Support Vector Machines

Harshil Shah
Computer Science
Colorado State University

October 18, 2015

Contents

---

1 Introduction To SVM & SVM With No Bias Term

In machine learning, Support Vector Machines analyze data and recognize patterns with the use of associated learning algorithms. Apart from linear classification, SVMs can efficiently perform a non-linear classification by implicitly mapping their inputs into high-dimensional feature spaces using Kernel functions.[1, 2]

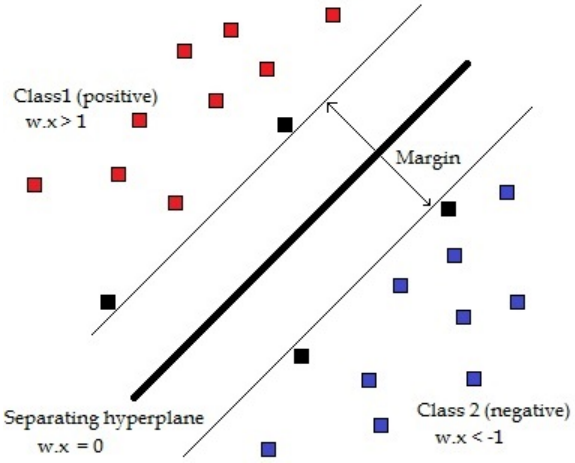

Figure 1: SVM Without Bias Term[4]

The margin of linear discriminant will be,

$$margin = \frac{\bar{w}^T(X_+ - X_-)}{2} \tag{1}$$

Suppose that, x_+ and x_- are at equal distance from decision boundary, then $w^T x_+ = a$ & $w^T x_- = -a$ will be the conditions with out including bias terms. Using this conditions, margin can be restructured as,

$$\bar{w}^T(X_+ - X_-) = 2a/||w|| \tag{2}$$

Where:
$\bar{w}$ = Unit Vector In Direction of w
$||w||$ = Norm of w
x_+&x_- = Support Vector Points

But if SVM fails to divide data examples into two classes clearly, the Soft Margin method will choose a hyperplane that splits the examples as cleanly as possible, with consideration of maximizing the margin too. The method includes non-negative slack variables, ξ_i, which represents the degree of misclassification of the example x_i.

$$\implies y_i(w^T x_i) \geq 1 - \xi_i$$

$$\xi_i \geq 0, i = 1, 2, ..n$$

Now the objective function ($\frac{||w||^2}{2}$) is then increased by a function which penalizes non-zero ξ_i. By considering the penalty function to be linear, for non separable example case, the optimization problem will look like as follows (according to equation : 2, to maximize margin, need to minimize $||w||$):

$$minimize(\frac{||w||^2}{2} + C\sum_{i=1}^{n}\xi_i)$$

with subject to

$$y_i(w^T x_i) \geq 1 - \xi_i)$$

$$\xi_i \geq 0, i = 1, 2, ..n$$

Where:
ξ_i = Slack Variable
C = Co-Efficient Parameter for Linear Equations

Above mentioned constraint can be solved using Lagrange Multipliers, with the objective of minimizing $||w||$. Lagrange equation to satisfy above conditions can be written as below.

$$\Lambda(w, \alpha, \xi) = \frac{||w||^2}{2} + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i(1 - \xi_i - y_i(w^T x_i)) - \sum_{i=1}^{n}\beta_i\xi_i,$$

$$\alpha_i, \beta_i \geq 0$$

Where:
α_i = Lagrangian multiplier for constraint function
β_i = Lagrangian multiplier for slack variables

Now according to Lagrange condition, constraints can be satisfied by solving $\nabla\Lambda(w, \alpha, \xi) = 0$. Here, $\nabla_x f(x)$ denotes differentiation of function $f(x)$ with respect to x.

