

# Apply machine learning on superconductors analysis

By

Mengda Xu

Senior Thesis

Submitted in partial satisfaction of the requirements for High Honors for the  
degree of

BACHELOR OF SCIENCE

in

Mathematical Analytics and Operations Research

in the

COLLEGE OF LETTERS AND SCIENCE

of the

UNIVERSITY OF CALIFORNIA,

DAVIS

Approved:

---

Jesús A. De Loera  
May, 2017

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                       | <b>3</b>  |
| <b>2</b> | <b>Basic concept in Machine Learning</b>  | <b>3</b>  |
| <b>3</b> | <b>Method</b>                             | <b>4</b>  |
| 3.1      | Sparse Logistic Regression . . . . .      | 4         |
| 3.2      | Artificial Neural Network (ANN) . . . . . | 8         |
| 3.3      | Support Vector Machines . . . . .         | 10        |
| 3.4      | Cross Validation . . . . .                | 16        |
| <b>4</b> | <b>Results</b>                            | <b>16</b> |
| 4.1      | Sparse logistic regression . . . . .      | 16        |
| 4.2      | Artificial Neural Network . . . . .       | 19        |
| 4.3      | Support Vector Machines . . . . .         | 20        |
| <b>5</b> | <b>Appendix</b>                           | <b>22</b> |

*Acknowledgments: I would like to thank my thesis advisor, Dr.Jesús De Lopera, for introducing me to this research topic and for the tremendous guidance he has offered during the course of my work. I also would like to thank Dr.Julia Zaikina, she spent tremendous times and offer valuable advises on our research project. Also, thanks my colleagues Matthew Corbelli,Elise Shen and Malanie Tran. I cannot complete this thesis without their help.*

# 1 Introduction

In this paper, We study superconductor materials and perform analysis to uncover correlation between superconducting property. I fetched the superconductor data from online database. Then applying different data mining and machine learning method on the data. The first stage of my research is to find the most important features. Those features are the most significant factors deciding whether a compound is a superconductor or not. The second stage of my research is to apply machine learning technology to classify the superconductor and non-superconductor.

A superconductor is a material in which its electrical resistance drops to 0 when it is cooled below its critical temperature,  $T_c$ . It experiences the Meissner effect, expulsion of a magnetic field, during its transition to the superconducting state. Our study focuses on iron-based compounds that all consist of layers of iron and arsenic.(see [14] and [15] for detail) We are able to predict superconductivity by identifying the most important structural parameters of these compounds using machine learning analysis. The whole data set contains 40 samples. Each sample contains 17 features and a  $T_c$  value. The threshold of whether a material is a superconductor is its  $T_c$ -temperature. If  $T_c$  of a material is bigger than 0, then it is a superconductor. Otherwise, it is not. The 17 features include: a, b, c, angle beta, Volume, temperature, z coordinates of As, Fe-As distance, Fe-Fe distance, alpha, Beta, A-As distance, A-Fe distance, S.O.F.A, S.O.F.As, S.O.F.Fe.

# 2 Basic concept in Machine Learning

*"Field of study that gives computers the ability to learn without being explicit programmed "*

*—A.Samuel*

Machine learning is a really hot topic right now. What is exactly Machine learning?

**Definition 2.1** *Machine learning: set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty[2]*

There are majorly three learning strategies: supervised learning, unsupervised learning and reinforcement learning.

**Definition 2.2** *Supervised learning: learn a mapping from input  $x$  to output  $y$ , given a labeled set of input-output pairs. When  $y$  is categorical, then the task is to assign a class to a new sample and it is called classification problem. When  $y$  is real-valued, then the task is called regression.[2]*

**Definition 2.3** *Unsupervised learning: Only given input data  $x$ , find the structure or patterns in the data.*[2]

**Definition 2.4** *Reinforcement learning: Given a certain environment, find the policy that maximizes the cumulative reward.*[2]

During this research, we heavily use supervised machine learning methods since we are given data set which include input: 17 chemical features and output: superconductor or not.

### 3 Method

#### 3.1 Sparse Logistic Regression

logistic regression is a well-known classification method and it can be found in chapter 8 of [2]. For example, if there are two categories of data and logistic regression will separate these two data according to their features. Logistic regression uses sigmoid function. Here is the graph of sigmoid function.

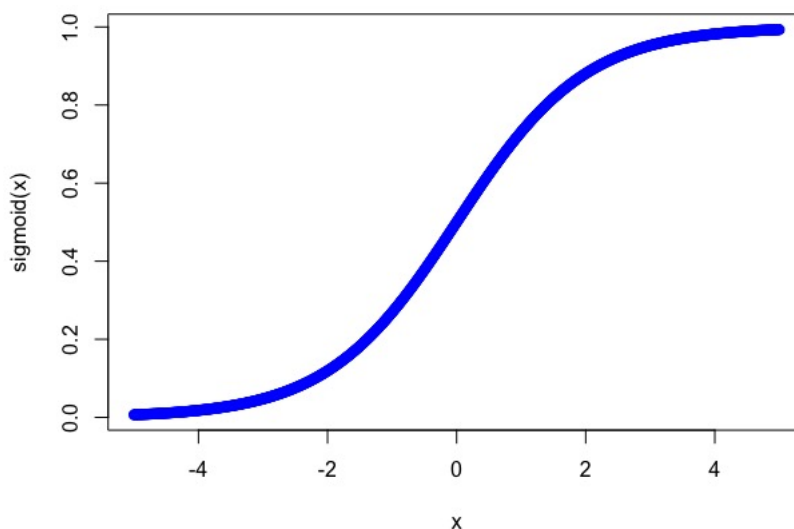


Figure 1: Sigmoid function plot

Logistic regression tries to find out the linear combination of features as input to sigmoid function. we use logistic regression to predict the probability of a

given sample is 1 versus the probability of a given sample is 0. We try to tune the coefficient  $w$  to satisfy following:

$$P(y = 1|x) = g_w(x) = \frac{1}{1 + e^{-w^T x}}. \quad (1)$$

$$P(y = 0|x) = 1 - P(y = 1|x) = 1 - g_w(x). \quad (2)$$

Equation(1) uses the sigmoid function to output the probability of being class 1. Equation(2) takes the compliment of  $P(y = 1|x)$  to express  $P(y = 0|x)$ . The outputs will be between 0 and 1. The output of Equation(1) is close to 1 if it belongs to class1 and the output is close to 0 if it belongs to class 0. We can

