ELL-409

ASSIGNMENT-2 REPORT

-AVADHESH PRASAD

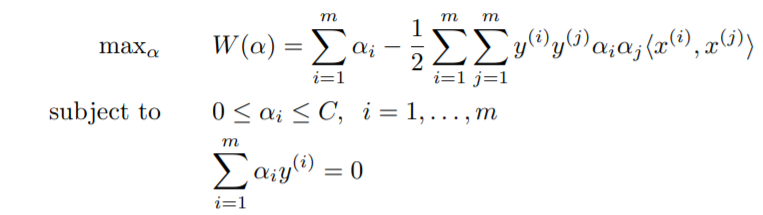
-2019MT60747

Objective: -

To experiment with the use of SVMs for both binary and multiclass classification problems, and understand the effects of varying various hyperparameters there in

**PART-1**

we have to carry out dual SVM for binary and multiclass classification problems.



For this, we will use standard python library LIBSVM.SVC and convex optimization package (CVX) for the following parts-

**Binary-classification: -**

**Method-1 LIBSVM**

**Let choose 3 class-pairs for the given data-**

|  |  |  |
| --- | --- | --- |
| Pair-id | class-numbers |  |
| 1 | **1,9** |  |
| 2 | **4,5** |  |
| 3 | **3,8** |  |

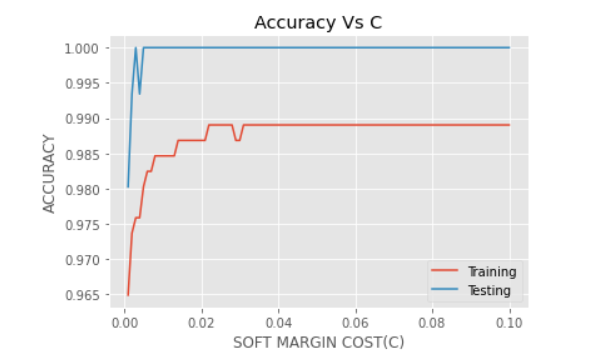
**For the above pairs, we will take 75% of the set for training and remaining 25 % of the set for validation.**

**Now we will tune hyperparameters and find out the best model for each pair for number of features 10 and 25 respectively.**

**PAIR-1 CLASS 1,9 Features-10**

**Hyper-parameter-tuning**

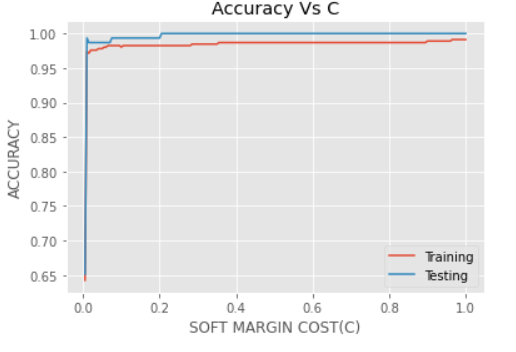
* First, we use linear kernel



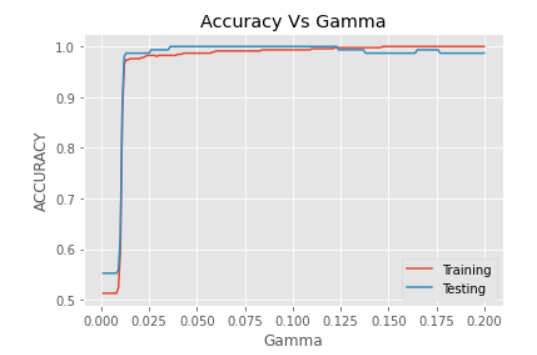
Here, we can see linear-kernel become constant for c>0.01 and c=0.01 is the best choice. Also, accuracy is very poor for low c and constant for high c. It is because if cost of penalty is low, we can misclassify few points, that will give us low accuracy.

Here, low c is underfitting and large c is overfitting case.

* Now, we choose polynomial-kernel



Here, accuracy is low for low c which is the same reasoning of the cost. Here, we can see polynomial-kernel has best training and testing accuracy for c=0.4 and now we will choose gamma using c=0.4

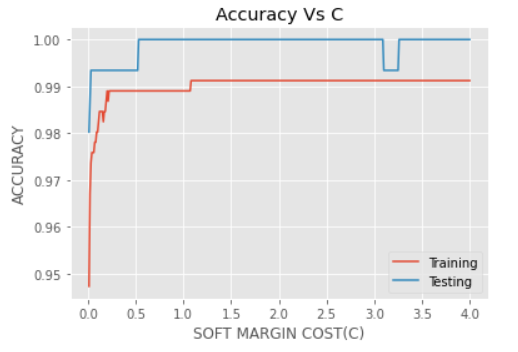


Here, the best gamma is 0.125.

We can observe gamma is low for low values of gamma then it increase and finally slightly decreases creating a peak.

Generally, gamma is approximately 1/features =0.1 which is same as gamma here.

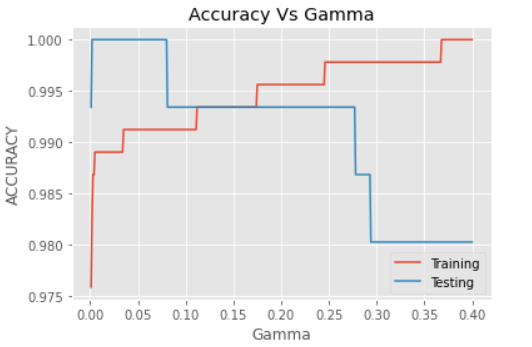
* Now, we will choose RBF kernel



Here, accuracy is very poor for low c and constant for high c. It is because if cost of penalty is low, we can misclassify few points, that will give us low accuracy.

Here, low c is underfitting and large c is overfitting case.

Here, we can see RBF-kernel has best training and testing accuracy for c=1.5 and now we will choose gamma using c=1.5



Here, the best gamma is 0.15 where both training and testing-error are optimal.

In this plot, we can observe gamma is low for low values of gamma then it increases and finally decreases gradually creating a peak.

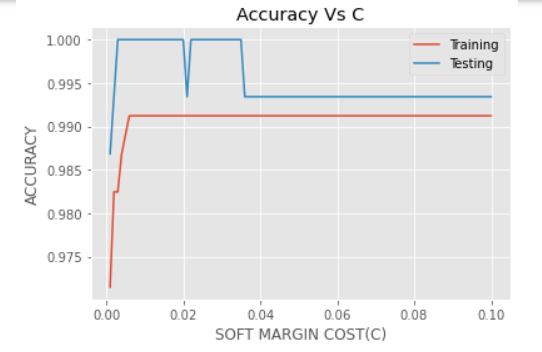
Generally, gamma is approximately 1/features =0.1 which is same as gamma here.

|  |  |  |
| --- | --- | --- |
| Kernel | cost(c) | gamma |
| linear | 0.01 | - |
| polynomial | 0.4 | 0.125 |
| rbf | 1.5 | 0.15 |

**PAIR-1 CLASS 1,9 Features-25**

**Hyper-parameter-tuning**

* First, we use linear kernel

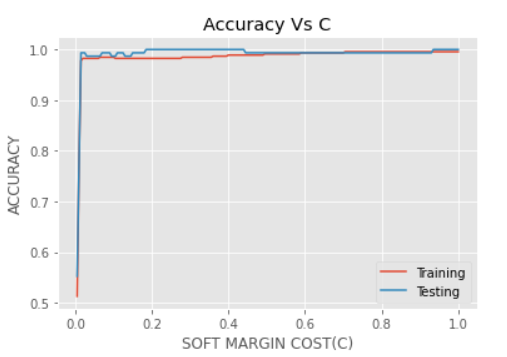


Here, we can see linear-kernel become constant for c>0.01 and c=0.01 is the best choice. Also, accuracy is very poor for low c and constant for high c. It is because if cost of penalty is low, we can misclassify few points, that will give us low accuracy.

Here, low c is underfitting and large c is overfitting case.

Here, c0 is almost same as the case of 10 features, because number of points are fixed, so independent of number of features, cost must also be same in optimum case.

* Now, we choose polynomial-kernel

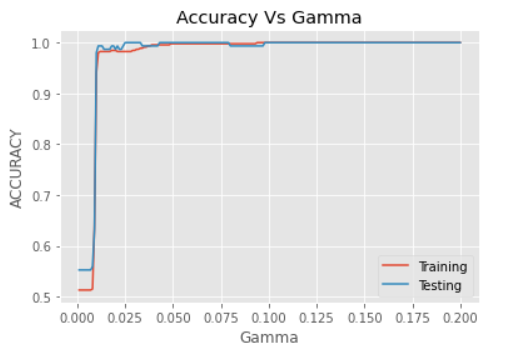


Here, we can see polynomial-kernel has best training and testing accuracy for c=0.4 and now we will choose gamma using c=0.4

Also, accuracy is very poor for low c and constant for high c. It is because if cost of penalty is low, we can misclassify few points, that will give us low accuracy.

Here, low c is underfitting and large c is overfitting case.

Here, c0 is almost same as the case of 10 features, because number of points are fixed, so independent of number of features, cost must also be same in optimum case.



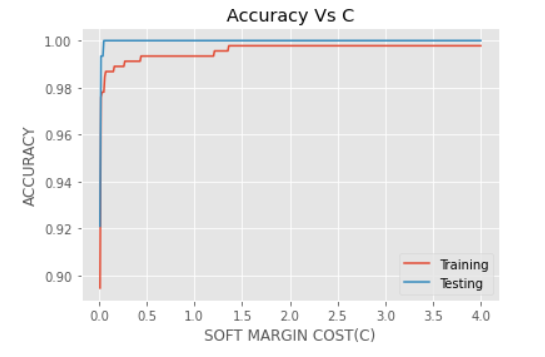
Here, the best gamma is 0.075.

