A Simple Implementation Example of SVC[1]

Koruk, Kasimcan (kasimcan.koruk@queensu.ca)

Abstract

Support Vector Machines (SVM) is a powerful method in the world of machine learning, because it can handle complex regression and classification problems. Today, many machine learning libraries offer robust modules for several methods, and Scikit-learn's Support Vector Classifier (SVC) is one of them. This article's objective is to determine the boundaries of a binary-class problem using SVC. In this context, the article shows a simple example of SVC on a 3-D drillhole data. To see the potential of SVC, it is applied on 3D pseudo-data which are comprised of composite samples of vertical drillholes distributed homogenously in space. Cross-validation is applied on the data to determine the optimum parameters of the method. Posterior probabilities are obtained thanks to Platt's method embedded in SVC module. The data is split into train and test datasets to assess the accuracy of the model. The article shows that SVC can provide complex and accurate models at a cost of possible misclassifications of outliers in the multi-dimensional space.

1. Introduction

Support Vector Machine (SVM) is a powerful method for machine learning because it can handle complex regression and classification problems (Bishop, 2006) and it has been a popular method for many years (Géron, 2017). The name of SVM comes from the subset of training samples, known as support vectors, utilized in the decision function. Support vectors let computer use less amount of training subsets, and thus the model becomes memory efficient (Pedregosa et al., 2011). In the case of a classification problem, support vectors optimize class margins to a maximum possible span. To do this, SVM utilizes a complex mathematical algorithm. In the following subsections, first the mathematics behind SVM are explained in a simple manner, then an implementation of SVC on a simple drillhole data is shown with the details of programming in Python. SVC is a strong classification module offered by Scikit-learn library in Python. An important notice for the implementation is that posterior probability results are obtained from SVC, which is not possible normally. Although there are some earlier approximations to compute posterior probability of SVM, Platt (1999) succeeded to obtain a better posterior probability using a sigmoid model. The method of Platt (1999) to obtain posterior probability is embedded in SVC of Scikit-learn, and it is activated in the implementation of SVC.

[1] Cite as: Koruk K (2021) A Simple Implementation Example of SVC, Predictive Geometallurgy and Geostatistics Lab, Queen's University, Annual Report 2021, paper 2021-07, 79-87.

2. Support Vector Machines

2.1. Theoretical Review of SVM

The mathematics behind SVM can be challenging for a learner. The best approach to understand the logic of SVM is to start from a simple two-class and two-parameter case. For this simple case, SVM model takes the linear form

$$y(x_n) = w^T \Phi(x_n) + b \tag{1}$$

Where x_n is training input from 1 to N number of inputs, $y(x_n)$ is target classes, $\Phi(x_n)$ is a fixed transformation function of x, w^T is a coefficient vector which is maximizing the distance between two classes and b is the parameter to control the bias (Bishop, 2006). The equation is quite similar to logistic regression technique, however the main difference between logistic regression and SVM is that SVM maximizes the distance between classes with the help of margins at each side of the class boundary (Figure 1). The samples located on these margins are called support vectors.

Going back to the equation, SVM tries to approximate best values to the unknowns, w^T and b, to maximize the margins of the boundary. Optimum condition is provided when $1/\|w\|$ is maximized. If classification can be performed without letting any misclassification, the classification is called hard margin classification like in Figure 1. In its simplest manner, the problem takes the form

$$t_n(w^T \Phi(x_n) + b) = 1 \tag{2}$$

where t_n is target for samples from 1 to N number of samples.

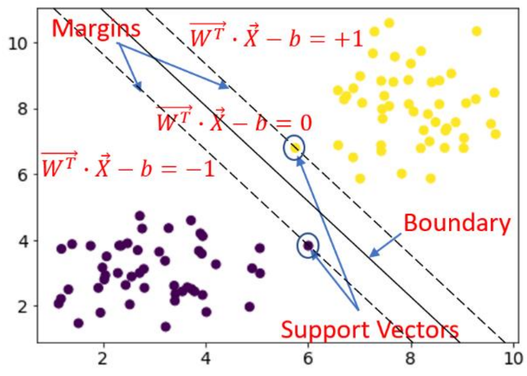


Figure 1 Hard margin linear SVM classification

However, in most cases hard margin classification is not possible. At this point, soft margin classification comes as a solution. Soft margin classification lets the model misclassify some samples to reach the optimum condition (Figure 2). To make misclassification possible, slack variables, ξ_n, are introduced into the equation (Bishop, 2006). $\xi_n = 0$ when data points are correctly classified, and the rest is $\xi_n = |t_n - y(x_n)|$, meaning that samples located on boundary or samples passing boundary are penalized up to 1. After the introduction of slack variables, the classification problem takes the form:

