Proximal Support Vector Machine Classification on Seismic Data – Program psvm3d

Overview

Support vector machine (SVM) is a recent supervised machine learning technique that is widely used in text detection, image recognition and protein classification. In exploration geophysics, it can be used in seismic facies classification, petrophysics parameter estimation, and correlation of seismic attributes with engineering data. Proximal support vector machine (PSVM) is a variant of SVM, which has comparable classification performance to standard SVM but at considerable computational savings (Fung and Mangasarian, 2001, 2005; Mangasarian and Wild, 2006) that is critical when handling large 3D seismic surveys. This documentation provides an overview of the arithmetic of PSVM and step-by-step instruction on an AASPI implementation of PSVM for seismic data – psvm3d.

Comparing to the most popular artificial neural network (ANN) algorithms that are available in many commercial software, SVM and its variants benefit from the fact that they are based on convex optimization which is free of local minima (Shawe-Taylor and Cristianini, 2004), therefore provide a constant and robust classifier once training samples and model parameters are determined. Such classifier can then generate stable, reproducible classification result (Bennett and Campbell, 2000). Also, SVM has fewer parameters to pick than ANNs and the number of kernel functions is automatically selected, which makes it easier to reach the optimal model (Bennett and Campbell, 2000). Some researchers have compared the capability of SVM with ANN in pressure-wave velocity prediction in mining geophysics (Verma et al., 2014) and other non-geophysics disciplines (Wong and Hsu, 2005; Balabin and Lomakina, 2011) and found SVM is superior in most cases.

Theory of PSVM
Because SVMs are originally developed to solve binary classification problems, the arithmetic we show here is the steps to generate a binary PSVM classifier. Strategy of extending binary PSVM to a multiclass classifier is in later part of this chapter.

Similarly to SVM, a PSVM decision condition is defined as (Figure 1):

\[
\begin{align*}
\mathbf{x}'\mathbf{\omega} - \gamma &= 0, \quad \mathbf{x} \in A^+; \\
\mathbf{x}'\mathbf{\omega} - \gamma &> 0, \quad \mathbf{x} \in A^+ \text{ or } A^-; \\
\mathbf{x}'\mathbf{\omega} - \gamma &< 0, \quad \mathbf{x} \in A^-,
\end{align*}
\]

where \( \mathbf{x} \in \mathbb{R}^n \) is a \( n \) dimensional vector data point to be classified, \( \mathbf{\omega} \in \mathbb{R}^n \) implicitly defines the normal of the decision-boundary, \( \gamma \in \mathbb{R} \) defines the location of the decision-boundary, and “\( A^+ \)” and “\( A^- \)” are two classes of the binary classification. PSVM solves an optimization problem and takes the form of (Fung and Mangasarian, 2001):

\[
\min_{\mathbf{\omega}, \gamma, \mathbf{y}} \frac{1}{2} \|\mathbf{y}\|^2 + \frac{1}{2} (\mathbf{\omega}'\mathbf{\omega} + \gamma^2),
\]
Figure 1. (a) Scratch of a two-class PSVM in 2-D space. Class “A+” and “A−” are approximated by two parallel lines that being pushed as far apart as possible. The decision boundary then sits right at the middle of these two lines. In this case, maximizing the margin is equivalent to minimizing $(\mathbf{W}^T \mathbf{W} + \gamma^2)^{1/2}$. (b) Two-class PSVM in 3D space. In this case the decision boundary becomes a plane.

\[ D(A\omega - e\gamma) + y = e. \] (3)

In this optimization problem, $y \in \mathbb{R}^m$ is the error variable; $A \in \mathbb{R}^{m \times n}$ is a sample matrix composed of $m$ samples, which can be divided into two classes, $A+$ and $A−$; $D \in \mathbb{R}^{m \times m}$ is a diagonal matrix of labels with a diagonal composed of “+1” for $A+$ and “−1” for $A−$; $\nu$ is a non-negative parameter; and $e \in \mathbb{R}^m$ is a column vector of ones. This optimization problem can be solved by using a Lagrangian multiplier $u \in \mathbb{R}^m$:

\[ L(\omega, \gamma, y, u) = \nu^2 ||y||^2 + \frac{1}{2} (\omega^T \omega + \gamma^2) - u^T (D(A\omega - e\gamma) + y - e). \] (4)

