8-7-2008

Clustering Via Supervised Support Vector Machines

Sepehr Merat

University of New Orleans

Clustering Via Supervised Support Vector Machines

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

by
Sepehr Merat
B.Sc. Simon Fraser University, 2006

August, 2008

I dedicate my work to my beloved parents for all their support.

Acknowledgment

I deeply thank my dear advisor Dr. Winters-Hilt for all of his help and support. He also taught me the importance of common sense, simplicity and maturity in research, work and life in general.

I also thank my beloved parents and brothers for their unconditional love and continuous support.

Contents

List of Figures

List of Tables

Abstract

An SVM-based clustering algorithm is introduced that clusters data with no a priori knowledge of input classes. The algorithm initializes by first running a binary SVM classifier against a data set with each vector in the set randomly labeled. Once this initialization step is complete, the SVM confidence parameters for classification on each of the training instances can be accessed. The lowest confidence data (e.g., the worst of the mislabeled data) then has its labels switched to the other class label. The SVM is then re-run on the data set (with partly re-labeled data). The repetition of the above process improves the separability until there is no misclassification. Variations on this type of clustering approach are shown.

Keywords: clustering, machine learning, pattern recognition, support vector machines, supervised learning, unsupervised learning

Chapter 1

Introduction

1.1 Motivation

The goal of clustering analysis is to partition objects into groups, such that the members of each group are more "similar" to each other than the members of other groups. The similarity is determined subjectively as it does not have a universally agreeable upon definition. In [1] the author suggests a formal perspective on the difficulty in finding such a unification, in the form of an impossibility theorem: for a set of three simple properties, there is no clustering function satisfying all three. Furthermore, the author demonstrates that relaxations of these properties expose some of the interesting (and unavoidable) trade-offs at work in well-studied clustering techniques such as single-linkage, sum-of-pairs, k-means, and k-median.

Ideally, one would like to solve the clustering problem given all the known and unknown objective functions. This is provably an NP-Hard problem (see [2]). This work is dedicated to provide a new perspective on clustering by introducing an algorithm that does not require an objective function. Hence, it does not inherit the limitations of an embedded objective function. The algorithm in its bare form is capable of suggesting solutions that can be later evaluated using known criterion functions (cluster validators).

1.2 Overview

This work is organized into 5 chapters. In Chapter 2 we will introduce kernel-based classification along with Support Vector Machines. Notations and conventions are also presented in this chapter. In Chapter 3 we introduce kernel polarization method and derive **Absdiff** and **Sentropic** regularized kernels. Two cluster validators, supervised and unsupervised, are then defined followed by an introduction to the SVM-Relabeler algorithm. This chapter is ended with methods of improving the SVM-Relabeler algorithm, and one particular improvement using stochastic methods is discussed in details. Results of the proposed algorithms are presented in Chapter 4 using two-dimensional, Iris, and DNA hairpin data sets. Finally, in Chapter 5 some important aspects of this work are discussed and direction for future work is given.

Chapter 2

Introduction to Classification

2.1 Theory of Classification

The problem of classification is to find a general rule, that based on a set of given observations a machine is trained to match a set of objects to their appropriate classes. In its simplest form the machine's task is to estimate a function $f : R^N \rightarrow \{\pm 1\}$, using a set of independent and identically distributed training data set according to an unknown probability distribution $P(\mathbf{x}, y)$, such that if future example pairs $(\mathbf{x}, y)$ are drawn from the same probability distribution $P(\mathbf{x}, y)$, f would be able to perfectly classify them as +1 if $f(x) \geq 0$ or -1 if otherwise. To clarity, each example is a pair that includes a vector of features (or observations), $\mathbf{x}$, and a corresponding label (or response), y. In the case of the above binary classification the examples are in fact:

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in R^N \times Y, \qquad Y = \{\pm 1\}$$

The best function, f, is theoretically obtained by minimizing the expected risk function (expected error) (cf. [3]):

$$R[f] = \int l(f(\mathbf{x}), y) dP(\mathbf{x}, y) \tag{2.1}$$

where l is a suitable loss function. For instance, in the case of' "0/1 loss"

$$l(f(\mathbf{x}), y) = \Theta(-yf(\mathbf{x}))$$

where Θ is the Heaviside function ($\Theta(z) = 0$ for $z < 0$ and $\Theta(z) = 1$ otherwise.) In most realistic cases $P(\mathbf{x}, y)$ is unknown and therefore the risk function above cannot be used to find the optimum function f. To overcome this fundamental limitation one has to use the information *hidden* in the limited training examples and the properties of the function class F to approximate this function. Hence instead of minimizing the expected risk in (2.1), one may minimize the *empirical risk*

$$R_{emp}[f] = 1/n \sum_{i=1}^{n} l(f(\mathbf{x_i}), y). \tag{2.2}$$

As a by product, the learning machine can ensure that for $n \to \infty$ the emperical risk will asymptotically converge to expected risk, but for a small training set the resulting deviations are often large. This leads to a phenomenon called "over-fitting." Then a small generalization error cannot be obtained by simply minimizing the training error (2.2). One way to avoid the over-fitting dilemma is to restrict the complexity of the function class that one chooses the function from [4]. The intuition, which will be formalized in the following is that a "simple" (e.g., linear) function that explains most of the data is preferable to a complex one (Occams razor). Typically one introduces a regularization term to limit the complexity of the function class from which the learning machine can choose. This way the model selection – finding an optimal complexity of the function – is necessary (cf. [5]).

A specific way of controlling the complexity of a function class is given by the Vapnik-Chervonenkis (VC) theory and the structural risk minimization (SRM) principle [4], [6]. Here the concept of complexity is captured by the VC dimension h of the function class F that the estimate f is chosen from. The following set of definition clarify the rest of this discussion.

Definition 1 (Shattering) *A Learning Machine f can shater a set of points $x_1, x_2, \ldots, x_n$*

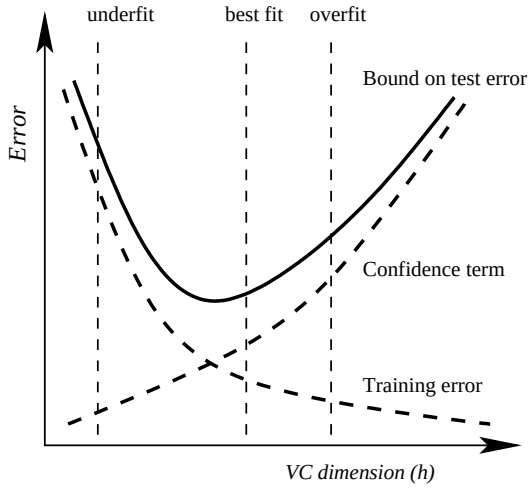


Figure 2.1: This figure illustrates relation (2.3). In practice the goal is to find the best trade-off between empirical error and the complexity. This figure is based on the illusteration in [4]

if and only if for every possible training set of the form $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ *there exists some parameter set* α *that gets zero training error (empirical risk).*

Definition 2 (VC Dimension) *Given a learning machine* f*, the VC-dimension* h *is the maximum number of points that can be arranged so that* f *shatter them.*

Roughly speaking, the VC dimension measures how many (training) points can be shattered (i.e., separated) for all possible labelings using functions of the class. Constructing a nested family of function classes $F_1 \subset \ldots \subset F_k$ with nondecreasing VC dimension the SRM principle proceeds as follows:

Definition 3 (SRM Principle) *Let* $f_1, \ldots, f_k$ *be the solutions of the empirical risk minimization (2.2) in the function classes* F_i*. SRM chooses the function class* F_i *(and the function* f_i *) such that an upper bound on the generalization error is minimized which can be computed making use of theorems such as the following one (see Figure (2.1)).*

Theorem 4 (Expected Risk Upperbound) *Let* h *denote the VC dimension of the function class* F *and let* R_{emp} *be defined by (2.2) using the "0/1 loss." For all*

delta > 0 *and* $f \in F$ *the inequality bounding the risk*

$$R[f] \leq R_{emp}[f] + \sqrt{\frac{h(ln\frac{2n}{h} + 1) - ln\frac{\delta}{4}}{n}} \tag{2.3}$$

holds with probability of at least $1 - \delta$ *for* $n > h$ *.([4], [6])*

Note, this bound is only an example and similar formulations are available for other loss functions (cf. [6]) and other complexity measures, e.g., entropy numbers [7]. In the equation (2.3): the goal is to minimize the generalization error $R[f]$, which can be achieved by obtaining a small training error $R_{emp}[f]$ while keeping the function class as small as possible. Two extremes can arise for equation (2.3): i) a very small function class (like F_1) yields a vanishing square root term, but a large training error might remain, while ii) a huge function class (like F_k) may give a vanishing empirical error but a large square root term. The best class is usually in between (see Figure (2.1)), as one would like to obtain a function that explains the data quite well and to have a small risk in obtaining that function. This is very much in analogy to the bias-variance dilemma scenario described for neural networks (see, e.g., [8]).

2.2 Kernel Feature Spaces

The so-called *curse of dimensionality* from statistics says essentially that the difficulty of an estimation problem increases drastically with the dimension N of the space, since in principle as N increases, the number of required patterns to sample grows exponentially. This statement casts doubts in using high dimensional feature vectors as input to learning machines. Fortunately, results from statistical learning theory [4] also show that the likelihood of data separability by linear learning machines is proportional to its dimensionality.

Therefore, instead of working in the $\mathbb{R}^N$, one can design algorithms to work in feature space, $\mathcal{F}$, where the data has much higher dimension. This can be done via the following mapping

$$\Phi : \mathbb{R}^N \to \mathcal{F}, \quad \mathbf{x} \mapsto \Phi(\mathbf{x}). \tag{2.4}$$

In (2.4) the data $\mathbf{x_1}, \ldots, \mathbf{x_n} \in \mathbb{R}^N$ is mapped into a potentially much higher dimensional feature space $\mathcal{F}$. For a given learning algorithm one now considers the same algorithm in $\mathcal{F}$ instead of $\mathbb{R}^N$. Hence, the learning machine works with the following sample:

$$(\Phi(\mathbf{x}_1), y_1), \ldots, (\Phi(\mathbf{x}_n), y_n) \in \mathcal{F} \times Y, \qquad Y = \{\pm 1\}$$

It is important to note that this mapping is also implicitly done for (one hidden layer) neural networks, radial basis networks (cf. [9]) and boosting algorithms (cf. [10]) where the input data is mapped to some representation given by the hidden layer, the radial basis function (RBF) bumps or the hypotheses space, respectively.