$$t_n y(x_n) \geq 1 - \xi_n \tag{3}$$

Theoretically, minimizing the summation of condition (3) for n from 1 to N number of samples can provide the best classification:

$$min \sum_{n=1}^{N} \xi_n + \frac{1}{2}\|w\|^2 \tag{4}$$

The equation (4) is scaled by a parameter C, and C, with constraint of being higher than 0, regularizes the complexity of the model.

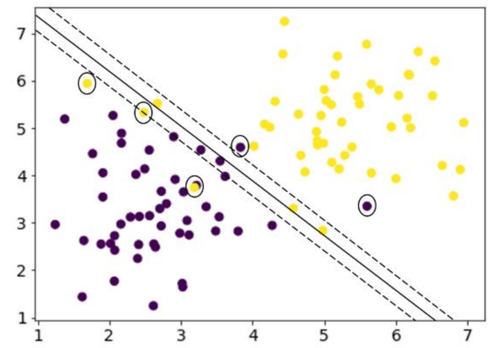


Figure 2 Soft margin linear SVM classification; encircled samples are misclassified samples

So far, the mathematical explanation of SVM is done as simple as possible with focus on linearly separable class problems. However, most of the real scenarios require non-linear solutions with quadratic function problems. Detailed explanation for quadratic function problems can be found in Bishop (2006). For the sake of an easier comprehension, the problem is kept simple. However, it is worth to mention kernel functions to better understand the non-linear solutions. The transformation functions we called earlier Φ(X_n) are kernel functions. Technically, Kernel functions works as if more features are added to sample spaces to make classes separable (Géron, 2017). Most popular kernel functions are tabulated in Table 1 below:

Table 1 Types of kernel functions

Kernel Types	*Function*
Linear Kernel	$k(x, x') = x^T x'$
Polynomial Kernel	$k(x, x') = (x^T x' + r)^d$
Radial Basis Kernel	$k(x, x') = e^{(-\gamma\|x-x'\|^2)}$
Sigmoid Kernel	$k(x, x') = \tanh(\gamma(x^T x') + r)$

Linear kernel, polynomial and radial basis kernel are derived from same the equation. While d equals 1 in polynomial kernel, polynomial kernel becomes linear kernel. Increasing d can be considered as increasing the amount of feature space in the training process. If d approximates to ∞, polynomial kernel approximates to radial basis function. The approximation is performed by making use of Taylor Expansion Series. Fundamentally, employing radial basis function can yield similar but much faster performance compared to polynomial kernel when d is ∞. Therefore, radial basis function (RBF) is preferred on the implementation of SVC.

Classifiers involving posterior probability are very useful in target recognition (Platt, 1999). Platt (1999) offered a modification to a previous multinomial likelihood non-sparse machine method. Platt offers sigmoid function for probabilities:

$$P(y = 1|y(x_n)) = \frac{1}{1 + \exp\,(A\,y(x_n) + B)} \tag{5}$$

where $y(x_n)$ is the equation (1), and A and B are the parameters found by minimizing a cross-entropy error function:

$$\min - \sum_{n} t_n \log(p_n) + (1 - t_n)\log\,(1 - p_n) \tag{6}$$

where p_i is equation (5). The method of Platt (1999) to obtain posterior probability is embedded in SVC of Scikit-learn, and it is activated in the implementation of SVC.

2.2. Implementation of SVC on a Simple Data

Jupyter notebook is employed to implement SVC. SVC is a fully developed module, and it offers solutions to some problems like unbalanced class problems. The critical parameters shaping models are regularization parameter C and gamma. The parameter C practically controls overfitting of the data. The parameter C with low values makes the model smooth, and C with high values may cause overfitting. Gamma defines how far a sample can have influence on the model. A low gamma makes the model general, and a large gamma can cause individuality of samples.