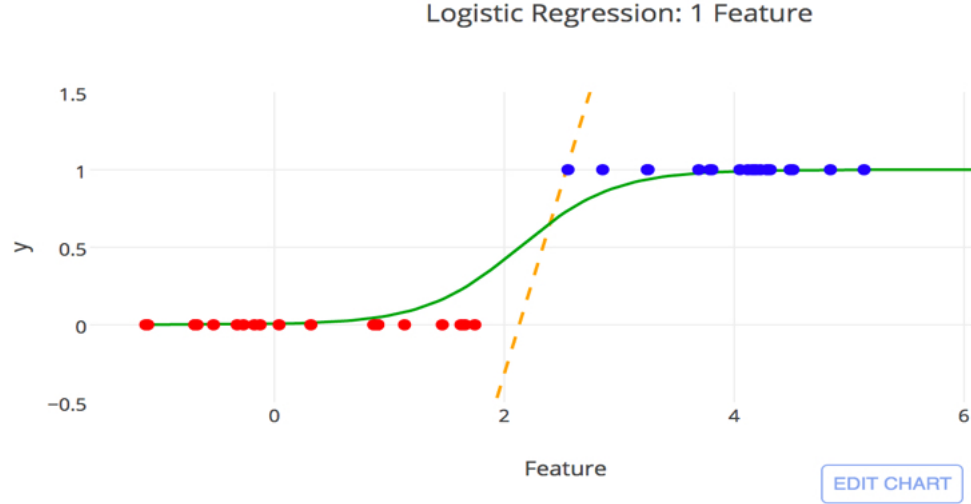


Figure 2: Sample Sigmoid output,pic from [16]

combine these two equations together as following:

$$P(y|x; w) = g(x; w)^y (1 - g(x; w)^{1-y}). \quad (3)$$

Suppose we have many samples and samples are i.i.d, then we can derive the Maximum Likelihood Estimation(MLE). MLE is an useful tool in statistics.It helps us to tune the parameter to maximize the probability given observations.

**Definition 3.1** *The maximum likelihood estimate (MLE) of is that value of  $w$  that maximizes  $lik(w)$ : it is the value that makes the observed data the “most probable”.*[5]

Suppose  $X_1, X_2, X_3, \dots, X_n$  have joint density denoted:

$$f_w(x_1, x_2, x_3, \dots, x_n) = f(x_1, x_2, x_3, \dots, x_n|w). \quad (4)$$

Given observation  $X_1 = x_1, X_2 = x_2, X_3 = x_3, \dots, X_n = x_n$ , the likelihood of

$w$  is the function:

$$lik(w) = f(x_1, x_2, x_3, \dots, x_n|w). \quad (5)$$

If the  $X_i$  are i.i.d, then the expression can be simplified to:

$$\prod_{i=1}^n f(x_i|w). \quad (6)$$

(See[5])

Since sometimes it is hard to maximize the lik function and log function is an increasing function, maximize lik function is equivalent to maximize log likelihood:

$$\sum_{i=1}^n \log(f(x_i|w)). \quad (7)$$

Let  $g(x^{(i)}; w) = \text{sigm}(w^T x) = \frac{1}{1+e^{-w^T x}}$  and  $h^{(i)}$  be the  $i^{th}$  row of matrix  $h$ .

Given a data set  $x$ , a corresponding label vector  $y$ , and a randomly initialized weight vector  $w$ , we want to maximize the likelihood of our labels  $y$  given  $x$  and  $w$ , which can be written as:

$$l(w) = \log p(D|w) = \sum_{i=1}^M \log p(y^{(i)}|x^{(i)}; w) \quad (8)$$

$$= \sum_{i=1}^M \log(g(x^{(i)}; w)^{y^{(i)}} (1 - g(x^{(i)}; w))^{1-y^{(i)}}) \quad (9)$$

$$= \sum_{i=1}^M y^{(i)} \log(g(x^{(i)}; w)) + 1 - y^{(i)} (1 - g(x^{(i)}; w)). \quad (10)$$

(See[11]for detail)

The Sparse Logistic Regression is nothing but to add penalty  $-\sum_1^n |w|$  on Equation(6). If we have  $n$  features while maximize a negative expression is the same as minimize it. Now the equation we try to maximize becomes:

$$p(y^{(i)}|x^{(i)}; w) = g(x^{(i)}; w)^{y^{(i)}} (1 - g(x^{(i)}; w))^{1-y^{(i)}} - \lambda \sum_1^n |w_i|. \quad (11)$$

The tuning parameter  $\lambda$  controls the magnitude of penalty. If  $\lambda$  equals to 0, the Equation(11) degenerates to simple logistic regression. As  $\lambda$  increases, coefficients begin to shrink. As  $\lambda$  is big enough, some coefficients drop to 0 and variable selection has been done during this process. The remaining nonzero variables have the most significant correlation between the output  $y$ . Generally, the bias increases when  $\lambda$  increases and variance decreases when  $\lambda$  increases. (See [6]) Here is the plot of misclassification error versus  $\log \lambda$  in the research:

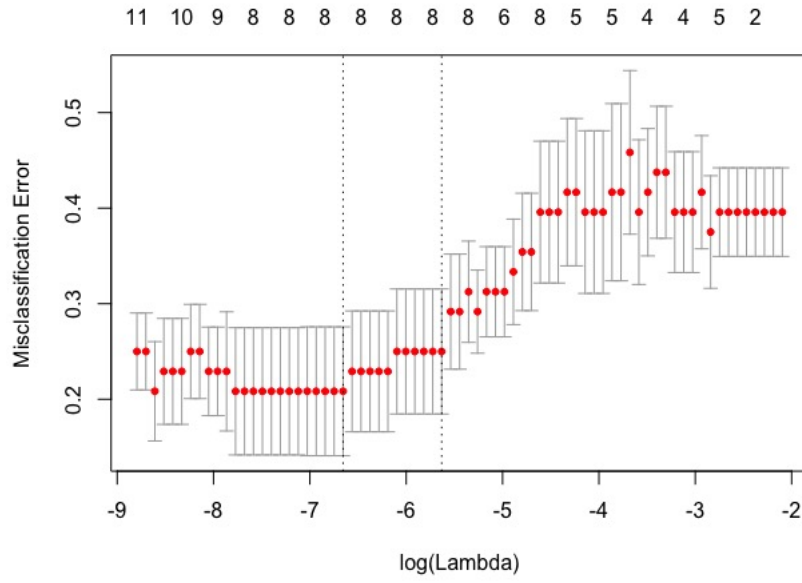


Figure 3: misclassification error versus  $\log \lambda$

### 3.2 Artificial Neural Network (ANN)

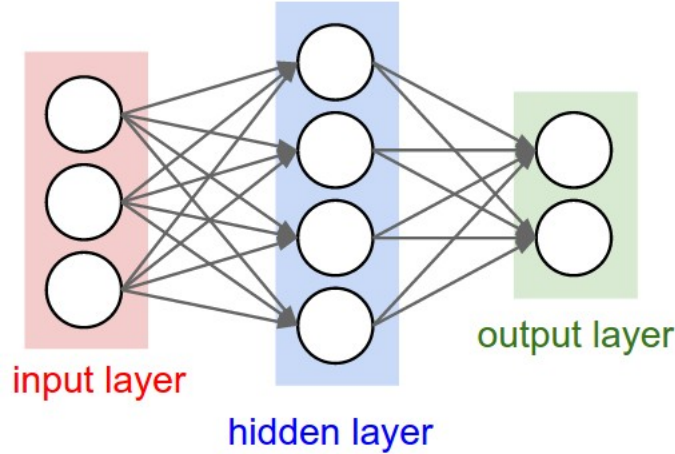


Figure 4: Feed forward network

Artificial Neural Network is analogy with the nature neural network from human brain. Instead of using the natural neural, ANN uses nodes and links to simulate the nature neural network. The inventor of the first neurocomputer, Dr. Robert Hecht-Nielsen define a neural network as- "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." [10] There are different kind of ANN. We used the feed-forward network in our research.

**Definition 3.2** *A feed-forward network has connections only in one direction which forms a acyclic graph. Every node receive input from "upstream" nodes and deliver output to "downstream" nodes, with activation functions usually step or logistic function. [3]*

**Remark 1** *A feed-forward network can model any smooth function to any accuracy level*

Usually, there are three major parts of a feed forward network, including one input layer, one or multiple hidden layers and one output layer. Each layer is composed of multiple nodes which simulate the neurons of human brain. Each layer is connected to the other layer by link. The data is feed into ANN through input layer and output by the output layer. The neurons are connected by links and they transfer data with each other.



**Definition 3.3** A link from unit  $i$  to unit  $j$  serves to propagate the activation  $a_i$  from  $i$  to  $j$ . Each link also has a numeric weight  $w_{i,j}$  associated with it, which determines the strength and sign of the connection. [3]

The nodes can receive input data and perform operations on the data. The output of these operations is passed to other nodes through links. The output at each node is called its activation value. The output of each node is determined by the activation function. Formally, each unit  $j$  first computes a weighted sum of its inputs:

$$in_j = \sum_{i=0}^n w_{i,j} a_i. \quad (12)$$

Then feed this input into an activation function  $g$  to derive the output

$$a_j = g(in_j) = g(\sum_{i=0}^n w_{i,j} a_i). \quad (13)$$

The activation function is usually a perceptron with a threshold or a sigmoid function. [3]

**Definition 3.4** A perceptron is a binary classifier which maps a real-valued vector  $x$  to an output value  $f(x)$

$$f(x) = \begin{cases} 1, & \text{if } w \cdot x + b > 0 \\ 0, & \text{otherwise} \end{cases} \quad (14)$$

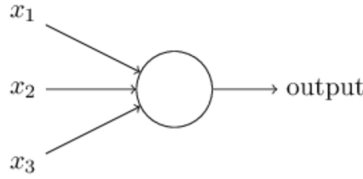


Figure 5: perceptron

However, we prefer the sigmoid function because we can take the advantage of differentiable. So far, we have introduced the basic structure of a simple feed-forward network. Next, we explain how does it works. The first step is to feed the data into input layer. For example, if a subject has 7 features, we can put 7 input nodes in the input layer. Then, each node in the input layer will feed the input into its activation function. The output of the input layer will send to the next layer. Repeat this procedure until data propagate to the final output layer. This whole procedure we call it forward propagation. In most cases, the output has error and we need a way to tune the network to adjust the coefficient  $w_{i,j}$  on each link. The method we use is called back propagation.

