Comparing performance of k-Nearest Neighbors, Parzen Windows and SVM Machine Learning Classifiers on QSAR Biodegradation Data across Multiple Dimensions

Anup K. Mishra
akmm94@mail.missouri.edu
Department of Electrical and Computer Engineering
University of Missouri, Columbia

Abstract

Machine learning and pattern recognition are the most popular artificial intelligence techniques to model systems, those can learn from data. These techniques efficiently help in Classification, Regression, Clustering and Anomaly detection etc. k-Nearest Neighbors, Parzen Windows and Support Vector Machine (SVM) are some of the widely used Machine Learning classification techniques. This project aims to experimentally compare several features of these classification techniques using QSAR Biodegradable dataset over different dimensions using Principal Component Analysis (PCA). The results of the experiment demonstrate that SVM performs way better than k-Nearest Neighbor and Parzen window and, k-Nearest Neighbor performs a little better over Parzen window on classification accuracy. Also, I establish that ERBF performs better than other kernel functions (RBF, Polynomial and Linear) when used for SVM.

Keywords: Machine Learning, Pattern Recognition, k-Nearest Neighbors, Parzen Window, SVM, PCA, ERBF
1. Introduction

Advances in information technologies specifically in the area of data storage and computation power in the recent years have made it possible to collect, store and process massive and often highly complex (multi-dimensional) datasets. Analysis of such datasets provides valuable information such as trends and patterns, which can further be utilized to improve decision making. In the past, different data analysis techniques geared towards classification and regression has been utilized in academia as well as in practice to extract knowledge from the raw data. With increase in storage capability, multi-dimensional data is collected and linear/nonlinear relationship among them is assessed. k-Nearest Neighbors [1] and Parzen Window [2] are two of the most common machine learning techniques to explore such relations. Both of them are non-parametric methods for classification. These are amongst the simplest of all machine learning algorithms. Both of these methods consider a single test sample at a time and evaluate each and every training sample to determine a class that the test sample belongs to. The k-Nearest Neighbor method considers a particular number (kN) of training samples closest to the test sample and checks the presence of number of samples from each class to classify the test data. So for instance, a test sample is defined as it belongs to class A out of class A and B (for a two class classification), if there are more samples of class A than B in the considered sample space (kN closest training samples). The Parzen window works pretty much in the same way but it does not consider a fixed number of samples (like kN in k-Nearest Neighbor) but defines a window size (hn) and considers training samples inside the window. And (using the previous example) the test sample is classified as class A if there are higher number of training samples of class A in the window than class B. Even though they are pretty similar in their implementation, they both have their own importance as per requirement and dataset used. Most commonly the square root of the number of training samples is considered as an ideal kN value for the k-NN classifier, but choosing the most appropriate kN really depends on the dataset and its distribution. The kN and hn values are usually determined by looking at how the dataset performs with each of them. Both of these methods estimate the underlying distribution of the data in a non-parametric fashion as we have no idea about the distribution of the data.

These two methods may fall sort because of their computationally expensive algorithms, as they consider all the samples of training data set. To address this problem, sparse kernel methods can be used, which not only provide higher flexibility in mapping nonlinear relationships but also, are computationally inexpensive and help draw inferences from the data by attaining high predictive performances as compared to other methods [3].

Realizing the growing relevance of sparse kernel methods, in our present project, I evaluate and compare the performance of a widely recognized sparse kernel classification method SVM with k-Nearest Neighbors and Parzen Window methods on QSAR Biodegradation Dataset. But, before I begin the comparison exercise, I also evaluate the optimality of the different kN values for the k-Nearest Neighbor method, hn values for Parzen window method and kernel functions with different parameter values for SVM. Having established the appropriate values I move towards comparing k-NN, Parzen window and SVM on metrics such as classification accuracy provided by sensitivity and specificity its generalizability.
SVM is a classification method that minimizes the classification error without requiring a statistical data model. It has been widely recognized due to its simple implementation and consistently high classification accuracy when applied to many real-world classification situations. Due to its inherent kernel properties, SVM classifier can separate data that are often not easily separable in the original data space (i.e., two-dimensional x, y) by mapping data into a higher dimensional (transformed) space. SVM uses the kernel function to find a hyper-plane that maximizes the distance (margin) between the two classes, while minimizing training error. The resultant model is sparse, depending on only a few training samples (“support vectors”) [4].