We can observe gamma is low for low values of gamma then it increase and finally slightly decreases creating a peak.

Generally, gamma is approximately 1/features =0.04 which is same as gamma here.

Here, gamma is not same as 10 features case because optimal gamma which is 1/features is depending on the features.

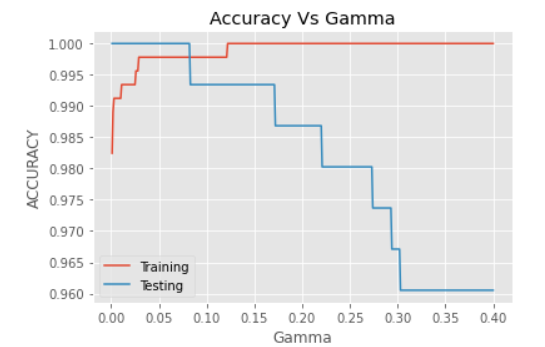
* Now, we will choose RBF kernel



Here, we can see RBF-kernel has best training and testing accuracy for c=1.5 and now we will choose gamma using c=1.5. Also, accuracy is very poor for low c and constant for high c. It is because if cost of penalty is low, we can misclassify few points, that will give us low accuracy.

Here, low c is underfitting and large c is overfitting case.

Here, c0 is almost same as the case of 10 features, because number of points are fixed, so independent of number of features, cost must also be same in optimum case.



Here, the best gamma is 0.05 where both training and testing-error are optimal. We can observe gamma is low for low values of gamma then it increases and finally slightly decreases creating a peak.

Generally, gamma is approximately 1/features =0.04 which is same as gamma here.

Here, gamma is not same as 10 features case because optimal gamma which is 1/features is depending on the features.

|  |  |  |
| --- | --- | --- |
| Kernel | cost(c) | gamma |
| linear | 0.01 | - |
| polynomial | 0.4 | 0.075 |
| rbf | 1.5 | 0.05 |

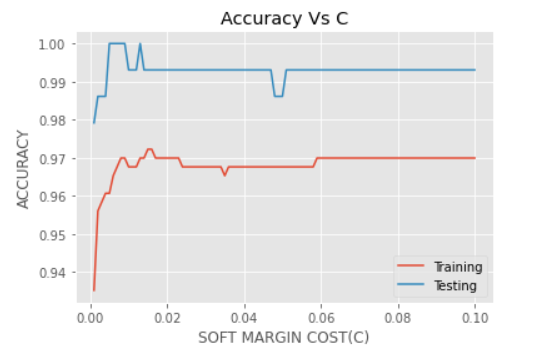
We can see that cost c is same for both features 10 and 25 but gamma changes. It is because gamma is 1/ (no. of features).

So, increasing features will decrease gamma.

**PAIR-2 CLASS 4,5 Features-10**

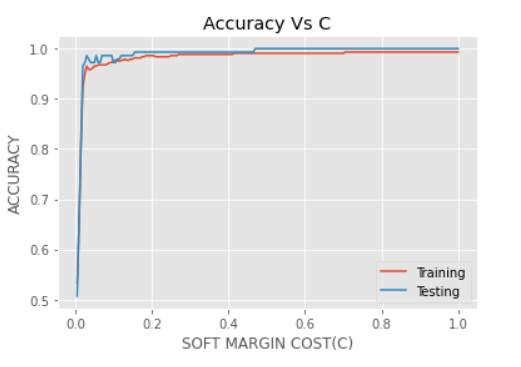
**Hyper-parameter-tuning**

* First, we use linear kernel

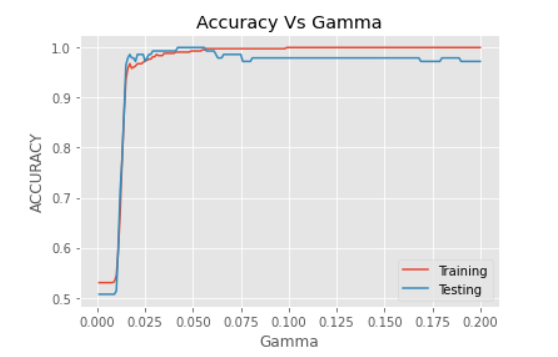


Here, we can see linear-kernel become constant for c>0.06 and c=0.06 is the best choice.

* Now, we choose polynomial-kernel

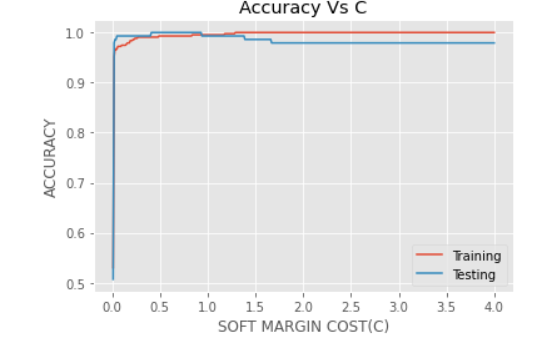


Here, we can see polynomial-kernel has best training and testing accuracy for c=0.6 and now we will choose gamma using c=0.6

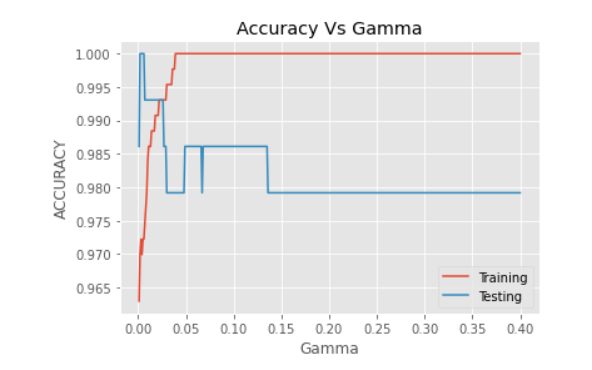


Here, the best gamma is 0.1

* Now, we will choose RBF kernel



Here, we can see RBF-kernel has best training and testing accuracy for c=2 and now we will choose gamma using c=2



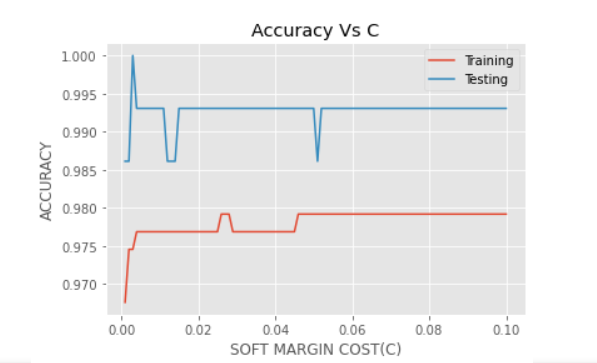
Here, the best gamma is 0.1 where both training and testing-error are optimal.

|  |  |  |
| --- | --- | --- |
| Kernel | cost(c) | gamma |
| linear | 0.06 | - |
| polynomial | 0.6 | 0.1 |
| rbf | 2 | 0.1 |

**PAIR-2 CLASS 4,5 Features-25**

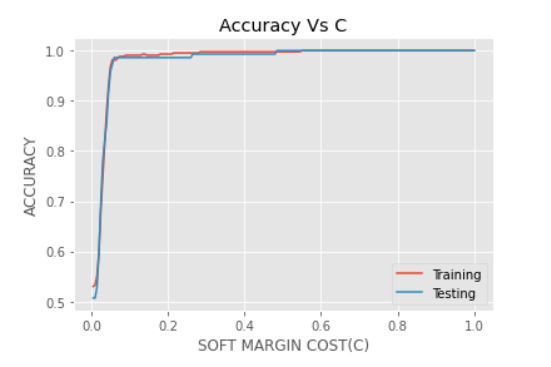
**Hyper-parameter-tuning**

* First, we use linear kernel

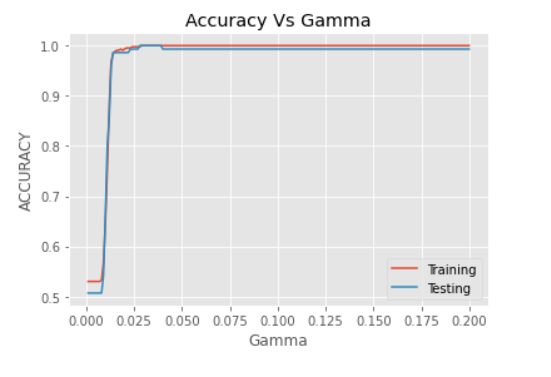


Here, we can see linear-kernel become constant for c>0.06 and c=0.06 is the best choice.

* Now, we choose polynomial-kernel

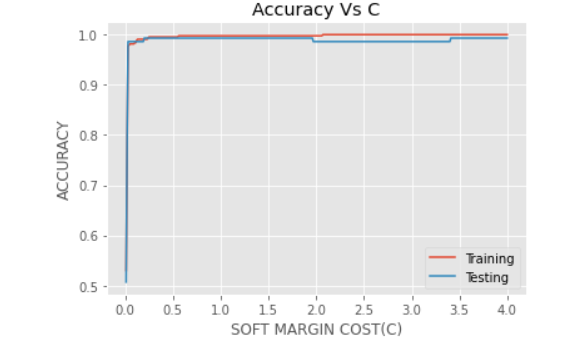


Here, we can see polynomial-kernel has best training and testing accuracy for c=0.6 and now we will choose gamma using c=0.6

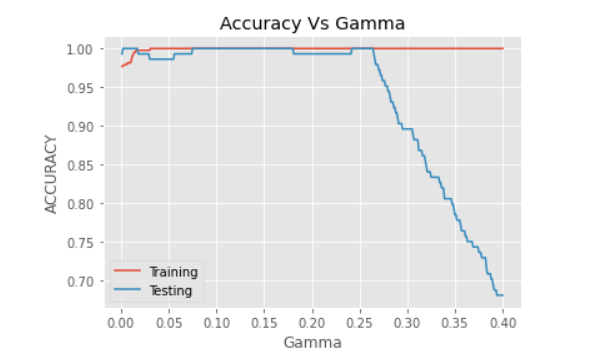


Here, the best gamma is 0.025

* Now, we will choose RBF kernel



Here, we can see RBF-kernel has best training and testing accuracy for c=2 and now we will choose gamma using c=2



Here, the best gamma is 0.08 where both training and testing-error are optimal.

