Support Vector Machines (SVM) Regression

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. The illustration below shows the basic idea behind SVMs. Here we see the original objects (left side of the schematic) mapped, i.e., rearranged, using a set of mathematical functions, known as kernels. The process of rearranging the objects is known as mapping (transformation). Note that in this new setting, the mapped objects (right side of the schematic) is linearly separable and, thus, instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the green and the red objects.


SVM is primarily a classier method that performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels. SVM supports both regression and classification tasks and can handle multiple continuous and categorical variables. For categorical variables a dummy variable is created with case values as either 0 or 1. Thus, a categorical dependent variable consisting of three levels, say (A, B, C), is represented by a set of three dummy variables:

A: \{1 0 0\}, B: \{0 1 0\}, C: \{0 0 1\}
To construct an optimal hyperplane, SVM employs an iterative training algorithm, which is used to minimize an error function. According to the form of the error function, SVM models can be classified into four distinct groups:

1. Classification SVM Type 1 (also known as C-SVM classification)
2. Classification SVM Type 2 (also known as nu-SVM classification)
3. Regression SVM Type 1 (also known as epsilon-SVM regression)
4. Regression SVM Type 2 (also known as nu-SVM regression)

1. Classification SVM Type 1

Here, training involves the minimization of the error function:

$$\frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$$

subject to the following constraints:

$$y_i (w^T \phi(x_i) + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, i = 1, \ldots, N$$

where $C$ is the capacity constant, $w$ is the vector of coefficients, $b$ is a constant, and $\xi_i$ represents parameters for handling non-separable data (inputs). The index $i$ labels the $N$ training cases. Note that $y \in \{\pm 1\}$ represents the class labels and $x_i$ represents the independent variables. The kernel $\phi$ is used to transform data from the input (independent) to the feature space. It should be noted that the larger the $C$, the more the error is penalized. Thus, $C$ should be chosen with care to avoid over fitting.

2. Classification SVM Type 2

In contrast to Classification SVM Type 1, the Classification SVM Type 2 model minimizes the error function:

$$\frac{1}{2} w^T w - \nu \rho + \frac{1}{N} \sum_{i=1}^{N} \xi_i$$

subject to the following constraints:

$$y_i (w^T \phi(x_i) + b) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, \ldots, N \text{ and } \rho \geq 0$$
In a regression SVM, you have to estimate the functional dependence of the dependent variable \( y \) on a set of independent variables \( x \). It assumes, like other regression problems, that the relationship between the independent and dependent variables is given by a deterministic function \( f \) plus the addition of some additive noise.

**Regression SVM**

\( y = f(x) + \text{noise} \)

The task is then to find a functional form for \( f \) that can correctly predict new cases that the SVM has not been presented with before. This can be achieved by training the SVM model on a sample set, i.e., training set, a process that involves, like classification (see above), the sequential optimization of an error function. Depending on the definition of this error function, two types of SVM models can be recognized:

3. **Regression SVM Type 1**

For this type of SVM the error function is:

\[
\frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i + C \sum_{i=1}^{N} \xi_i^* 
\]

which we minimize subject to:

\[
\begin{align*}
    w^T \phi(x_i) + b - y_i & \leq \varepsilon + \xi_i^* \\
    y_i - (w^T \phi(x_i) + b) & \leq \varepsilon + \xi_i \\
    \xi_i, \xi_i^* & \geq 0, i = 1, \ldots, N
\end{align*}
\]

4. **Regression SVM Type 2**

For this SVM model, the error function is given by:

\[
\frac{1}{2} w^T w - C \left( \nu \varepsilon + \frac{1}{N} \sum_{i=1}^{N} (\xi_i + \xi_i^*) \right) 
\]

which we minimize subject to:

\[
\begin{align*}
    (w^T \phi(x_i) + b) - y_i & \leq \varepsilon + \zeta_i \\
    y_i - (w^T \phi(x_i) + b_i) & \leq \varepsilon + \zeta_i^* \\
    \zeta_i, \zeta_i^* & \geq 0, i = 1, \ldots, N, \varepsilon \geq 0
\end{align*}
\]
There are number of kernels that can be used in Support Vector Machines models. These include linear, polynomial, radial basis function (RBF) and sigmoid:

\[ K(X_i, X_j) = \begin{cases} X_i \cdot X_j & \text{Linear} \\ \gamma (X_i \cdot X_j + C)^d & \text{Polynomial} \\ \exp(-\gamma |X_i - X_j|^2) & \text{RBF} \\ \tanh(\gamma X_i \cdot X_j + C) & \text{Sigmoid} \end{cases} \]

that is, the kernel function, represents a dot product of input data points mapped into the higher dimensional feature space by transformation \( \phi \) (Gamma is an adjustable parameter of certain kernel functions).