Let's look at saddle point equations. Below saddle point equations are obtained using above mentioned Lagrangian equation only.

$$\frac{\partial\Lambda}{\partial w} = 0 \implies w = \sum_{i=1}^{n}\alpha_i y_i x_i$$

&

$$\frac{\partial\Lambda}{\partial\xi_i} = 0 \implies C = \alpha_i + \beta_i$$

Constraints optimization problem of minimizing $\frac{||w||^2}{2} + C\sum_{i=1}^{n}\xi_i$ with subject to $y_i(w^T x_i) \geq 1 - \xi_i$ is solved by below mentioned Karush–Kuhn–Tucker (KKT) conditions. [3]

$$\implies \nabla\Lambda(w, \alpha, \xi) = 0$$

$$Where: \Lambda(w, \alpha, \xi) = \frac{||w||^2}{2} + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i(1 - \xi_i - y_i(w^T x_i)) - \sum_{i=1}^{n}\beta_i\xi_i,$$

$$\implies (1 - \xi_i - y_i(w^T x_i)) \leq 0$$

$$\implies \alpha_i \geq 0$$

&

$$\implies \alpha_i(1 - \xi_i - y_i(w^T x_i)) = 0$$

The classification rule can be restructured in its unconstrained dual form. Dual depicts that the maximum-margin hyperplane is only a function of the subset of the data that lie on the margin (i.e. support vectors). By plugging saddle point conditions($w = \sum_{i=1}^{n}\alpha_i y_i x_i$ & $C = \alpha_i + \beta_i$) into Lagrangian equation, one can derive dual conditions. The dual will be,

$$W(\alpha) = \frac{1}{2}(\sum_{i=1}^{n}\alpha_i y_i x_i)^T(\sum_{j=1}^{n}\alpha_j y_j x_j) + \sum_{i=1}^{n}(\alpha_i + \beta_i)\xi_i + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}(\alpha_i\xi_i) - (\sum_{i=1}^{n}\alpha_i y_i x_i)^T(\sum_{j=1}^{n}\alpha_j y_j x_j) + \sum_{i=1}^{n}(\beta\xi_i)$$

$$\implies W(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}(\sum_{i=1}^{n}\alpha_i y_i x_i)^T(\sum_{j=1}^{n}\alpha_j y_j x_j)$$

$$\implies W(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

So the dual condition will be as follows:

$$maximize(\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j)$$

with subject to

$$0 \leq \alpha \leq C$$

.

Professor Asa, has derived formula for hard-margin SVM with inclusion of bias term. Primal condition will surely be different than the one that has been derived here.

$$minimize(\frac{||w||^2}{2})$$

with subject to

$$y_i(w^T x_i) \geq 1, i = 1, 2, 3, ..., n$$

As discussed earlier in this section, Dual merely depends on support vectors, so the Dual of hard-margin SVM with inclusion of bias and Dual of soft-margin SVM without inclusion of bias are almost the same. They both differ by only one constraint condition which is there in hard-margin SVM(i.e. $\sum_{i=1}^{n} \alpha_i y_i = 0$).

Now, in hard-margin SVM, SMO optimize two α s at a time. Same thing will also be true in soft-margin SVM, if one choose to apply SMO to solve SVM, as dual condition for both of them are same. So in the nutshell, just like hard-margin, **SMO will solve soft-margin SVM, by optimizing <u>two</u> alphas(α) at a time**.

2 Soft-margin SVM For Separable Data

As usual, SVM's primal conditions will be:

$$minimize(\frac{||w||^2}{2} + C\sum_{i=1}^{n} \xi_i)$$

with subject to

$$y_i(w^T x_i) \geq 1 - \xi_i)$$

$$\xi_i \geq 0, i = 1, 2, ..n$$

As soft-margin SVM does try to consider slack variables ξ_i, one can say that, soft margin tries to reduce effect of degree of misclassification (ξ_i) on decision boundary (always $\xi_i \geq 0$).
$\xi_i \leq 1 \implies$ Points are correctly classified but they exists between decision boundary and margin boundary
$\xi_i > 1 \implies$ Points are misclassified.

Now for linearly classified dataset, one can ensure that ξ_i will falls between 0 and 1 (i.e. $0 \leq \xi_i \leq 1$), but **it is <u>not</u> required that, ξ_i needs to be always zero.**

On contrary hard margin SVMs don't consider ξ_is. So hard-margin SVMs will always plot margin boundary such that there exists no example points between decision boundary and margin boundary. Margin in hard-margin SVM will always be significantly lower than the one in soft-margin SVM. There may reside some probability of misclassification in hard margin after training of algorithm is done. So in the nutshell, **it is <u>not</u> necessarily better to use a hard margin SVM over a soft-margin SVM**

3 Analysis of SVM

In upcoming sections, SVM has been analyzed based on two kernel function. Accuracy of respective kernels has been plotted on 3D graph as function of kernel function's free parameters.