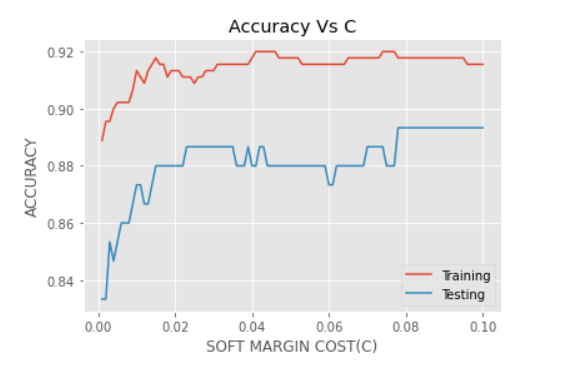
|  |  |  |
| --- | --- | --- |
| Kernel | cost(c) | gamma |
| linear | 0.06 | - |
| polynomial | 0.6 | 0.025 |
| rbf | 2 | 0.08 |

We can see that cost c is same for both features 10 and 25 but gamma changes. It is because gamma is 1/ (no. of features). So, increasing features will decrease gamma.

**PAIR-3 CLASS 3,8 Features-10**

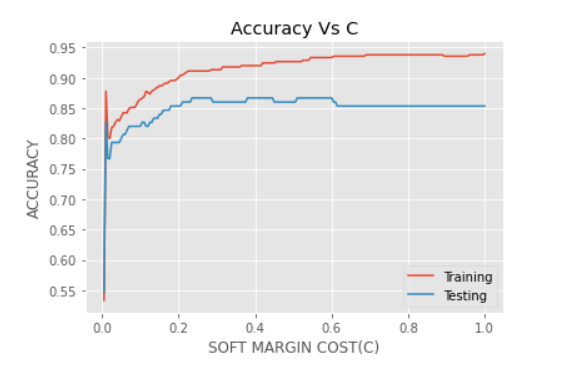
**Hyper-parameter-tuning**

* First, we use linear kernel

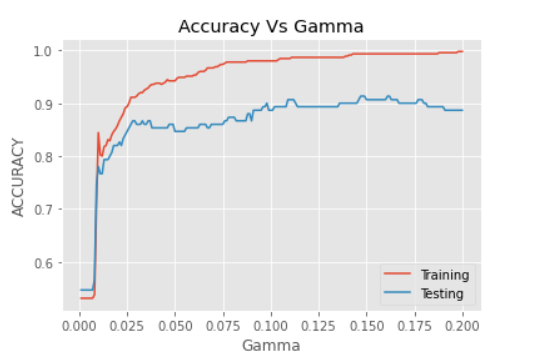


Here, we can see linear-kernel become constant for c>0.08 and c=0.08 is the best choice.

* Now, we choose polynomial-kernel

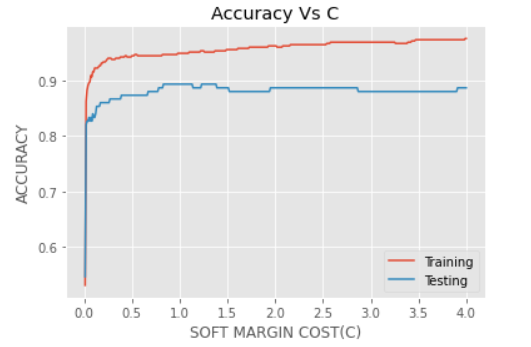


Here, we can see polynomial-kernel has best training and testing accuracy for c=1 and now we will choose gamma using c=1

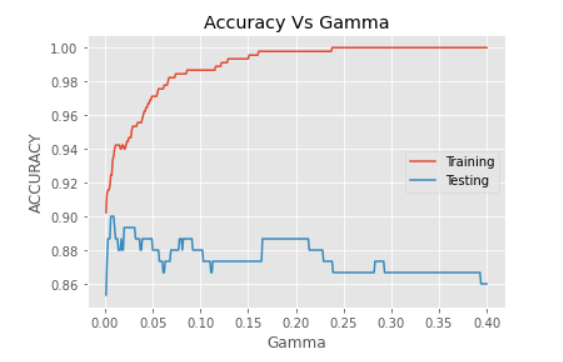


Here, the best gamma is 0.15

* Now, we will choose RBF kernel



Here, we can see RBF-kernel has best training and testing accuracy for c=2.5 and now we will choose gamma using c=2.5



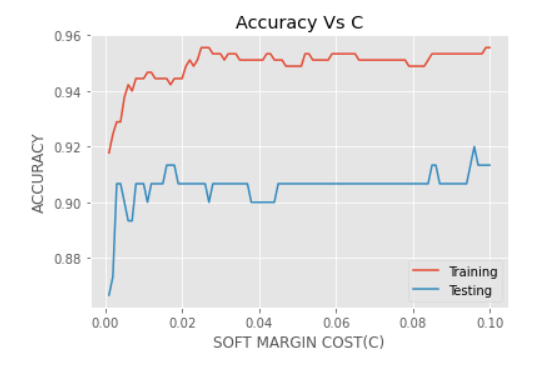
Here, the best gamma is 0.09 where both training and testing-error are optimal.

|  |  |  |
| --- | --- | --- |
| Kernel | cost(c) | gamma |
| linear | 0.08 | - |
| polynomial | 1 | 0.15 |
| rbf | 2.5 | 0.09 |

**PAIR-3 CLASS 3,8 Features-25**

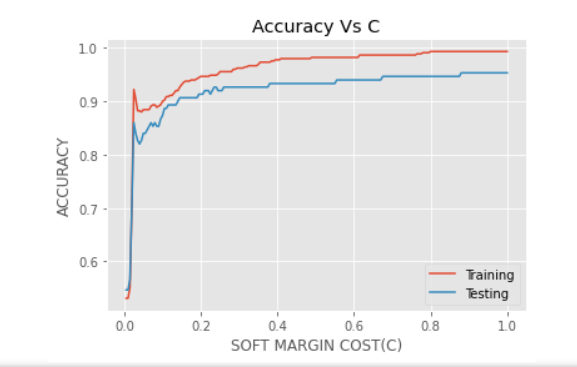
**Hyper-parameter-tuning**

* First, we use linear kernel

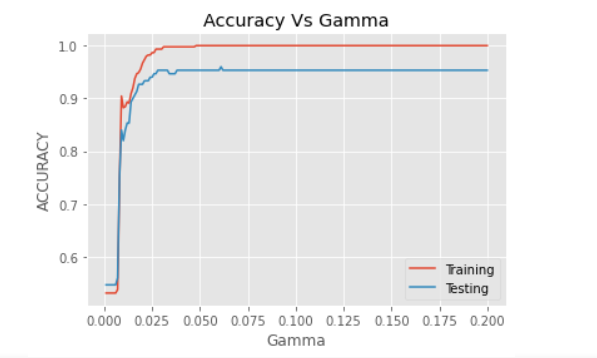


Here, we can see linear-kernel become almost constant for c>0.08 and c=0.08 is the best choice.

* Now, we choose polynomial-kernel

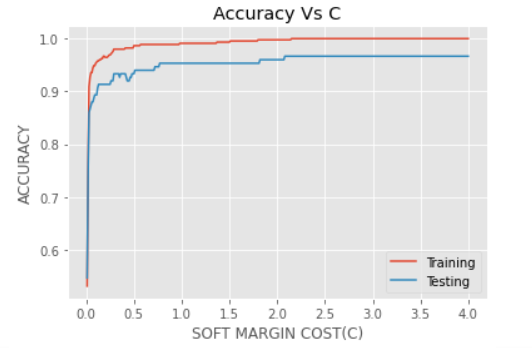


Here, we can see polynomial-kernel has best training and testing accuracy for c=1 and now we will choose gamma using c=1

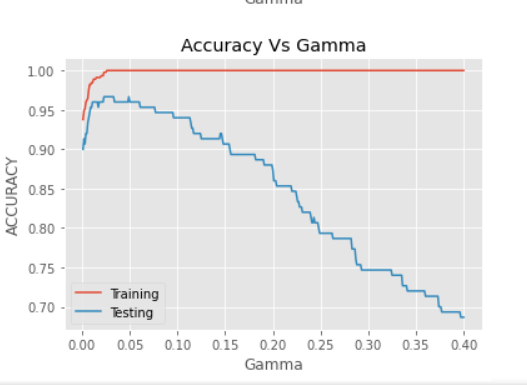


Here, the best gamma is 0.05

* Now, we will choose RBF kernel



Here, we can see RBF-kernel has best training and testing accuracy for c=2.5 and now we will choose gamma using c=2.5



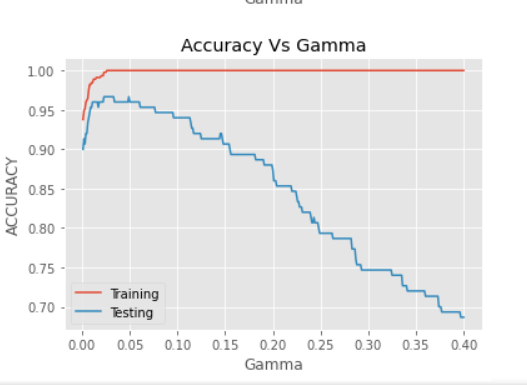
Here, the best gamma is 0.05 where both training and testing-error are optimal.

|  |  |  |
| --- | --- | --- |
| Kernel | cost(c) | gamma |
| linear | 0.08 | - |
| polynomial | 1 | 0.05 |
| rbf | 2.5 | 0.05 |

We can see that cost c is same for both features 10 and 25 but gamma changes. It is because gamma is 1/ (no. of features).

