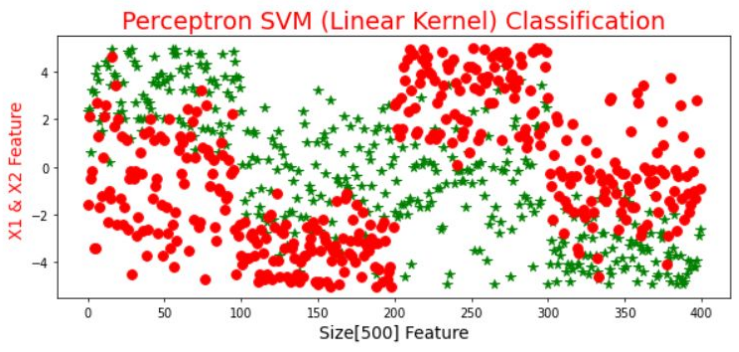
Tabulated the coefficient, accuracy, weighted vectors and intercept along with Plotting the accuracy graph/tables and Plotting the Datasets features are done here.

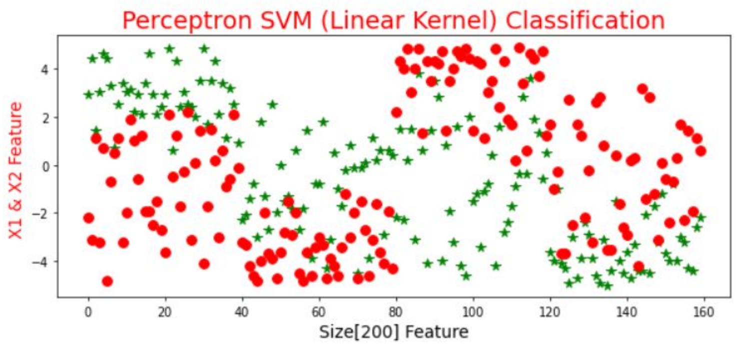
The patterns are generated between the range[-5,5], classified in [C1.,C2,C3,C4, R31,R24,R14,R24,NC1,NC2] based on the above below. If it satisfies the Class - it is marked YES else NO- similar for Regions and NC. Also NC are Labelled as [0,1] and for Kernel Algo - we have [Y] Label since we need to compute for2 Dimension.

	X1	X2	X1Square	X2Square	Y	C1	C2	C3	C4	R13	R14	R24	R23	NC1	NC2	NC-Class
0	2.60	-2.80	6.76	7.84	1	Yes	No	Yes	No	Yes	No	No	No	Yes	No	1
1	3.40	3.30	11.56	10.89	1	Yes	No	Yes	No	Yes	No	No	No	Yes	No	1
2	3.00	0.30	9.00	0.09	1	Yes	No	Yes	No	Yes	No	No	No	Yes	No	1
3	2.60	0.80	6.76	0.64	1	Yes	No	Yes	No	Yes	No	No	No	Yes	No	1
4	4.90	0.40	24.01	0.16	1	Yes	No	Yes	No	Yes	No	No	No	Yes	No	1

5	0.70 No	-3.40 No	0.49 Yes	11.56 0	1	Yes	No	No	Yes	No	Yes	No
6	0.20 No	-3.60 No	0.04 Yes	12.96 0	1	Yes	No	No	Yes	No	Yes	No
7	-0.50 No	-1.30 No	0.25 Yes	1.69 0	1	Yes	No	No	Yes	No	Yes	No
8	3.20 No	-4.70 No	10.24 Yes	22.09 0	1	Yes	No	No	Yes	No	Yes	No
9	-1.70 No	-3.00 No	2.89 Yes	9.00 0	1	Yes	No	No	Yes	No	Yes	No
10	-0.20 Yes	0.20 No	0.04 Yes	0.04 0	0	No	Yes	Yes	No	No	No	No
11	-4.40 Yes	4.40 No	19.36 Yes	19.36 0	0	No	Yes	Yes	No	No	No	No
12	0.00 Yes	1.40 No	0.00 Yes	1.96 0	0	No	Yes	Yes	No	No	No	No
13	-1.30 Yes	2.70 No	1.69 Yes	7.29 0	0	No	Yes	Yes	No	No	No	No
14	-2.00 Yes	3.30 No	4.00 Yes	10.89 0	0	No	Yes	Yes	No	No	No	No
15	-3.20 No	-1.90 Yes	10.24 No	3.61 1	0	No	Yes	No	Yes	No	No	Yes
16	-4.20 No	-2.10 Yes	17.64 No	4.41 1	0	No	Yes	No	Yes	No	No	Yes
17	-3.80 No	-3.30 Yes	14.44 No	10.89 1	0	No	Yes	No	Yes	No	No	Yes
18	-4.40 No	-1.40 Yes	19.36 No	1.96 1	0	No	Yes	No	Yes	No	No	Yes
19	-3.50 No	-0.40 Yes	12.25 No	0.16 1	0	No	Yes	No	Yes	No	No	Yes