As a consequence of the above discussion, the dimensionality of the data does not detract us from finding a good solution, but it is rather the complexity of the function class F that contributes the most to the complexity of the problem at hand. Similarly, in practice one would never need to know the mapping function Φ, and therefore the complexity and intractability of computing the actual mapping is also irrelevant to the complexity of the problem of classification. To this end, algorithms are transformed to take advantage of a method called the *Kernel Trick*.

Definition 5 (Kernel Trick) *Achieve the following two objectives when using a learning machine:*
1) Rewrite our learning algorithm so that instead of using $\Phi(\mathbf{x_1})$ *and* $\Phi(\mathbf{x_2})$ *directly, it only uses the dot-product* $K(\mathbf{x_1}, \mathbf{x_2}) = \Phi(\mathbf{x_1}) \cdot \Phi(\mathbf{x_2})$*, and*
2) Compute the dot-products $K(\mathbf{x_1}, \mathbf{x_2})$ *in a manner, that avoids computing* $\Phi(\mathbf{x_1})$ *and* $\Phi(\mathbf{x_2})$ *explicitly.*

Kernel trick in definition (5) requires the knowledge of the appropriateness of a given function, K, to be used as a kernel. *Mercer's Condition* is a way to test for admission, and is stated in the following theorem.

Theorem 6 *[Mercer's Condition][3] For a given function,* K*, there exists a mapping* Φ *and an expansion:*

$$K(\mathbf{x}, \mathbf{y}) = \sum_i \Phi(\mathbf{x})_i \Phi(\mathbf{y})_i$$

if and only if, for any function, $g(\boldsymbol{x})$ *such that*

$$\int g(\boldsymbol{x})^2 d\boldsymbol{x}$$

is finite, then

$$\int\int K(\mathbf{x},\mathbf{y})g(\boldsymbol{x})g(\boldsymbol{y})d\boldsymbol{x}d\boldsymbol{y} \geq 0$$

Note, that it is usually difficult to apply Theorem (6) directly, since the final integration is hard to perform for complex candidate kernel functions.

To summarize, any linear classifier that can be formulated to avoid explicit calculation of the mapping function Φ, could be used to classify nonlinear data. The argument advanced in the next section expands upon the theory of admissible kernels.

2.3 Reproducing Kernel Hilbert Spaces – RKHS

In section (2.2) the notions of *Kernel* and *higher dimension maps* were introduced. Basically, kernels can be used to convert some linear learning algorithms to non-linear algorithms, without concerning us with the actual mapping. Admission of functions to be kernels depends on the disscussion that follows in this section. Formally put,

Definition 7 *Kernel is any symmetric function* $K : \chi \times \chi \to \mathbb{R}$. *Kernel is said to be:*

(i) Positive Definite if $\sum K(x_i, x_j)c_i c_j \geq 0$

(ii) Negative Definite if $\sum K(x_i, x_j)c_i c_j \leq 0$, *such that* $\sum c_i = 0$

for all $n \in \mathbb{N}$, $x_1, x_2, ..., x_n \in \chi$ *and* $c_1, c_2, ..., c_n \in \mathbb{R}$.

Consider the *Hilbert space*[1] $L_2 : \langle u, w \rangle = \int u(t)v(t)dt$. L_2 contains too many non-smooth functions and Reproducing Hilbert Kernel Spaces provides us with the restriction necessary to obtain a smaller well behaved Hilbert space. Given a kernel $K(x, x')$, the goal is to construct a Hilbert space in which K is a dot product. That is,

Definition 8 *A kernel, K, on set χ is pd iff $\exists$ a Hilbert space, H, and a map Φ : $\chi \to H$ such that $\forall x, y \in \chi$,*

$$K(x, y) = \langle \Phi(x), \Phi(y) \rangle$$

Further define the following *reproducing kernel map*:

$$\phi : x \mapsto K(\cdot, x)$$

that is, to each vector x in the input space χ we assign a function $K(\cdot, x)$. We can now construct RKHS, a vector space from the linear combinations of the range of the above map,

$$f(\cdot) = \sum_i \alpha_i K(\cdot, x_i).$$

The inner product of $f(\cdot)$ and $g(\cdot)$ for $g(\cdot) = \sum_j \beta_j K(\cdot, x'_j)$ is then defined as:

$$\langle f, g \rangle = \sum_i \sum_j \alpha_i \beta_j K(x_i, x'_j). \tag{2.5}$$

[1] A Hilbert space is a vector space $\mathcal{H}$ over a topological field (either $\mathbb{R}$ or $\mathbb{C}$) with an inner product $\langle \cdot, \cdot \rangle$. That is given vectors u, v, and $w \in \mathcal{H}$ and scalars λ and μ,

$$\begin{aligned} \langle \lambda u + \mu v, w \rangle &= \lambda \langle u, w \rangle + \mu \langle v, w \rangle \\ \langle u, v \rangle &= \langle v, u \rangle \\ \langle u, u \rangle &\geq 0 \\ \langle u, u \rangle = 0 &\to x = 0 \end{aligned}$$

From $\langle \cdot, \cdot \rangle$ we get a norm $|| \cdot ||$ via $||u|| = \sqrt{\langle u, u \rangle}$. This norm allows us to define notions of convergence. Adding all limit points of Cauchy sequences to our space yields a Hilbert space – a complete inner product space.

Note that for any given $f(\cdot)$ as defined above,

$$\langle K(\cdot, x), f \rangle = \sum_i \alpha_i K(x_i, x) = f(x)$$

shows that the kernel, k, is the *representer of evaluation.* Finally,

$$\langle K(\cdot, x), K(\cdot, x')f \rangle = K(x, x')$$

is the *reproducing property* of the kernel.

2.4 Component Analysis

2.4.1 Principal Component Analysis – PCA

The goal of PCA is to find a new set of dimensions that better capture the variability of the data. In fact, the amount of variability captured by the new dimensions decreases with higher dimensions. That is the first dimension captures most of the variability; the second dimension is orthogonal to the first and captures as much variability as allowed by providing that constraint and so on. In a more mathematical way, PCA's tenet is to compute the most meaningful *basis* to re-express a noisy data set. The hope is that this new basis will filter out the noise an reveal the hidden structure. If the data is represented using a matrix with columns being the like attributes, PCA results in the following set of properties:

1- The covariance of each new two columns is *zero*.

2- Columns are ordered with respect to the amount of variability they captured.

3- First column contains as much and most of the variability of the data,

4- Subject to the constraint of orthogonality, each subsequent column contains as much variability as possible.

Futhermore, PCA contains the following assumptions:

1- Linearity,

2- Mean and variance are sufficient statistics[2],

3- Large variance is of structural importance, and

4- The principal components are orthogonal.

Supposing that the data matrix, X, is centered – meaning that the mean of each column is 0. Our objective is to find transformation matrix P such that the new

[2]A statistic $S(X)$ is sufficient for underlying parameter α if the conditional probability distribution of the input data X, given the statistic $S(X)$, is independant of the parameter, α. Formally,

$$Pr(X = x|S(X) = s, \alpha) = Pr(X = x|S(X) = s). \tag{2.6}$$

Sufficient statisitics are described in more detail in [11].

matrix Y can be written as:

$$Y = PX$$

where the rows of P are the *principal components* of X. Since it is desired that the columns are Y have *zero* covariance, the covariance matrix of Y should have all zeros on its off diagonals. Therefore, the covariance matrix $C = \frac{1}{n-1}YY^T$ needs to be diagonalized. With a substitution $A = XX^T$ we can rewrite C as:

$$C = \frac{1}{n-1}P(XX^T)TP^T = \frac{1}{n-1}PAP^T \tag{2.7}$$

Note A is symmetric it must be orthogonally diagonalizable. since:

$$A^T = (EDE^T)^T = E^{TT}D^TE^T = EDE^T = A \tag{2.8}$$

In PCA the trick is that P is chosen so that each row of P is an eigenvector of A and therefore, XX^T. Combining with (2.7), (2.8),

$$C = \frac{1}{n-1}PAP^T = \frac{1}{n-1}D \tag{2.9}$$

It is shown that P diagonalizes C and therefore the rows of P, principal components of X are eigenvectors of XX^T. Algorithm (1) demonstrates PCA procedure.

Algorithm 1 PCA Procedure

Require: data set X

1: Subtract the mean for each column of X
2: Calculate the covariance matrix S
3: Find the eigenvectors, P, and the eigenvalues, $\mathbf{v}$ of XX^T
4: Sort P rows based on $\mathbf{v}$'s values
5: Project the data set: $Y = PX$

2.4.2 Kernel Principal Component Analysis – KPCA

Derivation

Kernel PCA uses the kernel trick in section (2.2) to relax the linearity assumption in the regular PCA. There for KPCA is a generalization PCA to the case where we are not interested in principal components in input space, but rather in principal components of features which are non-linearly related to the input features. This amounts to taking higer-order correlations between input features.

Recall that the input space covariance matrix $S_I = 1/(n-1)XX^T$ can be written as $S_I = 1/(n-1)\sum_{j=1}^{n} \mathbf{x}_j \mathbf{x}_j^T$, where X is centered input feature matrix. In feature space this can be modified to:

$$S \equiv S_F = 1/(n-1)\sum_{j=1}^{n} \Phi(\mathbf{x}_j)\Phi(\mathbf{x}_j)^T.$$

Also recall that the principal components are calculated via:

$$\lambda P = SP = \frac{1}{n-1}\sum_{j=1}^{n}(\Phi(\mathbf{x}_j)\cdot P)\Phi(\mathbf{x}_j). \tag{2.10}$$

It can be seen from (2.10) that $P \in span\{\Phi(\mathbf{x}_1), \dots \Phi(\mathbf{x}_n)\}$ and therefore,

$$P = \sum_{j=1}^{n} \alpha_j \Phi(\mathbf{x}_j). \tag{2.11}$$

by multiplying $\Phi(\mathbf{x}_k)$ by λP in (2.10) we obtain:

$$\lambda(\Phi(\mathbf{x}_k)\cdot P) = \Phi(\mathbf{x}_k)\cdot SP \tag{2.12}$$

Furthermore, by combining (2.10) and (2.12) we obtain

$$\lambda\sum_{j=1}^{n}\alpha_j(\Phi(\mathbf{x}_k)\cdot\Phi(\mathbf{x}_j)) = \frac{1}{n-1}\sum_{j=1}^{n}\alpha_j(\Phi(\mathbf{x}_k)\sum_{i=1}^{n}\Phi(\mathbf{x}_i))(\Phi(\mathbf{x}_j)\cdot\Phi(\mathbf{x}_i)) \tag{2.13}$$
$$\forall k = 1, \dots, n$$

