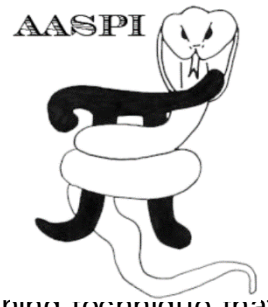


Proximal Support Vector Machine Classification on Well Logs

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 well log data – **psvm_welllogs**.

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-Tayler 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):

$$\mathbf{x}'\boldsymbol{\omega} - \gamma \begin{cases} > 0, & \mathbf{x} \in A+; \\ = 0, & \mathbf{x} \in A+ \, or \, A-; \\ < 0, & \mathbf{x} \in A-, \end{cases} \quad (1)$$

where $\mathbf{x} \in R^{\boldsymbol{n}}$ is a n dimensional vector data point to be classified, $\boldsymbol{\omega} \in R^{\boldsymbol{n}}$ implicitly defines the normal of the decision-boundary, $\gamma \in R$ 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):

$$\min_{\boldsymbol{\omega},\gamma,\mathbf{y}} \nu \frac{1}{2}\|\mathbf{y}\|^2 + \frac{1}{2}(\boldsymbol{\omega}'\boldsymbol{\omega} + \gamma^2), \quad (2)$$

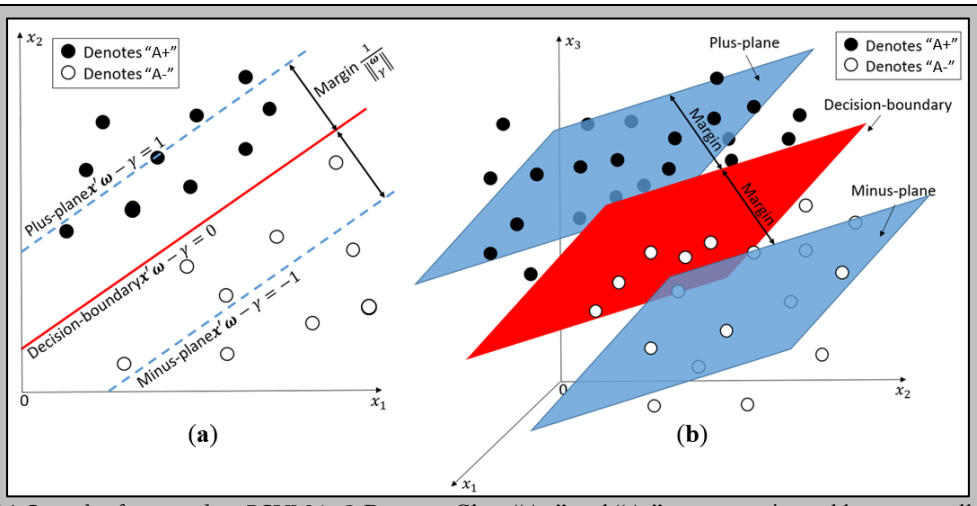


Figure 1. (a) Scratch of a two-class PSVM in 2-D space. Class "A+" and "A-" are approximated by two parallel lines that being pushed as far apart as possible. The decision boundary then sits right at the middle of these two lines. In this case, maximizing the margin is equivalent to minimizing $(\mathbf{W}^T\mathbf{W}+\gamma^2)^{1/2}$. (b) Two-class PSVM in 3D space. In this case the decision boundary becomes a plane.

$$\mathbf{D}(\mathbf{A}\boldsymbol{\omega}-\mathbf{e}\gamma)+\mathbf{y}=\mathbf{e}. \tag{3}$$

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

$$L(\boldsymbol{\omega},\gamma,\mathbf{y},\mathbf{u})=\nu\frac{1}{2}\|\mathbf{y}\|^2+\frac{1}{2}(\boldsymbol{\omega}'\boldsymbol{\omega}+\gamma^2)-\mathbf{u}'(\mathbf{D}(\mathbf{A}\boldsymbol{\omega}-\mathbf{e}\gamma)+\mathbf{y}-\mathbf{e}). \tag{4}$$

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

$$\min_{\boldsymbol{\omega},\gamma,\mathbf{y}}\nu\frac{1}{2}\|\mathbf{y}\|^2+\frac{1}{2}(\mathbf{u}'\mathbf{u}+\gamma^2), \tag{5}$$

subject to

$$\mathbf{D}(\mathbf{A}\mathbf{A}'\mathbf{D}\mathbf{u}-\mathbf{e}\gamma)+\mathbf{y}=\mathbf{e}. \tag{6}$$