	X1	X2	X1Square	X2Square	Y	C1	C2	C3	C4	R13	R14	R24	R23	NC1	NC2	NC-Class
0	5.00	3.60	25.00	12.96	1	Yes	No	Yes	No	Yes	No	No	No	Yes	No	1
1	3.70	1.20	13.69	1.44	1	Yes	No	Yes	No	Yes	No	No	No	Yes	No	1
2	2.30	-2.70	5.29	7.29	1	Yes	No	Yes	No	Yes	No	No	No	Yes	No	1
3	3.60	2.40	12.96	5.76	1	Yes	No	Yes	No	Yes	No	No	No	Yes	No	1
4	2.50	1.90	6.25	3.61	1	Yes	No	Yes	No	Yes	No	No	No	Yes	No	1
5	-0.90	-3.30	0.81	10.89	1	Yes	No	No	Yes	No	Yes	No	No	No	Yes	0
6	-2.70	-4.70	7.29	22.09	1	Yes	No	No	Yes	No	Yes	No	No	No	Yes	0
7	1.30	-3.30	1.69	10.89	1	Yes	No	No	Yes	No	Yes	No	No	No	Yes	0
8	1.90	-3.80	3.61	14.44	1	Yes	No	No	Yes	No	Yes	No	No	No	Yes	0
9	0.30	-4.50	0.09	20.25	1	Yes	No	No	Yes	No	Yes	No	No	No	Yes	0
10	-1.10	2.70	1.21	7.29	0	No	Yes	Yes	No	No	No	No	Yes	No	Yes	0
11	0.20	3.80	0.04	14.44	0	No	Yes	Yes	No	No	No	No	Yes	No	Yes	0
12	-2.50	1.60	6.25	2.56	0	No	Yes	Yes	No	No	No	No	Yes	No	Yes	0
13	-1.80	1.10	3.24	1.21	0	No	Yes	Yes	No	No	No	No	Yes	No	Yes	0
14	-2.30	4.20	5.29	17.64	0	No	Yes	Yes	No	No	No	No	Yes	No	Yes	0
15	-2.50	-1.30	6.25	1.69	0	No	Yes	No	Yes	No	No	Yes	No	Yes	No	1
16	-4.50	-2.60	20.25	6.76	0	No	Yes	No	Yes	No	No	Yes	No	Yes	No	1
17	-1.80	-1.40	3.24	1.96	0	No	Yes	No	Yes	No	No	Yes	No	Yes	No	1
18	-4.00	-2.20	16.00	4.84	0	No	Yes	No	Yes	No	No	Yes	No	Yes	No	1
19	-1.40	0.30	1.96	0.09	0	No	Yes	No	Yes	No	No	Yes	No	Yes	No	1





Experimental Observation:

Perceptron Algorithm:

The weights of the Perceptron algorithm are estimated from training data for different data set patterns [200,300,400], [500], [500,200,300,400] along with the accuracy between the Predicted and Desired Values.

Experimented for DP [200,300,400]

Weight Vector and Accuracy are calculated for all above patterns.

