Machine Learning Lab (CS353) Report
Support Vector Machine with Gaussian kernel Algorithm

Team Members:

- Srikrishna Ganapati Yaji 181CO153
- Sai Nishanth K R 181CO145

Introduction

SVM-Introduction:

Support Vector Machine or SVM is one of the Supervised Learning algorithms, which is primarily used for Classification problems in Machine Learning. The goal of the SVM is to create the best line or decision boundary, called hyperplane, that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.
Important terms related to SVM are:

Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vectors. Since these vectors support the hyperplane, hence called a Support vector.

Margin: The concept of the margin of a separating line can be used to find the best separator line. The margin of the separator line is defined as the double of the shortest perpendicular distance between data points and the separator line.

Maximum Margin Line: The line with the highest value for margin is called the "maximum margin line" or the "optimal separating line". This line is otherwise called a "support vector machine"

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

Generally, SVM is of two types based on the dataset under consideration, namely:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM (Kernel SVM): Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data, and classifier used is called as Non-linear SVM classifier.

Kernel Functions and Kernel SVM: When the dataset under consideration is not linearly separable, the traditional SVM algorithm cannot be used to determine the hyperplane. So, SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of a kernel is to take data as input and transform it into a linear form.
The most commonly used kernel functions are:

- **Polynomial Kernel:**

  $k(x_i, x_j) = (x_i . x_j + 1)^d$ where d is the degree of the polynomial
- **Gaussian Kernel:**

  $k(x_i, x_j) = e^{-||x_i - x_j||^2/2\sigma^2}$ where σ^2 is the variance.
- **Gaussian Radial Basis Function:**

  $k(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}$ where γ is a hyperparameter.

It is a general-purpose kernel; used when there is no prior knowledge about the data.
The SVMs which use any of the kernel functions, to make the nonlinear data under consideration linearly separable and hence classifiable to classes, such SVMs are called Kernel SVMs.

SVM with Gaussian Kernel:

The SVM which uses Gaussian Kernel to classify the non-linear dataset under consideration is called Gaussian Kernel SVM.
The Gaussian kernel computed with a support vector is an exponentially decaying function in the input feature space, the maximum value of which is attained at the support vector and which decays uniformly in all directions around the support-vector, leading to hyper-spherical contours of the kernel function. The SVM classifier with the Gaussian kernel is simply a weighted linear combination of the kernel function computed between a data point and each of the support vectors.
The intuition behind applying the Gaussian kernel is, it gives the dot product between the mapped values of the support vectors in a particular dimension (say, m) to points in infinite dimension.
The parameter $1/2\sigma^2$ in the Gaussian Kernel is called the regularisation parameter, where σ^2 is the variance. The value σ^2 is a hyper parameter and should be considered based on the accuracy of the classifier with various values.

The Gaussian Radial Basis kernel is same as the Gaussian Kernel when γ in the Gaussian Radial Basis is considered equal to $1/2\sigma^2$.

SVM with Gaussian Kernel Algorithm:

Step 1: Calculate Euclidean distance between data points in different classes and identify the support vectors (s1,s2,..,sn)

Step 2: Use a bias to modify the support vectors

Step 3: Apply support vector equations and solve to find the α values.

$\alpha 1\, \Phi(s1)\,.\,\Phi(s1) + \alpha 2\, \Phi(s2)\,.\,\Phi(s1) + \ldots\ldots + \alpha n\, \Phi(sn)\,.\,\Phi(s1)$ = class label of s1…(so on)

till, $\alpha 1\, \Phi(s1)\,.\,\Phi(sn) + \alpha 2\, \Phi(s2)\,.\,\Phi(sn) + \ldots\ldots + \alpha n\, \Phi(sn)\,.\,\Phi(sn)$ = class label of sn,

where Φ is the kernel function. Since, the kernel function used is Gaussian Kernel,

$\Phi(si)\,.\,\Phi(sj) = e^{-(||si - sj||^2)/2\sigma^2}$.

Step 4: Apply α values in weight vector w˜ equation w˜= $\Sigma\, \alpha i\, si$.: y=wx+b produce maximum margin hyperplane.

Pros and Cons of Gaussian Kernel SVM:

Pros:

1. It is effective in the higher dimension.
2. Effective when the number of features are more than training examples.
3. Best algorithm when classes are separable
4. The hyperplane is affected by only the support vectors thus outliers have less impact.

Cons:

1. For a larger dataset, it requires a large amount of time to process.
2. Does not perform well in case of overlapped classes.
3. Selecting, appropriately hyperparameters of the SVM that will allow for sufficient generalization performance.