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 $\mathbf{A}\mathbf{A}'$ in Equation 6. Utilizing the Lagrangian multiplier again (this time we denote the multiplier as $\mathbf{v}$), we can minimize the new optimization problem against $\mathbf{u}$, γ, $\mathbf{y}$ and $\mathbf{v}$. By setting the gradients of these four variables to zero, we can express $\mathbf{u}$, γ and $\mathbf{y}$ explicitly by $\mathbf{v}$ and other knowns, where $\mathbf{v}$ is solely a dependent on the data matrices. Then for $\mathbf{x}\in R^{1\times n}$ we write the decision conditions as

$$\mathbf{x}'\mathbf{A}'\mathbf{D}\mathbf{u}-\gamma\begin{cases}>0, & \mathbf{x}\in A+;\\ =0, & \mathbf{x}\in A+\ or\ A-;\\ <0, & \mathbf{x}\in A-,\end{cases} \tag{7}$$

with

$$\mathbf{u}=\mathbf{D}\mathbf{K}'\mathbf{D}\left(\frac{\mathbf{I}}{\nu}+\mathbf{G}\mathbf{G}'\right)^{-1}\mathbf{e}, \tag{8}$$

$$\gamma=e'D\left(\frac{\mathbf{I}}{\nu}+\mathbf{G}\mathbf{G}'\right)^{-1}\mathbf{e}, \tag{9}$$

and

$$\mathbf{G} = \mathbf{D}[\mathbf{K} \quad -\mathbf{e}]. \tag{10}$$

Instead of **A**, we have **K** in equations 8 and 10, which is a Gaussian kernel function of **A** and $\mathbf{A}'$ that has the form:

$$\mathbf{K}(\mathbf{A}, \mathbf{A}')_{ij} = \exp\left(-\sigma \left\| \mathbf{A}'_{i\cdot} - \mathbf{A}'_{j\cdot} \right\|^2\right), i, j \in [1, m], \tag{11}$$

where σ is a scalar parameter. Finally, by replacing $\mathbf{x}'\mathbf{A}'$ by its corresponding kernel expression, the decision condition can be written as:

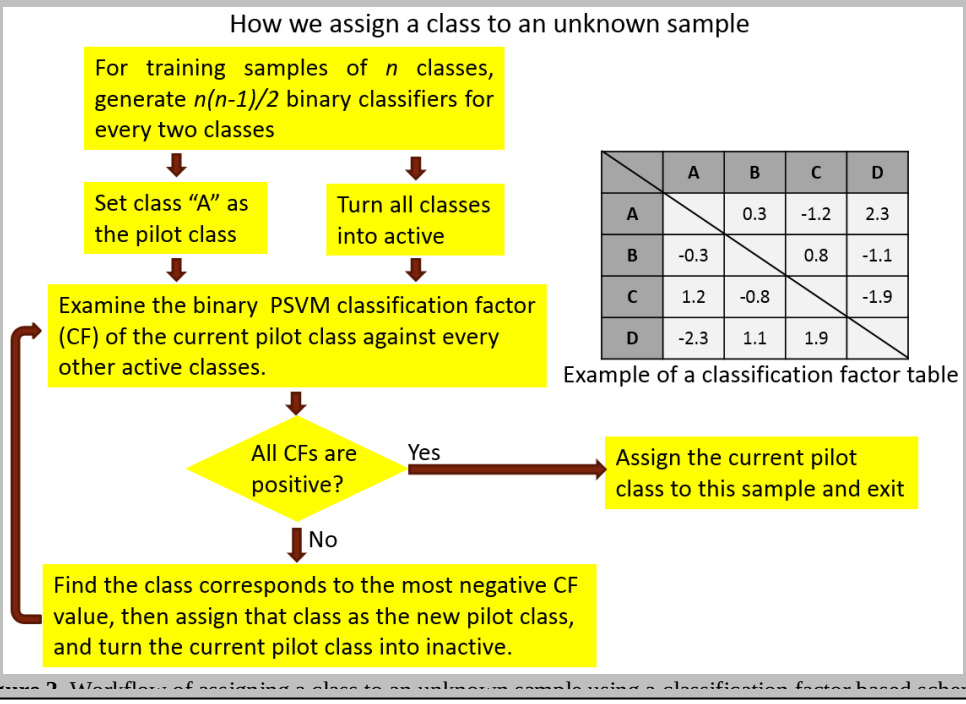
$$\mathbf{K}(\mathbf{x}', \mathbf{A}')\mathbf{D}\mathbf{u} - \gamma \begin{cases} > 0, & \mathbf{x} \in A+; \\ = 0, & \mathbf{x} \in A+ \text{ or } A-; \\ < 0, & \mathbf{x} \in A-. \end{cases} \tag{12}$$

and

$$\mathbf{K}(\mathbf{x}', \mathbf{A}')_{ij} = \exp(-\sigma \|\mathbf{x} - \mathbf{A}'_{i\cdot}\|^2), i \in [1, m]. \tag{13}$$

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.