Weighted Vector for Perceptron for 200 - [0.26, -0.29, 0.31, -0.38]

Weighted Vector :::: 200 {'Weight': array([[25.3 , -18.3 , 24.69, -30.27]]), 'intercept_': array([5.]), 'accuracy': 0.975} Weighted Vector :::: 300 {'Weight': array([[5.5 , -1.4 , 16.07, -21.48]]), 'intercept_': array([4.]), 'accuracy': 0.9166666666666666} Weighted Vector :::: 400 {'Weight': array([[16. , -5.6 , 22.38, -25.54]]), 'intercept_': array([4.]), 'accuracy': 0.95}

Accuracy for Perceptron average = 0.97 + 0.91 + 0.95

Experimented for DP [500]

Weighted Vector :::: 500 {'Weight': array([[80.9 , -63.1 , 72.17, -69.57]]), 'intercept_': array([1.]), 'accuracy': 0.99}

Accuracy 0.99

Experiment for DP [500,200,300,400]

Weighted Vector :::: 500 {'Weight': array([[73.7 , -63.1 , 79.97, -71.63]]), 'intercept_': array([-7.]), 'accuracy': 1.0}

Weighted Vector :::: 200 {'Weight': array([[58.4 , -60. , 54.86, -94.68]]), 'intercept_': array([-6.]), 'accuracy': 0.925}

Weighted Vector :::: 300 {'Weight': array([[44.4 , -50.2 , 48.28, -50.34]]), 'intercept_': array([0.]), 'accuracy': 1.0}

Weighted Vector :::: 400 {'Weight': array([[64.3 , -72.7 , 80.57, -79.81]]), 'intercept_': array([-3.]), 'accuracy': 1.0}

Accuracy ---> (1 + .92 + 1 + 1)/4

W Vector --->

500 DP ----------------- - 73.7 , -63.1 , 79.97, -71.63

400 DP ------------------ 58.4 , -60. , 54.86, -94.68

300 DP------------------ 44.4 , -50.2 , 48.28, -50.34

200 DP ------------------- 64.3 , -72.7 , 80.57, -79.81

Error seen for small epoch -max-ite values

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_stochastic_gradient.py:570: ConvergenceWarning: Maximum number of iteration reached before convergence. Consider increasing max_iter to improve the fit.

  warnings.warn("Maximum number of iteration reached before "
```

Also tried with tuning the max-item value.
max_iter=50

Kernel SVM

The weights of the Kernel SVM algorithm are estimated from training data for different data set patterns [200,300,400], [500], [500,200,300,400].

Experiment with different Gamma and C Parameter Values to improve accuracy.

Initially accurate was low around .6, .7 and .8. and Later fined tuned to get > .9.

It was a challenge in getting the value to > .9 .

Some Trial methods used for fine tuning - "gamma" and "C" Parameter

```
#for g,c in [ (0.1,1), (0.1,2), (0.1,3), (0.1,4),(0.2,1), (0.2,2), (0.2,3), (0.2,4), (0.3,1), (0.3,2), (0.3,3), (0.3,4),

   #(0.2,2), (0.2,1), (0.2,3), (0.3,4),(0.3,2), (0.3,1), (0.3,3), (0.3,4 ), (0.4,1), (0.3,3), (0.4,4 ) , (0.7,1), (0.7,2), (0.7,3), (0.7,4),(0.8,1), (0.8,2), (0.8,3), (0.8,4)]:

   #for g,c in [ (0.2,2), (0.2,1), (0.2,3), (0.3,4),(0.3,2), (0.3,1), (0.3,3), (0.3,4 ), (0.4,1), (0.3,3), (0.4,4 ) ]:


   #for g,c in [ (0.7,1), (0.7,2), (0.7,3), (0.7,4),(0.8,1), (0.8,2), (0.8,3), (0.8,4), (1,1), (1,2), (1,3), (1,4) ]:
```

Experiment for DP [200,300,400]

Weight Vector and Accuracy are calculated for all above patterns.

Gamma and C Parameter

0.1, 5