By setting the gradients of $L$ to zero, we obtain expressions for $\omega$, $\gamma$ and $y$ explicitly in the knowns and $u$, where $u$ can further be represented by $A$, $D$ and $v$. Then by changing $\omega$ in equations 2 and 3 using its dual equivalent $\omega = A^T D u$, we can arrive at (Fung and Mangasarian, 2001):

\[ \min_{\omega, \gamma, y} \nu^2 ||y||^2 + \frac{1}{2} (u^T u + \gamma^2), \] (5)

subject to

\[ D(AA^T D u - e\gamma) + y = e. \] (6)

Equations 5 and 6 provide a more desirable version of the optimization problem since one can now insert kernel methods to solve nonlinear classification problems made possible by the term $AA^T$ in Equation 6. Utilizing the Lagrangian multiplier again (this time we denote the multiplier as $v$), we can minimize the new optimization problem against $u$, $\gamma$, $y$ and $v$. By setting the gradients of these four variables to zero, we can express $u$, $\gamma$, $y$ and $v$ explicitly by $v$ and other knowns, where $v$ is solely a dependent on the data matrices. Then for $x \in \mathbb{R}^{1 \times n}$ we write the decision conditions as

\[ x^T A^T D u - \gamma \begin{cases} > 0, & x \in A+; \\ = 0, & x \in A + or A−; \\ < 0, & x \in A−, \end{cases} \] (7)

with

\[ u = D K^T \left( \frac{1}{v} + GG^T \right)^{-1} e, \] (8)

\[ \gamma = e^T D \left( \frac{1}{v} + GG^T \right)^{-1} e, \] (9)
Volumetric Classification: Program psvm3d

and

\[ G = D[K - e]. \] \hspace{1cm} (10)

Instead of \( A \), we have \( K \) in equations 8 and 10, which is a Gaussian kernel function of \( A \) and \( A' \) that has the form:

\[ K(A, A')_{i,j} = \exp\left(-\sigma \| A'_i - A'_j \|^2 \right), \quad i, j \in [1, m], \] \hspace{1cm} (11)

where \( \sigma \) is a scalar parameter. Finally, by replacing \( x'A' \) by its corresponding kernel expression, the decision condition can be written as:

\[ K(x', A')D_u - \gamma \begin{cases} > 0, & x \in A +; \\ = 0, & x \in A + or A -; \\ < 0, & x \in A -. \end{cases} \] \hspace{1cm} (12)

and

\[ K(x', A')_{i,j} = \exp(-\sigma \| x - A'_i \|^2), \quad i \in [1, m]. \] \hspace{1cm} (13)

The formulations above represent a nonlinear PSVM classifier.

To extend this binary classifier to handle multiclass classification problems, some strategies have been developed by researchers, which generally lie into three categories: “one-versus-all”, “one-versus-one” and “all together”. The former two strategies, as one can tell from the names, build several binary classifiers individually \((n(n - 1))/2 \) for “one-versus-one” and \( n \) for “one-versus-all”, where \( n \) is the number of class), then use these classifiers to conclude the final classification decision. While “all together” will solve multiclass problems in one step. Experiments conducted by some researchers indicate a superiority of “one-versus-one” methods on large problems for practical use (Hsu and Lin, 2002). There are two popular particular algorithms for “one-versus-one” strategies, namely “Max Wins” (Kreßel, 1999) and directed acyclic graph (DAG) (Platt et al., 2000). Both algorithms can give comparable results while surpassing the “one-versus-all” method in accuracy and computational efficiency. In our implementation, an approach similar to DAG is adopted and is described below.