The QSAR Dataset [5] contains values for 41 biochemical attributes or features (molecular descriptors) [Refer Appendix A] and has 1055 chemical samples in it. All these chemicals belongs to either of the two classes or categories named, Ready Biodegradable and Not Ready Biodegradable.

Also, I implemented PCA (Principal Component Analysis) on the dataset and compared the classifiers on different dimensions of the data. PCA uses orthogonal transformation to convert a set of correlated data in to a set of linearly uncorrelated variables called principal components. The transformation is defined in such a way that the principal components are arranged with their variance values [7].

In the following sections, I first describe the different methods I am using in this project like, k-NN, Parzen Window and SVM, followed by my implementation strategies and different experiments I did to compare the methods and finally conclude with our findings from the three experiments.

2. Background and Motivation

k-NN, Parzen window, SVM and PCA are the four important methods I used in this project where k-NN, Parzen Window and SVM are the three non-parametric classifiers and PCA is used for dimensionality reduction.

k-NN, Parzen Window, SVM and PCA are the methods [10] I am replicating. There have been many work in pattern classification and comparison of classifiers [11] including supervised and unsupervised methods. Method comparison is really important when you have a dataset having no idea about its distribution. Comparing different classifiers can give us an idea and confidence in choosing a classifier when considering a data similar to that or a new dataset with similar attributes.
3. Proposed Method

In this project, I compare the three widely used machine learning classification technique to classify QSAR Biodegradable data. The data has two classes; ready biodegradable and not ready biodegradable. The data contains 356 ready biodegradable and 699 not ready biodegradable data. All of them are represented with 41 bio chemical characteristics. 1055 total samples look really less in number as compared to 41 dimensions or attributes given. Therefore I propose a method to classify the data using different classification techniques, considering different dimensions at a time, to test how the classification accuracy varies over different dimensions. I used PCA for dimension reduction.

I divided the data in to training and testing considering what percent of data to be trained and what percent should be left out for testing. The experiments conducted here were with considering 50% of the data as training data each time.

For the first set of experiments I considered all the dimensions and experimented with all the classification methods. I repeated the experiments considering 20 and 5 top Eigen valued dimensions from PCA out of 41 dimensions in further experiments. I also considered the 20 and 5 least Eigen valued dimensions from PCA and did another set of experiments to compare the results.

The basic reason behind all these experiments is, no two data set are similar. The experiments were to determine how the different classifiers perform with different dimensions of the data. This is very important to play around with the data to get an optimum solution of classification and understand the data distribution.

Every test results a confusion matrix and the program determines Specificity, Sensitivity, Precision, Accuracy and Error of classification [8].

Sensitivity= Number of True Positive/ (Number of True Positive + Number of False Negative)
Specificity= Number of True Negative/ (Number of True Negative + Number of False Positive)
Accuracy= (Number of True Positive + Number of True Negative) / (Number of Data Samples)
Error= 1- Accuracy
Precision= Number of True Positive / (Number of True Positive + Number of False Positive)

Also, I am plotting the Receiver Operating Characteristic (ROC) curves [8] for each classification to compare their performance in a much clearer way. ROC is considered as the gold standard to compare classifiers.

3.1 k-NN

The experiments were simple and I started classifying the data with k-NN classifier considering all the dimensions at first and then different the PCA transformed data with different dimensions. Considering
the kN value of k-NN is always a tough job because it’s hard to predict it. Many people use square root of number of training data as the kN value, considering it ideal. But in my experiments I literally run the kNN algorithm 50 times to get the best kN value out of them. There is no specific value of kN for a particular data too. I really depends on the number of dimensions. So every time I considered a different dimension I run the kNN algorithm 50 times again to make sure that I have a clear idea about the best kN values for the particular scenario. kN value is always an integer because it defines the number of training samples considered around the test data to classify it into a class.