As before let K_{ij} be an $n \times n$ matrix defined by:

$$K_{ij} = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j). \tag{2.14}$$

Equation (2.13) is therefore simplified to:

$$(n-1)\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha} \tag{2.15}$$

The matrix K is positive semi-definite and hence the eigenvalues are all positive. First we sort eigenvalues using $\lambda_1 \leq \ldots \leq \lambda_n$ with corresponding eigenvectors as $\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^n$. Then we normalize $\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^n$ by requiring that the corresponding vectors in $\mathcal{F}$ be normalized such that $P^k \cdot P^k = 1$ for all $k \in \{\text{index of non-zero eigenvalues}\}$. This condition can be derived from (2.10) and (2.10) by requiring that

$$1 = \lambda_k(\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k) \tag{2.16}$$

Finally, to extract features of a new observation $\boldsymbol{x}$ one can project the mapped pattern $\Phi(\boldsymbol{x})$ onto P^k using

$$\begin{aligned} P^k \cdot \Phi(\boldsymbol{x}) &= \sum_{j=1}^{n} \alpha_j^k (\Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x})) \\ &= \sum_{j=1}^{n} \alpha_j^k k(\mathbf{x}_j, \mathbf{x}). \end{aligned} \tag{2.17}$$

Implementation

So far we assumed that the mapped data is centered in $\mathcal{F}$. That is:

$$\sum_{j=1}^{n} \Phi(\mathbf{x}_j) = 0 \tag{2.18}$$

We do not have to assume this since

$$\tilde{\Phi}(\mathbf{x}_i) \equiv \Phi(\mathbf{x}_i) - 1/n \sum_{j=1}^{n} \Phi(\mathbf{x}_j) \tag{2.19}$$

is centered for any given set of features $\mathbf{x}_j$, for all j between 1 to n. Therefore the previous analysis holds if we replace all Φ with $\tilde{\Phi}$. We can also write $\tilde{K}$ in terms of K which results in:

$$\begin{aligned}
\tilde{K_{ij}} &= \tilde{\Phi}(\mathbf{x}_i)^T \tilde{\Phi}(\mathbf{x}_j) \\
&= (\Phi(\mathbf{x}_i) - 1/n \sum_{k=1}^{n} \Phi(\mathbf{x}_k))^T (\Phi(\mathbf{x}_j) - 1/n \sum_{k=1}^{n} \Phi(\mathbf{x}_k)) \\
&= \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j) - 1/n \sum_{k=1}^{n} \Phi(\mathbf{x}_k)^T \Phi(\mathbf{x}_j) - 1/n \sum_{k=1}^{n} \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_k) \\
&\quad + 1/n^2 \sum_{k,k'=1}^{n} \Phi(\mathbf{x}_k)^T \Phi(\mathbf{x}_{k'})
\end{aligned}$$

By letting $(1_n)_{ij} \equiv 1/n$, we can rewrite the above as:

$$\tilde{K_{ij}} = K - 1_n K - K 1_n + 1_n K 1_n. \tag{2.20}$$

This modification does not affect the formulation as long as K is replaced with equation (2.20). This process is demonstrated in algorithm (2).

Algorithm 2 KPCA Procedure

Require: data set X
Require: Feature vector $\boldsymbol{x}$
1: Calculate kernel matrix K
2: Calculate $\tilde{K}$ from (2.20)
3: Diagonalize $\tilde{k}$ using the eigenvalue decomposition in (2.10)
4: Normalize eigenvectors with constraint in (2.16)
5: Project $\boldsymbol{x}$ onto eigenvectors using (2.17)

2.5 Supervised Learning

2.5.1 Margins

As justified in section (2.2) we can assume that the training sample is separable by a hyperplane, and therefore the following learning machine.

$$f(\mathbf{x}) = (\omega \cdot \mathbf{x}) - b \tag{2.21}$$

In [4] it is shown that the for this class of hyperplanes the VC dimension can be bounded in terms of another quantity, the margin. The margin is defined as the minimal distance of a sample to the decision surface (see section 2.5.2). It is rather intuitive that the thicker the margin the better the training would be, and in fact this quantity could be measured by the length of the weight vector, ω in (2.21). since we assumed that the training data is separable we can rescale and normalize ω such that the points closest to the hyperplane is a unit away from the hyperplane (i.e., the canonical representation of the hyperplane). This can be done by requiring that $|(\omega \cdot \mathbf{x}) - b)| = 1$. Now consider two samples $\mathbf{x_1}$ and $\mathbf{x_2}$ from different classes with $(\omega \cdot \mathbf{x_1}) - b = 1$ and $(\omega \cdot \mathbf{x_2}) - b = -1$, respectively. The margin is given by the distance of these two points, measured perpendicular to the hyperplane and therefore, $\omega/||\omega||_2 \cdot (\mathbf{x_1} - \mathbf{x_2}) = 2/||\omega||_2$. The concept of margins links this intuitive result to another result from VC dimension of linear hyperplanes [4]. In fact, using the notation from [4] and Theorem (9), we can control its VC dimension if we bound the function class to be below $2/\Lambda$.

Theorem 9 *[4] For the class of separating hyperplane with margin $||\omega||_2$ the inequality:*

$$h \leq \Lambda^2 R^2 + 1 \tag{2.22}$$

and

$$||\omega||_2 \leq \Lambda \tag{2.23}$$

holds where R is the radius of the smallest sphere enclosing the data.

2.5.2 Support Vector Machines

In this section we treat a supervised separating hyperplane learning machine, called Support Vector Machines (SVMs), which incorporate the results in section (2.5.1). As a consequence SVMs are less susceptible to over-fit even with a poor choice of kernel and parameters. The rest of this section focuses on the specifics of derivation and implementation of such learners.

Derivation

Using the convention of implied sum on repeated Greek indices, throughout the rest of this derivation, feature vectors are denoted by x_{ik}, where index i labels the feature vectors ($1 \leq i \leq M$) and index k labels the N feature vector components ($1 \leq k \leq N$). Similarly, labeling of training data is done using label variable $y_i = \pm 1$ (with sign according to whether the training instance was from the positive or negative class). For hyperplane separability, elements of the training set must satisfy the following conditions: $\omega_\beta x_{i\beta} - b = +1$ for i such that $y_i = +1$, and $\omega_\beta x_{i\beta} - b = -1$ for $y_i = -1$, for some values of the weight coefficients $\omega_1, \ldots, \omega_N$, and b. This can be written more concisely as: $y_i(\omega_\beta x_{i\beta} - b) - 1 = 0$. Data points that satisfy the equality in the above are known as "support vectors" (or "active constraints").

Once training is complete, discrimination is based solely on position relative to the discriminating hyperplane: $\omega_\beta x_{i\beta} - b = 0$. The boundary hyperplanes on the two classes of data are separated by a distance $2/||\omega||_2$, known as the "margin" (section 2.5.1), where $||\omega||_2^2 = \omega_\beta \omega_\beta$. Note that by increasing the margin between the separated data as much as possible the optimal separating hyperplane is obtained. Furthermore, the goal to maximize $||\omega||_2^{-1}$ can be restated as the goal to minimize $||\omega||_2^2$. The Lagrangian variational formulation then selects an optimum defined at a saddle point of

$$L(\omega, b; \alpha) = \frac{\omega_\beta \omega_\beta}{2} - \alpha_\gamma y_\gamma (\omega_\beta x_{\gamma\beta} - b) - \alpha_0 \tag{2.24}$$
$$\text{Subject to:} \quad \alpha_0 = \sum_\gamma \alpha_\gamma, \ \alpha_\gamma \geq 0 \ (1 \leq \gamma \leq M).$$

Therefore, the saddle point is obtained by minimizing with respect to $\{\omega_1, \ldots, \omega_N\, b\}$ and maximizing with respect to $\{\alpha_1, \ldots, \alpha_M\}$.

Now, if $y_i(w_\beta x_{i\beta} - b) - 1 \geq 0$, then maximization on α_i is achieved for $\alpha_i = 0$. On the other hand, if $y_i(w_\beta x_{i\beta} - b) - 1 = 0$, then there is no constraint on α_i and if $y_i(w_\beta x_{i\beta} - b) - 1 < 0$, there is a constraint violation, and $\alpha_i \to \infty$. In the case of absolute separability, the last case will eventually be eliminated for all α_i. But if separability is not possible it is natural to limit the size of α_i by some constant upper bound, i.e., $max(\alpha_i) = C$, for all i. This is equivalent to another set of inequality constraints with $\alpha_i = C$. Introducing sets of Lagrange multipliers, ξ_γ and μ_γ for $1 \leq \gamma \leq M$, to achieve this, the Lagrangian becomes:

$$L(\omega, b; \alpha, \xi, \mu) = \frac{\omega_\beta \omega_\beta}{2} - \alpha_\gamma[y_\gamma(\omega_\beta x_{\gamma\beta} - b) + \xi_\gamma] + \alpha_0 + \xi_0 C - \mu_\gamma \xi_\gamma \tag{2.25}$$

$$\text{subject to: } \xi_0 = \sum_\gamma \xi_\gamma,\ \alpha_0 = \sum_\gamma \alpha_\gamma \text{and} \alpha_\gamma \geq 0 \text{and} \xi_\gamma \geq 0 \quad (1 \leq \gamma \leq M)$$

At the variational minimum on the $\{\omega_1, \ldots, \omega_N, b\}$ variables, $\omega_\beta = \alpha_\gamma y_\gamma x_{\gamma\beta}$, and the Lagrangian (2.25) simplifies to:

$$L(\alpha) = \alpha_0 - \frac{\alpha_\delta y_\delta x_{\delta\beta} \alpha_\gamma y_\gamma x_{\gamma\beta}}{2} \tag{2.26}$$

$$\text{subject to: } 0 \leq \alpha_\gamma \leq C (1 \leq \gamma \leq M) \quad \text{and} \quad \alpha_\gamma y_\gamma = 0,$$

where only the variations that maximize in terms of the α_γ remain (also known as the *Wolfe Transformation*). In this form the computational task can be greatly simplified.

By introducing an expression for the discriminating hyperplane, $f_i = \omega_\beta x_\beta - b = \alpha_\gamma y_\gamma x_{\gamma\beta} x_{i\beta} - b$, the variational solution for $L(\alpha)$ reduces to the following set of relations (known as the Karush-Kuhn-Tucker, or KKT, relations):

$$\text{(i)} \qquad \alpha_i = 0 \quad \Leftrightarrow \quad y_i f_i \geq 1 \tag{2.27}$$

$$\text{(ii)} \qquad 0 < \alpha_i < C \quad \Leftrightarrow \quad y_i f_i = 1 \tag{2.28}$$

$$\text{(iii)} \qquad \alpha_i = C \quad \Leftrightarrow \quad yifi \leq 1 \tag{2.29}$$

When the KKT relations are satisfied for all of the α_γ (with $\alpha_\gamma y_\gamma = 0$ maintained) the solution is achieved. Note that the constraint $\alpha_\gamma y_\gamma = 0$ is satisfied for the initial choice of multipliers by setting the α's associated with the positive training instances to $1/N^+$ and the α's associated with the negatives to $1/N^-$, where N^+ is the number of positives and N^- is the number of negatives.