Intuitively, each layer should be responsible for the fraction of the error. During the back propagation, we can back propagate the error from the output layer to the hidden layer[3] and adjust the coefficients between each layer. In our research, the ANN was tuned to use a learning rate  $\alpha = 0.07$ , one hidden layer, and four nodes within the hidden layer. Let  $g$  be the sigmoid function ; let  $w$  be a set of initially random weights; let  $a^{(l)}$  mean all nodes in layer 1; and let  $a_i^{(l)}$  mean the  $i^{th}$  node in layer  $l$ . During the feed-forward portion of the training, each activation node  $a_i^{(l)}$  was updated with the rule

$$a_i^{(l)} = g(w^{(l-1)T} a^{(l-1)}). \quad (15)$$

The update rule for back propagation is

$$w_{kj}^{(l-1)} := w_{kj}^{(l-1)} - \alpha \frac{\partial RSS}{\partial w_{kj}^{(l-1)}}. \quad (16)$$

where  $\frac{\partial RSS}{\partial w_{kj}^{(l-1)}}$  is equal to  $-\delta_k^{(l)} \cdot a_j^{(l-1)}$  and

$$\delta_j^{(l-1)} = \sum_k \delta_k^{(l)} \cdot w_{kj}^{(l-1)} \cdot (1 - a_j^{(l-1)}) \cdot a_j^{(l-1)}. \quad (17)$$

for nodes not in the final layer  $l$ . If a node is in the final layer,

$$\delta_k^{(l)} = -(y_k - a_k^{(l)})(1 - a_k^{(l)})(a_k^{(l)}). \quad (18)$$

Here is the detailed algorithm in pseudocode:

### 3.3 Support Vector Machines

A Support Vector Machine (SVM) is a classifier defined by a separating hyperplane. Given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new examples.[13]

We tried SVM with different kernel functions. First, let us see the definition of the kernel function.

**Definition 3.5** *Each kernel  $k$  has an associated feature mapping  $\phi$ .  $\phi$  takes input  $x \in \chi$  and maps it to  $F$  (feature space). Kernel  $k(x, z)$  takes two inputs and gives their similarity in  $F$ -space*

$$\phi : \chi \rightarrow F \quad (19)$$

$$k : \chi \times \chi \rightarrow R, k(x, z) = \phi(x)^T \phi(z). \quad (20)$$

$F$  needs to be a vector space with a dot product defined on it, also called a Hilbert Space[7]

A function can be used as a kernel function if it satisfy Mercer's Condition.

```

function BACK-PROP-LEARNING(examples, network) returns a neural network
inputs: examples, a set of examples, each with input vector x and output vector y
         network, a multilayer network with L layers, weights  $w_{i,j}$ , activation function g
local variables:  $\Delta$ , a vector of errors, indexed by network node

repeat
  for each weight  $w_{i,j}$  in network do
     $w_{i,j} \leftarrow$  a small random number
  for each example (x, y) in examples do
    /* Propagate the inputs forward to compute the outputs */
    for each node i in the input layer do
       $a_i \leftarrow x_i$ 
    for  $\ell = 2$  to L do
      for each node j in layer  $\ell$  do
         $in_j \leftarrow \sum_i w_{i,j} a_i$ 
         $a_j \leftarrow g(in_j)$ 
    /* Propagate deltas backward from output layer to input layer */
    for each node j in the output layer do
       $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$ 
    for  $\ell = L - 1$  to 1 do
      for each node i in layer  $\ell$  do
         $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$ 
    /* Update every weight in network using deltas */
    for each weight  $w_{i,j}$  in network do
       $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$ 
until some stopping criterion is satisfied
return network

```

Figure 6: pseudocode from [3]

**Definition 3.6** For  $k$  to be a kernel function, there must exist a Hilbert Space  $F$  for which  $k$  defines a dot product if  $K$  is a positive definite function. [7]

**Remark 2** let  $k_1, k_2$  be two kernel functions, then kernels can be constructed by following rules:

$$k(x, z) = k_1(x, z) + k_2(x, z). \quad (21)$$

$$k(x, z) = c \cdot k_1(x, z). \quad (22)$$

$$k(x, z) = k_1(x, z) * k_2(x, z). \quad (23)$$

Why we need kernel functions? The main function of kernel tricks is mapping data to higher dimensions where it exhibits linear patterns. The kernel tricks make linear model work in non-linear setting.[7] First, we use linear kernel. Intuitively, linear kernel is to separate data by a linear combination of given features. If the data set in a problem is linearly separable, then linear kernels will have best performance. Here is the formal definition of linear kernel.

**Definition 3.7** If  $\phi(x) = x$ , we get the linear kernel.  $\kappa(x, z) = x^T z$  [7]

We need to find a hyperplane  $w \cdot x + b = 0$  which can separate the data into two categories. In our case, it is superconductor and non-superconductor. We hope

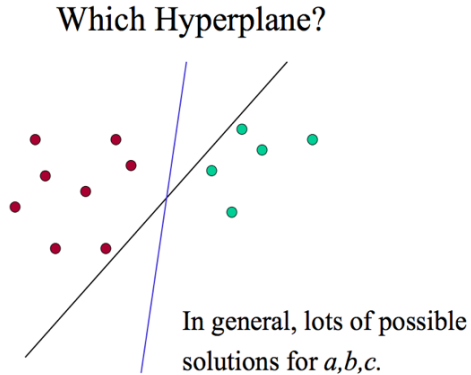


Figure 7: which hyperplane?(picture from [12])

we can find a hyperplane could satisfy:

$$w \cdot x + b > 1$$

when  $y=1$  (superconductor)

$$w \cdot x + b < -1$$

when  $y=-1$ (non superconductor)

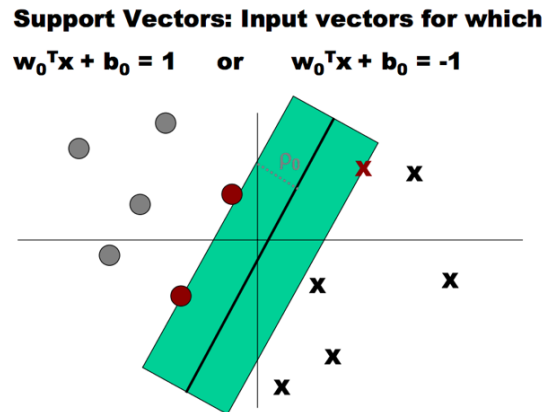


Figure 8: Support Vector picture from [12]