For the implementation of SVC, a 3-D pseudo drillhole data is created on which SVC is applied. The data is comprised of 56 vertical drillholes. Number of drillholes along East and North axes are 8 and 7 respectively. Each drillhole has 25 composite samples with 2 meter-length with total of 1400 samples in the data, and samples are classified under 2 classes: host rocks as -1 and ores as 1 (Figure 3). Summary statistics of the data can be seen in Table 2.

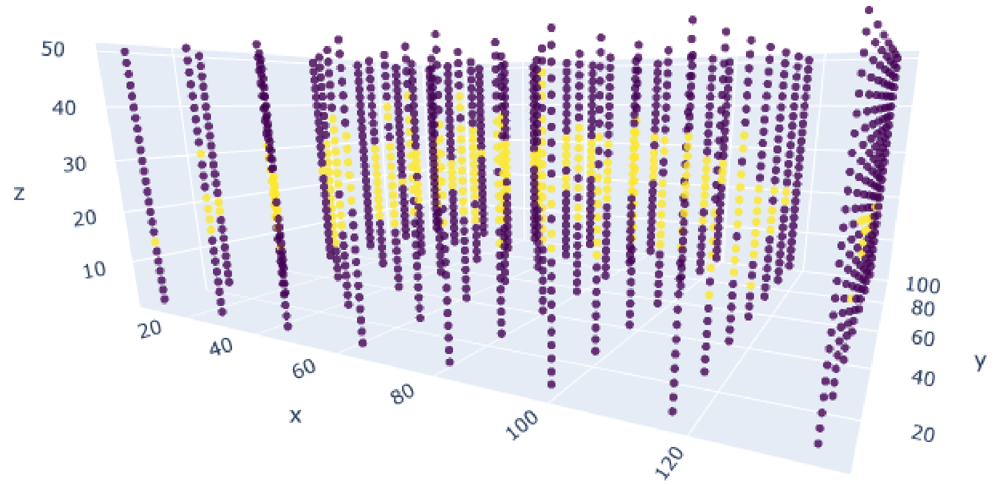


Figure 3 Distribution of pseudo data in the 3-D space. Data is illustrated using Plotly. Class 1 and -1 are shown with yellow and blue colors respectively.

Table 2 Summary statistics of the data

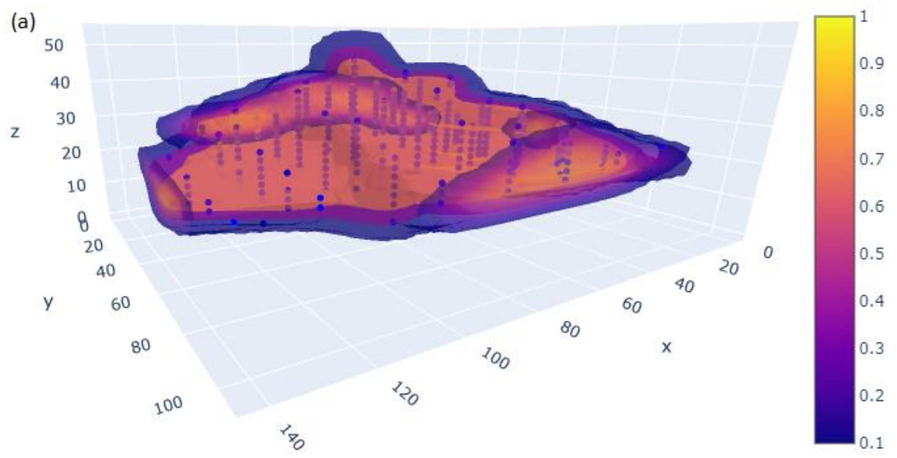
	Count	Mean	St. Dev.	Min.	25%	50%	75%	Max.
East	1400	72.00	41.258	9.0	40.5	72.0	103.5	135.0
North	1400	53.00	30.011	8.0	23.0	53.0	83.0	98.0
Elevation	1400	26.00	14.427	2.0	14.0	26.0	38.0	50.0
Value	1400	-0.55	0.833	-1	-1	-1	-1	1