```
Weighted Vector :::: 200 {'Weight': 0, 'intercept_': array([-0.23]), 'accuracy': 0.9166666666666666}
Weighted Vector :::: 300 {'Weight': 0, 'intercept_': array([-0.15]), 'accuracy': 0.825}
Weighted Vector :::: 400 {'Weight': 0, 'intercept_': array([-0.06]), 'accuracy': 0.875}
```

Kernel Intecept for 200 - 0.23,

Kernel Intecept for 300 - 0.15,

Kernel Intecept for 2400 - 0.06

Linear SVM

From the experiment it can be seen that Perceptron accuracy > Linear SVM > Kernel SVM.

The 'C' Parameter fined tuned for better accuracy for SVM Linear algorithm using the experiment approach of running the svm linear for different 'C' until optimal accuracy is achieved.

Weighted Vector for Linear SVM for 200 - [0.26, -0.29, 0.31, -0.38], intercept 0.02

Weighted Vector for Linear SVM for 300 - [0.01, -0.01, 0.07, -0.24], intercept 1.05

Weighted Vector for Linear SVM for 400 -[] 0.03, -0.03, 0.15, -0.23], intercept 0.31

C Parameter' = 0.025

The 'C' and "Gamma" Parameter fined tuned for better accuracy for SVM Linear algorithm using an experimental approach of running the svm kernel for different 'C' and 'gamma' until optimal accuracy is achieved.

Repeated the experiment for [500] and values tabulated.

Similarly for For Data set [500]

Accuracy for Linear SVM average = 1

Accuracy for Perceptron average = 0.99

Accuracy for Kernel SVM = 0.97

From the experiment it can be seen that Linear SVM > Perceptron accuracy > Kernel SVM.

The 'C' Parameter is fine tuned for better accuracy for SVM Linear algorithm using the experiment approach of running the svm linear for different 'C' until optimal accuracy is achieved.

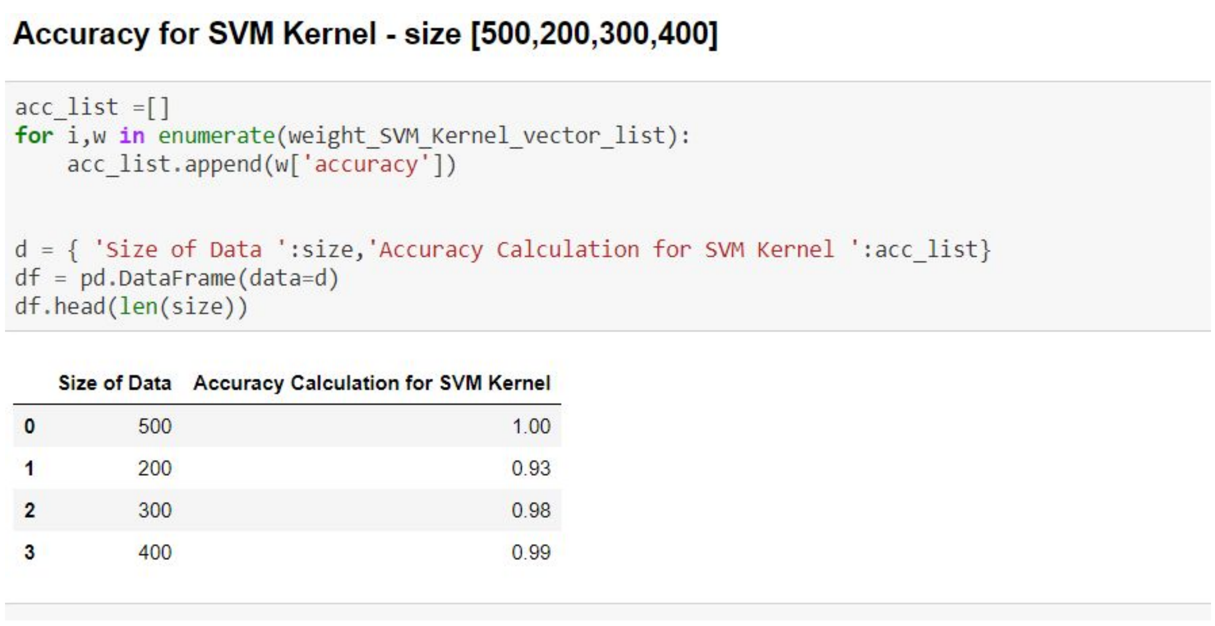
Based on the experiment done for linear and non-linear patterns, with fine tuning gamma and C parameters. It is observed that Linear SVM has performed better than Perceptron and Kernel SVM and overall all the algorithms provide good accuracy results.

The behavior of the model is very sensitive to the gamma parameter. If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting.

When `gamma` is very small, the model is too constrained and cannot capture the complexity or "shape" of the data. The region of influence of any selected support vector would include the whole training set resulting misclassification.

For some intermediate values of gamma got equally performing models when C becomes very large.

In practice though it might still be interesting to simplify the decision function with a lower value of C so as to favor models that use less memory and that are faster to predict.



Accuracy for SVM Kernel - size [500,200,300,400]