3.1 Accuracy Based on Free Parameters of Kernel Functions

SCOP (Structural Classification of Proteins) dataset has been used for analyzing accuracy of SVM. Two basic kernels - Gaussian & Polynomial - are used. Accuracy is measured using the area under the ROC curve as a function of both the soft-margin parameter of the SVM, and the free parameter of the kernel functions. Kernel functions of both Gaussian and Polynomial are as shown in below equations (consider : equations 3, 4). Accuracy is measured in five-fold cross-validation. So in the nutshell, for each free parameters of kernel functions, for each soft-margin parameters & for each fold of the dataset, accuracy will be computed. So for each specific parameters combination, there will be 5 different accuracy for each fold. Mean of the accuracy will be taken as final accuracy for the dataset for a given combination of free parameters of kernel function and the soft-margin parameter.

$$K_{poly}(X, X^{'}) = (X^{T}X^{'} + 1)^{p} \tag{3}$$

$$K_{gauss}(X, X^{'}) = e^{(-\gamma||X-X'||^{2})} \tag{4}$$

3.1.1 Analysis of Polynomial Kernel

Now, to analyze accuracy of kernel function (equation : 3), implementation of algorithm is done in python. Data has been already parsed [6] and not been normalized. Scikit's in-built functions are used to create classifiers, 5-fold-validation function and other important functionalities. Python code regarding that is shown in appendix A .[5] As shown in code, for each combination of parameters, classifier has been defined and fitted in dataset. Using 5-fold-cross-validation by dividing dataset randomly, accuracy has been calculated & mean of the results is taken as final accuracy for given combination of parameters.

Soft-margin parameter - C - is taken on logarithmic scale. Kernel function's free parameter - degree of polynomial - is chosen in incremental order. Accuracy is calculated by taking mean of accuracy generated for each fold for specific parameters. 3D graph of Parameter C v/s Degree of Polynomial v/s Accuracy has been drawn as shown below. (consider figure: 2)

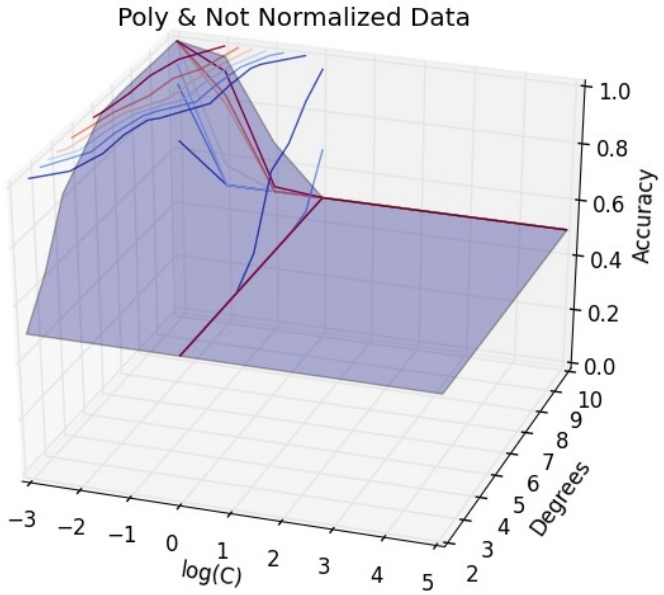

Figure 2: Accuracy By Using Polynomial Kernel On Non-Normalized Raw Data

Accuracy dataset for different combinations of free parameter of kernel and soft margin parameter is described in below table.