The variables East, North and Elevation are utilized for training, and the variable Class is employed for target. The samples are split into train and test datasets to check accuracy of the model. Test ratio of the dataframe is set as 0.25. At the preprocessing stage, standardization is applied on the training data before fitting the data to the model. Class weight of SVC is set to balanced to consider the imbalanced ratio of classes while modelling. Normally, SVM is not capable of predicting probability, however SVC offers probability prediction using Platt's method (Platt, 1999). During modelling, prediction probability is also activated. To apply SVC, cross validation is applied first to determine ideal parameters. Cross validation is performed for the values and conditions expressed in Table 3. The optimum parameters determined for C and gamma are 0.1 and 5 respectively. However, C is increased slightly, and it was chosen as 1 to avoid high regularity effect.

Table 3 Cross-validation parameters of SVC

Parameters	***Cross-Validation***						
C	0.1	0.5	1	10	100	1000	
Gamma	5	2	1	0.1	0.01	0.001	0.0001

Time spent for the modelling was less than 1 second, meaning that training time is not a concern for SVC especially for such a simple dataset. The result of the model is both successful visually (Figure 4a) and accuracy according to confusion matrix (Figure 4b). Figure 4a shows distribution of probabilities and ore samples of drillholes (blue dots). As seen in Figure 4a high probability zones correspond to the ore samples. Both accuracy of class -1 and 1 are over 85% (Figure 4b), and overall accuracy is 88% which is quite promising.



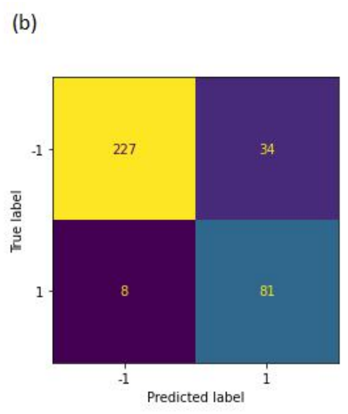


Figure 4 (a) 3-D visualization of probabilities and (b) confusion matrix of prediction results

To better investigate the model, cross-sections are created along North axis at each line of mesh grids. Two of the cross-sections overlain by drillholes are illustrated in Figure 5. Background color shows the map of probability. Blue and yellow crosses are host rock and ore samples of drillholes, respectively, and green circles show support vectors. For most of the areas, the model represents drillhole samples well. However, notice that there are some areas intended to be misclassified in terms of probability, which is desirable.