	A	B	C	D
A		0.3	-1.2	2.3
B	-0.3		0.8	-1.1
C	1.2	-0.8		-1.9
D	-2.3	1.1	1.9	

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 **psvm_welllogs** has three running modes: **testing**, **predicting**, and **cross-validation**. "**Testing**" reads in a training file and a testing file both with known labels (both in ASCII format, will describe in details in the later parts of this documentation), and output testing correctness, correlation coefficient, and a file containing the testing label outputs. "**Predicting**" reads in a training file with known labels and a testing file without labels (to be predicted), and output a file containing the predicted label outputs. "**Cross-validation**" is another method of testing, which reads in only a training file with labels, then randomly selects a user defined size from this file for testing, and uses the remaining portion as training. Times of iterations can be assigned for cross-validation, and the training and testing samples are selected randomly for each iteration. The output file of "**cross-validation**" consists a probability distribution after all iterations. Flow chart is shown in Figure 3.

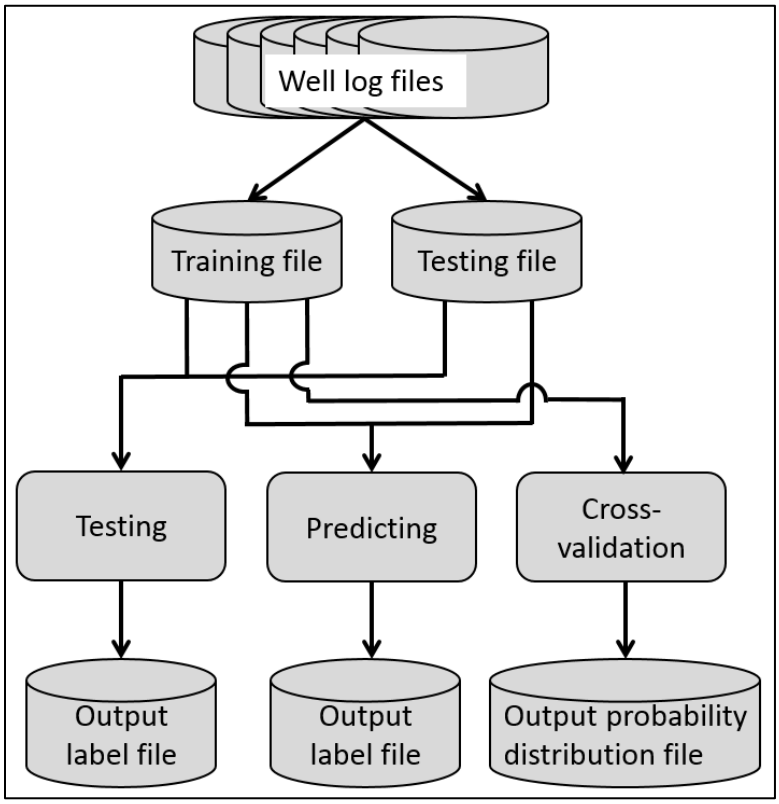


Figure 3. Flow chart of program **psvm_welllogs**consists of three running modes.

Step-by-step instruction on program PSVM Well Log Analysis

This Program **psvm_welllogs** is launched from the *Formation Attributes* in the main **aaspi_util** GUI (Figure 4).

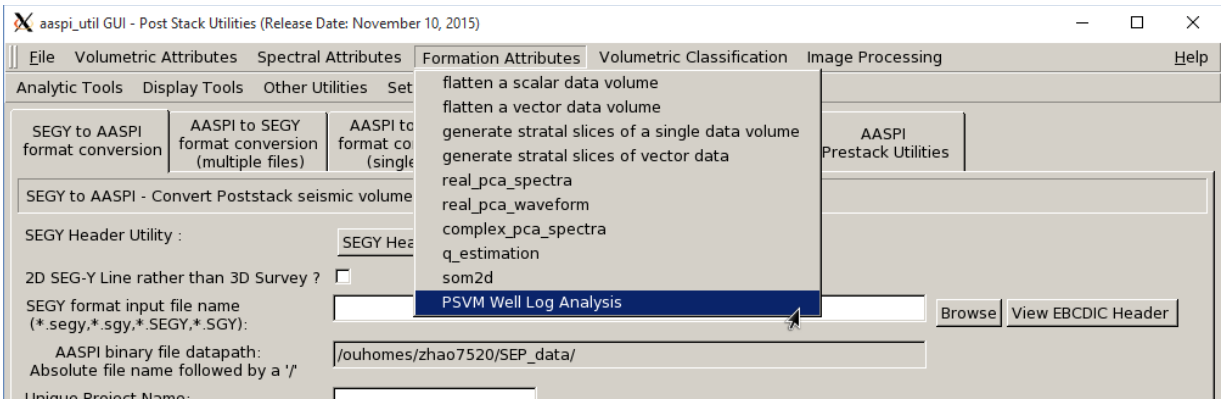


Figure 4. How to launch program **psvm_welllogs**.