The RBF is by far the most popular choice of kernel types used in SVMs. This is mainly because of their localized and finite responses across the entire range of the real x-axis.


**Worked Example**

In this worked example, we will develop a regression SVM type 1 with an RBF kernel. Our aim will be to forecast one step ahead predictions for retail trade in Australia. The retail trade data is provided by Australian Bureau of Statistics (Cat.8501.0 Retail Trade, Australia). The retail trade will be our dependent variable. Predictors, Australian population 15+, CPI, unemployment rate, ASX index, annual personal Income and consumer confidence index, are obtained from a number of Australian Bureau of Statistics and, Australian Stock Exchange reports.

Following histograms give us an idea on the distribution of our dependent and predictor variables.
The $p$ value of the Lilliefors test as reported by STATISTICA suggests a non-normally distributed retail trade. Calculating the Lilliefors test by hand also suggest that the data is not normally distributed.

Table 1

<table>
<thead>
<tr>
<th>Lilliefors test:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>0.086</td>
</tr>
<tr>
<td>$D_{(standardized)}$</td>
<td>0.832</td>
</tr>
<tr>
<td>$p$-value</td>
<td>0.085</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.05</td>
</tr>
</tbody>
</table>

As the computed $p$-value is greater than the significance level $\alpha=0.05$, one cannot reject the null hypothesis $H_0$ (that the data follows normal distribution).

Further, the following histograms show that most of the predictors are not normally distributed except the unemployment rate. This suggests that regression models based on the assumption of normality is not suitable for modelling the socio-economic data we have. Therefore, SVMs are suitable alternatives in modelling this data.
Figure 3

Histogram: Population_15+
K-S d=.12233, p<.15 ; Lilliefors p<.01

Figure 4

Histogram: Annual_Personal_Income(000)
K-S d=.10547, p> .20; Lilliefors p<.05
Figure 5

Histogram: CPI
K-S d = 0.07771, p > 0.20; Lilliefors p < 0.20
Expected Normal

Figure 6

Histogram: Unemployment_Rate
K-S d = 0.05537, p > 0.20; Lilliefors p < 0.20
Expected Normal
Figure 7

Histogram: ASX_Index
K-S d=.11500, p<.20; Lilliefors p<.01
Expected Normal

Figure 8

Histogram: Consumer_Confidence_Index
K-S d=.10356, p>.20; Lilliefors p<.05
Expected Normal
The following figure shows the model fit as performed by STATISTICA.

Figure 9

![Histogram of RetailTrade(000000) (Observed)](image)

Figure 10

![Histogram of RetailTrade(000000) (Predictions)](image)

The comparison of observed versus predicted in our test sample indicates a less than desired model fit.
Lastly, the observed versus predicted chart above shows that our SVM regression model clearly fails to capture the seasonality present in the retail trade; having said that seasonality adjusted data is likely to have much better prediction results when treated with SVM regressions. The chart above clearly hints at this hypothesis.

For an automated production, the following C++ code (as generated by STATISTICA) will implement the SVM model we discussed in this document.

```cpp
#include <math.h>
int noContInputs = 6;
int noCatInputs = 0;
int noInputs = 6;
int noDummyInputs = 6;
int noContOutputs = 1;
int noCatOutputs = 0;
static double dummy[6];
int type = 3;
int ktype = 2;
double coef0 = 0;
double gamma = 1.66666666666670e-001;
double order = 0;
int noSVs = 28;
int noNominals = 0;
int noOfSVs= 28;
static int label[2];
static int noOfSVsPerClass[2];
static double SV[] =
{
  6.64727877025317e-003, 8.48532910388577e-002, 3.42105263157895e-001, 6.50000000e-001, 1.34191356613203e-001,
  7.57575757575758e-001, 1.32945575405063e-002, 1.35606661379857e-001, 3.42105263157895e-001, 5.66666666666666e-001,
  1.75774997550060e-001, 7.60710553814002e-001,
};
```