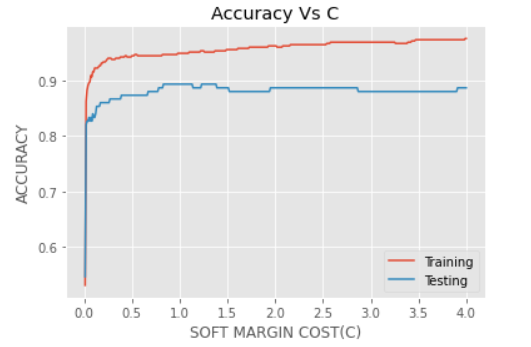
So, increasing features will decrease gamma.

**Case of underfitting and overfitting**



Here, we can see that first the accuracy is increasing for both training and testing data and then accuracy started to decrease as gamma increased after gamma=0.05. So, we can say that gamma=0.05 is best-fit and before that it is underfit and after that it is overfit.

Now, we will see another case.



Here, we can see accuracy at c=2.5 is maximum but after that it decreases and before it also it decreases. So, c=2.5 is best fit situation and after that is overfit as accuracy for training is increasing and before that it is underfitting.

**Observations: -**

* **With increase in C, first accuracy increases then decreases .**
* **Gamma is approximately 1/ (# features) at its optimum value. So, it differs as number of features changes.**
* **C is independent of number of features**
* **Polynomial and gaussian has high accuracy than linear because they can classify more rigorous shape due to their non-linearity because for non-linear classified data like centered data, we cannot classify using linear kernel.**

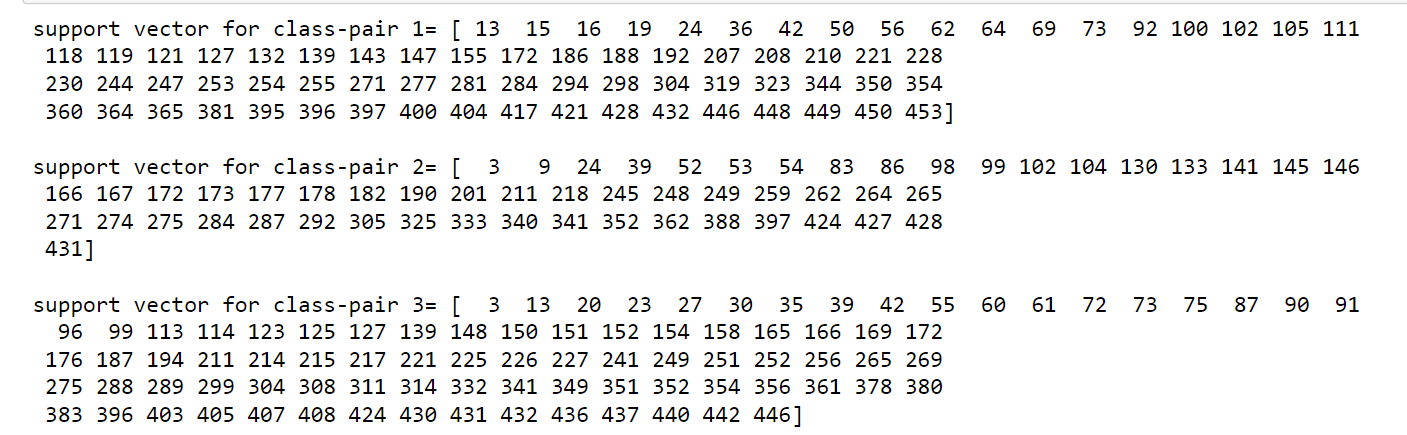
**Accuracy and time for tuned hyperparameter in LIBSVM**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | PAIR | | 1,9 | PAIR | 4,5 | pair | 3,8 |
| features |  | **10** | **25** | | **10** | **25** | **10** | **25** |
| linear | **Time** | **0.0019** | **0.001** | | **0.0029** | **0.003** | **0.0049** | **0.003** |
|  | **Accuracy** | **1.0** | **1.0** | | **0.9930** | **0.99** | **0.8733** | **0.90** |
|  |  |  |  | |  |  |  |  |
| polynomial | **Time** | **0.0029** | **0.002** | | **0.003** | **0.003** | **0.0049** | **0.005** |
|  | **Accuracy** | **0.9934** | **1.0** | | **0.9791** | **0.99** | **0.8866** | **0.95** |
|  |  |  |  | |  |  |  |  |
| rBF | **Time** | **0.0050** | **0.014** | | **0.01** | **0.014** | **0.01** | **0.016** |
|  | **Accuracy** | **0.9934** | **0.99** | | **0.9791** | **1.0** | **0.8866** | **0.90** |

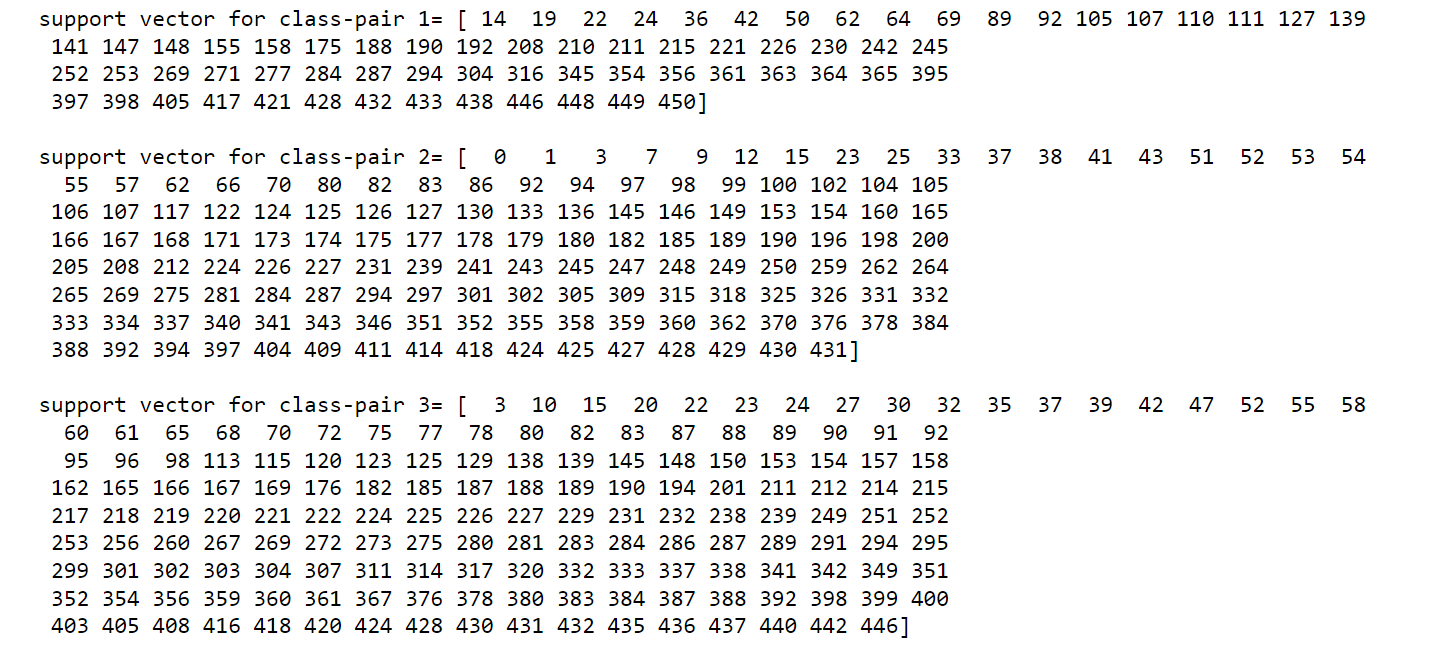
Here, accuracy is 0.8 in 3rd pair while 1 in starting 2 pairs. So, 3rd pair is not a good class.