How we assign a class to an unknown sample

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.3</td>
<td>-1.2</td>
<td>2.3</td>
</tr>
<tr>
<td>B</td>
<td>-0.3</td>
<td>0.8</td>
<td>-1.1</td>
</tr>
<tr>
<td>C</td>
<td>1.2</td>
<td>-0.8</td>
<td>-1.9</td>
</tr>
<tr>
<td>D</td>
<td>-2.3</td>
<td>1.1</td>
<td>1.9</td>
</tr>
</tbody>
</table>

Example of a classification factor table

For training samples of \( n \) classes, generate \( n(n-1)/2 \) binary classifiers for every two classes

Set class “A” as the pilot class

Turn all classes into active

Examine the binary PSVM classification factor (CF) of the current pilot class against every other active classes.

All CFs are positive?

Yes

Assign the current pilot class to this sample and exit

No

Find the class corresponds to the most negative CF value, then assign that class as the new pilot class, and turn the current pilot class into inactive.

Figure 2. Workflow of assigning a class to an unknown sample using a classification factor based scheme.
Our approach uses a classification factor table to assign classes to unknown samples. A classification factor of an unknown sample point for a certain pilot class “A” is the normalized distance to the binary decision boundary between “A” and the other class used when generating this binary decision boundary. An example of a classification factor table is shown in Figure 2, and based on this table, the unknown sample point belongs to class “D”.

**Computation flow chart**

The program **psvm3d** takes an ASCII format training file to build the PSVM classifier, then apply it to seismic taking multiple AASPI format seismic attribute input volumes. The training file consists of input/output pairs and can be constructed either from hand picking facies, or edited well logs. Several commercial software (e.g. point set in Petrel) and **make_training_clusters** can be used to manually pick facies of interest to be used as output in the training file. A well log property can also be used as output, as long as such property is discretized. After having the output ready, users can use **make_training_clusters** to extract attributes at corresponding locations to generate the input attributes in the training file. The flow chart of **psvm3d** is shown in Figure 3.

![Flow chart of program psvm3d](image)

**Figure 3.** Flow chart of program **psvm3d**.
Step-by-step instruction on program psvm3d

This Program psvm3d is launched from the Volumetric Classification in the main aaspi_util GUI (Figure 4).

Figure 4. How to launch program PSVM Well Log Analysis.

The interface of psvm3d is shown below. We will go through all the options in detail.
Figure 5. Interface of psvm3d.

**Button 1**: Browse training file.

Note: The program can only handle ASCII format training file, so please hand edit or using program make_training_clusters to generate a .txt or .dat file in the following format (Figure 6). The format is: from left to right, each column is an input dimension (e.g. one attribute), then a column of label (positive integer numbers for facies or a discrete property). All other columns after "label" are ignored. You can have arbitrary number of header lines, which will be skipped during importing the files. **The sequence of input attributes in the training file must agree with the input attributes file list!**