There are many lines can separate the data but which one is the best? SVM selects the line that has the maximum distance from one or more samples from each class. Moreover, we also need this hyperplane could maximize the

margin which is the distance between the data point to  $w \cdot x + b = 0$ . This margin- $d^+$  is calculated by  $\frac{1}{\|w\|}$ . Similarly, we the margin- $d^-$  is also  $\frac{1}{\|w\|}$ . And we add it together to get  $\frac{2}{\|w\|}$ . We need to maximize  $\frac{2}{\|w\|}$  to get the optimal hyper-plane and it is equal to minimize  $\frac{1}{2}\|w\|^2$ .(see [12] for detail) Finally, we can form the problem into an optimization problem as following:

$$\min \frac{1}{2}\|w\|^2. \quad (24)$$

It is constrained by

$$y^i(w^T x^i + b) > 1. \quad (25)$$

where  $i=1,2,\dots,m$

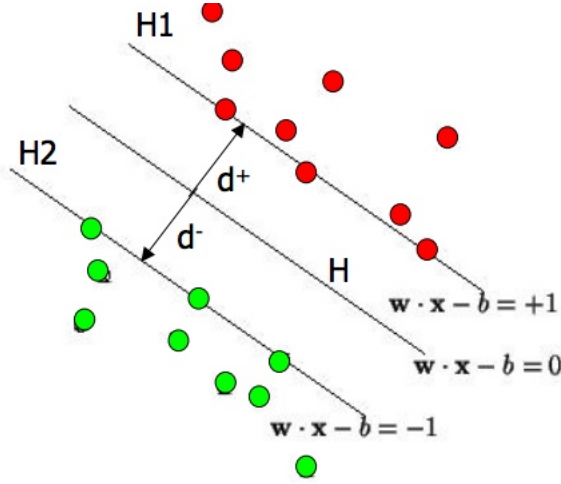


Figure 9: SVM picture from [12]

To solve this, we need to introduce Lagrange multipliers and we formulate the optimization problem into Lagrange function. Let's start we with a simple example:

$$\max f(x) \quad (26)$$

$$s.t.g(x) = 0. \quad (27)$$

If we do not have the constrain, we can simply set the gradient of  $f$ , denoted  $\nabla f$ , equal to 0 because the direction of gradient of  $f$  is the direction of most rapid increase of  $f$ . However, we do have a constrain here so the gradient now should be orthogonal to the set of feasible solution. And the gradient of  $g$  at  $x$  is also orthogonal to the feasible set at  $x$  for every point  $x$  in the feasible set. Therefore,  $\nabla f$  is parallel to  $\nabla g$  at every point  $x$  in feasible solution. We can write this observation into a system of equation[1]:

$$g(x) = 0. \quad (28)$$

$$\nabla f(x) = y \nabla g(x). \quad (29)$$

If there are m constraints:

$$\max f(x) \quad (30)$$

s.t

$$g_1(x) = 0 \quad (31)$$

$$g_2(x) = 0.$$

.

.

(32)

$$g_m(x) = 0. \quad (33)$$

By the same logic, we can conclude:

$$\nabla f(x) = \sum_{i=1}^m y_i \nabla g_i(x). \quad (34)$$

Alternatively, we can use Lagrangian function to get the same answer.(See [1] for detail)

**Definition 3.8** *Lagrangian function:*

$$L(x, y) = f(x) - \sum_i y_i g_i(x). \quad (35)$$

Then we take derivative with respect to x and y and set them equal to 0:

$$\frac{\partial L}{\partial x_j} \frac{\partial L}{\partial x_j} = \frac{\partial f}{\partial x_j} - \sum_i y_i \frac{\partial g_i}{\partial x_j} = 0. \quad (36)$$

$$\frac{\partial L}{\partial y_i} = -g_i = 0. \quad (37)$$

The equations we derived is exactly the same as we use get in geometric way. [1]Solving the equations above we can get the best hyperplane in SVM. What we discussed above is so called linear kernel. Next, we introduced Radial Basis Function kernel.

**Definition 3.9** *Radial Basis Function(RBF) Kernel:*

$$k(x, z) = e^{-\gamma \|x-z\|^2}. \quad (38)$$

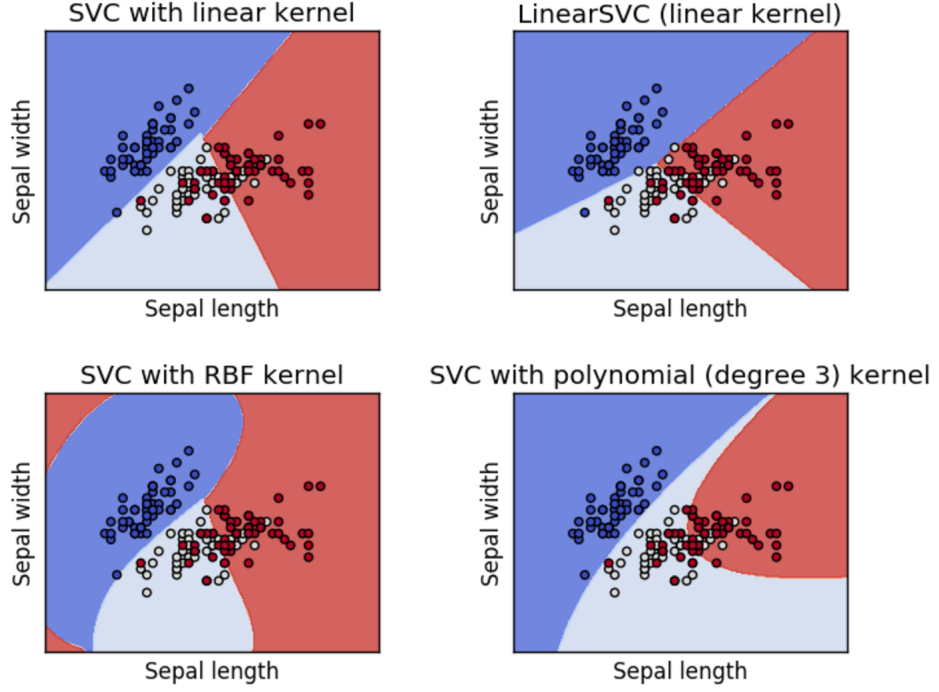


Figure 10: SVM with different kernel(this nice pic comes from [8])