The interface of **psvm_welllogs** is shown below. We will go through all the options in detail.

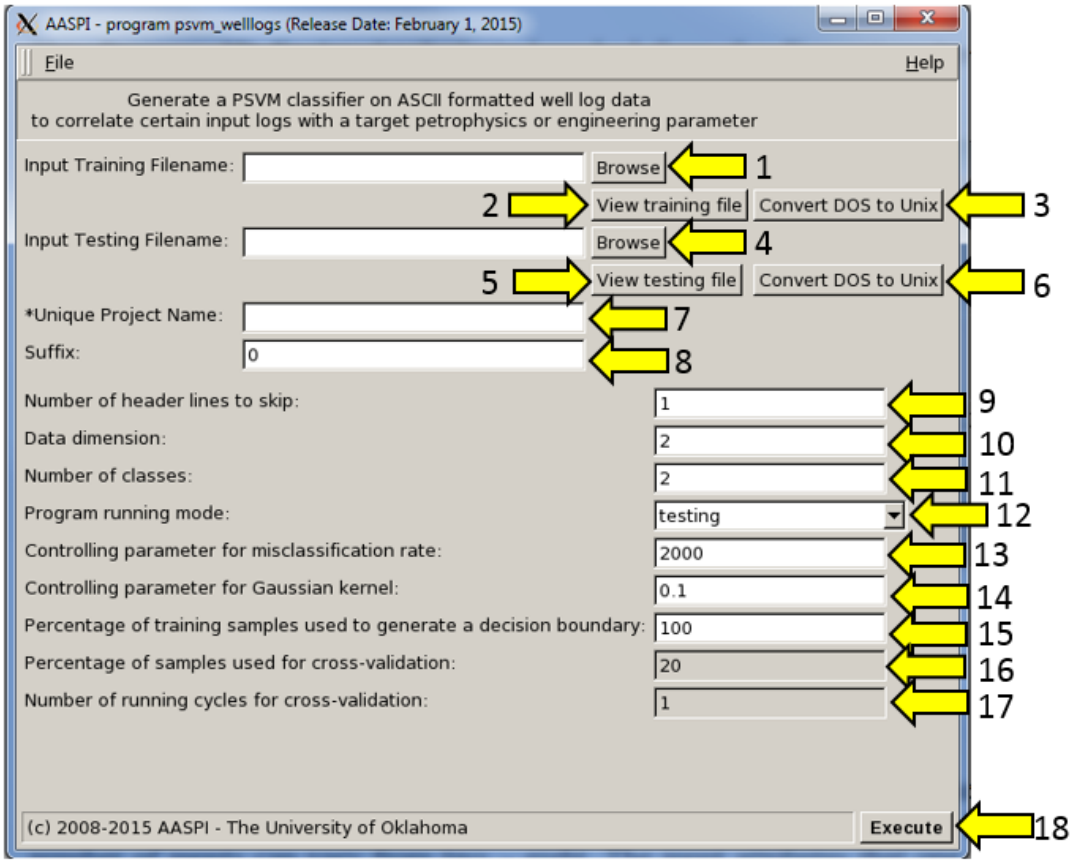


Figure 5. Interface of **PSVM Well Log Analysis**.

Button 1 and **4**: Browse input training and testing files. Testing file turns gray when **Button 12** is in "cross-validation", in which only training file is used.

Note: currently the program can only handle simple ASCII format files, so please generate a .txt or .dat file in the following format (Figure 6) from your well log files (.las). The format is: from left to right, each column is an input dimension (e.g. one well log or other types of data), then is a column of label (positive integer numbers). All other columns after "label" are ignored. You can have arbitrary number of header lines, which will be skipped during importing the files. Also remember to **put the same property in the same column in both training and testing files**.