Once the Wolfe transformation is performed it is apparent that the training data (Support Vectors in particular, relation (2.28)) enter into the Lagrangian solely via the inner product $x_{i\beta}x_{j\beta}$. Likewise, the discriminator f_i, and KKT relations, are also dependent on the data solely via the $x_{i\beta}x_{j\beta}$ inner product. Generalization of the SVM formulation to data-dependent inner products other than $x_{i\beta}x_{j\beta}$ are possible using the discussion advanced in section (2.2) and in particular definition (5). Thoroughout the rest of this section, this inner product is notated as: $x_{i\beta}x_{j\beta} \mapsto \Phi(x_i)_\beta \Phi(x_j)_\beta = K_{ij}$.

Implementation

To implement the SVM problem one has to solve the (convex) quadratic programming (QP) problem in equation (2.26). There has been a significant effort to solve this QP problem (cf. [12]) but unfortunately only a few methods and algorithms are free of unrealistic assumptions about the geometry of the training data [13]. Here, we describe two of these methods that allow for reasonably fast convergence with small memory footprint.

Chunking A key observation in solving large scale SVM problems is the sparsity of the solution. Depending on the problem, many of the α's will either be zero or C. If we could easily identify $\alpha = 0$, the corresponding calculation could be avoided without changing the value of the quadratic form. Furthermore, recalling that the optimality is encoded in the KKT relations (starting at 2.27). A method called chunking [14] is described, making use of the sparsity and the KKT conditions. At every step chunking solves the problem containing all $\alpha \neq 0$ plus some of the KKT violating αs. The size of this problem varies but is finally equal to the number of non-zero αs. While this technique is suitable for fairly large problems it is still limited by the maximal number of support vectors that it can handle. This is because for every

chunk a quadratic optimizer is still necessary. Further details on this implementation is found at [15].

Sequential Minimal Optimization – SMO Sequential Minimal Optimization (SMO), first proposed by [13], is an extreme case of chunking, such that in each iteration it solves a quadratic problem for only two αs. The benefit is apparent when it is shown that this can be done analytically, and therefore no quadratic optimizer is necessary.

The method described here follows the description of [13] and begins by selecting a pair of Lagrange multipliers, $\{\alpha_1, \alpha_2\}$, where at least one of the multipliers has a violation of its associated KKT relations. For simplicity it is assumed in what follows that the multipliers selected are those associated with the first and second feature vectors: $\{x_1, x_2\}$. The SMO procedure then "freezes" variations in all but the two selected Lagrange multipliers, permitting much of the computation to be circumvented by use of analytical reductions:

$$\begin{aligned} L(\alpha_1, \alpha_2; \alpha_{\beta' \geq 3}) = \quad & \alpha_1 + \alpha_2 - \frac{(\alpha_1^2 K_{11} + \alpha_2^2 K_{22} + 2\alpha_1 \alpha_2 y_1 y_2 K_{12})}{2} \\ & -\alpha_1 y_1 v_1 - \alpha_2 y_2 v_2 + \alpha_{\beta'} U_{\beta'} - \frac{\alpha_{\beta'} \alpha_{y'} y_{\beta'} K_{\beta' y'}}{2} \end{aligned} \tag{2.30}$$

where $\beta', \gamma' = 3$, and $v_i = \alpha_{\beta'} y_{\beta'} K_{i\beta'}$. Due to the constraint $\alpha_\beta y_\beta = 0$, we have the relation: $\alpha_1 + s\alpha_2 = -\gamma$, where $\gamma = y_1 \alpha_{\beta'} y_{\beta'}$ with $\beta' \geq 3$ and $s = y_1 y_2$. Substituting the constraint to eliminate references to α_1, and performing the variation on α_2:

$$\frac{\partial L(\alpha_2; \alpha_{\beta' \geq 3})}{\partial \alpha_2} = (1 - s) + \eta \alpha_2 + s\gamma(K11 - K22) + s y_1 v_1 - y_2 v_2,$$

where $\eta = (2K12 - K11 - K22)$. Since v_i can be rewritten as $v_i = \omega_\beta x_{i\beta} - \alpha_1 y_1 K_{i1} - \alpha_2 y_2 K_{i2}$, the variational maximum $\partial L(\alpha_2; \alpha_{\beta' \geq 3})/\partial \alpha_2 = 0$ leads to the following update rule:

$$\alpha_2^{new} = \alpha_2^{old} - \frac{y_2((\omega_\beta x_{1\beta} - y_1) - (\omega_\beta x_{2\beta} - y_2))}{\eta} \tag{2.31}$$

Once α_2^{new} is obtained, the constraint $\alpha_2^{new} = C$ must be re-verified in conjunction with the $\alpha_\beta y_\beta = 0$ constraint. If the $L(\alpha_2; \alpha_{\beta'} \geq 3)$ maximization leads to a α_2^{new}

that grows too large, α_2^{new} must be "clipped" to the maximum value satisfying the constraints. For example, if $y_1 \neq y_2$, then increases in α_2 are matched by increases in α_1. So, depending on whether α_2 or α_1 is nearer its maximum of C, we have:

$$max(\alpha_2) = argmin\{\alpha_2 + (C - \alpha_2); \alpha_2 + (C - \alpha_1)\}.$$

Similar arguments provide the following boundary conditions:

$$\begin{aligned} (i) \quad & s = -1 \rightarrow max(\alpha_2) = argmin\{\alpha_2; C + \alpha_2 - \alpha_1\} \bigwedge \\ & min(\alpha_2) = argmax\{0; \alpha_2 - \alpha_1\}, \text{and} \\ (ii) \quad & s = +1 \rightarrow max(\alpha_2) = argmin\{C; \alpha_2 + \alpha_1\} \bigwedge \\ & min(\alpha_2) = argmax\{0; \alpha_2 + \alpha_1 - C\}. \end{aligned}$$

In terms of the new $\alpha_2^{new,clipped}$, clipped as indicated above if necessary, the new α_1 becomes:

$$\alpha_1^{new} = \alpha_1^{old} + s(\alpha_2^{old} - \alpha_2^{new,clipped}) \tag{2.32}$$

where $s = y_1 y_2$ as before. After the new α_1 and α_2 values are obtained there still remains the task of obtaining the new b value. If the new α_1 is not "clipped" then the update must satisfy the non-boundary KKT relation: $y_1 f(x_1) = 1$, i.e., $f^{new}(x1) - y_1 = 0$. By relating f^{new} to f^{old} the following update on b is obtained:

$$b_1^{new} = b - (f^{new}(x_1) - y_1) - y_1(\alpha_1^{new} - \alpha_1^{old})K_{11} - y_2(\alpha_2^{new,clipped} - \alpha_2^{old})K_{12} \tag{2.33}$$

If α_1 is clipped but α_2 is not, the above argument holds for the α_2 multiplier and the new b is:

$$b_2^{new} = b - (f^{new}(x_2) - y_2) - y_2(\alpha_2^{new} - \alpha_2^{old})K_{22} - y_1(\alpha_1^{new,clipped} - \alpha_1^{old})K_{12} \tag{2.34}$$

If both α_1 and α_2 values are clipped then any of the b values between b_1^{new} and b_2^{new}

is acceptable, and following the SMO convention, the new b is chosen to be:

$$b^{new} = \frac{b_1^{new} + b_2^{new}}{2} \tag{2.35}$$

In [16] the authors described an improved SMO algorithm that instead of updating a single threshold, b, it updates the bounds on permissible thresholds. They report substantial improvement in speed, especially for extreme C values.

2.6 Unsupervised Learning (Clustering)

2.6.1 K-means

K-means is a simple yet popular algorithm for clustering a set of un-labled features vectors $\boldsymbol{X} : \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ that are drawn independently from the mixture density $p(\boldsymbol{X}|\boldsymbol{\theta})$ with a parameter set $\boldsymbol{\theta}$.

Derivation

At the heart of K-means algorithm is optimization of the sum-of-squared-error criterion function (SSE), J_i defined in definition (10).

Definition 10 (Sum-of-squared-error) *Given a cluster χ_i, the sum-of-squared, J_I is defined by*

$$J_i = \sum_{\boldsymbol{x} \in \chi_i} ||\boldsymbol{x} - \boldsymbol{m}_i||^2$$

where $\boldsymbol{m}_i$ is the mean of the samples belonging to χ_i.

The geometric interpretation of this criterion function is that for a given cluster χ_i the mean vector $\boldsymbol{m}_i$ is the centroid of the cluster by minimizing the length of the vector $\boldsymbol{x} - \boldsymbol{m}_i$. This can be shown by taking the variation of J_i with respect to the

"centroid" $\boldsymbol{m}_i$ and setting it zero,

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{m}_i} J_i &= \frac{\partial}{\partial \boldsymbol{m}_i} \sum_{\boldsymbol{x} \in \chi_i} (\boldsymbol{x} - \boldsymbol{m}_i)^T (\boldsymbol{x} - \boldsymbol{m}_i) \\ &= 2 \sum_{\boldsymbol{x} \in \chi_i} (\boldsymbol{x} - \boldsymbol{m}_i) = \boldsymbol{0}\end{aligned}$$

and solving for $\boldsymbol{m}_i$,

$$\boldsymbol{m}_i = 1/n_i \sum_{\boldsymbol{x} \in \chi_i} \boldsymbol{x} \tag{2.36}$$

where $n_i = |\chi_i|$ is the number of feature vectors belonging to χ_i. The total SSE for all of the clusters, J_e is the sum of SSE for individual clusters. The value of J_e depends on the cluster membership of the data, (i.e. the shape of the clusters), and the number of clusters. The optimal clustering is the one that minimizes J_e for a given number of clusters, k, and K-means tries to do just that. Algorithm (3) outlines the high-level implementation of the K-means algorithm.

Algorithm 3 K-means Clustering

Require: Number of clusters: k
Feature vectors: $\boldsymbol{x}_1, \ldots \boldsymbol{x}_n$
Initial mean vectors: $\boldsymbol{m}_1, \ldots \boldsymbol{m}_k$

1: **repeat**
2: classify feature vectors based on the mean vectors
3: re-compute the mean vectors
4: **until** the mean vectors remain unchanged

2.6.2 Kernel K-means

Kernel K-means [17] is another example of linear algorithms that since they could be formulated in terms of dot products, they could make use of kernel methods (KPCA described in (2.4.2) is another example).