$\gamma$  here is so called the kernel bandwidth.[6]

How do we exactly use Kernel functions? kernel represents a dot product in feature space  $F$ . Hence, any dot products can be kernelized eg,  $\phi(x_i)^T \phi(x_j) = k(x_i, x_j)$  Before we continue, let's first introduce dual problem of Lagrangian. If we convert previous Lagrangian optimization into dual problem, we can get the follow dual problem:

$$L_d(w, b, \epsilon, \alpha, \beta) = \sum_{n=1}^N a_n - \frac{1}{2} \sum_{m,n=1}^N \alpha_m \alpha_n y_m y_n (x_m^T x_n). \quad (39)$$

subject to

$$\sum_{n=1}^N a_n y_n = 0. \quad (40)$$

We can replace  $X_m^T X_n$  with  $\phi(x_m^T) \phi(x_n) = k(x_m, x_n) = \text{SomeKernelFunction}$  (See [6] for detail) SVM now learns a linear separator in the feature Space  $F$ . This is how we convert a non-linear problem into a linear problem.

### 3.4 Cross Validation

I applied Cross Validation to all above methods. To be specific, I applied K-fold cross validation. Here is the detailed description from [9]

*Cross validation is a model evaluation method that is better than residuals. The problem with residual evaluations is that they do not give an indication of how well the learner will do when it is asked to make new predictions for data it has not already seen. One way to overcome this problem is to not use the entire data set when training a learner. Some of the data is removed before training begins. Then when training is done, the data that was removed can be used to test the performance of the learned model on “new” data. This is the basic idea for a whole class of model evaluation methods called cross validation. K-fold cross validation is one way to improve over the holdout method. The data set is divided into  $k$  subsets, and the holdout method is repeated  $k$  times. Each time, one of the  $k$  subsets is used as the test set and the other  $k - 1$  subsets are put together to form a training set. Then the average error across all  $k$  trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set  $k - 1$  times. The variance of the resulting estimate is reduced as  $k$  is increased. The disadvantage of this method is that the training algorithm has to be rerun from scratch  $k$  times, which means it takes  $k$  times as much computation to make an evaluation. A variant of this method is to randomly divide the data into a test and training set  $k$  different times. The advantage of doing this is that you can independently choose how large each test set is and how many trials you average over.*

## 4 Results

### 4.1 Sparse logistic regression

Each  $w_j$  is corresponding to one of features in a, b, c, Volume, z of As, As-As interlayer distance, Fe-As distance, Fe-Fe distance, alpha, Beta, A-As, A-Fe, S.O.F.A, S.O.F. As, S.O.F.Fe.



| features                  | $w_j$         | rank |
|---------------------------|---------------|------|
| a                         | 5.559108e+01  | 12   |
| b                         | 2.133013e-12  | 8    |
| c                         | .             | 4    |
| Volume                    | .             | 4    |
| z.of.As                   | -1.952826e+03 | 16   |
| As-As.interlayer.distance | .             | 4    |
| Fe-As.distance            | .             | 4    |
| Fe-Fe distance            | .             | 4    |
| alpha..degrees            | .             | 4    |
| Beta..degrees             | 5.440771e+00  | 11   |
| A-As distance             | -1.023784e+02 | 12   |
| A-Fe distance             | .             | 4    |
| S.O.F.A                   | -1.350225e+00 | 9    |
| S.O.F.As                  | -1.356316e+00 | 10   |
| S.O.F.Fe                  | -3.068363e+02 | 14   |

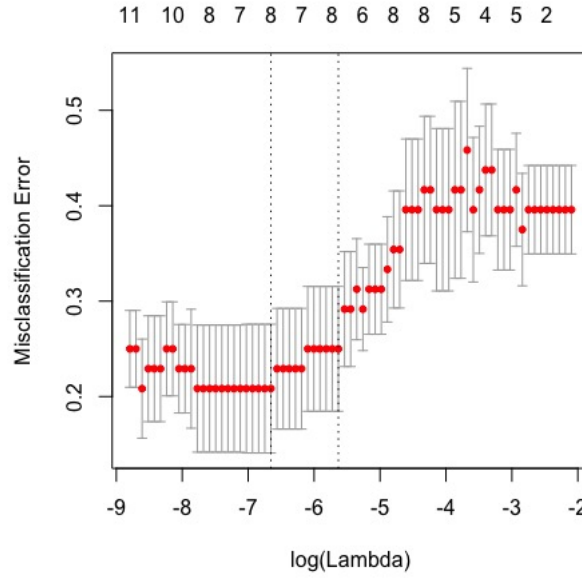


Figure 11: misclassification error versus  $\log \lambda$

Arranged by magnitude of  $w_j$ , we could get most important six feature: a, z of As, Beta degrees, A-As, S.O.F.As, S.O.F.Fe  
Here is the plot of misclassification error versus  $\log \lambda$  in the research. We can

clearly see that the error increase as  $\lambda$  increases. The minimum misclassification rate happen at  $\lambda = 0.001287396$ . The misclassification rate below 1 standard deviation is at  $\lambda = 0.003582255$

## 4.2 Artificial Neural Network

I used one hidden layer and I tried 3,6,9,12 nodes in the hidden layer. After 3000 iteration, the accuracy could reach 70 percents. The error significantly drop as iteration increases. y-axis is the magnitude of training error and x-axis is the number of iteration The yellow denotes 3 nodes, the green denotes 6 nodes, the red denotes 9 nodes, and the blue denotes 12 nodes. From the graph we can see that the training error significantly drop after large iteration. The accuracy of testing set is around 70 percents.