log1	log2	log3	log4	label	extra columns... (e.g. MD)
51841.64063	26935.35742	1.704349708	3811383228	1	7502
33774.44531	17715.05469	1.634891917	1656611034	6	7660.5
50315.04688	26632.26758	1.56927077	3504752383	1	7923
53169.19922	27760.99414	1.6681764	4051182345	1	8502.5
34095.53906	16539.62695	2.249557077	1845954646	2	7868.5
48726.94141	24698.78906	1.89212672	3458510965	1	8597.5
38455.96484	19724.96484	1.800974794	2169707788	2	7705
45310.55469	24011.8457	1.560800845	3026602008	7	8262.5
52936.375	27013.35938	1.840176505	4029289962	1	7490
28180.46094	15991.72363	1.105314593	1059605743	6	8099
54165.91797	27750.04492	1.809998668	4244495243	1	8516.5
29320.0625	17191.31055	0.908785852	1142603987	8	8299.5
54166.74609	27742.24023	1.812259132	4257033610	1	8517
53954.65625	27858.05859	1.7510788	4216501385	1	8534.5
31412.4375	19265.32031	0.658586632	1171121773	9	8225.5
51861.29688	27149.27539	1.648970041	3758128825	1	7399.5
30382.01758	18928.20703	0.576405696	1066315103	8	8403
52596.90234	27241.33789	1.727893156	3928474660	1	7465
45016.66406	24360.98633	1.414734147	2743924255	1	7679
54823.67969	28755.60742	1.634889911	4182686811	1	8633.5
51951.76172	27140.26953	1.664141484	3777469633	1	7413.5
35710.83203	20000.30664	1.188061043	1720712563	8	8205.5
50521.89844	26961.88867	1.511225421	3523451743	1	7910
27744.17773	14988.05859	1.426517131	1118232594	2	8076.5
52579.58984	27344.14258	1.697479367	3906468779	1	7393.5
50714.21875	26632.41211	1.626089165	3592529109	1	7430
52662.21875	27469.63477	1.67529773	3906856425	1	7826
39965.76563	20995.17969	1.623573544	2275429956	1	7901.5
34559.80859	19429.59961	1.163842922	1624689227	4	8118.5
32354.3418	19198.0332	0.8402178	1324971963	8	8214
51962.21094	27150.45313	1.662866463	3792142635	1	7412
49247.73047	25697.74219	1.672675537	3424101737	1	7946
30129.89648	17727.28516	0.888755428	1134988504	8	8018
28798.59961	17923.15039	0.581749341	1000516282	10	8178.5
52114.26563	27085.1582	1.702120226	3833284544	1	7460.5
31283.79102	17445.36719	1.215722451	1320017984	5	8092
53879.60938	28386.38281	1.602705695	4044150337	1	8528.5

Figure 6. An example of a supported file format.

Button 2 and **5**: View the training and testing file contents (Figure 7).
Button 3 and **6**: 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 7). Use these two buttons 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 files**.
Blank 7 and **8**: Project name and suffix. You can put the parameters as suffix.
Blank 9: Number of header lines to skip when importing files. Currently this value is used for both training and testing files so please keep these two files in the same format.
Blank 10: Number of input dimensions, i.e. number of columns before "label".
Blank 11: Number of classes within the data.

Note: Classes that do not appear in the training file cannot be predicted.

Button 12: Program running mode selection. Please refer to the **Computation flow chart** chapter for details.

Blank 13 and **14**: PSVM classifier parameters (must be positive real numbers). Generally, **Blank 13** controls how tight the classifier fits the training data, which will scarify the ability of generalization. **Blank 14** is the standard deviation of a Gaussian function used in kernel mapping. **The classifier's performance is more sensitive to Blank 14 based on our study**.

Blank 15: 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.

Blank 16 and **17**: Defines how many samples out of the training file will be randomly selected and used for testing in cross-validation, and the number of cross-validation cycles a user wants to run. For each cross-validation cycle, a new testing group will be randomly selected.

Button 18: Run the program.

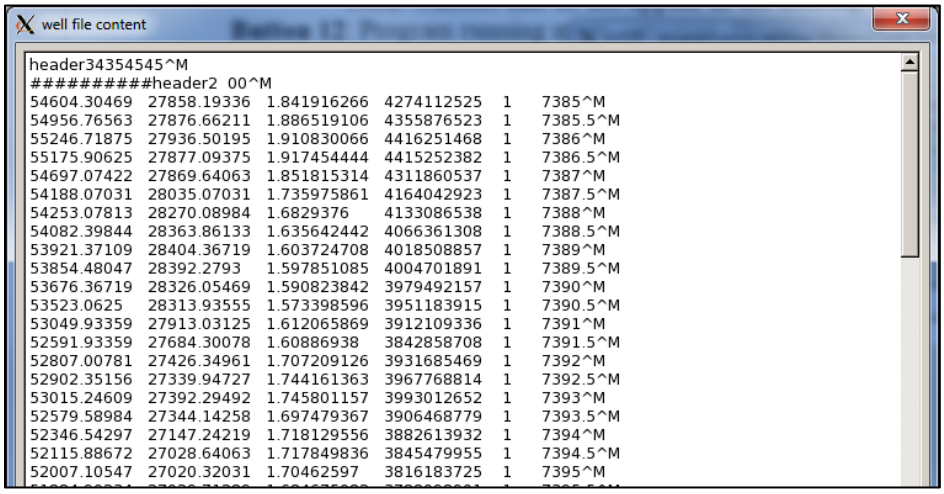


Figure 7. An example of viewing a well file content. Carriage returns are visible as "^M".