Applications of Gaussian Kernel SVM:

- Text Classification
- Face Detection
- Image classification
- Handwriting recognition
- Bioinformatics
- Protein fold and remote homology detection
- Generalized predictive control(GPC)
- Geo-spatial data-based applications
- Security-based applications

Numerical Implementation:

A sample dataset which shows whether a golf match will occur or not based on the climatic conditions of temperature and humidity is considered, on which SVM using Gaussian Kernel is applied.

Dataset:

Temperature	Humidity	Play
85.0	85.0	no
80.0	90.0	no
83.0	78.0	yes
70.0	96.0	yes
68.0	80.0	yes
65.0	70.0	no
64.0	65.0	yes
72.0	95.0	no
72.0	90.0	yes
71.0	80.0	no

The vectors belonging to the yes class are considered as y1, y2...yn
The vectors belonging to the no class are considered as n1, n2, n3..nn
The class 'yes' is taken as +1
The class 'no' is taken as -1
The distance matrix was computed where the distance metric followed was *'Euclidean Distance'*

	n1	n2	n3	n4	n5
y1	7.3	12.4	19.16	20.24	12.16
y2	18.6	11.7	26.476	2.236	16.031
y3	17.7	15.6	10.44	15.52	3

y4	29.0	29.7	5.099	31.048	16.552
y5	13.9	8.0	21.189	5	10.049

The minimum distance d_min from the distance matrix can be obtained as 2.236.

This implies the support vectors are:
Assuming bias (b) = 1 ,
Support Vectors:

s1 = y2 = [70, 96, 1]
s2 = n4 = [72, 95, 1]

Now,
Taking kernel function as Gaussian Kernel with variance (σ^2) = 1.00
Now, $1/(2\sigma^2)$ = 0.5

$\alpha 1\ (\Phi(s1)\,.\ \Phi(s1))\ +\ \alpha 2\ (\Phi(s2)\,.\ \Phi(s1))\ =\ +\ 1$ …..(equation - 1)
$\alpha 1\ (\Phi(s1)\,.\ \Phi(s2))\ +\ \alpha 2\ (\Phi(s2)\,.\ \Phi(s2))\ =\ -\ 1$ …..(equation - 2)

Upon finding $\Phi(si)\ \Phi(sj)$ for i = {1, 2} and j = {1, 2} equation -1 and equation - 2 reduces to:

$\alpha 1\ (1.0)\ +\ \alpha 2\ (0.082)\ =\ +\ 1$ …(i)
$\alpha 1\ (0.082)\ +\ \alpha 2\ (1.0)\ =\ -\ 1$ …(ii)

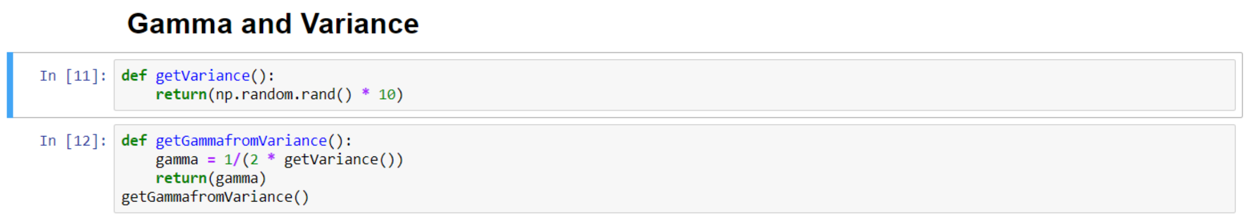
Upon solving (i) and (ii) we get:
$\alpha 1\ =\ 1.0894$
$\alpha 2\ =\ -\ 1.0894$

We know that the Gaussian kernel maps the finite dimensional vector (here dimension = 3) to infinite dimensional space.
So, extending the vectors using the formula:
$x\ \rightarrow e^{-x^2/2\sigma^2}(1\ ,\ \sqrt{(2/1!)}\ x\ ,\ \sqrt{(2^2/2!)}\ x^2$,....(till infinity)) [The taylor expansion]

We get the weight vector to be:
W~ = [1.54, -1.54, 106.91 …..(till infinity)]

IMPLEMENTATION

The Variance chosen in the Gaussian kernel is a hyper-parameter which is computed by defining the function *getVariance* as shown below. We know that the $\gamma = 1/2\sigma^2$ when the kernel is Gaussian kernel. The second function *getGammafromVariance* computes the γvalue given σ^2.