```
acc_list =[]
for i,w in enumerate(weight_SVM_Kernel_vector_list):
    acc_list.append(w['accuracy'])


d = { 'Size of Data ':size,'Accuracy Calculation for SVM Kernel ':acc_list}
df = pd.DataFrame(data=d)
df.head(len(size))
```

	Size of Data	Accuracy Calculation for SVM Kernel
0	500	1.00
1	200	0.93
2	300	0.98
3	400	0.99

Accuracy for SVM Linear - size [500,200,300,400]

```
acc_list =[]
for i,w in enumerate(weight_SVM_Linear_vector_list):
    acc_list.append(w['accuracy'])


d = { 'Size of Data ':size,'Accuracy Calculation for SVM Linear ':acc_list}
df = pd.DataFrame(data=d)
df.head(len(size))
```

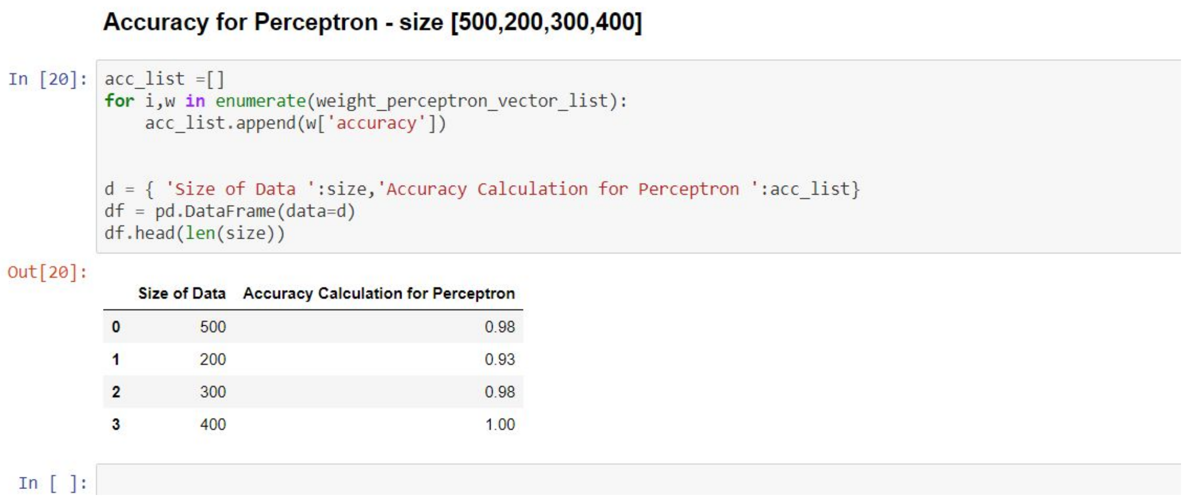
	Size of Data	Accuracy Calculation for SVM Linear
0	500	1.00
1	200	0.97
2	300	0.98
3	400	1.00

Accuracy for SVM Linear - size [500,200,300,400]

```
acc_list =[]
for i,w in enumerate(weight_SVM_Linear_vector_list):
    acc_list.append(w['accuracy'])


d = { 'Size of Data ':size,'Accuracy Calculation for SVM Linear ':acc_list}
df = pd.DataFrame(data=d)
df.head(len(size))
```

	Size of Data	Accuracy Calculation for SVM Linear
0	500	1.00
1	200	0.97
2	300	0.98
3	400	1.00



Accuracy for Perceptron - size [500,200,300,400]

In [20]:
```
acc_list =[]
for i,w in enumerate(weight_perceptron_vector_list):
    acc_list.append(w['accuracy'])


d = { 'Size of Data ':size,'Accuracy Calculation for Perceptron ':acc_list}
df = pd.DataFrame(data=d)
df.head(len(size))
```

Out[20]:

	Size of Data	Accuracy Calculation for Perceptron
0	500	0.98
1	200	0.93
2	300	0.98
3	400	1.00

In [ ]:

End of Analysis Report

Thank you