**Support vector for LIBSVM**

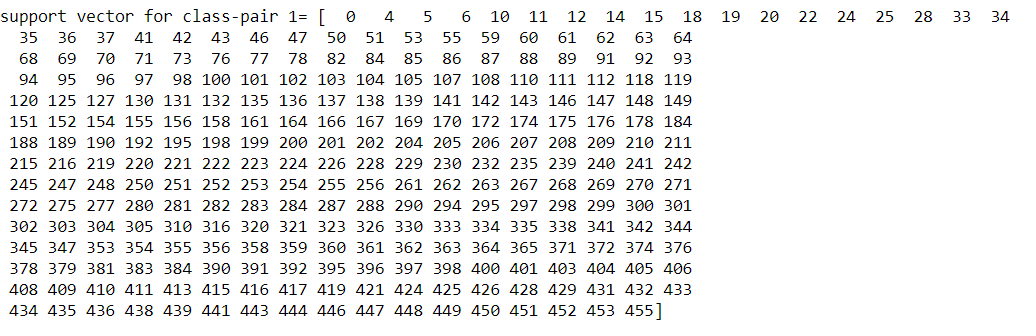
FOR LINEAR-KERNEL

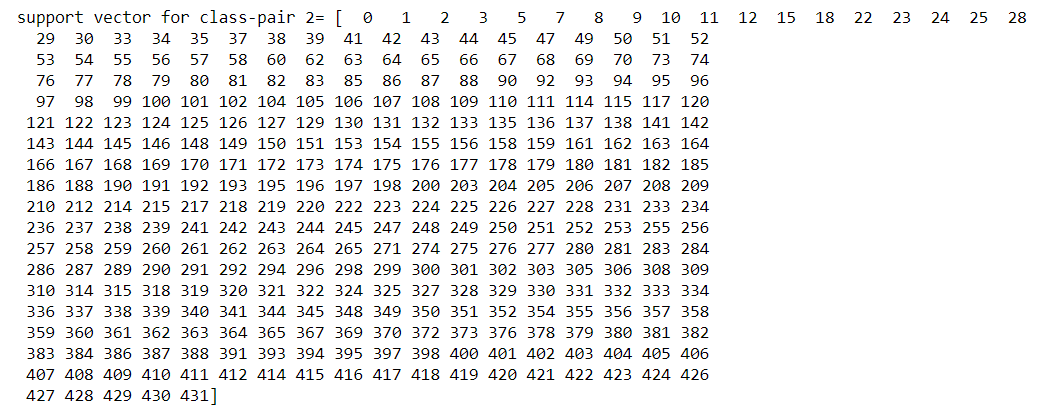


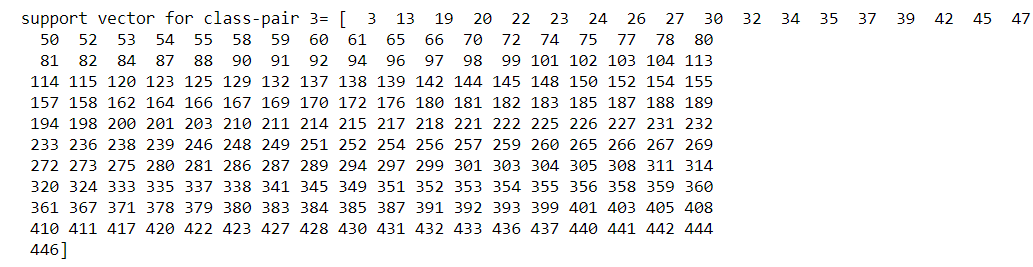
FOR POLYNOMIAL-KERNEL



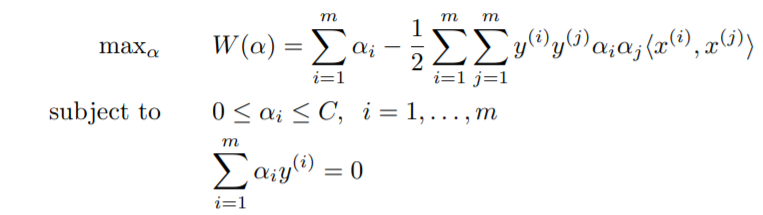
FOR RBF-KERNEL

****



****

**CVX OPTIMIZATION: -**



This is the condition we have to solve using convex optimization package.

As we know,

We can solve the dual svm problem using quadratic programming(qp) solver.

For, this we first find Lagrangian for the dual svm problem and then convert it into the convex optimization problem using KKT constraints.

Below is accuracy and time for above class pairs-

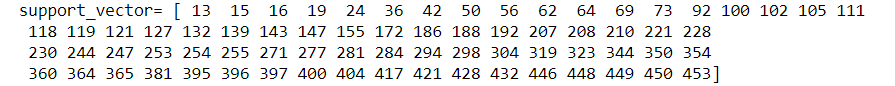
**Accuracy and time for tuned hyperparameter in CVX**

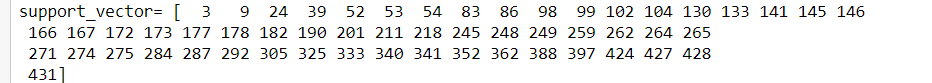
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | PAIR | | 1,9 | PAIR | 4,5 | pair | 3,8 |
| features |  | **10** | **25** | | **10** | **25** | **10** | **25** |
| linear | **Time** | **0.50** | **0.52** | | **0.41** | **0.42** | **0.53** | **0.49** |
|  | **Accuracy** | **0.99** | **0.99** | | **0.98** | **0.97** | **0.78** | **0.84** |
|  |  |  |  | |  |  |  |  |
| polynomial | **Time** | **0.57** | **0.59** | | **0.56** | **0.57** | **0.61** | **0.68** |
|  | **Accuracy** | **0.97** | **0.99** | | **0.93** | **0.92** | **0.83** | **0.82** |
|  |  |  |  | |  |  |  |  |
| rBF | **Time** | **1.57** | **1.57** | | **1.39** | **1.44** | **1.56** | **1.65** |
|  | **Accuracy** | **0.93** | **0.94** | | **0.91** | **0.95** | **0.84** | **0.88** |

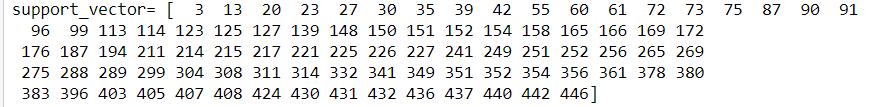
**Support-vectors**

FOR LINEAR-KERNEL

Support vectors for pairs 1,2,3 respectively is: -

****

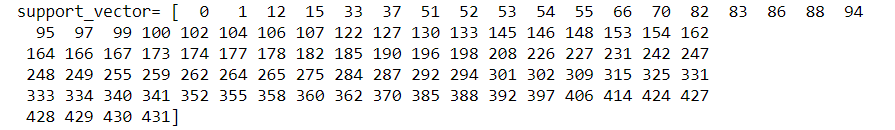
****

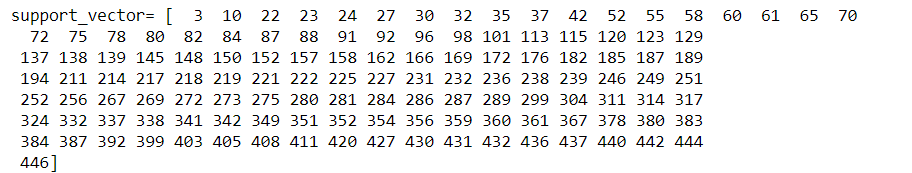
****

FOR POLYNOMIAL-KERNEL

Support vectors for pairs 1,2,3 respectively is: -

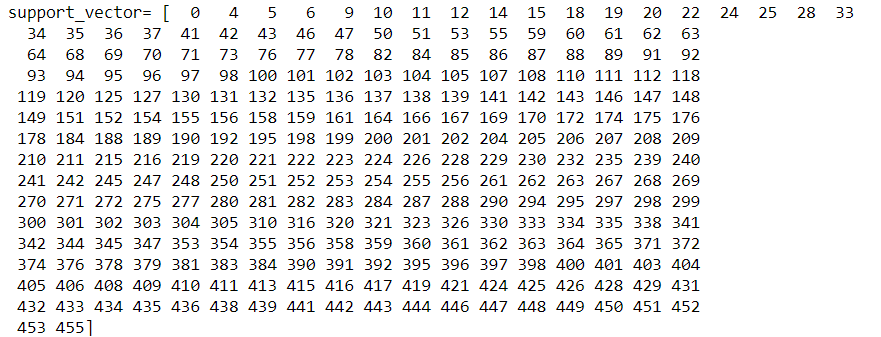
****

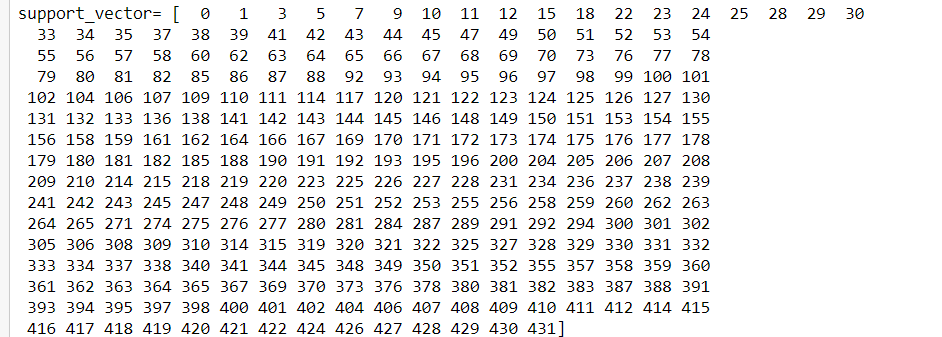
****

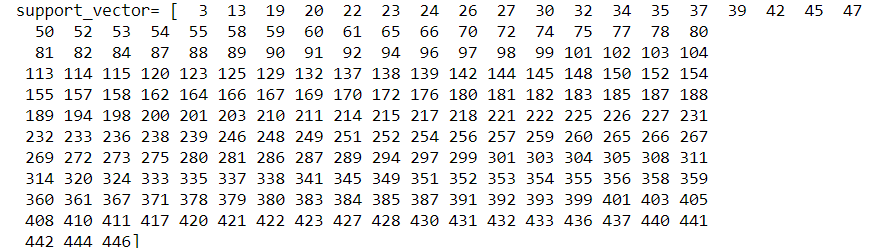
****

FOR RBF-KERNEL

Support vectors for pairs 1,2,3 respectively is: -

****

****

****

**Conclusion for support- vectors**

Here in above support vectors for LIBSVM and CVX, we can see number of support vector varies a little but all the support vectors are common in both.

So, we can say that cvx and libsvm gives almost same results but differ slightly as number of support vectors differ.

**Compare between number of features**

We can see in above results that on increasing number of features time for computation increases and accuracy also increases, it is because we are using more information, so it helps more in classification but it take time to comprehend.

**Compare between classes**

We can see in above results that class 1 and 2 gives accuracy up to 0.95 and class 3 gives output of accuracy 0.85. so, we can say class 1 and 2 have data that can be divided by line more easily while class 3 data may be concentric or of another style.

**Compare between kernels**

We can see that linear kernel have less accuracy than polynomial and gaussian kernel because these kernels can occupy more shapes.