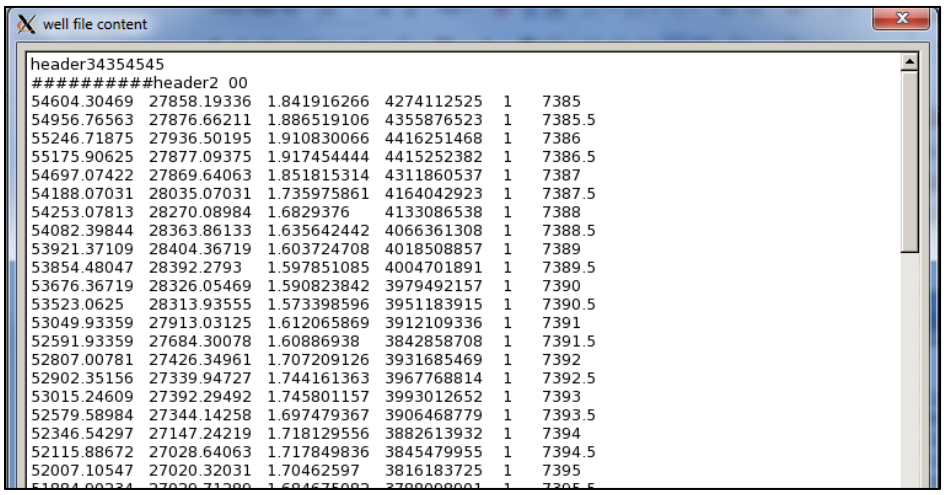


Figure 8. An example of a well file after deleting carriage returns.

Here we see an application using the program **psvm_welllogs** to predict brittleness index (BI) from four well log derived rock properties.

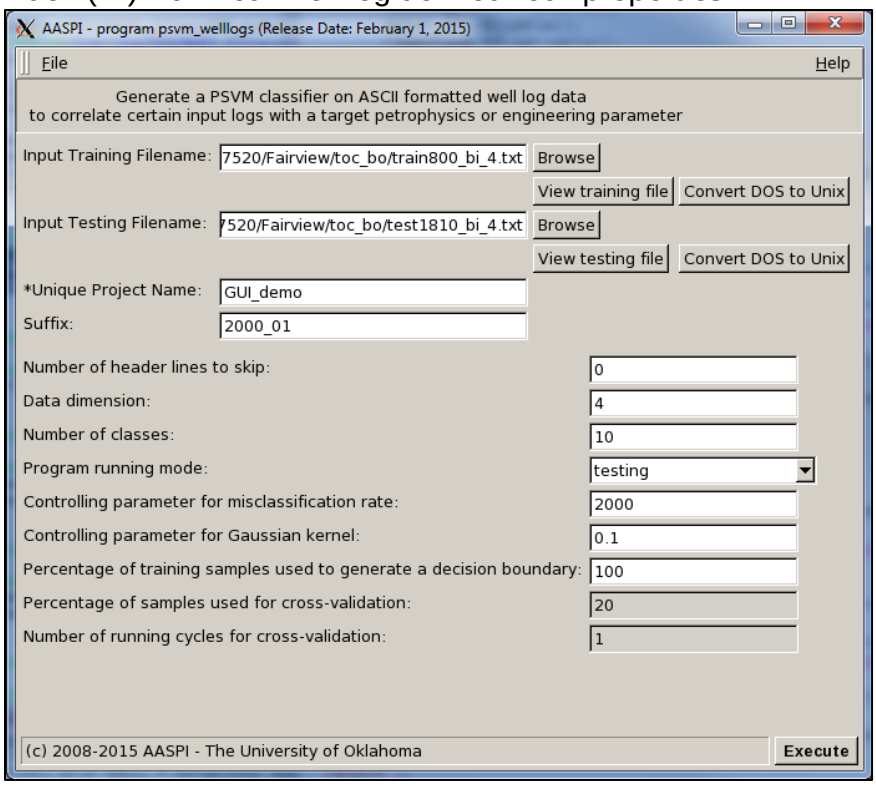


Figure 9. Parameter settings for BI prediction.

As shown in Figure 9, we prepared a training file (shown in Figures 7 and 8) and a testing file from one study well, and use the parameters listed in the panel. The input logs are P-impedance, S-impedance, Lambda/ Mho, and Young's Modulus/ Poisson's Ratio, where the target properties is brittleness which is digitalized in to 10 classes, 1 being the least brittle and 10 being the most. After clicking the **Execute** button, users are able to view the running progress (Figure 10).