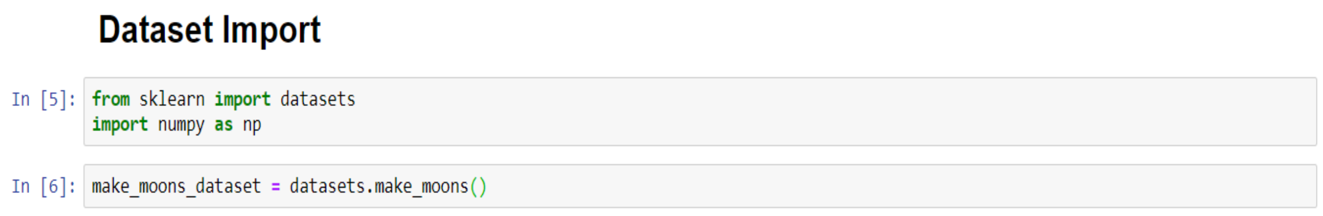
Gamma and Variance

```
In [11]: def getVariance():
             return(np.random.rand() * 10)
```

```
In [12]: def getGammafromVariance():
             gamma = 1/(2 * getVariance())
             return(gamma)
         getGammafromVariance()
```

The dataset used for implementations is an inbuilt dataset called make_moons in the sklearn library. The best value of variance(σ^2) for the above dataset was shown to be *0.4648*.

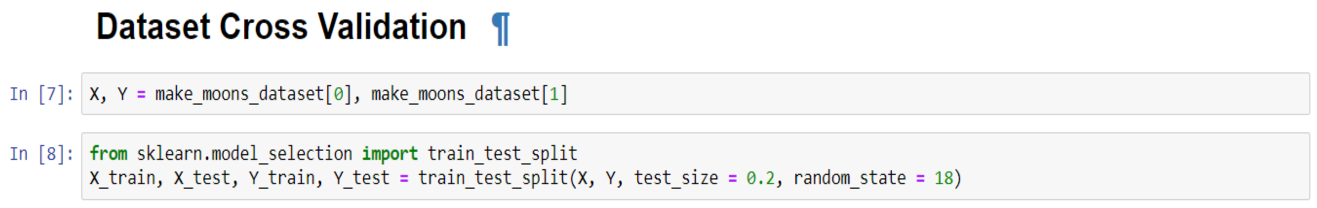
Python (using libraries)



Dataset Import

```
In [5]: from sklearn import datasets
        import numpy as np
```

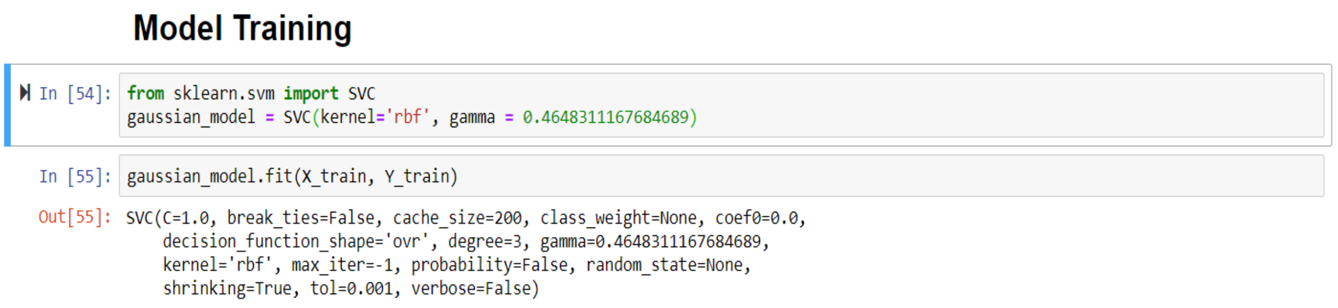
```
In [6]: make_moons_dataset = datasets.make_moons()
```



Dataset Cross Validation ¶

```
In [7]: X, Y = make_moons_dataset[0], make_moons_dataset[1]
```

```
In [8]: from sklearn.model_selection import train_test_split
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 18)
```