**Pseudo code for implementing k-NN**

1. Load the data and normalize it
2. Split the data into training and testing data
3. Initialize the kN value to 1 and iterate till 50
4. Initialize a confusion matrix with zeroes
5. For a particular test data of a class, find the Euclidian distances of all the training data.
6. Find the kN closest training data samples
7. Find the number of samples belonging to each class and find the class with highest training sample
8. Update confusion matrix
9. Repeat Step 5 to 8, until all test data sample updates the confusion matrix
10. Get the final confusion matrix and calculate Accuracy, Specificity, Sensitivity, Precision and Error and print them keeping them all in a matrix
11. Repeat until kN value is 50
12. Plot ROC
13. Stop

Repeat this pseudo code for different dimensions. The distance calculation method used here is, Euclidian distance between the test sample and the training sample around it.

**3.2 Parzen Window**

This method considers a window size to classify a particular test sample. The window size consideration is equally difficult as selecting a kN value. So I used the same strategy without keeping the hn value constant. I varied the window size from 0.5 to 1.5 with an increment of .1 to see which window size works best for the data set with a particular number of dimension consideration. For different dimensions I got different best working hn values. Similar to k-NN method, I calculated distance of training samples around the test sample using Euclidian distance method, to check if the training samples are inside the Parzen window considered each time.

**Pseudo code for implementing Parzen Window**

1. Load the data and normalize it
2. Split the data into training and testing data
3. Initialize the hn value to .5 and iterate till 1.5 with .1 increments
4. Initialize a confusion matrix with zeroes
Step 5: For a particular test data of a class, find the Euclidian distances of all the training data.
Step 6: Find the training samples inside the hn Parzen Window
Step 7: Find the number of training samples belonging to each class and find the class with highest training sample
Step 8: Update confusion matrix
Step 9: Repeat Step 5 to 8, until all test data sample updates the confusion matrix
Step 10: Get the final confusion matrix and calculate Accuracy, Specificity, Sensitivity, Precision and Error and print them keeping them all in a matrix
Step 11: Repeat until hn value is 1.5
Step 12: Plot ROC
Step 13: Stop

This pseudo code is repeated for different dimensions of data.

3.3 SVM

SVM has different parameters to set to get good results. Although it outperformed k-NN and Parzen Window classification but we can find really bad results if do not set the parameters right. In the extant machine learning literature there are two commonly used kernel functions for feature mapping. They are,

I. ERBF also known as exponential radial basis function $K(x, y) = \exp [- |x - y |/2\sigma^2]$
II. Gaussian function $K(x, y) = \exp [- |x - y |^2/2\sigma^2]$, where parameter $\sigma^2$ is the variance,

ERBF has been proved to produce improved results as compared to Gaussian. Thus, as part of experiment, I am trying to replicate this finding by comparing the performance of the two kernel functions (ERBF, RBF) for the dataset.

In this project the $\sigma^2$ is varied from .1 to 3.0 with an increment of .2 each time. In this implementation I used the SVM Code from Author: Steve Gunn [9].

SVM Code of Steve Gunn

This is a basically a MATLAB library for SVM. I am using four functions of the library and they are:

I. $[nSVs, \alpha, \text{bias}] = \text{svc}(\text{trainX}, \text{trainT}, \text{ker}, C)$;
   Svc take the training data samples and training ones column vector along with the kernel function and the coefficient of slack variable and returns number of support vectors, alpha value and the bias value
II. $\text{svcplot}(\text{trainXpca(:,1:2)}, \text{trainT}, \text{ker}, \alpha, \text{bias})$;
   svcplot is used to plot the training data with 2 dimensions( created by considering best two PCA dimensions)
III. predicted$Y = \text{svoutput}(\text{trainX}, \text{trainT}, \text{testX}, \text{ker}, \alpha, \text{bias})$;
   svoutput provides the output of the svm classification to calculate the confusion matrix
IV. test$Y = \text{softmargin}(\text{predictedY})$;
softmargin function calculates the ones matrixes using the output values from the above function to classify the data comparing test T(ones column matrix of test samples)