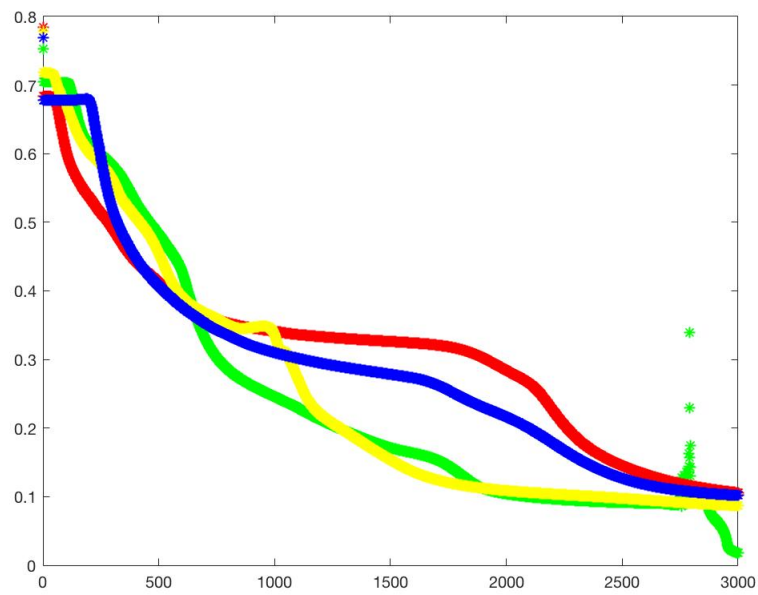


Figure 12: Caption

### 4.3 Support Vector Machines

We use 5-folder cross validation to measure the accuracy. The linear Kernel has 56.4444 percents accuracy.

In terms of the polynomial Kernel, the following table is generated for different degrees polynomials.

| degree | scale | C    | accuracy  | Kappa       |
|--------|-------|------|-----------|-------------|
| 1      | 0.001 | 0.25 | 0.6044444 | 0.00000000  |
| 1      | 0.001 | 0.50 | 0.6044444 | 0.00000000  |
| 1      | 0.001 | 1.00 | 0.6044444 | 0.00000000  |
| 1      | 0.010 | 0.25 | 0.6044444 | 0.00000000  |
| 1      | 0.010 | 0.50 | 0.6044444 | 0.00000000  |
| 1      | 0.010 | 1.00 | 0.5822222 | -0.04324324 |
| 1      | 0.100 | 0.25 | 0.6022222 | 0.01389961  |
| 1      | 0.100 | 0.50 | 0.6022222 | 0.01389961  |
| 1      | 0.100 | 1.00 | 0.6444444 | 0.15104247  |
| 2      | 0.001 | 0.25 | 0.6044444 | 0.00000000  |
| 2      | 0.001 | 0.50 | 0.6044444 | 0.00000000  |
| 2      | 0.001 | 1.00 | 0.6044444 | 0.00000000  |
| 2      | 0.010 | 0.25 | 0.6044444 | 0.00000000  |
| 2      | 0.010 | 0.50 | 0.5822222 | -0.04324324 |
| 2      | 0.010 | 1.00 | 0.6022222 | 0.01389961  |
| 2      | 0.100 | 0.25 | 0.6444444 | 0.15104247  |
| 2      | 0.100 | 0.50 | 0.6444444 | 0.15064935  |
| 2      | 0.100 | 1.00 | 0.6244444 | 0.11168831  |
| 3      | 0.001 | 0.25 | 0.6044444 | 0.00000000  |
| 3      | 0.001 | 0.50 | 0.6044444 | 0.00000000  |
| 3      | 0.001 | 1.00 | 0.6044444 | 0.00000000  |
| 3      | 0.010 | 0.25 | 0.6244444 | 0.05714286  |
| 3      | 0.010 | 0.50 | 0.6022222 | 0.01389961  |
| 3      | 0.010 | 1.00 | 0.6222222 | 0.07104247  |
| 3      | 0.100 | 0.25 | 0.6666667 | 0.19428571  |
| 3      | 0.100 | 0.50 | 0.6644444 | 0.11168831  |
| 3      | 0.100 | 1.00 | 0.7044444 | 0.24604540  |

From the table, we can see degree = 3, scale = 0.1 and C = 1 is the best model with the accuracy 70 percents.

RBF kernel with different C value:

| C    | Accuracy     | Kappa         |
|------|--------------|---------------|
| 0.25 | 0.6044444444 | 0.0000000000  |
| 0.50 | 0.5844444444 | -0.0380952381 |
| 1.00 | 0.6444444444 | 0.1476190476  |

The final values used for the model were  $\sigma = 0.08413063677$  and  $C = 1$  with the highest accuracy 64 percents.