Denote $M_{i\nu}$ to be the cluster assignment variables such that $M_{i\nu} = 1$ if and only if $\boldsymbol{x}_i$ belongs to cluster ν and 0 otherwise. As for K-means, the goal is to minimize the J_ν (definition 10) for all clusters, ν in feature space, by trying to find k means $\Phi(\boldsymbol{m}_\nu)$ such that each observation in the data set when mapped using Φ is close to at least one of the means. But the means lie in the span of $\Phi(\boldsymbol{x}_1, \dots (\boldsymbol{x}_n)$. Therefore, we can write them as:

$$\boldsymbol{\mu}_\nu \equiv \Phi(\boldsymbol{m}_\nu) = \sum_{j=1}^{n} \gamma_{\nu j} \Phi(\boldsymbol{x}_j). \tag{2.37}$$

We can then substitute this in the J_i of definition (10) to obtain,

$$\begin{aligned} J_\nu &= \sum_{\boldsymbol{x} \in \chi_i} ||\Phi(\boldsymbol{x}) - \boldsymbol{\mu}_\nu||^2 \\ &= \sum_{\boldsymbol{x} \in \chi_i} ||\Phi(\boldsymbol{x}) - \sum_{j=1}^{n} \gamma_{\nu j} \Phi(\boldsymbol{x}_j)||^2 \\ &= K(\boldsymbol{x}, \boldsymbol{x}) - 2 \sum_{j=1}^{n} \gamma_{\nu j} K(\boldsymbol{x}, \boldsymbol{x}_j) + \sum_{i,j=1}^{n} \gamma_{\nu i} \gamma_{\nu j} K(\boldsymbol{x}_i, \boldsymbol{x}_j) \end{aligned} \tag{2.38}$$

We initially assign random feature vectors to means. Then Kernel K-means proceeds iteratively as follows: each new remaining feature vectors, $\boldsymbol{x}_{t+1}$, is assigned to the closes mean μ_α:

$$M_{t+1,\alpha} = \begin{cases} 1 & \text{if for all } \nu \neq \alpha, ||\Phi(\boldsymbol{x}_{t+1}) - \boldsymbol{\mu}_\alpha||^2 < ||\Phi(\boldsymbol{x}_{t+1}) - \boldsymbol{\mu}_\nu||^2 \\ 0 & \text{otherwise} \end{cases} \tag{2.39}$$

or, in terms of the kernel function,

$$M_{t+1,\alpha} = \begin{cases} 1 & \text{if for all } \nu \neq \alpha, \\ & \sum_{i,j=1}^{n} \gamma_{\alpha i} \gamma_{\alpha j} K(\boldsymbol{x}_i, \boldsymbol{x}_j) - 2 \sum_{j=1}^{n} \gamma_{\alpha j} K(\boldsymbol{x}_{t+1}, \boldsymbol{x}_j) \\ & < \sum_{i,j=1}^{n} \gamma_{\nu i} \gamma_{\nu j} K(\boldsymbol{x}_i, \boldsymbol{x}_j) - 2 \sum_{j=1}^{n} \gamma_{\nu j} K(\boldsymbol{x}_{t+1}, \boldsymbol{x}_j) \\ 0 & \text{otherwise} \end{cases} \tag{2.40}$$

The update rule for the mean vector is then given by,

$$\boldsymbol{\mu}_{\alpha}^{t+1} = \boldsymbol{\mu}_{\alpha}^{t} + \Delta \left(\Phi(\boldsymbol{x}_{t+1} - \boldsymbol{\mu}_{\alpha}^{t} \right), \tag{2.41}$$

where,

$$\Delta \equiv \frac{M_{t+1,\alpha}}{\sum_{i=1}^{t+1} M_{i\alpha}} \tag{2.42}$$

Kernel k-means algorithm is identical to algorithm (3). Line 2 of this algorithm can be obtained using equation (2.38) and line 3 can be calculated using equation (2.41).

2.6.3 SVM-Internal Clustering

The SVM-Internal approach to clustering was originally defined by [18]. Data points are mapped by means of a kernel to a high dimensional feature space where we search for the minimal enclosing sphere. In what follows, Keerthi's method [16] is used to solve the dual (see implementation).

The minimal enclosing sphere, when mapped back into the data space, can separate into several components; each enclosing a separate cluster of points. The width of the kernel (say Gaussian) controls the scale at which the data is probed while the soft margin constant helps to handle outliers and over-lapping clusters. The structure of a data set is explored by varying these two parameters, maintaining a minimal number of support vectors to assure smooth cluster boundaries.

This section is dedicated to outline the derivation and the implementation of this algorithm.

Derivation

Let $\mathbf{x}$ be a data set of N points in R^d. As explained in section (2.2) using a non-linear transformation Φ (equation 2.4), we transform $\mathbf{x}$ to some high-dimensional space called Kernel space and look for the smallest enclosing sphere of radius R. Hence we have: $||\Phi(x_j) - a||_2^2 \leq R^2$ for all $j = 1, \ldots, N$; where a is the center of the

sphere. Soft constraints are incorporated by adding slack variables ζ_j':

$$||\Phi(x_j) - a||_2^2 \leq R^2 \quad \forall j = 1, \ldots, N \qquad (2.43)$$
$$\text{Subject to: } \zeta_j$$

We introduce the Lagrangian as:

$$L = R^2 - \sum_j \beta_j (R^2 + \zeta_j - ||\Phi(x_j) - a||_2^2) - \sum_j \zeta_j \mu_j + C \sum_j \zeta_j \qquad (2.44)$$
$$\text{Subject to: } \quad \beta_j \geq 0, \mu_j \geq 0$$

where C is the cost for outliers and hence $C \sum_j \zeta_j$ is a penalty term. Setting to zero the derivative of L w.r.t. R, a, and ζ we have: $\sum_j \beta_j = 1, a = \sum_j \beta_j \Phi(\mathbf{x}_j)$, and $\beta_j = C - \mu_j$. By substituting the above equations into the Lagrangian, we get the following dual formalism:

$$W = 1 - \sum_{i,j} \beta_i \beta_j K_{ij} \qquad (2.45)$$
$$\text{for } 0 \leq \beta_j \leq C \text{ and } K_{ij} = e^{-||\mathbf{x}_i - \mathbf{x}_j||/2\sigma^2}$$
$$\text{Subject to: } \sum_i \beta_i = 1$$

By KKT conditions we have: $\zeta_j \mu_j = 0$ and $\beta_j (R2 + \zeta_j - ||\Phi(\mathbf{x}_j) - a||_2^2) = 0$. Hence, in the kernel space of a data point x_j:

i) if $\zeta_j > 0$, then $\beta_j = C$ and hence it lies outside of the sphere i.e. $R^2 < ||\Phi(\mathbf{x}_j) - a||_2^2$. This point becomes a bounded support vector or *BSV*.

ii) if $\zeta_j = 0$, and $0 < \beta_j < C$, then it lies on the surface of the sphere i.e. $R^2 = ||\Phi(\mathbf{x}_j) - a||_2^2 2$. This point becomes a support vector or *SV*.

iii) If $\zeta_j = 0$, and $\beta_j = 0$, then $R^2 > ||\Phi(\mathbf{x}_j) - a||_2^2$ and hence this point is enclosed with-in the sphere.

Implementation

The following is based on the work of [16] and [19].

Referring to the dual representation at relation (2.45), for any data point x_k, the distance of its image in kernel space from the center of the sphere is given by

$$R^2(x_k) = 1 - 2\sum_i \beta_i K_{ik} + \sum_{i,j} \beta_i \beta_j K_{ij}.$$

The radius of the sphere, R, is $\{R(x_k)$ st. x_k is a Support Vector$\}$, hence data points which are Support Vectors lie on cluster boundaries. Outliers are points that lie outside of the sphere and thus do not belong to any cluster i.e. they are Bounded Support Vectors. All other points are enclosed by the sphere and therefore they lie inside their respective cluster. KKT Violators are given as:

(i) If $0 < \beta_i < C$ and $R(x_i) \neq R$,

(ii) If $\beta_i = 0$ and $R(x_i) > R$, and

(iii) If $\beta_i = C$ and $R(x_i) < R$.

The Wolfe dual is: $f(\beta) = min_\beta\{\sum_{i,j} \beta_i \beta_j K_{ij} - 1\}$. In the SMO decomposition, in each iteration we select β_i and β_j and change them such that $f(\beta)$ reduces. All other β's are kept constant for that iteration. Let us denote β_1 and β_2 as being modified in the current iteration. Also $\beta_1 + \beta_2 = (1 - \sum_{i=3} \beta_i) = s$, is a constant.

Furthermore, let $\sum_{i=3} \beta_i K_{ik} = C_k$ to obtain the SMO form:

$$f(\beta_1, \beta_2) = \beta_2\beta_1 + \sum_{i,j=3} \beta_i \beta_j K_{ij} + 2\beta_1\beta_2 K_{12} + 2\beta_1 C_1 + 2\beta_2 C_2$$

By eliminating β_1 we have:

$$f(\beta_2) = (s - \beta_2)^2 + \beta_2^2 + \sum_{i,j=3} \beta_i \beta_j K_{ij} + 2(s - \beta_2)\beta_2 K_{12} + 2(s - \beta_2)C_1 + 2\beta_2 C_2.$$

To minimize $f(\beta_2)$, we take the first derivative w.r.t. β_2 and equate it to zero, thus

$$f'(\beta_2) = 0 = 2\beta_2(1 - K_{12}) - s(1 - K_{12}) - (C_1 - C_2),$$

and we get the update rule:

$$\beta_2^{new} = \frac{C_1 - C_2}{2(1 - K_{12})} + s/2.$$

We also have an expression for $C_1 - C_2$ from:

$$R(x_1^2) - R(x_2^2) = 2(\beta_2 - \beta_1)(1 - K_{12}) - 2(C_1 - C_2),$$

thus

$$C_1 - C_2 = (R(x_2^2) - R(x_1^2))/2 + (\beta_2 - \beta_1)(1 - K_{12}),$$

Substituting, we have:

$$\beta_1^{new} = \beta_1^{old} - \frac{R(x_2^2) - R(x_1^2)}{4(1 - K_{12})} \tag{2.46}$$

This update rule along with the rest of the logic is shown in algorithms (4, 5, and 6) in a high level fashion.