**Pseudo code for implementing SVM**

Step 1: Load the data and normalize it  
Step 2: Split the data into training and testing data  
Step 3: Create two column matrixes with 1’s and 1’s for the two different classes of training and testing data. (This would help in creating the confusion matrix)  
Step 4: Initialize the \( \sigma^2 \) value to .1 and iterate till 3 with .2 increments  
Step 5: Consider a coefficient of slack variable and calculate the support vectors and alpha value and bias  
Step 6: Use PCA to get the best two Eigen value dimensions and plot the support vector classification results  
Step 7: Calculate the output values to calculate the confusion matrix and calculate Accuracy, Specificity, Sensitivity, Precision and Error and print them keeping them all in a matrix  
Step 8: Repeat step 3 to 7 until \( \sigma^2 \) value is 3  
Step 9: Plot ROC  
Step 10: Stop

**3.4 PCA**

PCA is one of the common approaches of dimension reduction. In this project I am implementing a very straightforward PCA replicating the actual algorithm and the result is a matrix with the dimensions with decreasing Eigen values. I consider different subsets of PCA output data considering different dimensions at a time.

**Pseudo code for implementing SVM**

Step 1: Get the input data and subtract mean for each dimension  
Step 2: Calculate covariance matrix  
Step 3: Find the Eigenvectors and Values from the covariance matrix  
Step 4: Extract diagonal of the matrix as a vector  
Step 5: Sort the variances in decreasing order  
Step 6: Transform the original dataset

**3.5 Plot ROC**

ROC [8] is considered as one of the best graphical representation of performance of classifiers considering True Positive Rates and False Positive Rates. For each classification I considered just the increasing values of (1-specificity) to plot a graph between sensitivity and 1-specificity to avoid bad graphs. The resulted plots were interesting and helped a lot to compare the performance of different classifiers for different dimension considerations.
4. Experiments and Results

I have individually experimented with the three different classifiers and compared there ROC curve results and compared the performance considering their accuracy of classification.

4.1 Experimenting with k-NN (with all dimensions; first 20 and 5 PCA dimensions)

![Image: ROC for k-NN with all dimensions and PCA: 20 and 5 best dimensions]

Figure 4.1 shows the k-NN classification results with all dimensions, and reduced dimensions (PCA: 20 and 5 dimensions). Clearly the classification results with all the dimensions are better than top 20 and 5 dimensions. Again the top 20 PCA dimensions are able to classify the data nearly equal to all dimension classification. When the dimensions reduce to 5 the results decreased to a greater extent. It is because of data loss because of dimension reduction.

Here k-NN gives a decent performance with the following accuracies.

<table>
<thead>
<tr>
<th></th>
<th>All 41 Dimensions</th>
<th>PCA: 20 top Dimensions</th>
<th>PCA: 5 top dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-NN Accuracy</td>
<td>78.5579</td>
<td>78.3681</td>
<td>74.3833</td>
</tr>
</tbody>
</table>

Table: 4.1: Accuracies for different top dimensions using k-NN Classifier
The table 4.1 makes it clearer, as the accuracy decreases with the decrease in dimensions. With a reduction to 20 best dimensions using PCA the accuracy decreased to 0.1898 and reducing it further to 5 decreased it more 0.1746.

The decrease in accuracy is higher as we decrease the dimensions.

4.2 Experimenting with SVM (with all 41 dimensions, top 20 and 5 PCA dimensions)

The ROC plot in Fig 4.2.1 show that the SVM’s plot with all 41 dimensions is really good, getting all or most of the points exactly at top left corner, making it look like they all are a single point. This shows a very high rate of accuracy in classification because the area under the curve is almost 1. In this experiment the sigma square (spread for ERBF) value considered was from .1 to 3.0 with a .2 increment and the coefficient of slack variable C was $10^5$. 