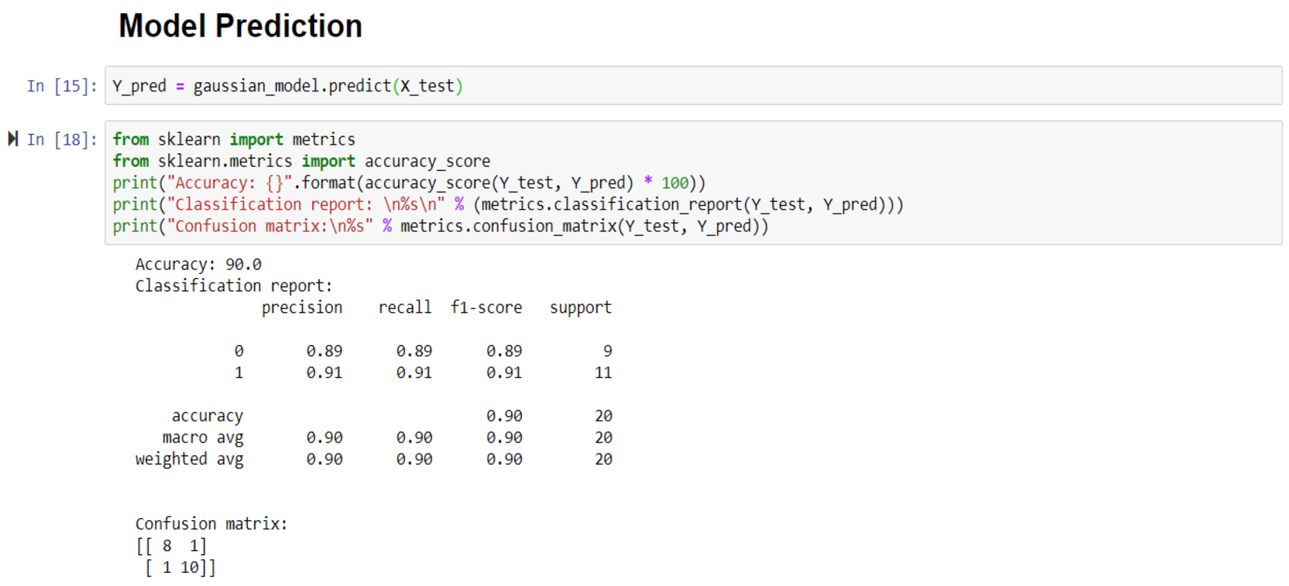


Model Training

```
In [54]: from sklearn.svm import SVC
         gaussian_model = SVC(kernel='rbf', gamma = 0.4648311167684689)
```

```
In [55]: gaussian_model.fit(X_train, Y_train)
```

```
Out[55]: SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma=0.4648311167684689,
             kernel='rbf', max_iter=-1, probability=False, random_state=None,
             shrinking=True, tol=0.001, verbose=False)
```



Model Prediction

```
In [15]: Y_pred = gaussian_model.predict(X_test)
```

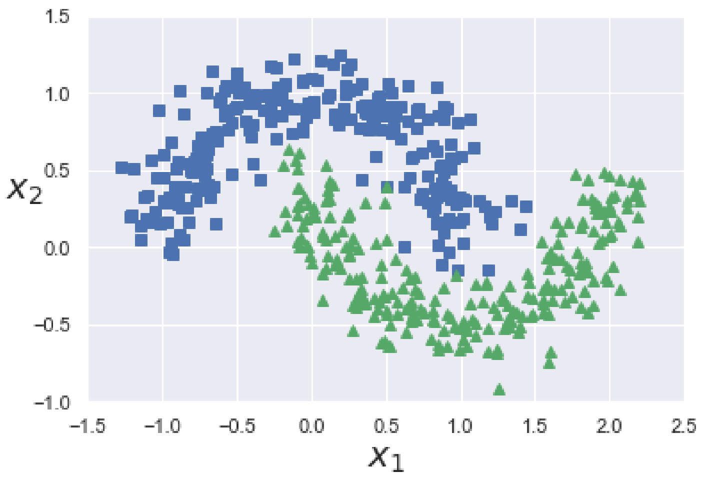
```
In [18]: from sklearn import metrics
         from sklearn.metrics import accuracy_score
         print("Accuracy: {}".format(accuracy_score(Y_test, Y_pred) * 100))
         print("Classification report: \n%s\n" % (metrics.classification_report(Y_test, Y_pred)))
         print("Confusion matrix:\n%s" % metrics.confusion_matrix(Y_test, Y_pred))
```

```
Accuracy: 90.0
Classification report:
              precision    recall  f1-score   support

           0       0.89      0.89      0.89         9
           1       0.91      0.91      0.91        11

    accuracy                           0.90        20
   macro avg       0.90      0.90      0.90        20
weighted avg       0.90      0.90      0.90        20


Confusion matrix:
[[ 8  1]
 [ 1 10]]
```