**CASE OF MULTICLASS**

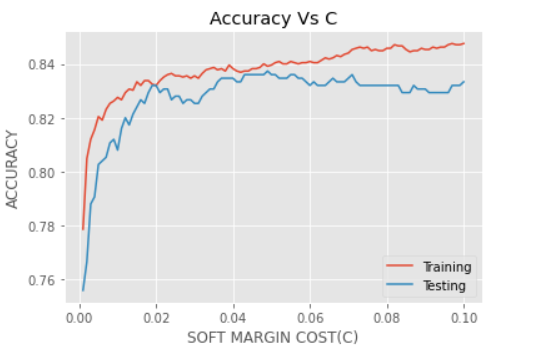
**Features-10**

Here, we will classify all labels using libsvm. Libsvm uses one to all method such that it takes one set as class 1 and all other as -1.

We will use validation for 75% training and 25% testing data.

**Hyper-parameter-tuning**

* First, we use linear kernel

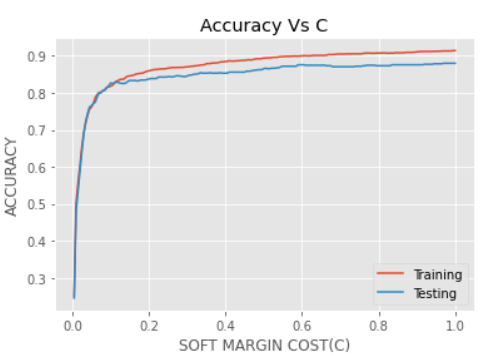


Here, we can see training and testing error for c=0.07 is the best choice. Also, accuracy is very poor for low c and constant for high c. It is because if cost of penalty is low, we can misclassify few points, that will give us low accuracy.

Here, low c is underfitting and large c is overfitting case.

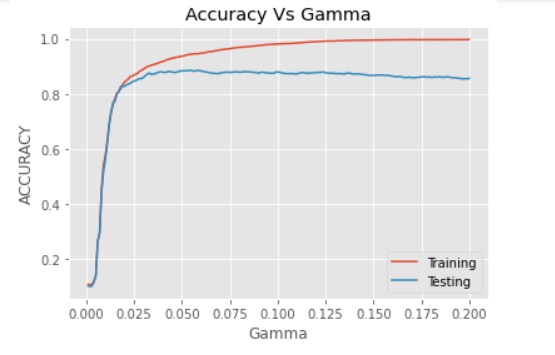
Accuracy is 0.84 lower than all other kernels.

* Now, we choose polynomial-kernel



Here, we can see polynomial-kernel has best training and testing accuracy for c=1 and now we will choose gamma using c=1.

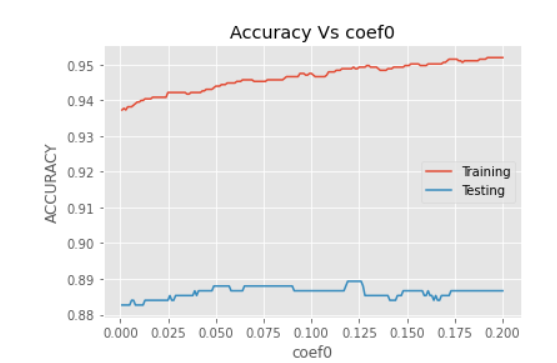
Here, accuracy is 0.91 which is better than linear kernel and here accuracy is quite good.



Here, the best gamma is 0.1.

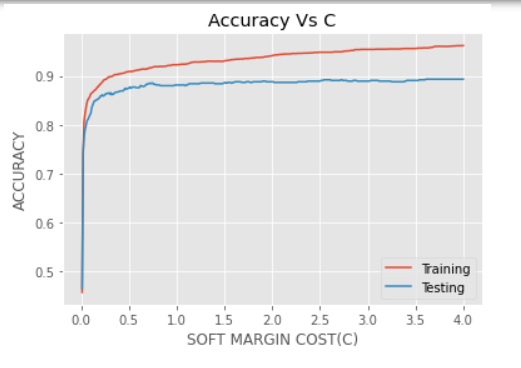
Here gamma is same as (1/#features) =0.1 which is the optimal case.

Now, we take c=1 and gamma=0.1

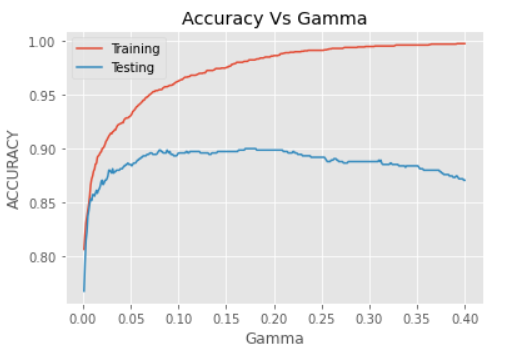


Here, the best coef0 is 0.12

* Now, we will choose RBF kernel



Here, we can see RBF-kernel has best training and testing accuracy for c=1 and now we will choose gamma using c=1



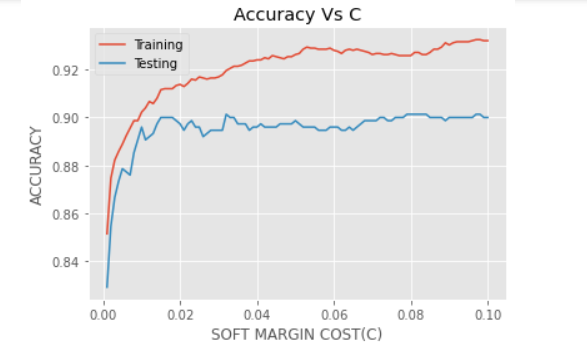
Here, the best gamma is 0.1 where both training and testing-error are optimal.

|  |  |  |  |
| --- | --- | --- | --- |
| Kernel | cost(c) | gamma | COEF0 |
| linear | 0.07 | - | - |
| polynomial | 1 | 0.1 | 0.12 |
| rbf | 1 | 0.1 | - |

**Features-25**

**Hyper-parameter-tuning**

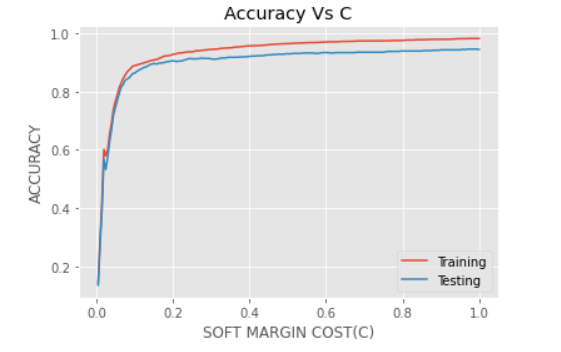
* First, we use linear kernel



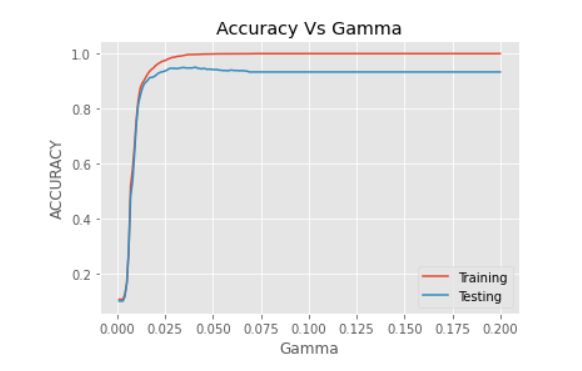
Here, we can see testing error is best for c=0.07. So, c=0.07 is the best choice.

Here, accuracy increases than 10 features case because we have more information.So, we can say accuracy increases by increasing features.

* Now, we choose polynomial-kernel

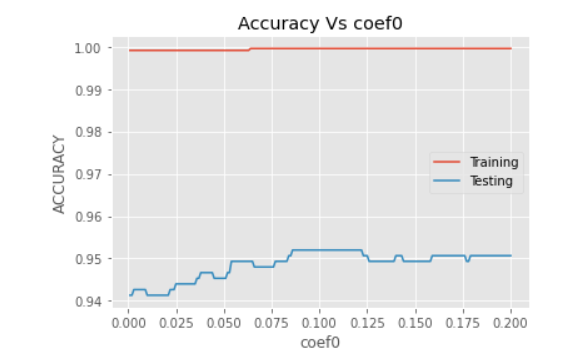


Here, we can see polynomial-kernel has best training and testing accuracy for c=1 and now we will choose gamma using c=1



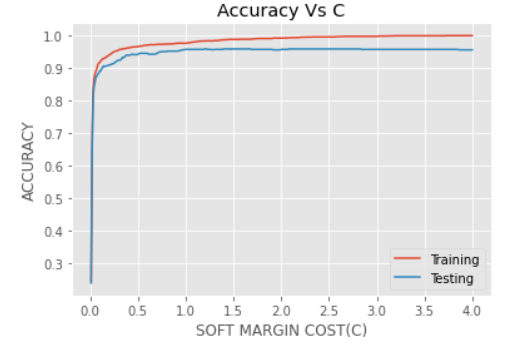
Here, the best gamma is 0.05

Now, we take c=1 and gamma=0.05 to tune coef0

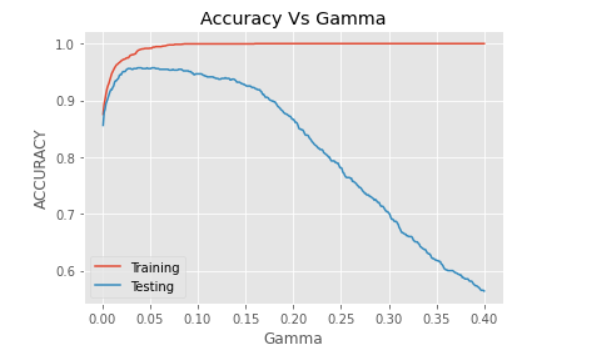


Here, the coef0 is 0.12

* Now, we will choose RBF kernel



Here, we can see RBF-kernel has best training and testing accuracy for c=1 and now we will choose gamma using c=1



Here, the best gamma is 0.04 where both training and testing-error are optimal.

|  |  |  |  |
| --- | --- | --- | --- |
| Kernel | cost(c) | gamma | COEF0 |
| linear | 0.07 | - | - |
| polynomal | 1 | 0.05 | 0.12 |
| rbf | 1 | 0.04 | - |

**LIBSVM ways for multiclass classification**

For Multiclass, LIBSVM.SVC use ***one to rest*** type of mechanism. i.e., it makes first data as 1 and all other data to -1 and classified them and iterates it for all classes.