**Fig 4.2.1 ROC plot of SVM (all dimensions) with k-NN (all, PCA:20, PCA:5 dimensions)**
Fig 4.2.2 ROC plot of SVM (all dimensions and PCA 20 dimensions) with k-NN (all, PCA:20, PCA:5 dimensions)

Figure 4.2.3: ROC plot of SVM and PCA with all 41, PCA: 20 and PCA 5 dimensions and k-NN
The figure 4.2.2 and 4.2.3 are provided here for a better understanding of the ROC curve of the SVM classifier when considered PCA 20 high dimensions and PCA 5 high dimensions because all these three ROC curves are overlapped. The change in dimension (considering the high PCA dimensions really doesn’t change the accuracy and performance of SVM as per the above graphs. But giving a closer look can change our perception. Even though the accuracy decreased in a very minimal level, the point here is accuracy decreased. The following table contains the accuracy values for different dimensional classification by SVM and k-NN.

<table>
<thead>
<tr>
<th></th>
<th>All 41 dimensions</th>
<th>PCA 20 high dimensions</th>
<th>PCA 5 high dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>99.3258</td>
<td>99.3213</td>
<td>99.2806</td>
</tr>
<tr>
<td>k-NN</td>
<td>78.5579</td>
<td>78.3681</td>
<td>74.3833</td>
</tr>
</tbody>
</table>

Table: 4.2 Accuracy of SVM and k-NN classification

The accuracy of SVM is way higher than k-NN classifier in this observation. When decreased to 20 dimensions the accuracy decreased by 0.0045\% and again decreasing the dimensions to 5 it decreased to 0.0452\%. The difference observed here is minimal and we can consider this as there is absolutely no change.

4.3 Experimenting with Parzen Window (with all 41 dimensions, top 20 and 5 PCA dimensions)

![ROC Plot for SVM, kNN and Parzen Window](image)

Fig 4.3: ROC of Parzen window classification (with all, PCA 20 and PCA 5 dimensions) vs k-NN and SVM
The Fig 4.3 shows an overall comparison of all the classifiers with three different dimensionalities (all 41 dimensions, top PCA 20 and 5 Eigen value attributes). The Parzen window classification with all dimensions has a little lesser accuracy than with PCA top 20 dimensions and the Parzen window classifier with PCA top 5 Eigen value dimensions performs equally well as with all dimensions.

<table>
<thead>
<tr>
<th></th>
<th>All Dimensions</th>
<th>PCA top 20 dimensions</th>
<th>PCA top 5 dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>99.3258</td>
<td>99.3213</td>
<td>99.2806</td>
</tr>
<tr>
<td>k-NN</td>
<td>78.5579</td>
<td>78.3681</td>
<td>74.3833</td>
</tr>
<tr>
<td>Parzen Window</td>
<td>70.9677</td>
<td>73.2448</td>
<td>72.6755</td>
</tr>
</tbody>
</table>

Again comparing the Parzen window with other two classifiers here. Parzen window always gave less accuracy than the other two. But there is a dramatic increment and decrement in the accuracy values when the dimensions are reduced further.

### 4.4 Experimenting with different Kernel functions for SVM

In this experiment I considered two kernel functions; ERBF, RBF because the dataset I am using is much overlapped and considering these two functions I can fit a good SVM classifier.

Considering different values of $\sigma^2$ I compared the accuracy of SVM classifier using both ERBF and RBF kernel functions. ERBF performed most of the time, so I considered ERBF as my kernel function of SVM classification. Table 4.4 gives a comparison of the performance of ERBF and RBF functions considering different $\sigma^2$ values.