Algorithm 4 High Level SVM-Internal Clustering

Require: Percentage of outliers is n, size of data (rows) is N

1: $C = 100/(N * n)$
2: initialize β
3: Initialize two different randomly chosen indices to values less than C such that $\sum_i \beta_i = 1$
4: Compute $R(x_i)^2$ for all i based on the current value of β
5: Divide data into three sets:
(I) $0 < \beta_i < C$,
(II) $\beta_i = 0$, and
(III) $\beta_i = C$.
6: Compute $R^2_{low} = max\{R(x_i)^2 | 0 \leq \beta_i < C\}$ and $R^2_{up} = min\{R(x_i)^2 | 0 < \beta_i \leq C\}$.
7: **repeat**
8: Switch between the two loops alternatively:
9: (1)
10: **for all** examples **do**
11: call examineExample()
12: **end for**
13: (2)
14: **for all** examples belonging to Set (I) **do**
15: call examinExample()
16: quit if $R^2_{low} - R^2_{up} < 2$
17: **end for**
18: **until** There are no KKT violaters left.

Algorithm 5 examineExample() Procedure

Require: an example

1: Check if the example is a KKT violator, that is if:
Set II and $R^2(x_i) > R^2_{up}$; choose R^2_{up} for joint optimization
Set III and $R^2(x_i) < R^2_{low}$; choose R^2_{low} for joint optimization
Set I and $R^2(x_i) > R^2_{up} + 2(tolerance) \bigvee R^2(x_i) < R^2_{low} - 2(tolerance)$; choose R^2_{low} or R^2_{up} for joint optimization depending on which gives a worse KKT violator
2: call jointOptimizer()

Algorithm 6 jointOptimizer() Procedure

Require: examples x_1 and x_2

$\eta = 4(1 - K_{12})$
$D = (R^2(x_2) - R^2(x_1))/\eta$
$L_1 = min\{(C - \beta_2), \beta_1\}$
$L_2 = min\{(C - \beta_1), \beta_2\}$
if $D > 0$ **then**
    $D = min\{D, L_1\}$
else
    $D = max\{D, -L_2\}$
end if
update: $\beta_2 = \beta_2 + D$
update: $\beta_1 = \beta_1 - D$
Re-compute $R^2(x_i)$ for all i based on the changes in β_1 and β_2
Re-compute R^2_{low} and R^2_{up} based on elements in Set I, $R^2(x_1)$ and $R^2(x_2)$

Chapter 3

SVM-Relabeler: An External Method of SVM Clustering

3.1 Introduction

Although the internal approach to SVM (see svm-internal) clustering is only weakly biased towards the shape of the clusters in the input space (the bias is for spherical clusters in the feature space), it still lacks robustness. In the case of most real-world problems and strongly overlapping clusters, the SVM- Internal Clustering algorithm above can only delineate the relatively small cluster cores. Additionally, the implementation of the formulation is tightly coupled with the initial choice of kernel; hence the static nature of the formulation and implementation does not accommodate numerous kernel tests. To remedy this excessive geometric constraint, an external-SVM clustering algorithm, called SVM-Relabeler, is introduced that clusters data vectors with no a priori knowledge of each vector's class.

SVM-Relabeler algorithm works by first running a Binary SVM against a data set, with each vector in the set randomly labeled, until the SVM converges. In order to obtain convergence, an acceptable number of KKT violators must be found. This is done through running the SVM on the randomly labeled data with different numbers of allowed violators until the number of violators allowed is near the lower bound of violators needed for the SVM to converge on the particular data set. Choice of an

appropriate kernel and its parameters also will affect convergence. After the initial convergence is achieved, the generalization error will be very high. The algorithm now improves this result by iteratively re-labeling only the worst misclassified vectors, which have confidence factor values beyond some threshold, followed by rerunning the SVM on the newly relabeled data set. This continues until no more progress can be made. Progress is determined by a decreasing value of generalization error, hopefully near zero.

This algorithm is not biased towards the shape of the clusters, and unlike the internal approach the formulation is not fixed to a single kernel class. Nevertheless, there are robustness and consistency issues that must be addressed in the SVM-Relabeler clustering approach. To do this, an external approach to SVM clustering is prescribed in section (3.5) that takes into account the robustness required in realistic applications.

Before we describe this algorithm we derive two of the used kernels and explain the methods of supervised and unsupervised cluster validation.

3.2 Kernel Construction Using Polarization

A set of kernels are examined that generalize the difference measure of the Gaussian Kernel. All of the 'Occam's Razor' kernels in this generalization group perform strongly on the channel current data analyzed, with some regularly outperforming the Gaussian Kernel itself. The kernels fall into two classes: regularized distance (squared) kernels; and regularized information divergence kernels. The first set of kernels strongly models data with classic, geometric, attributes or interpretation. The second set of kernels is constrained to operate on $\mathbb{R}^{+N}$, the feature space of positive, non-zero, real-valued feature vector components. The space of these kernels is often also restricted to feature vectors obeying an L_1-norm $= 1$ constraint (i.e., the feature vector is a probability vector). Without the restriction on domain, the regularized divergence kernels are proven to violate Mercer's condition (see appendix A), since not in the form of the regularized distance kernels, which are found to satisfy Mercer's condition. If restricted to the $\mathbb{R}^{+N}$ domain, however, computational tests show no violation of Mercer's condition on the DNA-hairpin data. There could, thus, be a dense Mercer-satisfiability property in terms of the training sets actually used in kernel computations by the computer. It may prove to be the case that the non-symmetric nature of most practical datasets helps to complete the Mercer-satisfiability for divergence kernels. The latest results from this analysis will be described in this section[1].

3.2.1 "Occam's Razor" Kernels

Recall that Lemma (13) extended the Definition (8) and Theorem (12) by introducing a relationship between positive-definite and negative-definite kernels. There is yet another such theorem (see semigroup book) that is useful for the remainder of this section.

Theorem 11 *A Kernel, S on χ is negative-definite if and only if $e^{-\lambda^2 d}$ is positive-definite for all $\lambda \in \mathbb{R}$.*

[1]This material was part of a poster presentation at MCBIOS 2008 in Oklahoma City

A direct consequence of Theorem (12) is that a metric space (χ, d) embeds isometrically into a Hilbert space if and only if d^2 is a negative definite kernel on χ. In the light of this interpretation Theorem (11) states that given any metric space (χ, d) one can build a postive-definite kernel of the form $e^{-\lambda^2 d}$.

In this section we introduce **Absdiff** and **Sentropic** kernels that are respectively based on regularized distance and regularized information divergence.

3.2.2 Regularized Distance Kernels

Regularized distance kernels are based on the notion of euclidian distance. An example of such a kernel is the well-known Gaussian (Radial Basis Function) kernel most commonly written as:

$$K_{RBF}(\boldsymbol{x}, \boldsymbol{y}) = exp(-\frac{||\boldsymbol{x} - \boldsymbol{y})||_2^2}{2\sigma^2}) = exp(\frac{-\sum_i (x_i - y_i)^2}{2\sigma^2})$$

For all real-valued σ. We define *polarization* of K_{RBF} by taking the component-wise variation of lnK_{RBF},

$$\frac{\partial lnK_{RBF}(\boldsymbol{x}, \boldsymbol{y})}{\partial x_i} = -\left[\frac{x_i - y_i}{\sigma^2}\right]$$

and,

$$\frac{\partial lnK_{RBF}(\boldsymbol{x}, \boldsymbol{y})}{\partial y_i} = +\left[\frac{x_i - y_i}{\sigma^2}\right]$$

Suppose that the discrimination is governed by the sign of $(x_i - y_i)$, then the reduction of $(x_i - y_i)$ by $signum(x_i - y_i)$ might produce a useful kernel $K_{Reduced}$. Meaning the reduction,

$$\frac{\partial lnK_{Reduced}(\boldsymbol{x}, \boldsymbol{y})}{\partial x_i} = -\left[\frac{signum(x_i - y_i)}{\sigma^2}\right]$$

and,

$$\frac{\partial lnK_{Reduced}(\boldsymbol{x}, \boldsymbol{y})}{\partial y_i} = +\left[\frac{signum(x_i - y_i)}{\sigma^2}\right]$$

recovers another well-known kernel, the *Laplace* kernel,

$$K_{Reduced} \equiv K_{Laplace}(\boldsymbol{x}, \boldsymbol{y}) = exp(-\frac{|\boldsymbol{x} - \boldsymbol{y}|_1}{2\sigma^2}) = exp(\frac{-\sum_i |x_i - y_i|}{2\sigma^2}).$$

For these kernels it is clear that far away points result in large polarization values and small kernel values. Similarly, for $\boldsymbol{x} \approx \boldsymbol{y}$ the polarization terms approaches zero. We can take the other exterme, namely, forece the polarization terms to approach $\pm\infty$. An example of such polarization is the *Indicator Polarization* defined as,

$$\begin{aligned}\frac{\partial lnK_{Indicator}(\boldsymbol{x},\boldsymbol{y})}{\partial x_i} &= -\frac{1}{\sigma^2}\left[\frac{signum(x_i - y_i)}{\sqrt{\sum_i |x_i - y_i|}}\right] \\ &= -\frac{\sqrt{\sum_i |x_i - y_i|}}{2\sigma^2}\end{aligned}$$

and,

$$\begin{aligned}\frac{\partial lnK_{Indicator}(\boldsymbol{x},\boldsymbol{y})}{\partial y_i} &= -\frac{1}{\sigma^2}\left[\frac{signum(x_i - y_i)}{\sqrt{\sum_i |x_i - y_i|}}\right] \\ &= +\frac{\sqrt{\sum_i |x_i - y_i|}}{2\sigma^2}.\end{aligned}$$

Note that for $\boldsymbol{x} \approx \boldsymbol{y}$ the polarization term blows up. Integrating these polarization terms in turn recovers the *Absdiff* kernel,

$$K_{Indicator} \equiv K_{Absdiff}(\boldsymbol{x},\boldsymbol{y}) = exp(-\frac{\sqrt{|\boldsymbol{x} - \boldsymbol{y}|_1}}{2\sigma^2}) \tag{3.1}$$

3.2.3 Regularized Divergence Kernels

The polarization for the Regularized Distance kernel introduced in the previous section are symmetric, such that, reversing the roll of x_i and y_i only changes the sign of the polarization term, and not the magnitude. By allowing the polarization to absorb

a magnitude change (asymmetry) we can recover a class of Kullback–Leibler type divergence kernels. For instance, the **Sentropic** kernel uses the following polarization,

$$\begin{aligned}
\frac{\partial ln K_{Sentropic}(\boldsymbol{x},\boldsymbol{y})}{\partial x_i} &= -\frac{1}{\sigma^2}\left[1-\frac{y_i}{x_i}+ln\left(\frac{x_i}{y_i}\right)\right] \\
&= -\frac{1}{\sigma^2}\frac{\partial}{\partial x_i}\left[(x_i-y_i)ln\left(\frac{x_i}{y_i}\right)\right] \\
\frac{\partial ln K_{Sentropic}(\boldsymbol{x},\boldsymbol{y})}{\partial y_i} &= -\frac{1}{\sigma^2}\left[1-\frac{x_i}{y_i}+ln\left(\frac{y_i}{x_i}\right)\right] \\
&= -\frac{1}{\sigma^2}\frac{\partial}{\partial y_i}\left[(y_i-x_i)ln\left(\frac{y_i}{x_i}\right)\right],
\end{aligned}$$

to recover,

$$K_{Sentropic}(\boldsymbol{x},\boldsymbol{y}) = exp\left(-\frac{1}{\sigma^2}\left[D(\boldsymbol{x}||\boldsymbol{y})+D(\boldsymbol{y}||\boldsymbol{x})\right]\right), \tag{3.2}$$

where $D(\boldsymbol{x}||\boldsymbol{y})+D(\boldsymbol{y}||\boldsymbol{x}) = \sum_i (x_i-y_i)ln\,(x_i/y_i)$ is the symmetric Kullback–Leibler divergence.