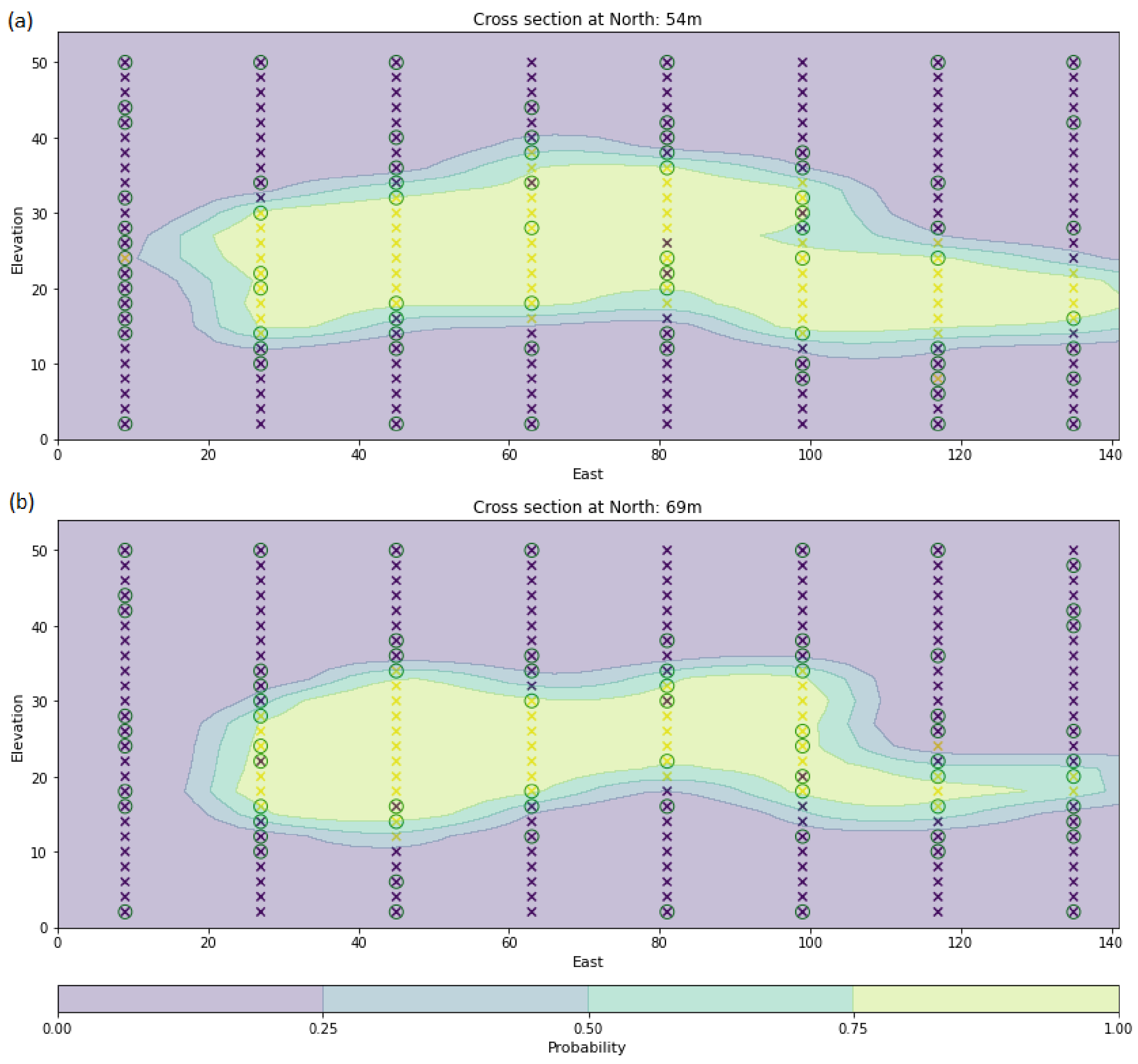
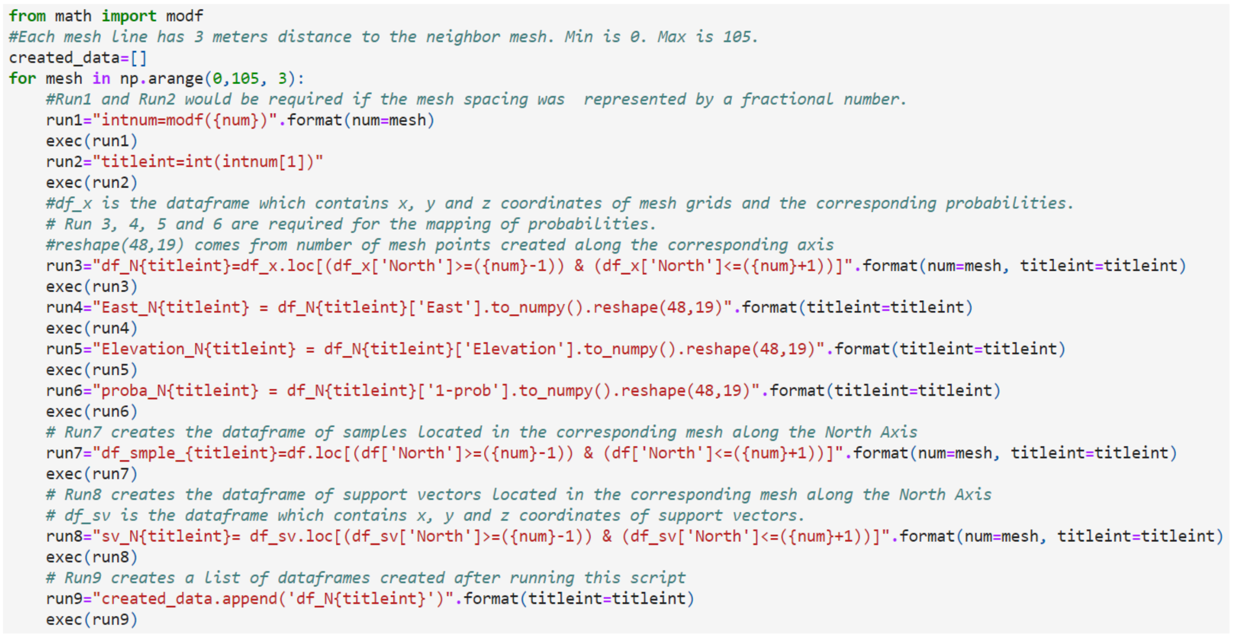