<table>
<thead>
<tr>
<th>$\sigma^2$</th>
<th>RBF</th>
<th>ERBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>99.2891</td>
<td>99.2857</td>
</tr>
<tr>
<td>.5</td>
<td>99.2958</td>
<td>99.3258</td>
</tr>
<tr>
<td>.7</td>
<td>99.2042</td>
<td>99.3258</td>
</tr>
<tr>
<td>1</td>
<td>99.1151</td>
<td>99.3228</td>
</tr>
</tbody>
</table>

**Table 4.4 Comparing Kernel functions (RBF and ERBF)**

The results clearly shows that ERBF performed better than RBF at most of the times with higher accuracy in SVM classification.
4.5 Experimenting with different $\sigma^2$ values for ERBF function

Fig 4.5.1: SVM classification with $S2=.05$

Accuracy = 99.3055

Fig 4.5.1: SVM classification with $S2=.2$

Accuracy = 99.2891

The Fig 4.5.1 and 4.5.2 represent two SVM classification graphs; considering different $\sigma^2$ ($S2$) values for the ERBF kernel. When the $\sigma^2$ value is really low (in Fig 1) it overt fits the classes and even though the classification accuracy is high but it can be a bad classifier for a different set of data. Whereas in the second fig the boundary looks smooth so I would consider the classification to be better and smoother than the first one.
4.6 Experimenting with PCA 10 least Eigen value dimensions

In this experiment I considered the last 10 Eigen valued dimensions and used SVM, k-NN and Parzen window to classify the data. The accuracies were really low as shown in the below table.

<table>
<thead>
<tr>
<th></th>
<th>All dimensions</th>
<th>PCA last 10 dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>99.3258</td>
<td>0.0130</td>
</tr>
<tr>
<td>k-NN</td>
<td>78.5579</td>
<td>0.0000</td>
</tr>
<tr>
<td>Parzen Window</td>
<td>70.9677</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Table 4.5: Classification performance with PCA last 10 dimensions

The results show that the PCA last 10 dimensions really doesn’t help in proper classification. They horribly decrease the classifier performance.

5. Conclusion

The five experiments conducted on the QSAR biodegradation data come up with good results. From the first experiment it is clear that decreasing the dimensions using PCA decreased the accuracy of the k-NN classifier. It can be because the other dimensions (lost dimensions) had some data which were useful samples which helped k-NN to classify the data when it had all the dimension to classify each test sample. The second experiment was on SVM classifier and the classification accuracy was way too higher than k-NN. The classification accuracy decreased negligibly by decreasing the dimensions. It seems like SVM has a very high classification accuracy even if we consider just the best Eigen value PCA dimensions. In the third experiment the Parzen window performed a little different than what I expected. The classification accuracy got increased when I decreased the dimensions from all 41 to just 20 best Eigen valued PCA dimensions. And again it decreased its performance when I decreased the dimensions from 20 to 5. It could be because the lower dimension were the reason of some misclassification. So when the classification was done with just 20 best dimensions the classification was more accurate. Again decreasing the dimensions further to 5 decreased the accuracy in Parzen window is just a similar case as k-NN and SVM. In the fourth experiment we can clearly visualize that ERBF performs better than RBF kernel function. There may not be an outstanding difference between their performances but I always prefer to choose a better kernel function over something which has a lesser classifying kernel function. Here ERBF does a better support vector fit than RBF. The fifth experiment was to choose a better \( \sigma^2 \) value for ERBF kernel function. Making small enough i.e. close to 0.01, SVM makes an over fit classification which I think is not a good way of classification. So a value close .1 makes it a better classification than much lower value. The fifth experiment was a bit interesting where I compared the results of classification when the considered dimensions were the least 10 Eigen valued dimensions of the dataset. The classifiers just couldn’t classify the data properly. The accuracies were way lower. It can be because the dimensions with least Eigen values doesn’t contain a decent correlation of the two classes and so no way helping the classifiers to classify the data.
The basic reason why I think SVM worked better than k-NN and Parzen window is the data distribution. Fig 4.5.1 and 4.5.2 we can see that the data of both the classes are very much overlapped. SVM could consider support vectors to classify the data properly but k-NN and Parzen window always considered the training samples around them and when the test samples were close to the boundary they always got misclassified.

The efficiency of k-NN and Parzen window could be improved by considering a better dimension selection algorithm or using weighted k-NN and Parzen window classification. From the literature we know that C often act as a regularization term and can be tweaked to obtain a sparser SVM classification. Thus incorporating a varying C value and identifying an upper bound for C will strengthen the findings of our project.