<table>
<thead>
<tr>
<th>label</th>
<th>extra columns... (e.g. MD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7502</td>
</tr>
<tr>
<td>2</td>
<td>7600.5</td>
</tr>
<tr>
<td>3</td>
<td>7923</td>
</tr>
<tr>
<td>4</td>
<td>8502.5</td>
</tr>
<tr>
<td>5</td>
<td>8768.8</td>
</tr>
<tr>
<td>6</td>
<td>8957.5</td>
</tr>
<tr>
<td>7</td>
<td>7705.5</td>
</tr>
<tr>
<td>8</td>
<td>8262.5</td>
</tr>
<tr>
<td>9</td>
<td>8516.5</td>
</tr>
<tr>
<td>10</td>
<td>8299.5</td>
</tr>
<tr>
<td>11</td>
<td>8517.5</td>
</tr>
<tr>
<td>12</td>
<td>8354.5</td>
</tr>
<tr>
<td>13</td>
<td>8225.5</td>
</tr>
<tr>
<td>14</td>
<td>7399.5</td>
</tr>
<tr>
<td>15</td>
<td>8403.5</td>
</tr>
<tr>
<td>16</td>
<td>7465.5</td>
</tr>
<tr>
<td>17</td>
<td>7679.5</td>
</tr>
<tr>
<td>18</td>
<td>8633.5</td>
</tr>
<tr>
<td>19</td>
<td>7413.5</td>
</tr>
<tr>
<td>20</td>
<td>8205.5</td>
</tr>
<tr>
<td>21</td>
<td>7910.5</td>
</tr>
<tr>
<td>22</td>
<td>8076.5</td>
</tr>
<tr>
<td>23</td>
<td>7393.5</td>
</tr>
<tr>
<td>24</td>
<td>7420.5</td>
</tr>
<tr>
<td>25</td>
<td>7826.5</td>
</tr>
<tr>
<td>26</td>
<td>7901.5</td>
</tr>
<tr>
<td>27</td>
<td>8118.5</td>
</tr>
<tr>
<td>28</td>
<td>8214.5</td>
</tr>
<tr>
<td>29</td>
<td>7412.5</td>
</tr>
<tr>
<td>30</td>
<td>7946.5</td>
</tr>
<tr>
<td>31</td>
<td>8018.5</td>
</tr>
<tr>
<td>32</td>
<td>8178.5</td>
</tr>
<tr>
<td>33</td>
<td>7460.5</td>
</tr>
<tr>
<td>34</td>
<td>8092.5</td>
</tr>
<tr>
<td>35</td>
<td>8528.5</td>
</tr>
</tbody>
</table>

Figure 6. An example of a supported training file format.

**Button 2**: View the training file content (Figure 7).

**Button 3**: If the file is generated from Windows based software (e.g. Petrel), they will have the annoying carriage return (^M) at the end of each line (Shown in Figure 7). Use this button to delete those carriage returns if you prefer to (result shown in Figure 8).

Note: This function depends on your Linux environment therefore may not always works. However it will not affect reading in the file.

**Buttons 4**: Browse input AASPI format attribute files. The files must agree with the order in the training file.

**Blank 5 and 6**: Project name and suffix. You can put the parameters as suffix.

**Blank 7**: Number of header lines to skip in the training file.

**Blank 8**: Number of input dimensions, i.e. number of attributes.
Blank 9: Number of classes (facies) within the data.  
Note: Classes that do not appear in the training file cannot be predicted.  
Blank 10 and 11: PSVM classifier parameters (must be positive real numbers).  
Generally, Blank 10 controls how tight the classifier fits the training data, which will  
scarcify the ability of generalization. Blank 11 is the standard deviation of a Gaussian  
function used in kernel mapping. The classifier’s performance is more sensitive to  
Blank 11 based on our test.  
Blank 12: Amount of samples out of the training file that are actually used for training.  
More training samples will have more computation cost, and sometimes not using all the  
available training samples may provide a more generalized classifier.  
Checkmark 13: Check if want to normalize attributes using z-score. Default setting is to  
normalize attributes, but in some circumstances it may produce better result without  
normalization.  
Button 14: Run the program.

Figure 7. An example of viewing a training file content. Carriage returns are visible as “\n”.

Figure 8. An example of a training file after deleting carriage returns.
Volumetric Classification: Program psvm3d

In the **Operation Window** tab, all functions are explained in Figure 9 (The panel shown is from *som3d*, and the *psvm3d* has an identical operation window panel).