	Degrees								
C	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
10^{-3}	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
10^{-2}	0.705605100593	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
10^{-1}	0.820761035276	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
1.0	0.924155482315	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
10^{1}	0.988849429106	0.555812191017	0.5	0.5	0.5	0.5	0.5	0.5	0.5
10^{2}	0.99551685951	0.775578311023	0.5	0.5	0.5	0.5	0.5	0.5	0.5
10^{3}	0.997427158514	0.830714594147	0.5	0.5	0.5	0.5	0.5	0.5	0.5
10^{4}	0.996679650208	0.918117915231	0.508099438653	0.5	0.5	0.5	0.5	0.5	0.5
10^{5}	0.996845763165	0.960547071213	0.690221807062	0.5	0.5	0.5	0.5	0.5	0.5

Table 1: Accuracy Achieved By Using Polynomial Kernel on Raw Data

As it is observable, accuracy becomes stagnate after degree of 4 of polynomial, regardless of how variant free parameter(C) of soft-margin is. After certain number of C, for degree 2, 3 & 4, accuracy leaves its comfort zone and kept changing with the increasing change of C. It is not always true that, by increasing C, accuracy will always increase. Now for specific C, by increasing degree of polynomial, one can also say that, accuracy of system kept decreasing. But it seems that system's accuracy cannot go lower than its lower bound of the accuracy (here, 0.5). Reason behind such observable pattern is, classifier curve may over-fit the dataset.

3.1.2 Analysis of Gaussian Kernel

Again, to analyze accuracy of kernel function (equation : 4), implementation of algorithm is done in python. Data is not normalized. Scikit's in-built functions are used to achieve important functionalities. Python code regarding that is shown in appendix B [5]. As shown in code, procedure to calculate accuracy remains same as for polynomial, only parameters will be change to the input of classifier.

Soft-margin parameter - C - & Kernel function's free parameter - gamma - are taken on logarithmic scale. Calculation of accuracy remains same as polynomial's. 3D graph of Parameter C v/s Gamma v/s Accuracy has been drawn as shown below. (consider figure: 3)

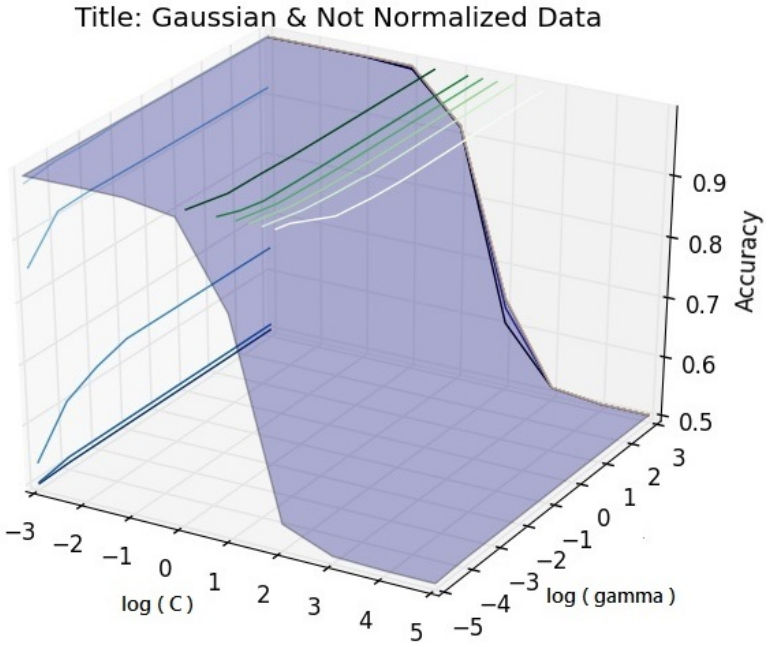

Figure 3: Accuracy By Using Gaussian Kernel On Non-Normalized Raw Data

Accuracy dataset for different combinations of free parameter of kernel and soft margin parameter is described in below table.