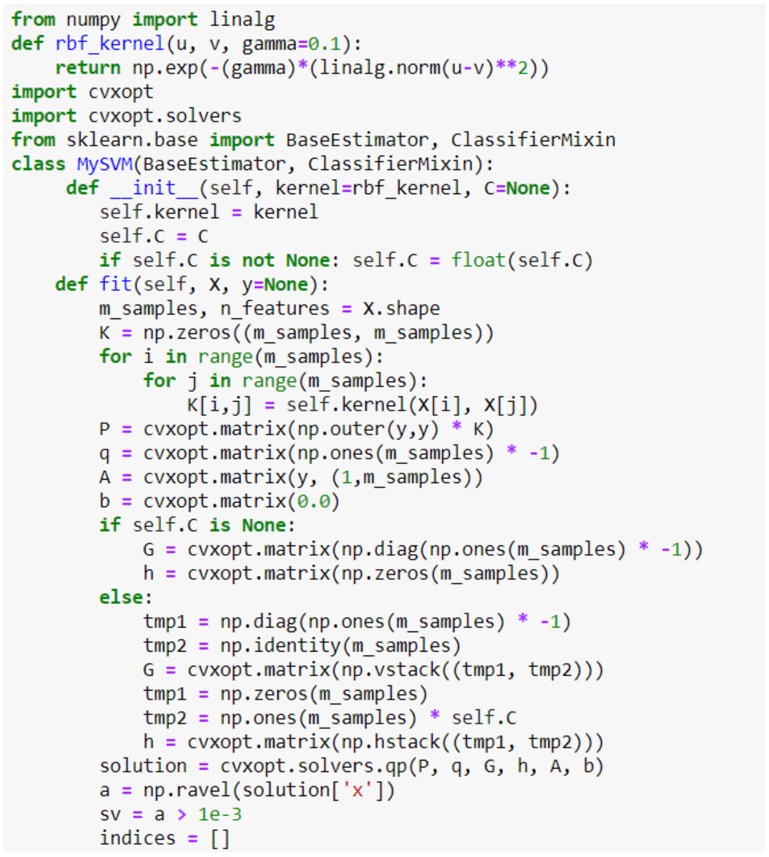
Python Implementation (Without Libraries)

Dataset Import and Cross Validation

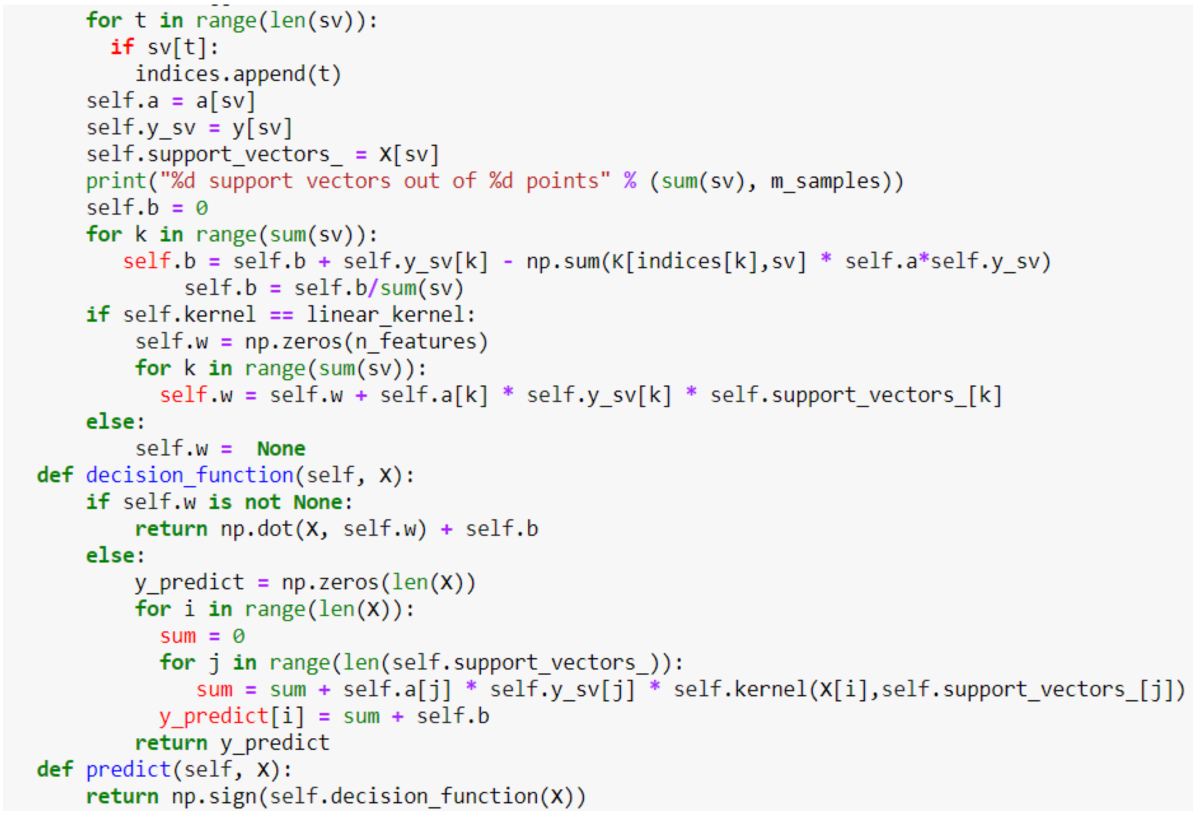
```
import sys
assert sys.version_info >= (3, 5)
import sklearn
assert sklearn.__version__ >= "0.20"
from sklearn.datasets import make_moons
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
import numpy as np
import matplotlib.pyplot as plt

X, y = make_moons (n_samples = 500, noise = 0.15, random_state = 49)

def plot_dataset(X, y, axes):
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs")
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^")
    plt.axis(axes)
    plt.grid(True, which='both')
    plt.xlabel(r"$x_1$", fontsize=20)
    plt.ylabel(r"$x_2$", fontsize=20, rotation=0)

plot_dataset(X,y,[-1.5, 2.5, -1, 1.5])
plt.show()

y = y*2-1.0

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)
```