**Figure 9.** Operation Window tab in *som3d* (identical to *psvm3d*).

**Figure 10.** (left) A gridded horizon file (EarthVision format). (right) An interpolated horizon file with five columns (ASCII free format).
Volumetric Classification: Program psvm3d
Horizon definition

The horizon definition panel will look the same for almost all AASPI GUIs:

1. Start time (upper boundary) of the analysis window.
2. End time (lower boundary of the analysis window.
3. Toggle that allows one to do the analysis between the top and bottom time slices described in 1 and 2 above, or alternatively between two imported horizons. If USE HORIZON is selected, all horizon related options will be enabled. If the horizons extend beyond the window limits defined in 1 and 2, the analysis window will be clipped.
4. Browse button to select the name of the upper (shallower) horizon.
5. Button that displays the horizon contents (see Figure 10).
6. Button to convert horizons from Windows to Linux format. If the files are generated from Windows based software (e.g. Petrel), they will have the annoying carriage return (^M) at the end of each line (Shown in Figure 10). Use these two buttons to delete those carriage returns. Note: This function depends on your Linux environment. If you do not have the program dos2unix it may not work. In these situations, the files may have been automatically converted to Linux and thus be properly read in.
7. Browse button to select the name of the lower (deeper) horizon.
8. Button that displays the horizon contents (see Figure 10).
9. Button to convert horizons from Windows to Linux format. (see 6 above).
10. Toggle that selects the horizon format. Currently gridded (e.g. EarthVision in Petrel) and interpolated (ASCII free format, e.g. SeisX) formats are supported. The gridded horizon are nodes of B-splines used in mapping and have no direct correlation to the seismic data survey. For example, gridded horizons may be computed simply from well tops. The x and y locations are aligned along north and east axes. In contrast interpolated horizons have are defined by line_no, cdp_no (crossline_no) and time triplets for each trace location. Examples of both format are shown in Figure 10. If interpolated is selected, the user needs to manually define each column in the file.
11. Number of header lines to skip in the interpolated horizon files.
12. Total number of columns in the interpolated horizon files.
13. Enter the column number containing the line_no (inline_no) of the interpolated data triplet.
14. Enter the column number containing the cdp_no (crossline_no) of the interpolated data triplet.
15. Enter the column number containing the time or depth value of the interpolated data triplet.
16. Znull value (indicate missing picks) in the horizon files.
17. Toggle to choose between positive down and negative down for the horizon files (e.g. Petrel uses negative down).
18. Choose the vertical units used to define the horizon files (either s, ms, kft, ft, km, or m).
Here we see an application using the program \texttt{psvm3d} to predict brittleness index (BI) from four inversion derived seismic attributes. Before running \texttt{psvm3d}, it is strongly advised to test the training data using program \texttt{PSVM Well Log Analysis} which is used for testing, validating, and predicting ASCII files. In \texttt{PSVM Well Log Analysis}, the user can manually divide the training file into two portions, training and testing, or use the whole training file in cross-validation model to find the best parameters. Once the optimal parameters are found, the user can use this training file, seismic attributes (in the corresponding order), and the PSVM parameters that tested out in \texttt{PSVM Well Log Analysis}, to perform classification using \texttt{psvm3d}. The parameters used in our example are shown in Figure 11.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{parameter_settings.png}
\caption{PSVM parameter settings for BI prediction.}
\end{figure}

As shown in Figure 11, we prepared a training file from one study well, and use the parameters listed in the panel. The input attributes are P-impedance, S-impedance, Poisson’s Ratio, and Lambda/Mhu Ratio, where the target properties is brittleness which is digitalized in to 10 classes, 1 being the least brittle and 10 being the most. In the \textbf{Operation Window} tab (Figure 12), we use two horizons as upper and lower limits of the operation window, and the horizons are \textit{interpolated}, which means we need to define the columns. And finally I choose to use 50 processors to speed up the job (Figure 13).
Volumetric Classification: Program psvm3d

Figure 12. Operation window options for BI prediction.

Figure 13. Extended (MPI) options for BI prediction.

After clicking the **Execute** button, the user is able to supervise the running progress (Figure 14).

Figure 14. Running progress windows.

Once finished, the user will locate the generated classification file in the same directory named as “psvm3d_classification_projectname_suffix.H”, which can be display using either the **AASPI_QC_Plot** or other commercial software. Figure 15 shows the final classification result displayed in Petrel.
Volumetric Classification: Program psvm3d

Figure 15. PSVM classification for brittleness index (Zhang et al., 2015).

References