	Gamma							
C	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	**1.0**	10^{1}	10^{2}
10^{-3}	0.99860426	0.9993431	0.99852025	0.98600450	0.85388037	0.5352908	0.5023122	0.5
10^{-2}	0.99909688	0.99917898	0.99835796	0.99323901	0.9143738	0.60507738	0.51160117	0.50232558
10^{-1}	0.99917898	0.99917898	0.99802764	0.9925764	0.91735300	0.63040146	0.51160117	0.50232558
1.0	0.9993431	0.99901477	0.99917898	0.99720281	0.91678886	0.64556246	0.51160117	0.50232558
10^{1}	0.99917898	0.9993431	0.9993431	0.9980295	0.9164566	0.6455624	0.51160117	0.50232558
10^{2}	0.9993431	0.99950738	0.99950738	0.9980295	0.9164566	0.64556246	0.5116011	0.50232558
10^{3}	0.99950738	0.99950738	0.99950738	0.9980295	0.9164566	0.64556246	0.51160117	0.50232558
10^{4}	0.99967159	0.99950738	0.99950738	0.9980295	0.9164566	0.64556246	0.51160117	0.50232558
10^{5}	0.99967159	0.99950738	0.99950738	0.9980295	0.9164566	0.64556246	0.5116011	0.50232558

Table 2: Accuracy Achieved By Using Gaussian Kernel on Raw Data

Accuracy of system drastically falls down after certain value of gamma (here, 1.0). For all the values of gamma lesser than 1.0, accuracy of system remains quite similar (98-99%). One interesting thing is observed over here - unlike Polynomial kernel, with the increase of soft margin free parameter (*C*), by keeping gamma's value constant, accuracy does not change drastically (only fraction of accuracy is getting changed). One possible explanation of this kind of pattern can be, kernel function depends more on euclidean distance between two datasets rather than any other parameters.

3.2 Effect of Normalization & Optimal Values of Kernel Parameters

For normalizing the data, preprocessing functionality of scikit-learn was used. [6] Preprocessing was done in Python only, and code snippet regarding implementation is shown in appendix C. After normalization is done on input dataset, same methodology described in section 3.1.2 will be followed on normalized data too and relevant results will be derived.

3D graph of Parameter C v/s Gamma v/s Accuracy has been drawn as shown below. (consider figure: 4)

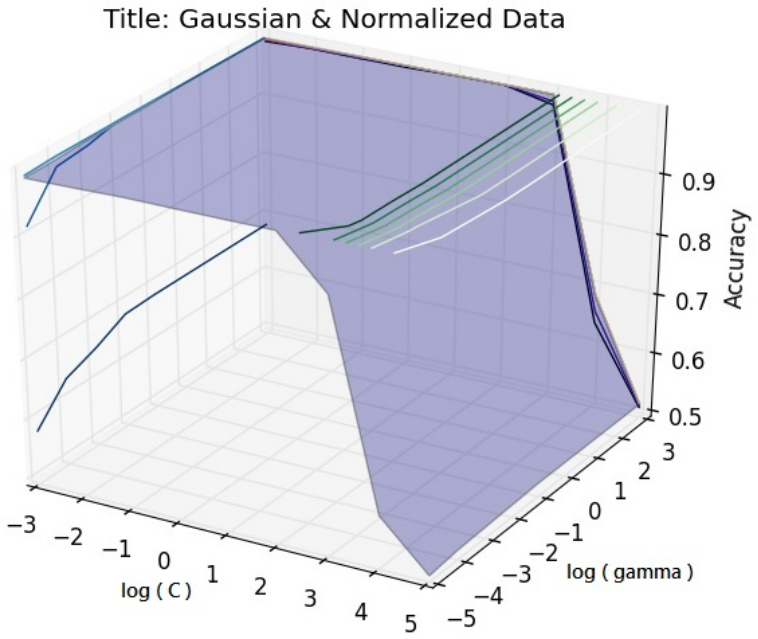

Figure 4: Accuracy By Using Gaussian Kernel On Normalized Data

Accuracy dataset for different combinations of free parameter of kernel and soft margin parameter is described in below table.