Defining the SVM Gaussian kernel



```
from numpy import linalg
def rbf_kernel(u, v, gamma=0.1):
    return np.exp(-(gamma)*(linalg.norm(u-v)**2))
import cvxopt
import cvxopt.solvers
from sklearn.base import BaseEstimator, ClassifierMixin
class MySVM(BaseEstimator, ClassifierMixin):
     def __init__(self, kernel=rbf_kernel, C=None):
        self.kernel = kernel
        self.C = C
        if self.C is not None: self.C = float(self.C)
    def fit(self, X, y=None):
        m_samples, n_features = X.shape
        K = np.zeros((m_samples, m_samples))
        for i in range(m_samples):
            for j in range(m_samples):
                K[i,j] = self.kernel(X[i], X[j])
        P = cvxopt.matrix(np.outer(y,y) * K)
        q = cvxopt.matrix(np.ones(m_samples) * -1)
        A = cvxopt.matrix(y, (1,m_samples))
        b = cvxopt.matrix(0.0)
        if self.C is None:
            G = cvxopt.matrix(np.diag(np.ones(m_samples) * -1))
            h = cvxopt.matrix(np.zeros(m_samples))
        else:
            tmp1 = np.diag(np.ones(m_samples) * -1)
            tmp2 = np.identity(m_samples)
            G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
            tmp1 = np.zeros(m_samples)
            tmp2 = np.ones(m_samples) * self.C
            h = cvxopt.matrix(np.hstack((tmp1, tmp2)))
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)
        a = np.ravel(solution['x'])
        sv = a > 1e-3
        indices = []
```



```
        for t in range(len(sv)):
          if sv[t]:
            indices.append(t)
        self.a = a[sv]
        self.y_sv = y[sv]
        self.support_vectors_ = X[sv]
        print("%d support vectors out of %d points" % (sum(sv), m_samples))
        self.b = 0
        for k in range(sum(sv)):
           self.b = self.b + self.y_sv[k] - np.sum(K[indices[k],sv] * self.a*self.y_sv)
                self.b = self.b/sum(sv)
        if self.kernel == linear_kernel:
            self.w = np.zeros(n_features)
            for k in range(sum(sv)):
              self.w = self.w + self.a[k] * self.y_sv[k] * self.support_vectors_[k]
        else:
            self.w =  None
    def decision_function(self, X):
        if self.w is not None:
            return np.dot(X, self.w) + self.b
        else:
            y_predict = np.zeros(len(X))
            for i in range(len(X)):
              sum = 0
              for j in range(len(self.support_vectors_)):
                 sum = sum + self.a[j] * self.y_sv[j] * self.kernel(X[i],self.support_vectors_[j])
              y_predict[i] = sum + self.b
            return y_predict
    def predict(self, X):
        return np.sign(self.decision_function(X))
```

Model Training and Prediction

```
svm_rbfmy = MySVM(kernel=rbf_kernel,C=2)
svm_rbfmy.fit(X_train, y_train)
expected = y_test
predicted = svm_rbfmy.predict(X_test)
```

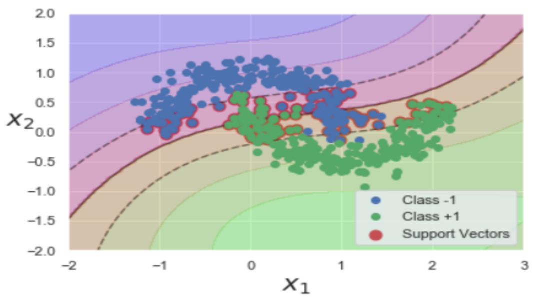
Plotting Results

```
import matplotlib.pyplot as plt

def plot_svm (clf, X, y, axes=[-2, 3, -2, 2]):
    x0s = np.linspace(axes[0], axes[1], 100)
    x1s = np.linspace(axes[2], axes[3], 100)
    x0, x1 = np.meshgrid(x0s,x1s)
    X_mesh = np.c_[x0.ravel(), x1.ravel()]
    y_pred = clf.predict(X_mesh).reshape(x0.shape)
    y_decision = clf.decision_function(X_mesh).reshape(x0.shape)

    plt.figsize=(16, 9)
    plt.plot(X[:, 0][y==-1], X[:, 1][y==-1], "bo", label="Class -1")
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "go", label="Class +1")
    plt.scatter(clf.support_vectors_[:,0], clf.support_vectors_[:,1], s=80, c="r", label="Support Vectors")
    plt.contourf(x0,x1, y_pred, cmap = plt.cm.brg, alpha = 0.1)
    plt.contourf(x0,x1, y_decision, cmap = plt.cm.brg, alpha = 0.2)
    plt.contour(x0, x1, y_decision, colors='k',
                levels=[-1, 0, 1], alpha=0.5,
                linestyles=['--', '-', '--'])
    plt.legend(loc="lower right")
    plt.axis("auto")

    plt.grid(True, which='both')
    plt.xlabel(r"$x_1$", fontsize=20)
    plt.ylabel(r"$x_2$", fontsize=20, rotation=0)
```