1.00000000000000e+001,
1.00000000000000e+001,
-1.00000000000000e+001,
-9.26808732157175e+000,
-2.52583206978974e+000,
-6.95827359809392e+000,
-1.45198042606107e+001,
-1.00000000000000e+001,
1.00000000000000e+001,
1.00000000000000e+001,
-1.00000000000000e+001,
2.97465032856124e+000,
};
static double constants[]=
{
  9.83707042341526e-001
};
static double ishift[]=
{
-6.8566805151641e+000,
-2.48215701823949e+000,
-3.15789473684211e-001,
-6.33333333333333e-001,
-1.5385620488028e+000,
-2.36206896551724e+000,
};
static double iscale[]=
{
4.1545923140839e-004,
7.93021411578113e-002,
2.63157894736842e+001,
1.66666666666667e+001,
3.26658609087642e-004,
2.61233019853710e+002,
};
double oshift=3.34413235492873e+000;
double oscale=1.59003998950574e-004;
*/
void create_inputs(double input[])
{
  int i, j, k=0;
  for(i=0; i<noDummyInputs; i++) dummy[i] = 0.0;
  for(i=0; i<noContInputs; i++) dummy[i] = input[i];
  for(i=0; i<noContInputs; i++)
  {
    dummy[i] = ishift[i] + iscale[i]*dummy[i];
  }
}
*/
double dot(double dummy[], double sv[])
{
  int i=0, j=0;
  double sum = 0;
  while(i!=noDummyInputs && j!=noDummyInputs)
  {
    if(i==j)
    {
      sum += dummy[i]*sv[j];
      i++;
      j++;
    }
    else
    {
      if(i>j) j++;
      else i++;
    }
    return sum;
  }
}*/
double kernel(double dummy[], int index)
{
  int i=0, j=0;
  double k, sum=0, d;
  double sv[6];
  for(i=0; i<noDummyInputs; i++) sv[i] = SV[i+index*noDummyInputs];
  i=0;
  switch(ktype)
  {

```c
case 0:
    k = dot(dummy, sv);
    break;

case 1:
    k = pow(gamma * dot(dummy, sv) + coef0, order);
    break;

case 2:
    { while (i != noDummyInputs && j != noDummyInputs)
      { if (i == j)
        { d = dummy[i] - sv[j];
          sum += d * d;
          i++;
          j++;
        }
        else
        { if (i > j)
          { sum += sv[j] * sv[j];
            j++;
          }
          else
          { sum += dummy[i] * dummy[i];
            i++;
          }
        }
      }
      while (i < noDummyInputs);
      { sum += dummy[i] * dummy[i];
        i++;
      }
      while (j < noDummyInputs)
      { sum += sv[j] * sv[j];
        j++;
      }
    }
    k = exp(-gamma * sum);
    break;

case 3:
    k = tanh(gamma * dot(dummy, sv) + coef0);
    break;
}

return k;
/**---------------------------------------------*/
double predict(double dummy[])
{ int i, j, k, index = 0;
  double sum = 0;
  if (type == 3 || type == 4)
    { for (i = 0; i < noOfSVs; i++)
      { sum += coefficient[i] * kernel(dummy, i);
      }
    sum -= constants[0];
    return sum;
  }
  else
    { int si, sj, ci, cj, p = 0;
      double *coef1, *coef2;
      int temp[2];
      int score[2];
      double kernels[28];
```
for(i=0; i<noOfSVs; i++) kernels[i] = kernel(dummy, i);

temp[0] = 0;
for(i=1; i<noNominals; i++) temp[i] = temp[i-1] + noOfSVsPerClass[i-1];
for(i=0; i<noNominals; i++) score[i] = 0;
for(i=0; i<noNominals; i++)
    
    sum = 0;
    si = temp[i];
    sj = temp[i];
    ci = noOfSVsPerClass[i];
    cj = noOfSVsPerClass[j];
    coef1 = coefficient;
    coef2 = coefficient;
    for(k=0; k<j-1; k++) coef1++;  
    for(k=0; k<i; k++) coef2++;   
    for(k=0; k<ci; k++) sum += coef1[si+k]*kernels[si+k];
    for(k=0; k<ci; k++) sum += coef2[sj+k]*kernels[sj+k];
    sum -= constants[p++];
    if(sum>0) ++score[i]; else ++score[j];
}
for(i=1; i<noNominals; i++)
    
    if(score[i]>score[index]) index=i;
}
return label[index];

double unscale(double prediction)
{
    return (prediction - oshift)/oscale;
}

#include <stdio.h>
#define MISSINGCODE -9999
int main(void)
{
    printf("STATISTICA Support Vector Machine test harness program.\n\n");
    printf("NOTE: Nominal variables should be numbered as they appear in the c file.\n\n");
    printf("Enter inputs below:\n\n");
    int i, cont;
    double input[6], result;
    while (1)
    {
        printf("\n");
        for(i=0; i<noInputs; i++)
            
            printf("Enter value for input %d: ", i+1);
            scanf("%lg", &input[i]);
            if(input[i] == MISSINGCODE)
                
                break;
            }
        create_inputs(input);
        result = unscale(predict(dummy));
        printf("\nSVM result: ");
        printf("Output: %.14e", result);
        printf("\n\n\n\nEnter 1 to continue, 0 to exit \n\n");
        scanf("%d", &cont);
        if(cont == 0) break;
    }
    return 0;
}