**Accuracy and time for multiclass**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | |  |
| features |  | **10** | **25** | |
| linear | **Time** | **0.07** | **0.073** | |
|  | **Accuracy** | **0.832** | **0.90** | |
|  |  |  |  | |
| polynomial | **Time** | **0.084** | **0.1** | |
|  | **Accuracy** | **0.889** | **0.952** | |
|  |  |  |  | |
| rBF | **Time** | **0.23** | **0.45** | |
|  | **Accuracy** | **0.896** | **0.94** | |

**CONCLUSION: -**

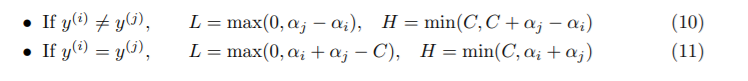
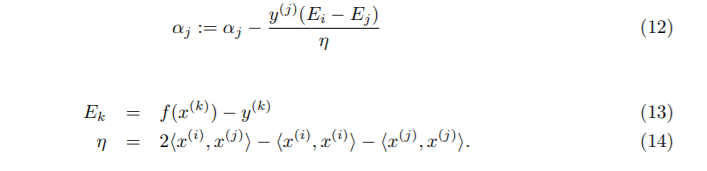
We get accuracy of 0.952 in case of polynomial kernel which is greater than case of linear kernel. And accuracy for 25 features is greater than 10 features because of the availability of remaining feature vectors to predict the data set.

**PART 1B**

Here, we will see the case of simplified-SMO which may converge or diverge depend on the data.

For our simplified version of SMO, we employ a much simpler heuristic. We simply iterate over all αi, i = 1, . . . m. If αi does not fulfil the KKT conditions to within some numerical tolerance, we select αj at random from the remaining m − 1 α’s and attempt to jointly optimize αi and αj.

First we will optimize αi and αj and find L AND H

Using all these equation a code been made for simpler version of smo.

Following are the output for binary classification for the given 3 pairs.

**Accuracy and time for tuned hyperparameter in simplified SMO**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | PAIR | | 1,9 | PAIR | 4,5 | pair | 3,8 |
| features |  | **10** | **25** | | **10** | **25** | **10** | **25** |
| linear | **Time** | **7.65** | **7.68** | | **8.6** | **16** | **12** | **22** |
|  | **Accuracy** | **0.99** | **0.99** | | **0.99** | **0.99** | **0.86** | **0.90** |
|  |  |  |  | |  |  |  |  |
| polynomial | **Time** | **28** | **30.2** | | **8.2** | **32** | **11** | **51** |
|  | **Accuracy** | **0.96** | **0.99** | | **0.93** | **0.97** | **0.72** | **0.89** |
|  |  |  |  | |  |  |  |  |
| rBF | **Time** | **76** | **123** | | **49** | **98** | **161** | **181** |
|  | **Accuracy** | **0.93** | **0.94** | | **0.91** | **0.95** | **0.84** | **0.87** |

**Observations: -**

* Time taken in case of SMO algorithm is almost 20~100 while in case of LIBSVM, it is 0.01 and in case of CVX it is 0.2.
* So, SMO is almost 1000 times slower than LIBSVM and CVX
* Here, we see the worst class pair 3 has accuracy 0.89 which is more than libsvm and cvx. So, we can say cvx is more accurate than CVX and LIBSVM.
* There is a trade off between accuracy and speed in the solvers. Increase the accuracy and speed is decreasing.

**Compare accuracy and time for CVX, SMO, LIBSVM**

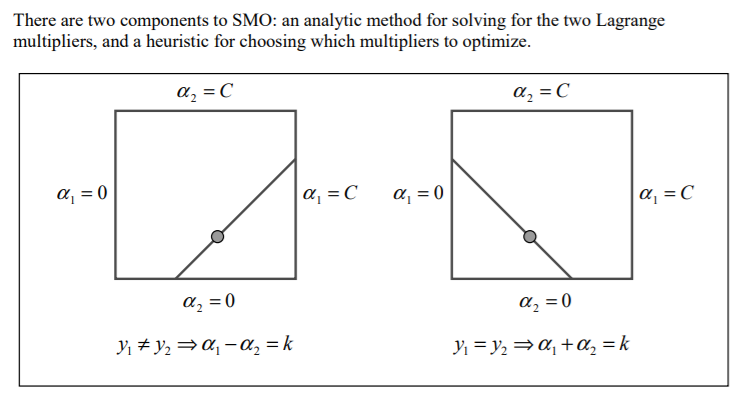
Here, we can see simplied-smo takes the most-time compare to cvx and libsvm but gives the best result

Accuracy: - SMO> LIBSVM>CVX

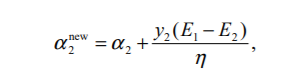
Speed: - LIBSVM > CVX > SMO

**PART 1C**

**Accuracy and time for tuned hyperparameter in FULL SMO**

****

OSUNA’S THEOREM

****

**Second choice heuristic: -**

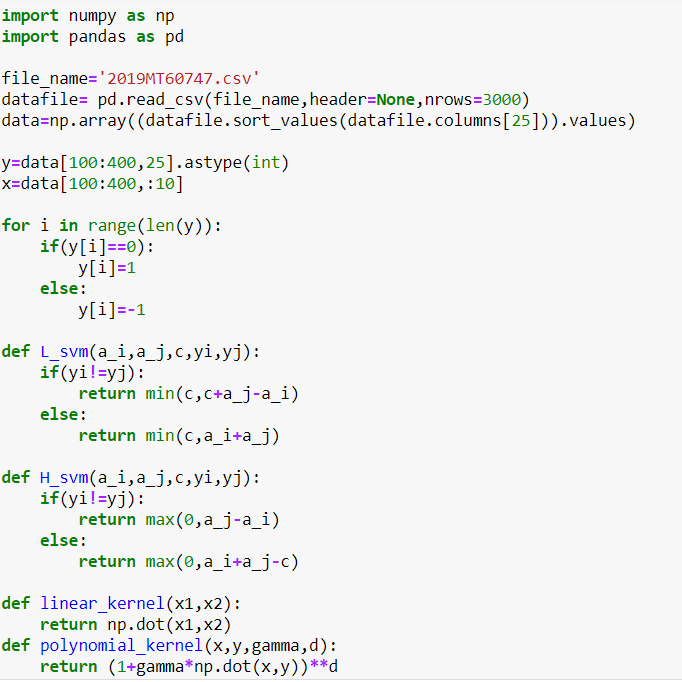
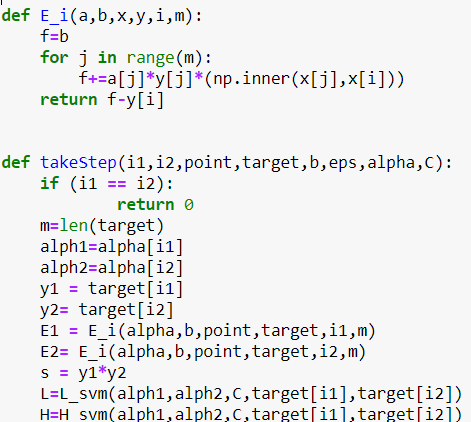
First smo chooses a Lagrange multiplier and then choose second Lagrange multiplier to maximize size of step taken during optimization.