Performance Results

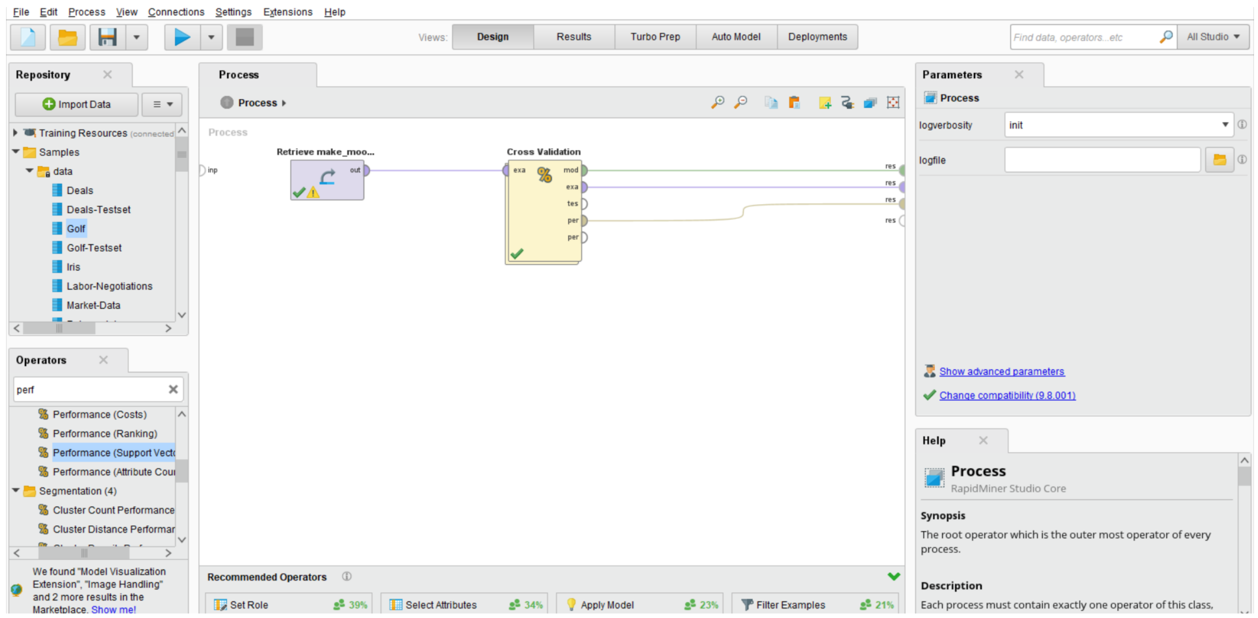
```
print("Classification report: \n%s\n" % (metrics.classification_report(expected, predicted)))
print("Confusion matrix:\n%s" % cm)
print("Accuracy={}".format(metrics.accuracy_score(expected, predicted)))
plot_svm(svm_rbfmy,X,y)
```

```
Optimal solution found.
130 support vectors out of 400 points
Classification report:
              precision    recall  f1-score   support

        -1.0       0.88      0.78      0.82        45
         1.0       0.83      0.91      0.87        55

    accuracy                           0.85       100
   macro avg       0.85      0.84      0.85       100
weighted avg       0.85      0.85      0.85       100


Confusion matrix:
[[35 10]
 [ 5 50]]
Accuracy=0.85
```



RAPIDMINER Implementation (Screenshots)

Overall Model


File Edit Process View Connections Settings Extensions Help
Views: Design Results Turbo Prep Auto Model Deployments
Repository
Import Data
Training Resources (connected)
Samples
data
Deals
Deals-Testset
Golf
Golf-Testset
Iris
Labor-Negotiations
Market-Data
Operators
perf
Performance (Costs)
Performance (Ranking)
Performance (Support Vecto
Performance (Attribute Cou
Segmentation (4)
Cluster Count Performance
Cluster Distance Performar
We found "Model Visualization Extension", "Image Handling" and 2 more results in the Marketplace. Show me!
Process
Retrieve make_moo...
Cross Validation
Recommended Operators
Set Role 39%
Select Attributes 34%
Apply Model 23%
Filter Examples 21%
Parameters
Help