Figure 5 Cross-sections at North 54m (a) and 69m (b).

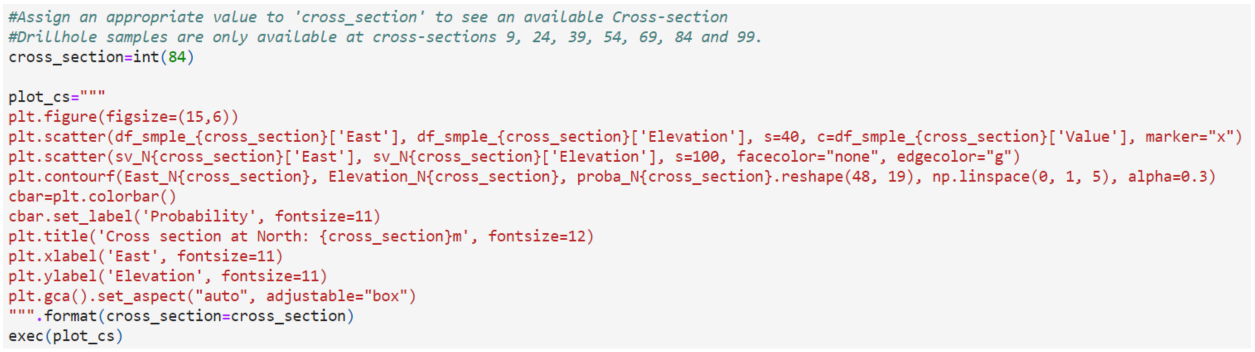
2.3. Scripts for Cross-sections

For illustration purposes, the library Plotly and Pyplot module of matplotlib are employed. Plotly is utilized for dynamic 3-D illustrations, and Pyplot is utilized for 2-D cross-sections. An automated script is written to create dataframes of a series of cross-sections (Figure 6). The dataframes created after the script shown in Figure 6 are utilized in a semi-automated script (Figure 7) which is written to plot cross-sections shown in Figure 5.



```python
from math import modf
#Each mesh line has 3 meters distance to the neighbor mesh. Min is 0. Max is 105.
created_data=[]
for mesh in np.arange(0,105, 3):
    #Run1 and Run2 would be required if the mesh spacing was  represented by a fractional number.
    run1="intnum=modf({num})".format(num=mesh)
    exec(run1)
    run2="titleint=int(intnum[1])"
    exec(run2)
    #df_x is the dataframe which contains x, y and z coordinates of mesh grids and the corresponding probabilities.
    # Run 3, 4, 5 and 6 are required for the mapping of probabilities.
    #reshape(48,19) comes from number of mesh points created along the corresponding axis
    run3="df_N{titleint}=df_x.loc[(df_x['North']>=({num}-1)) & (df_x['North']<=({num}+1))]".format(num=mesh, titleint=titleint)
    exec(run3)
    run4="East_N{titleint} = df_N{titleint}['East'].to_numpy().reshape(48,19)".format(titleint=titleint)
    exec(run4)
    run5="Elevation_N{titleint} = df_N{titleint}['Elevation'].to_numpy().reshape(48,19)".format(titleint=titleint)
    exec(run5)
    run6="proba_N{titleint} = df_N{titleint}['1-prob'].to_numpy().reshape(48,19)".format(titleint=titleint)
    exec(run6)
    # Run7 creates the dataframe of samples located in the corresponding mesh along the North Axis
    run7="df_smple_{titleint}=df.loc[(df['North']>=({num}-1)) & (df['North']<=({num}+1))]".format(num=mesh, titleint=titleint)
    exec(run7)
    # Run8 creates the dataframe of support vectors located in the corresponding mesh along the North Axis
    # df_sv is the dataframe which contains x, y and z coordinates of support vectors.
    run8="sv_N{titleint}= df_sv.loc[(df_sv['North']>=({num}-1)) & (df_sv['North']<=({num}+1))]".format(num=mesh, titleint=titleint)
    exec(run8)
    # Run9 creates a list of dataframes created after running this script
    run9="created_data.append('df_N{titleint}')".format(titleint=titleint)
    exec(run9)
```