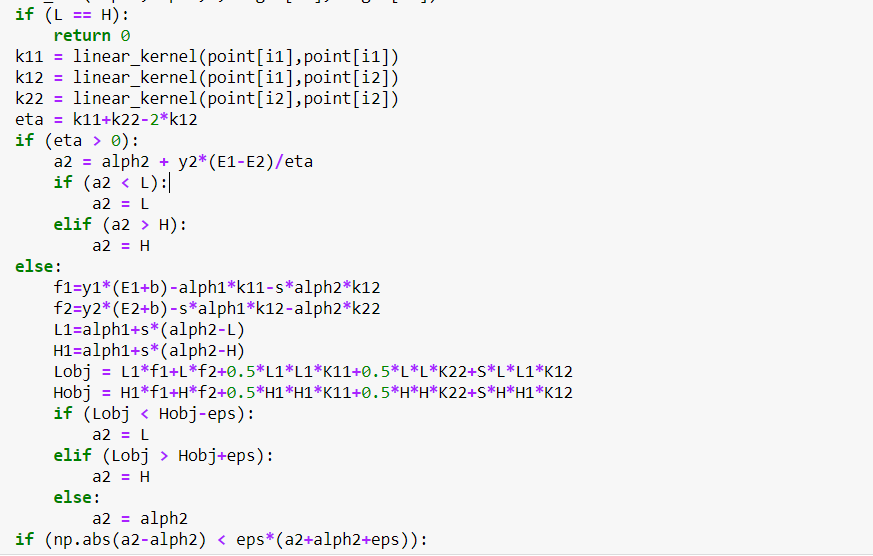
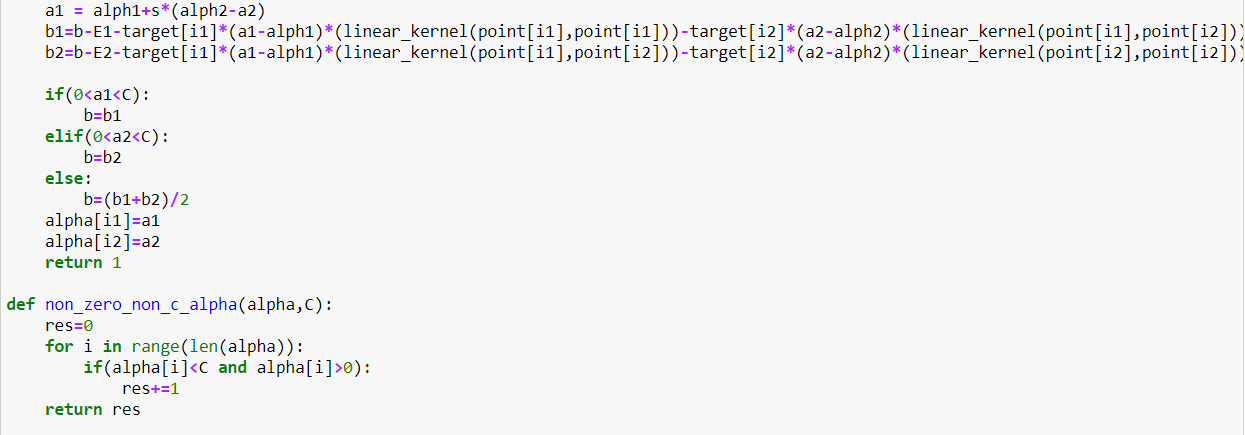
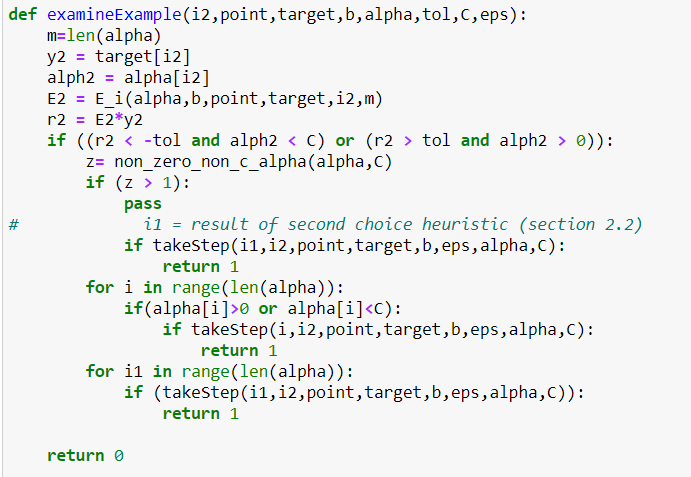
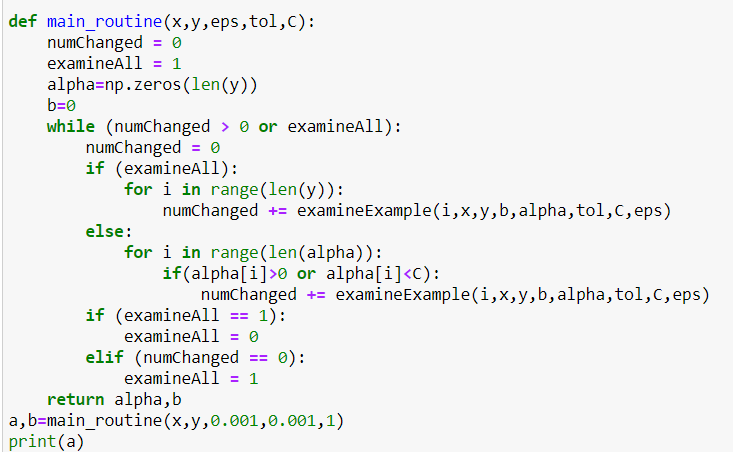
Because SMO always optimizes and alters two Lagrange multipliers at every step and at least one of the Lagrange multipliers violated the KKT conditions before the step, then each step will decrease the objective function according to Osuna’s theorem. Therefore, Convergence is thus guaranteed. In order to speed convergence, SMO uses heuristics to choose which two Lagrange multipliers to jointly optimize.

But it iterates over all data, so it is relatively slower than simplied-smo

**Speed: - LIBSVM > CVX > SIMPLIFIED\_SMO > SMO**

**CODE FOR FULL\_SMO: -**

**** ****

**** **** 

**PART 2**

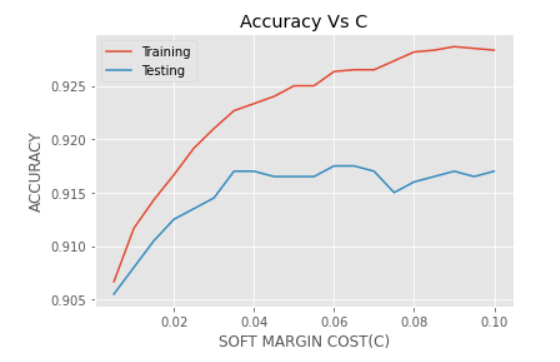
Given data set has 8000 instances in the same type of 25-dimensional feature space, and here we have to train a multiclass SVM model for predicting the labels on a target set of 2000 instances.

Here, we will use LIBSVM APPROACH because that is faster among all.

As data is too big, we have to choose the standard library.

First, we use cross-validation and breaks the set into 75% and 25% where 75% of the data is used for train and remaining 25% of the data required for validate.

**First, we will train for linear-kernel**

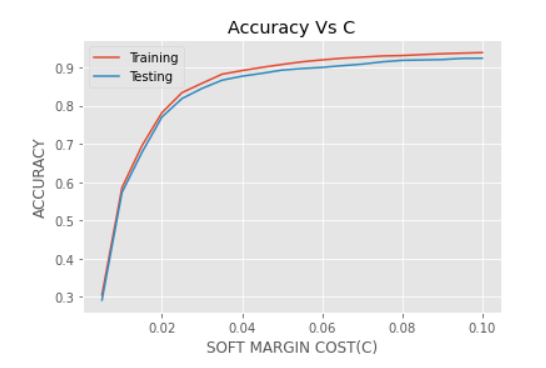


Here, we can see best testing accuracy is at c=0.06

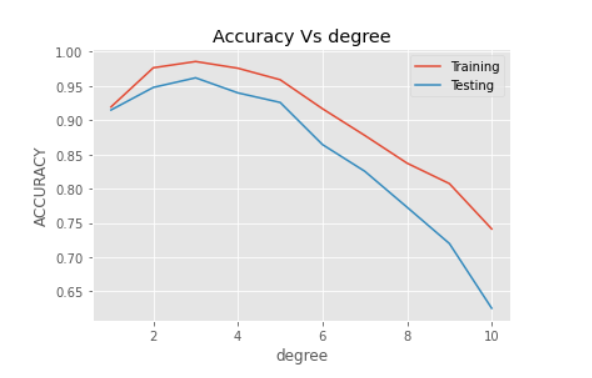
And corresponding accuracy is 0.92

**Now we will train for polynomial kernel**

Here, first we tune parameter c,

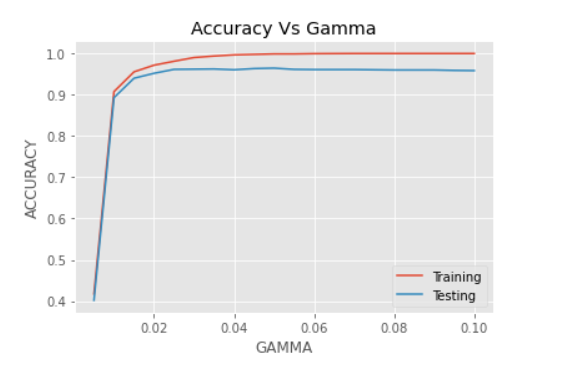


Here, we see that c is almost increasing so we will take terminal case c=0.01 and we will tune degree using this c.



Here, we can see degree 3 gives the best accuracy

Now we will tune gamma with varying data.



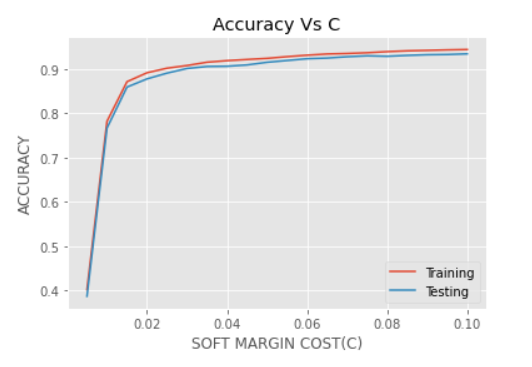
Here, we can see for gamma=0.1 it gives the best accuracy.

So, parameters are (degree, c, gamma) = (3,0.03,0.1)

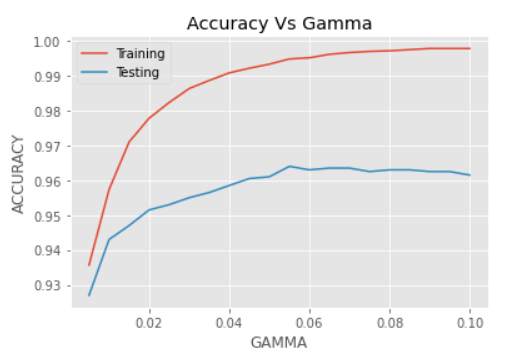
Accuracy for following data is: - 0.963 for (validation) and 0.99 for testing

**Now, we will train for RBF kernel**

First, we tune parameter c



Here, best c is 0.1 by above figure. Now, we will tune gamma using c=0.1



Here the best gamma is 0.05

So, parameters are (c, gamma) = (0.1,0.05)

Accuracy for following data is: - 0.963 for (validation) and 0.99 for testing

Putting both data model for polynomial and gaussian in Kaggle, both give accuracy 0.965 and 0.9625 respectively which is similar to our validation accuracy.

**END**