3.3 Supervised Cluster Validators

Externally derived class labels require external categorization that assume "correct" labels for each category. Unfortunately, unlike classification, clustering algorithms do not have access to the same level of fundamental truth. Thus, the performance of unsupervised algorithms, such as clustering, could not be measured with the same certitude as for the classification problems.

In this paper the result of the clustering is measured using the externally derived class labels for the patterns. Subsequently, we can use some of the **classification-oriented** measures to evaulate our results. These measures evaluate the extent to which a cluster contains patterns of a single class.

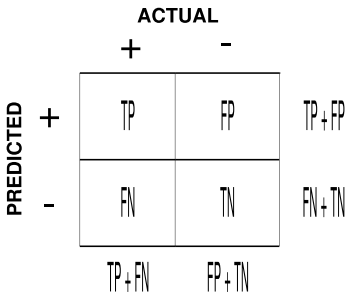


Figure 3.1: Confusion matrix for a 2-class problem

In supervised classification, Sensitivity, SN, and Sepecificity, SP, are defined using the confusion matrix in presented in figure (3.1) and are usually expressed as ([20]),

$$\begin{aligned} SN &\equiv \text{The fraction of positive patterns predicted correctly by the model} \\ &= \frac{TP}{TP + FN} \end{aligned} \tag{3.3}$$

$$SP \equiv \text{The fraction of negative patterns predicted correctly by the model}$$
$$= \frac{TN}{TN + FP}.$$

However, in the context of gene finding, since the frequency of coding nucleotides in genomic DNA sequences is much less than the frequency of coding nucleotides (see [21]), the value of TN is generally much greater than the value of FP. As a result the calculated value of SP is noninformative. Therefore, instead, in bioinformatics litrature SP is computed as

$$SP \equiv \text{The fraction of predicted patterns that turns out to be positive}$$
$$= \frac{TP}{TP + FP}. \tag{3.4}$$

Using this terminology, in classification problems where the input matrix is not sparse, and finding "positive" and "negative" patterns are equally important, the following two measures are also necessary.

$$nSN \equiv \text{The fraction of negative patterns predicted correctly by the model}$$
$$= \frac{TN}{TN + FP}. \tag{3.5}$$

$$nSP \equiv \text{The fraction of predicted patterns that turns out to be negative}$$
$$= \frac{TN}{TN + FN}. \tag{3.6}$$

In information retrieval community SN and SP equations in (3.3) and (3.4) are often referred to as, *recall* and *precision*.

3.3.1 Purity.

Let p_{ij} be the probability that an object in cluster i belongs to class j. Then *purity* for the cluster i, p_i, can be expressed as,

$$p_i = \max_j p_{ij} \tag{3.7}$$

Note that the probability that an object in cluster i belongs to class j can be written as the number of objects of class j in cluster i, n_{ij}, divided by the total number of objects in cluster i, n_i, (*i.e.*, p_{ij}=n_{ij}/ni.) Using this notation the overall validity of a cluster i using the measure p_i is the weighted sum of that measure over all clusters. Hence,

$$purity = \frac{1}{N}\sum_{i=1}^{k} n_i p_i \tag{3.8}$$

where, k is number of clusters and n is the total number of patterns. For our 2-class clustering problem equation (3.7) transforms to,

$$p_1 = max(SP, 1 - SP) \quad \text{and} \quad p_2 = max(nSP, 1 - nSP) \tag{3.9}$$

and after applying equations (3.4) and (3.3) to (3.8),

$$purity = max(\frac{TP + TN}{TP + FP + TN + FN}, 1 - \frac{TP + TN}{TP + FP + TN + FN}). \tag{3.10}$$

Although useful, *purity*, only considers either the frequency of patterns that are within a class or the frequency of patterns that are outside of a class but does not take into account the entire distribution.

3.4 Unsupervised Cluster Validator

Unlike purity, unsupervised evaluation techniques like do not depend on external class information. These measures are often optimization functions in many clustering algorithms. Sum-of-Squared-Error (SSE), measures the compactness of a single cluster and other measures evaluate the isolation of a cluster from other clusters. Unfortunately, SSE is calculated in the input space which does not necessarilly reflect the goodness of the clustering in the feature space. In this section we develope an unsupervised cluster validator that works in feature space instead.

3.4.1 Kernel-Sum-of-Squared-Error (Kernel SSE)

Positive-definite kernels are related to another kind of kernels, called *negative-definite kernels* (see definition (7)) that are in turn related to Hilbert spaces through the following theorem:

Theorem 12 *A kernel, S, on set χ is nd iff $\exists$ a Hilbert Space, H, and a map $f : \chi \to H$ such that $\forall x, y \in \chi$,*

$$S(x, y) = \| f(x) - f(y) \|^2$$

Furthermore, if $\{S = 0\}$ for all diagonal values, then $\sqrt{S}$ is a metric on χ

The following lemma is the relationship between positive and negative-definite kernels.

Lemma 13 *A kernel S is nd iff there exists a pd kernel, K, such that the following relation holds:*

$$S(x, y) = K(x, x) + K(y, y) - 2K(x, y)$$

Proof Suppose the map $f : \chi \to H$ such that $\forall x, y \in \chi$, then by theorem (12) we

can rewrite S as following:

$$\begin{aligned} S(x,y) &= \| f(x) - f(y) \|^2 \\ &= \langle f(x), f(x) \rangle + \langle f(y), f(y) \rangle - 2\langle f(x), f(y) \rangle \\ &= K(x,x) + K(y,y) - 2K(x,y) \end{aligned}$$

This is a useful since geometrically S is a measure of distance between x and y. Therefore, using lemma (13) for feature vectors $\mathbf{x}$ and $\mathbf{x}' \in \chi$ the inner cluster sum-of-squared (see definition (10)), J_i,

$$J_i = \frac{1}{2} n_i \overline{s}_i, \tag{3.11}$$

where n_i is the size of the ith cluster and for any nd kernel $S(\mathbf{x}, \mathbf{x}')$,

$$\overline{s}_i = \frac{1}{n_i^2} \sum_{\mathbf{x} \in \chi_i} \sum_{\mathbf{x}' \in \chi_i} S(\mathbf{x}, \mathbf{x}'), \tag{3.12}$$

can be written as:

$$J_i = \sum_{\mathbf{x} \in \chi_i} K(x,x) - \frac{1}{n_i} \sum_{x \in \chi_i} \sum_{x' \in \chi_i} K(x,x'). \tag{3.13}$$

Now, let, $\mathbf{K^m}$ be a Gram matrix, where $\mathbf{K}_{ij}^m = K(x_i, x_j)$ such that x_i and x_j are in the mth cluster χ_m, then for any normalized pd kernel, K, $(K(x,y) = 1 \Leftrightarrow x = y)$ and a column vector of m ones, 1,

$$J_i = n_i \left[1 - \frac{1^T K^m 1}{n_i^2} \right] \tag{3.14}$$

Similarly, for a non-normalized pd kernel, K,

$$\tilde{J}_i = trace(K^m) - \frac{1^T K^m 1}{n_i} \tag{3.15}$$

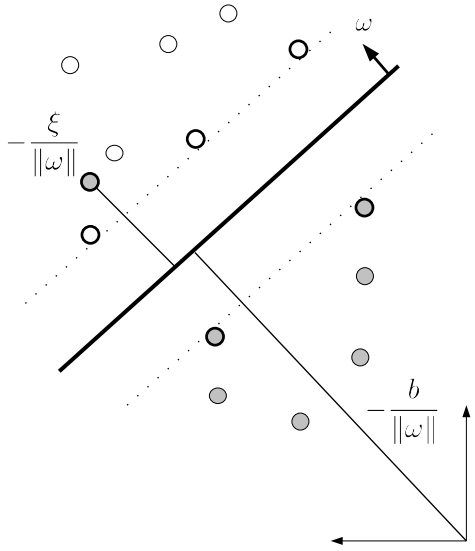


Figure 3.2: SVM on non-separable features

3.5 SVM-Relabeler Clustering Method

The SVM classification formulation is expanded to cluster a set of feature vectors with no *a priori* knowledge of the features' classification. Non-separable SVM guarantees convergence at the cost of allowing misclassification. As depicted in the figure (3.2) the extent of slack is controlled through the regularization constant, C, to penalize the slack variable, ξ. If the random mapping

$$((x_1, y_1), ..., (x_m, y_m)) \in \chi^m \times \mathbf{y}$$

is not linearly separable when ran through a binary SVM, the misclassified features are more likely to belong to the other cluster. Moreover, by relabeling those heavily misclassified features and by repeating this process we arrive at a separation between the two clusters. As formulated in the core of SVM formulation this separation has the largest margin given the regularization constant, C. The basics of this procedure is presented in Algorithm (7), where $\hat{\mathbf{y}}$ is the new cluster assignment for $\mathbf{x}$ and θ is the resulting SVM model.