Process

logverbosity: init

logfile:

Show advanced parameters

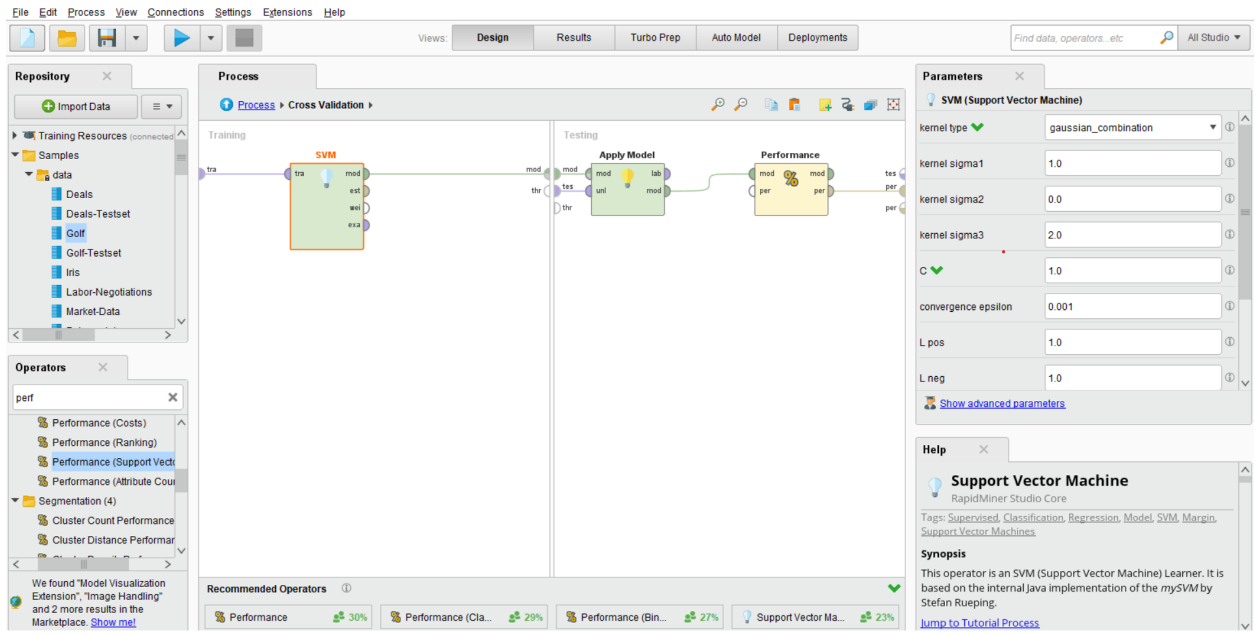
Change compatibility (9.8.001)

Process
RapidMiner Studio Core

Synopsis
The root operator which is the outer most operator of every process.

Description
Each process must contain exactly one operator of this class,

SVM Gaussian model inside Cross validation


File Edit Process View Connections Settings Extensions Help
Views: Design Results Turbo Prep Auto Model Deployments
Repository
Import Data
Training Resources (connected)
Samples
data
Deals
Deals-Testset
Golf
Golf-Testset
Iris
Labor-Negotiations
Market-Data
Operators
perf
Performance (Costs)
Performance (Ranking)
Performance (Support Vecto
Performance (Attribute Cou
Segmentation (4)
Cluster Count Performance
Cluster Distance Performar
We found "Model Visualization Extension", "Image Handling" and 2 more results in the Marketplace. Show me!
Process ▸ Cross Validation
Training
Testing
SVM
Apply Model
Performance
Recommended Operators
Performance 30%
Performance (Cla... 29%
Performance (Bin... 27%
Support Vector Ma... 23%
Parameters
Help

SVM (Support Vector Machine)

kernel type	gaussian_combination
kernel sigma1	1.0
kernel sigma2	0.0
kernel sigma3	2.0
C	1.0
convergence epsilon	0.001
L pos	1.0
L neg	1.0

Show advanced parameters

Support Vector Machine
RapidMiner Studio Core

Tags: Supervised, Classification, Regression, Model, SVM, Margin, Support Vector Machines

Synopsis
This operator is an SVM (Support Vector Machine) Learner. It is based on the internal Java implementation of the *mySVM* by Stefan Rueping.

Jump to Tutorial Process

Kernel SVM Performance Results

Kernel Model

```
Total number of Support Vectors: 100
Bias (offset): 0.494

w[att1] = 0.708
w[Column_1] = 3.721
w[Column_2] = -1.770
```

number_of_support_vectors

```
number_of_support_vectors: 89.900 +/- 0.316 (micro average: 89.900)
```