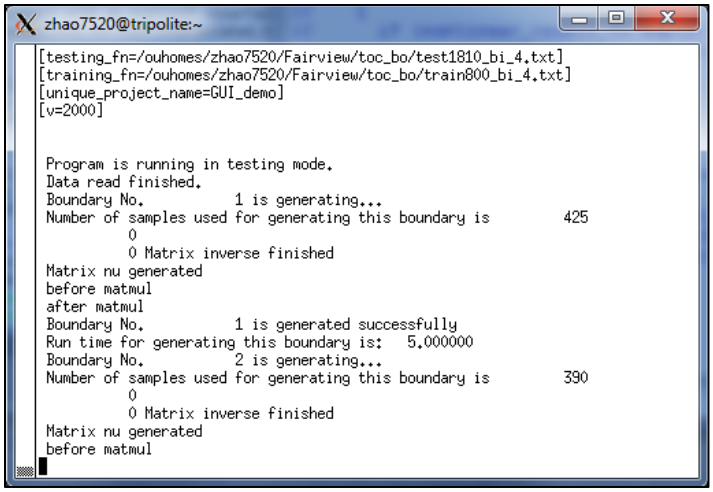


Figure 10. Running progress window.

Once finished, the correlation coefficient of the classification is given on the screen and a file containing testing result is generated (Figure 11). Users can plot the file in excel for QC. If the result is satisfactory, users can move to **predicting** mode and take a file need to be predicted as the testing file. The output file just contain one column of predicted labels, so it needs to be merged or imported to the corresponding well log file and further displayed in commercial software. Here we show a testing result on the previously used well displayed in Petrel (Figure 12). Roughly 30% of the samples are used for training.

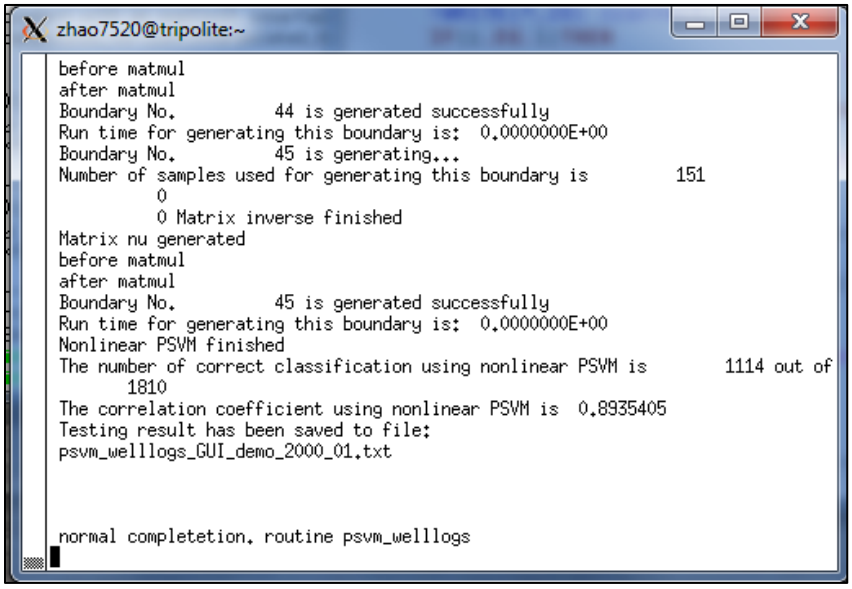


Figure 11. Completion of the program in **testing** mode.