	Gamma							
C	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	**1.0**	10^{1}	10^{2}
10^{-3}	0.992588	0.9938423	0.9958949	0.9957307	0.9957307	0.9962233	0.9155093	0.5792433
10^{-2}	0.9931034	0.9962233	0.9957307	0.995730	0.998029	0.9985221	0.9816636	0.6360253
10^{-1}	0.9956486	0.99605	0.9957307	0.998029	0.9985221	0.9985221	0.9871879	0.655583
1.0	0.9958949	0.99605	0.998029	0.9985221	0.9983432	0.9988358	0.998753	0.680827
10^{1}	0.9958949	0.9977011	0.9985221	0.9983432	0.9985074	0.9988358	0.998753	0.6828099
10^{2}	0.9978653	0.9985221	0.9983432	0.9985074	0.9985074	0.9988358	0.998753	0.682809
10^{3}	0.998522	0.9983432	0.9985074	0.9985074	0.9985074	0.9988358	0.998753	0.6828099
10^{4}	0.9983432	0.9985074	0.9985074	0.9985074	0.9985074	0.9985074	0.998507	0.682809
10^{5}	0.998507480224	0.9985074	0.9985074	0.9985074	0.9985074	0.9988358	0.9987537	0.6828099

Table 3: Accuracy Achieved By Using Gaussian Kernel on Normalized Data

Unlike Gaussian on raw data, accuracy of this system does not drastically fall down. Accuracy of system tries to remain on higher level as much as possible and then falls down (here, accuracy falls down at gamma value of 100.0). For all the values of gamma lesser than 100, accuracy of system remains quite similar (98-99%). Furthermore, with the increase of soft margin free parameter (C), by keeping gamma's value constant, accuracy does not change that much (only fraction of accuracy is getting changed). One possible explanation of this kind of pattern can be, kernel function still depends more on euclidean distance between two datasets rather than any other parameters. As accuracy tries to remain immune as parameters are changing, Gaussian kernel should be applied to normalized dataset.

Let's now throw some insights at the best optimal values of kernel parameters. To get idea regarding optimal parameter values of kernel, internal cross validation (i.e. nested cross validation) should be performed on training dataset. For performing nested cross validation, use of functionality provided by scikit-learn will be convenient. Grid search cross validation will be used, instead of random search, to compute internal cross validation. Implementation of this functionality has been done in Python and code snippet regarding that is shown in appendix D.

Sufficient values of soft-margin free parameter C and kernel's free parameter γ will be initialized. Best optimal value will be returned from combination of this two set only. For raw dataset, five folds will be done randomly. According classifier to nest cross validation will be fit in training and classifier's attribute will depict best optimal parameter values. Same things will also be done on normalized dataset. Below shown table, represents optimal values of parameters for both dataset - Raw dataset & Normalized dataset.

Name	**C**	**Gamma**	**Accuracy**
Gaussian Applied To Raw Data	0.001	0.0001	0.9993431
Gaussian Applied To Normalized Data	1.0	1.0	0.9988358

Table 4: Optimal Kernel Parameters

References

[1] Machine Learning - Wikipedia
https://en.wikipedia.org/wiki/Machine_learning

[2] Support Vector Machine - Wikipedia
https://en.wikipedia.org/wiki/Support_vector_machine

[3] Karush–Kuhn–Tucker conditions
https://en.wikipedia.org/wiki/Karush_Kuhn_Tucker_conditions

[4] Image Courtesy :
http://edequity.org/szlag.php?q=svm-light&page=5

[5] SVM : Scikit Learn
http://scikit-learn.org/stable/modules/svm.html

[6] Parsing Data : Scikit Learn
http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_svmlight_file.html

Appendices

A Code Snippet of Polynomial Kernel Implementation

```
def compare_soft_and_kernel_poly(X, y, isNorm) :
   Cs = np.logspace(-3, 5, 9)
   degrees_dynamic = [2, 3, 4, 5, 6, 7, 8, 9, 10]

   c_data = []
   degrees_data = []
   accuracy_data = []
   folds = cross_validation.StratifiedKFold(y, 5, shuffle=True)

   for changing_C in range(len(Cs)) :
      inner = []
      for changing_degree in range(len(degrees_dynamic)) :
         classifier_dynamic = svm.SVC(kernel='poly', degree=degrees_dynamic[changing_degree],
               C=Cs[changing_C])
         print classifier_dynamic
         cross_validation_result_dynamic = cross_validation.cross_val_score(classifier_dynamic,
               X, y, cv=folds, scoring='roc_auc')
         print "Mean Value: ", np.mean(cross_validation_result_dynamic)
         inner.append((np.mean(cross_validation_result_dynamic)))
      accuracy_data.append(inner)
      inner = []

   degrees_data = degrees_dynamic
   c_data = Cs

   c_data, degrees_data = np.meshgrid(c_data, degrees_data)
   accuracy_data = np.asarray(accuracy_data)
   if isNorm == 0 :
      print "Data Is Not Normalized"
   else :
      print "Data Is Normalized"

   print "C_Data", c_data
   print "Degrees_Data", degrees_data
   print "Accuracy_Data", accuracy_data
   plot_3D_graph_of_accuracy(c_data, degrees_data, accuracy_data, 0, isNorm)
```