Figure 6 Script written to create dataframes of a series of cross-sections



```python
#Assign an appropriate value to 'cross_section' to see an available Cross-section
#Drillhole samples are only available at cross-sections 9, 24, 39, 54, 69, 84 and 99.
cross_section=int(84)

plot_cs="""
plt.figure(figsize=(15,6))
plt.scatter(df_smple_{cross_section}['East'], df_smple_{cross_section}['Elevation'], s=40, c=df_smple_{cross_section}['Value'], marker="x")
plt.scatter(sv_N{cross_section}['East'], sv_N{cross_section}['Elevation'], s=100, facecolor="none", edgecolor="g")
plt.contourf(East_N{cross_section}, Elevation_N{cross_section}, proba_N{cross_section}.reshape(48, 19), np.linspace(0, 1, 5), alpha=0.3)
cbar=plt.colorbar()
cbar.set_label('Probability', fontsize=11)
plt.title('Cross section at North: {cross_section}m', fontsize=12)
plt.xlabel('East', fontsize=11)
plt.ylabel('Elevation', fontsize=11)
plt.gca().set_aspect("auto", adjustable="box")
""".format(cross_section=cross_section)
exec(plot_cs)
```

Figure 7 Script written to plot cross-sections

3. Discussion and Conclusion

Sci-kit learn offers a strong module for SVM. The documentation of SVC is very strong; therefore, all the answers can be found to any technical questions that arise during the application step. Moreover, the options like imposing the imbalance of classes, and posterior probabilities makes SVC valuable. The time spent on training was less than 1 second for a data set with 1400 samples, which is perfect for quick decision-making. Yet, considering training time increases exponentially, SVC still can be time-demanding for huge drillhole datasets. Again, the simple scenario was based on a binary class problem. When the number of targets is higher than 2, training becomes demanding for SVC since modelling scheme is not traditional like some other classifiers, e.g., decisiontreeclassifier. Therefore, multiclass problems can be demanding for SVC (Pedregosa et al., 2011). Although probabilities are accurate for the example, it is also important to notice that Sci-kit learn addresses that probability prediction has some issues (Pedregosa et al., 2011), therefore Platt's method can need verification and validation in real and more complex scenarios.

SVC is successfully applied to simple data. However, there are some points worth to mention about choosing the correct parameters and functions. Regularity of the model is an important concern. An overfitted model can be deceptive. Therefore, the regularity parameter was kept slightly higher than the optimum value obtained by cross-validation to prevent overfitting. Moreover, how the accuracy is determined for cross-validation is also important. A balanced accuracy can be more informative in terms of determining the optimum parameters. Another point, RBF as a kernel function is robust, however other kernel functions can also be effective under different scenarios. Lastly, analyzing and observing the results are at least as important as the accuracy of the model. To better observe the model, cross-sections are created along North axis. However, cross-sections created with different angles and directions can be required for some other cases. Therefore, either more developed cross-section modules can be utilized for the complex scenarios, or the scripts shared in the articles can be improved.

Overall, SVC proved that it can model a simple dataset. However, it is also important to notice that the application of SVC on a real drillhole data can be much more demanding.

4. References

Bishop, C.M., 2006. Kernel Methods & Sparse Kernel Machines. Pattern Recognition and Machine Learning. Springer Science+Business Media, LLC, Singapore.

Géron, A., 2017. Support Vector Machines. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow., 2nd Edition, O'Reilly Media, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, vol. 12, pp. 2825-2830.

Platt, J.C., 1999. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. MIT Press.