6. Reference:

9. Code from: Author: Steve Gunn (srg@ecs.soton.ac.uk)

7. Appendix

A. QSAR Biodegradation Data Features/Attributes

1) SpMax_L: Leading eigenvalue from Laplace matrix
2) J_Dz(e): Balaban-like index from Barysz matrix weighted by Sanderson electronegativity
3) nHM: Number of heavy atoms
4) F01[N-N]: Frequency of N-N at topological distance 1
5) F04[C-N]: Frequency of C-N at topological distance 4
6) NssssC: Number of atoms of type ssssC
7) nCb: Number of substituted benzene C(sp2)
8) C%: Percentage of C atoms
9) nCp: Number of terminal primary C(sp3)
10) nO: Number of oxygen atoms
11) F03[C-N]: Frequency of C-N at topological distance 3
12) SdssC: Sum of dssC E-states
13) HyWi_B(m): Hyper-Wiener-like index (log function) from Burden matrix weighted by mass
14) LOC: Lopping centric index
15) SM6_L: Spectral moment of order 6 from Laplace matrix
16) F03[C-O]: Frequency of C-O at topological distance 3
17) Me: Mean atomic Sanderson electronegativity (scaled on Carbon atom)
18) Mi: Mean first ionization potential (scaled on Carbon atom)
19) nN-N: Number of N hydrazines
20) nArNO2: Number of nitro groups (aromatic)
21) nCRX3: Number of CRX3
22) SpPosA_B(p): Normalized spectral positive sum from Burden matrix weighted by polarizability
23) nCIR: Number of circuits
24) B01[C-Br]: Presence/absence of C-Br at topological distance 1
25) B03[C-Cl]: Presence/absence of C-Cl at topological distance 3
26) N-073: Ar2NH / Ar3N / Ar2N-Al / R..N..R
27) SpMax_A: Leading eigenvalue from adjacency matrix (Lovasz-Pelikan index)
28) Psi_i_1d: Intrinsic state pseudoconnectivity index - type 1d
29) B04[C-Br]: Presence/absence of C-Br at topological distance 4
30) SdO: Sum of dO E-states
31) TI2_L: Second Mohar index from Laplace matrix
32) nCrt: Number of ring tertiary C(sp3)
33) C-026: R--CX--R
34) F02[C-N]: Frequency of C-N at topological distance 2
35) nHDon: Number of donor atoms for H-bonds (N and O)
36) SpMax_B(m): Leading eigenvalue from Burden matrix weighted by mass
37) Psi_i_A: Intrinsic state pseudoconnectivity index - type S average
38) nN: Number of Nitrogen atoms
39) SM6_B(m): Spectral moment of order 6 from Burden matrix weighted by mass
40) nArCOOR: Number of esters (aromatic)
41) nX: Number of halogen atoms
42) experimental class: ready biodegradable (RB) and not ready biodegradable (NRB)

B. Importing Data

The data set has two classes’ i.e. ready biodegradable or not ready biodegradable. The first 41 attributes represent different chemical characteristics of the chemical samples, whereas the last attribute (i.e. the 42nd attribute) defines the class of the sample. For ready biodegradable it denotes as ‘RB’ and for not ready degradable it denotes as ‘NRB’. This dataset was imported to MATLAB, changing the ‘RB’ values to 0 and ‘NRB’ values to 1 (getting bio.mat file).

C. DATA SET

https://archive.ics.uci.edu/ml/datasets/QSAR+biodegradation


Kamel Mansouri, Tine Ringsted, Davide Ballabio (davide.ballabio ‘@’ unimib.it), Roberto Todeschini, Viviana Consonni, Milano Chemometrics and QSAR Research Group (http://michem.disat.unimib.it/chm/), UniversitÃ degli Studi Milano â€“ Bicocca, Milano (Italy)

D. SVM Code

http://www.isis.ecs.soton.ac.uk/resources/svminfo/