B Code Snippet of Gaussian Kernel Implementation

```
def compare_soft_and_kernel_gaussian(X, y, isNorm) :
   Cs = np.logspace(-3, 5, 9)
   gammas_dynamic = np.logspace(-5, 3, 9)
   folds = cross_validation.StratifiedKFold(y, 5, shuffle=True)

   c_data = []
   gamma_data = []
   accuracy_data = [

   for changing_C in range(len(Cs)) :
      inner = []
      for changing_gamma in range(len(gammas_dynamic)) :
         classifier_dynamic = svm.SVC(kernel='rbf', gamma=gammas_dynamic[changing_gamma],
               C=Cs[changing_C])
         print classifier_dynamic
         cross_validation_result_dynamic = cross_validation.cross_val_score(classifier_dynamic,
               X, y, cv=folds, scoring='roc_auc')
         print "Mean Value: ", np.mean(cross_validation_result_dynamic)

         inner.append((np.mean(cross_validation_result_dynamic)))
      accuracy_data.append(inner)
      inner = []

   c_data = Cs
   gamma_data = gammas_dynamic

   c_data, gamma_data = np.meshgrid(c_data, gamma_data)
   accuracy_data = np.asarray(accuracy_data)

   print "C_Data", c_data
   print "Gamma_Data", gamma_data
   print "Accuracy_Data", accuracy_data
   if isNorm == 0 :
      print "Data Is Not Normalized"
   else :
      print "Data Is Normalized"

   plot_3D_graph_of_accuracy(c_data, gamma_data, accuracy_data, 1, isNorm)
```

C Code Snippet of Normalizing DataSet

```
from sklearn import preprocessing

def generate_parsed_data() :
   file_to_read = "scop_motif.data"
   with open(file_to_read) as f:
      content = f.readlines()

   for i in range(len(content)) :
      if content[i].find("rest") != -1 :
         content[i] = content[i].replace(content[i][:(content[i].find("rest") + 4)], "-1.0")
      else :
         content[i] = content[i].replace(content[i][:(content[i].find("a.1.1.2") + 7)], "1.0")
   file_to_updated = "scop_motif_updated.data"
   updated = open(file_to_updated, "w")
   for iterate in range(len(content)) :
      updated.write(content[iterate])
   X, y = load_svmlight_file(file_to_updated)

   return X, y
if __name__=='__main__' :
   X, y = generate_parsed_data()

   print "Size Of X", X.shape
   print "Size Of Y", y.shape

   print "Before Normalization", X
   X_normalized = preprocessing.normalize(X)
   print "After Normalization", X_normalized
```

D Code Snippet of Nested Cross Validation Implementation

```
def nested_cross_validate(X, y, isNorm) :
   Cs = np.logspace(-3, 5, 9)
   gammas = np.logspace(-5, 3, 9)

   for_gaussian = {'C': Cs, 'gamma': gammas, 'kernel': ['rbf']}
   print "For_Gaussian", for_gaussian

   final_param_grid = [for_gaussian]
   print "Final_Param", final_param_grid

   folds = cross_validation.StratifiedKFold(y, 5, shuffle=True)
   classifier =
      GridSearchCV(estimator=svm.SVC(), param_grid=final_param_grid, cv=folds, scoring='roc_auc')

   classifier.fit(X, y)

   if isNorm == 0 :
      print "Data Is Not Normalized"
   else :
      print "Data Is Normalized"

   print classifier.best_score_
   print classifier.best_estimator_
   print classifier.best_params_
```