## References

- [1] Vanderbei, R. J. (2008). Linear Programming: Foundations and Extensions. Boston, MA: Robert J. Vanderbei.
- [2] Murphy, K. P. (2013). Machine learning: a probabilistic perspective. Cambridge, Mass.: MIT Press.
- [3] Russell, S. J., Norvig, P. (2016). Artificial intelligence: a modern approach. Boston: Pearson.
- [4] Stanford UFLDL. (n.d.). Logistic Regression. Retrieved May 30, 2017, from <http://ufldl.stanford.edu/tutorial/supervised/LogisticRegression/>
- [5] Holmes, S. (2001, September 02). Maximum Likelihood Estimation. Retrieved from <http://statweb.stanford.edu/susan/courses/s200/lectures/lect11.pdf>
- [6] Tibshirani, R. (2013, March 21). Modern regression 2: The lasso. Retrieved from <http://www.stat.cmu.edu/ryantibs/datamining/lectures/17-modr2.pdf>
- [7] Rai, P. (2011, September 15). <https://www.cs.utah.edu/piyush/teaching/15-9-print.pdf>. Retrieved from <https://www.cs.utah.edu/piyush/teaching/15-9-print.pdf>
- [8] 1.4. Support Vector Machines¶. (n.d.). Retrieved May 30, 2017, from <http://scikit-learn.org/stable/modules/svm.html>
- [9] Schneider, J. (1997, February 9). Cross Validation. Retrieved from <https://www.cs.cmu.edu/schneide/tut5/node42.html>
- [10] T. (n.d.). Artificial Intelligence Neural Networks. Retrieved May 30, 2017, from <https://www.tutorialspoint.com/artificialintelligence/>
- [11] Tagkopoulos, I. (n.d.). Logistic Regression and Classification. Retrieved 2016, from <https://canvas.ucdavis.edu/courses/47607/files?preview=201598>
- [12] R. Berwick, Idiot, V. (n.d.). An Idiot’s guide to Support vector machines (SVMs). Retrieved from <http://www.svms.org/tutorials/Berwick2003.pdf>
- [13] Introduction to Support Vector Machines¶. (n.d.). Retrieved May 30, 2017, from <http://docs.opencv.org/2.4/doc/tutorials/ml>
- [14] Je ries J.R., Butch N.P., Kirshenbaum K., Saha S.R., Weir S.T., Vohra Y.K., Paglione J. (2012). The sup- pression of magnetism and the development of superconductivity within the collapsed tetragonal phase. 10.1103/PhyRevB(85)18451

- [15] Housecroft, Catherine E. Inorganic Chemistry. 4th ed. Harlow: Prentice Hall, 2012. Print.
- [16] Hartl, F. (2015, October 05). Logistic Regression – Geometric Intuition. Retrieved June 01, 2017, from <https://florianhartl.com/logistic-regression-geometric-intuition.html>

## 5 Appendix

Program 1

```
require(glmnet)
Data <- read.csv("/Users/mengdaxu/Desktop/chem/Dataset.csv")
x <- as.matrix(Data[,1:15])
y <- as.matrix(Data[,16])
y[y>0] <- 1
set.seed(999)
cv.lasso <- cv.glmnet(x, y, family='binomial', alpha=1, standardize=TRUE, type.m
# Results
plot(cv.lasso$glmnet.fit, xvar="lambda", label=TRUE)
cv.lasso$lambda.min
cv.lasso$lambda.1se
plot(cv.lasso)
coef(cv.lasso, s=cv.lasso$lambda.min)
rank(abs(coef(cv.lasso, s=cv.lasso$lambda.min)))
```

program 2

```
library(caret)
library(caTools)
Data <- read.csv("/Users/mengdaxu/Desktop/chem/Dataset.csv")
x <- as.matrix(Data[,1:15])
y <- as.matrix(Data[,16])
y[y>0] <- "yes"
y[y=0] <- "no"
train_control <- trainControl(method="cv", number=5)
train(x,y,trControl = train_control,method="svmLinear")
train(x,y,trControl = train_control,method="svmPoly")
train(x,y,trControl = train_control,method="svmRadial")
```

program 3

```
A=dataset('XLSFile','update.xlsx');
%%A=datasample(A,48,'Replace',false);
TC=double(A(1:48,16));
TC_train=TC(1:38);
TC_test=TC(39:48);
data=A(1:48,1:15);
data=double(data);
data=zscore(data);
for i=1:48
    if(TC(i,1)==0)
        TC(i)=0;
    else
        TC(i)=1;
    end
end
trainmatrix=ones(48,16);
testmatrix=ones(10,16);
for i=1:15
    trainmatrix(1:48,i+1)=data(1:48,i);
    testmatrix(1:10,i+1)=data(39:48,i);
end
w=lgr2(trainmatrix,TC,16,48);
count_0=0;
t1=-1*trainmatrix*w;
score_0=zeros(10,1);
result_0=zeros(10,1);
for i=1:38
    tempt=exp(t1(i));
    score_0(i)=1/(tempt+1);
    if(score_0(i)<0.6)
        result_0(i)=0;
    else
        result_0(i)=1;
    end
    if(result_0(i)==TC_train(i,1))
        count_0=count_0+1;
    end
end
end

for i=1:10
    if(TC_test(i,1)==0)
        TC_test(i,1)=0;
    else
        TC_test(i,1)=1;
    end
end
```



```

        end
    end
    count=0;
    t1=-1*testmatrix*w;
    score=zeros(10,1);
    result=zeros(10,1);
    for i=1:10
        tempt=exp(t1(i));
        score(i)=1/(tempt+1);
        if(score(i)<0.6)
            result(i)=0;
        else
            result(i)=1;
        end
        if(result(i)==TC_test(i,1))
            count=count+1;
        end
    end
end

```

```

program 4

function w=lgr2(X,y,n,m)
w=ones(n,1);
fprintf('n is %f ',n);
a=0.0000001;
sigmwx=ones(m,1);
temps=ones(m,1);
WX=-1*(X*w);

for i=1:m
    sigmwx(i)=1/(1+exp(WX(i)));
end
c=0;
while(1)
    wold=w;
    c=c+1;
    for j=1:n
        for k=1:m
            temps(k)=-1*(y(k)-sigmwx(k))*X(k,j);
        end
        w(j)=w(j)-a*(sum(temps)+100*w(j));
    end

    WX=-1*(X*w);
    for i=1:m
        sigmwx(i)=1/(1+exp(WX(i)));
    end

    test=norm(w-wold);
    if (norm(w-wold)<0.000001)
        fprintf('c is %f ',c)
        break;
    end
end

end

```