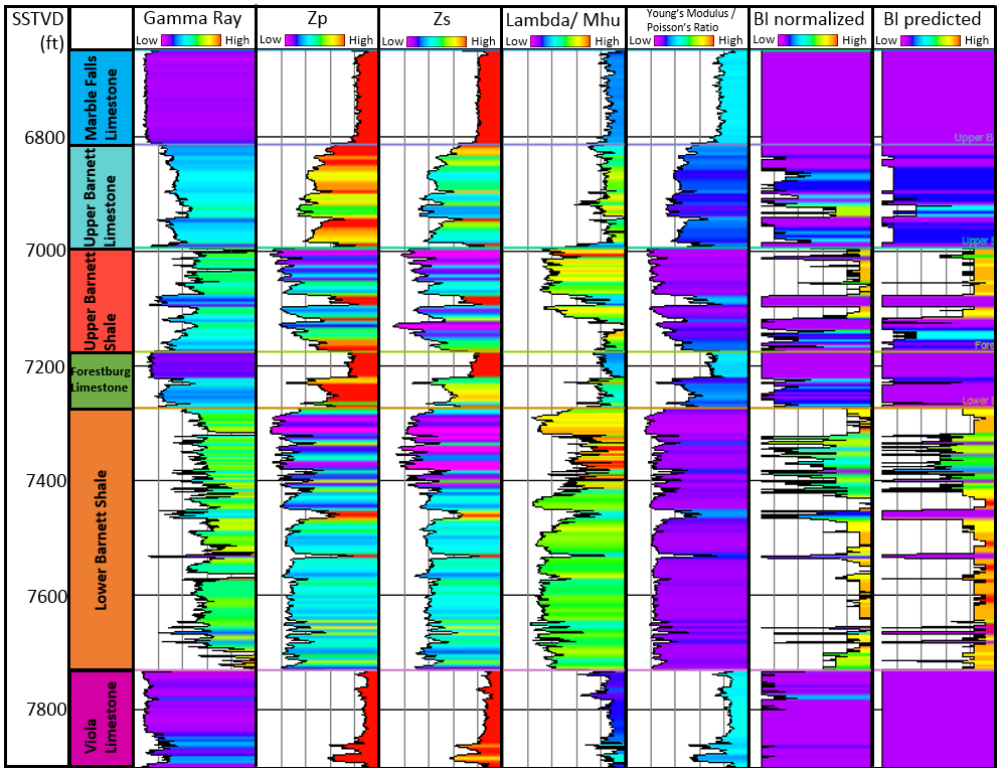


Figure 12. BI prediction on a well using four well log derived properties. The Gamma Ray log is plotted as a lithology reference. Good correlation can be identified between original and predicted BI logs.

In **cross-validation** mode (Figure 13), only a training file is used, and testing samples are randomly selected from the training file. In this example, for every one out of the ten cycles, 50% of the samples are used for testing, and the remaining portion are used for training. Once finished, it will generated a probability distribution file as shown in Figure 14.

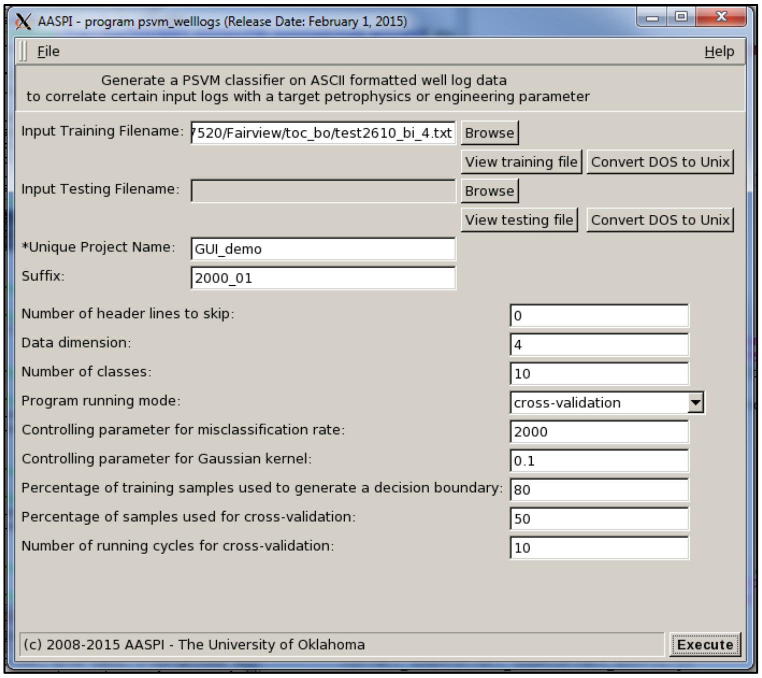


Figure 13. Cross-validation mode parameter setting.

Probability (%) of each sample against each class:

class 1	class 2	class 3	class 4	class 5	class 6	class 7	class 8	class 9	class 10
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 14. Probability distribution file generated by **cross-validation** mode (it happens to be solid class 1 for these samples).

References

Balabin, R. M. and E. I. Lomakina, 2011, Support vector machine regression (SVR/LS-SVM)—an alternative to neural networks (ANN) for analytical chemistry?

Comparison of nonlinear methods on near infrared (NIR) spectroscopy data: Analyst, **136**, 1703-1712.

Bennett, K. P. and A. Demiriz, 1999, Semi-supervised support vector machines: Advances in Neural Information Processing Systems 11: Proceedings of the 1998 Conference, 368-374.

Fung, G. and O. L. Mangasarian, 2001, Proximal support vector machine classifiers: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM 2001, 77-86.

Fung, G. M. and O. L. Mangasarian, 2005, Multicategory proximal support vector machine classifiers: Machine Learning, **59**, 77-97.

Mangasarian, O. L. and E. W. Wild, 2006, Multisurface proximal support vector machine classification via generalized eigenvalues: IEEE Transactions on Pattern Analysis and Machine Intelligence, **28**, 69-74.

Shawe-Taylor, J. and N. Cristianini, 2004, Kernel methods for pattern analysis: Cambridge University Press, New York, United States.

Verma, A. K., T. N. Singh and S. Maheshwar, 2014, Comparative study of intelligent prediction models for pressure wave velocity: Journal of Geosciences and Geomatics, **2**, 130-138.

Wong, W. and S. Hsu, 2006, Application of SVM and ANN for image retrieval: European Journal of Operational Research, **173**, 938-950.