It is reasonably assumed that the SVM procedure, $doSVM()$, converges regardless

Algorithm 7 SVM-Relabeler

Require: Feature vectors: $\mathbf{x}$

1: $\hat{y} \leftarrow$ Randomly chosen from $\{-1, +1\}$
2: **repeat**
3: $\quad \theta \leftarrow doSVM(\mathbf{x}, \hat{y})$
4: $\quad \hat{y} \leftarrow doRelabel(\mathbf{x}, \theta)$
5: **until** $\hat{y}$ remains constant

of the geometry of the data[2], in order to provide the *doRelabel*() procedure with the required model. After this procedure, *doRelabel*() reassignes some of the misclassified features to a the other cluster. If $D(x_i, \theta)$ is the distance between x_i feature and the trained SVM hyperplane (see footnote explaining the projection), then heavily misclassified features, $x_{j \in J}$, could be selected by comparing $D(x_j, \theta)$ to $D(x_{j'}, \theta)$ for all $j' \in J$. Algorithm (8) clarifies the basic implementation of this procedure by introducing a confidence parameter, α, that is the bottom percentile of misclassified features beyond which is relabeled.

Algorithm 8 *doRelabel*1() Procedure

Require: Input vector: $\mathbf{x}$
Cluster labeling: $\hat{y}$
SVM model: θ
Confidence Factor: α

1: Identify misclassified features:
$\hat{\mathbf{x}}^+ \leftarrow k$ misclassified features with $\hat{y} = +1$
$\hat{\mathbf{x}}^- \leftarrow l$ misclassified features with $\hat{y} = -1$
2: **for all** i^{th} component of $\hat{\mathbf{x}}^+$ **do**
3: $\quad$ **if** $D(\hat{x}_j^+, \theta) > percentile(1 - \alpha)$ **then**
4: $\quad\quad \hat{y}_i^+ \leftarrow -1$
5: $\quad$ **end if**
6: **end for**
7: **for all** i^{th} component of $\hat{\mathbf{x}}^-$ **do**
8: $\quad$ **if** $D(\hat{x}_j^-, \theta) < percentile(\alpha)$ **then**
9: $\quad\quad \hat{y}_i^- \leftarrow +1$
10: $\quad$ **end if**
11: **end for**

[2]SVMs were developed for both the separable and nonseparable data sets

3.6 Refinement Methods Using Simulated Annealing

The SVM-Relabeler algorithm does not use an objective function and the hope is that by running the algorithm in its purest form the resulting clusters are reliable solutions. However, running this algorithm in this basic fashion does not consistently provide us with a satisfying clustering solution. In fact, the solution space can be divided into three sets: successful, local-optimum, and unsuccessful (see figure (4.3)).

Trivially unsuccessful and the local optimum solutions are undesirable and should be ideally eliminated. Since, the solutions in the unsuccessful set are reasonably expected to be frequent in any experiment that calculates the the SSE of a randomly labeled data set, they can be simply eliminated by post-processing. For instance, in a similar experiment we have randomly labeled the dataset 5000 times and calculated the SSE distribution for the experiment. The resulting distribution has a good fit to Johnson's S_B distribution (see Appendix C) and is illustrated in the histogram of figure (3.3). using a fitted distribution one can calculate the p-value of a given SSE. For example, SSE threshold of 170.5 (accidentally very unlikely) can be employed to eliminate the unsuccessful set.

To substantially reduce the local optimum solutions, however, thresholding does not scale well. One solution is a to use a simple hill climbing algorithm which is to run the algorithm for a sufficiently long number of iterations to find the solution with the lowest SSE value. To do this the algorithm (7) is ran repeatedly and randomly initialized every time. A solution is accepted as the best solution so far if it has a lower SSE than the previously recorded value.

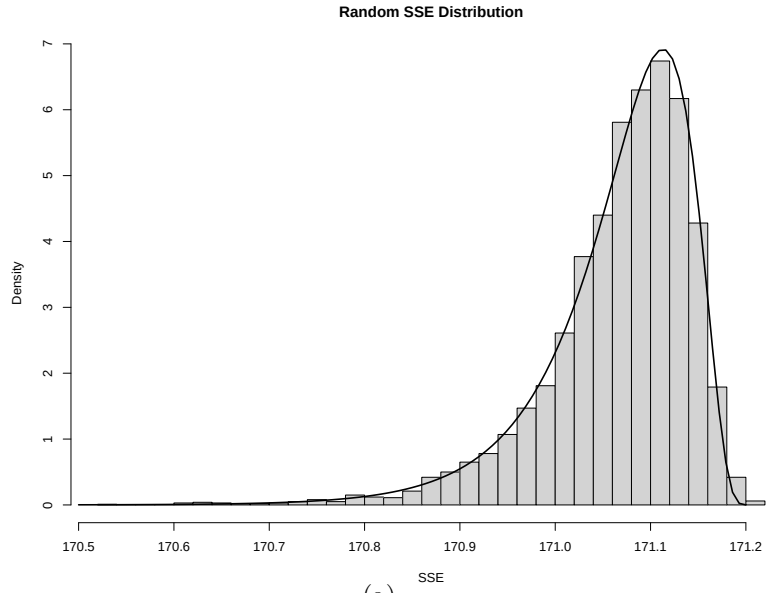
It is observed that random perturbation by flipping each label at some probability, p_{pert}, is often sufficient to switch to another subspace where a better solution could be found. (Note that $p_{pert} = 0.50$ has the effect of random reinitialization and $pert_{pert} = 1$ flips the entire labels.) The hope is that perturbation with $p_{pert} \neq 0.50$ results in a faster convergence for algorithm (7). To account for the cases where hill climbing (optimizing downward in the case of SSE) using perturbation results in a local- optimum, it is necessary to allow an occasional uphills.

Algorithm 9 SVM-Relabeler Using Simulated Annealing

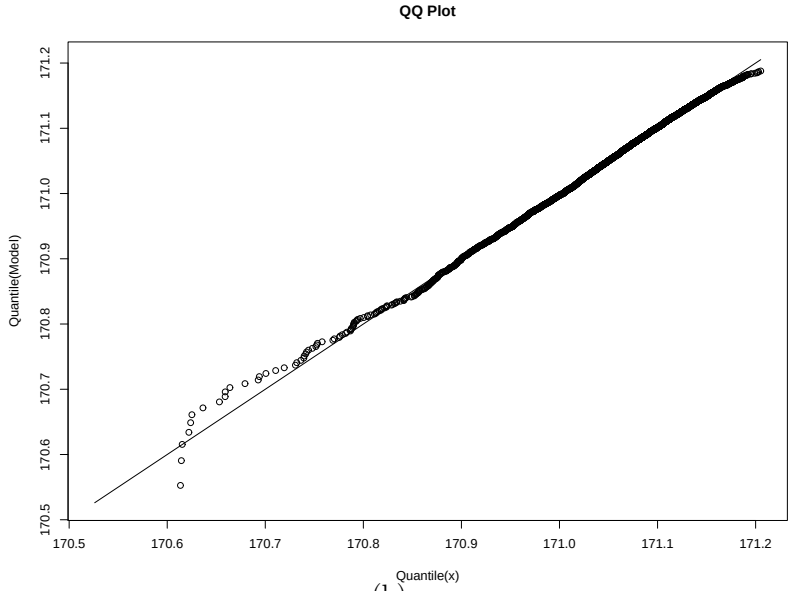
Require: Input vector: **x**
SVM-Relabeler: $doSVM()$
Initial system temperature: T
Annealing schedule: $anneal()$
Perturbation schedule: $pert()$
SSE function: $SSE()$
Random generator: $rand() \rightarrow (0, 1)$

initialize:
$y \leftarrow doSVM()$
$e \leftarrow SSE(y)$
$t \leftarrow T$
repeat
  perturb and evaluate:
  $y_{next} \leftarrow doSVM(pert(y))$
  $e_{next} \leftarrow SSE(y_{next})$
  if $(e_{next} < e) \bigvee ((e < e_{next}) \bigwedge (rand() < exp(-\frac{e_{next}-e}{t})))$ **then**
    accept the new state:
    $y \leftarrow y_{next}$
    $e \leftarrow e_{next}$
  end if
  $anneal()$
until threshold or maximum number of iteration reached

In summary, reliability can be acheived by searching through the solution space of algorithm (7). To do this efficiently, Monte Carlo Methods could be used by taking advantage of perturbation to evaluate the neighboring configuration. The procedure outlined in Algorithm (9) uses a modified version of Simulated Annealing to achieve this desired reliability.



(a)



(b)

Figure 3.3: (a) demonstrates the histogram and the fitted distribution by randomly labeling the 8GC/9GC data set using the Absdiff kernel ($\gamma = 1.8$) and computing SSE. The fitted Distribution is Johnson's S_B distribution (see Appendix) ($\gamma = -5.5405$, $\delta = 1.8197$, $\lambda = 2.7483$, $\xi = 168.46$) with the corresponding Q-Q plot in (b).

Chapter 4

SVM-Relabeler Results

SVM-Relabeler clustering algorithm is capable of clustering a wide range of data sets from simple 2-dimensional toy data (see Figure 4.1) sets to simple multi-dimensional data sets (e.g. Iris data set) to the complex Nanopore DNA hairpin data. The preliminary results were first published in [22] by this author and is now part of a more comprehensive presentation in this chapter.

4.1 Iris Data Set

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by Sir Ronald Aylmer Fisher in [23] as an example of discriminant analysis. However, it was first collected by Edgar Anderson to quantify the geographic variation of Iris flowers in the Gasp Peninsula (see [24]). The data set consists of 50 samples from each of three species of Iris flowers (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample, they are the length and the width of sepal and petal (see figure 4.2 and table 4.1). Based on the combination of the four features, Fisher developed a linear discriminant model to predict classes to which each sample belongs.

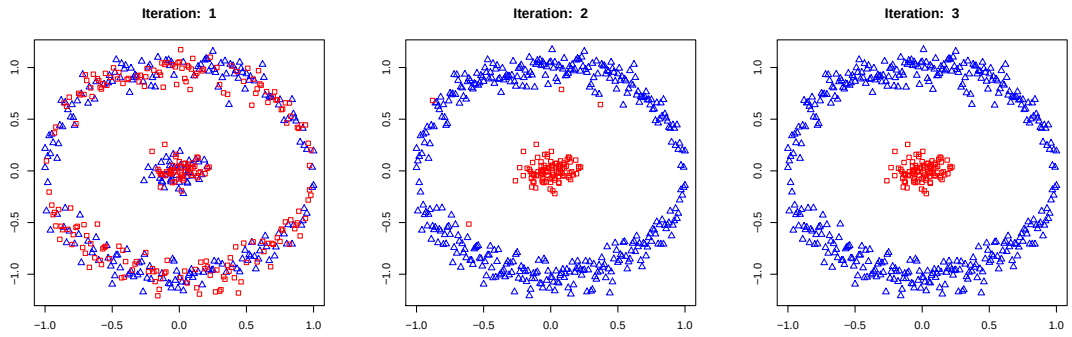


Figure 4.1: SVM-Relabeler using a polynomial kernel

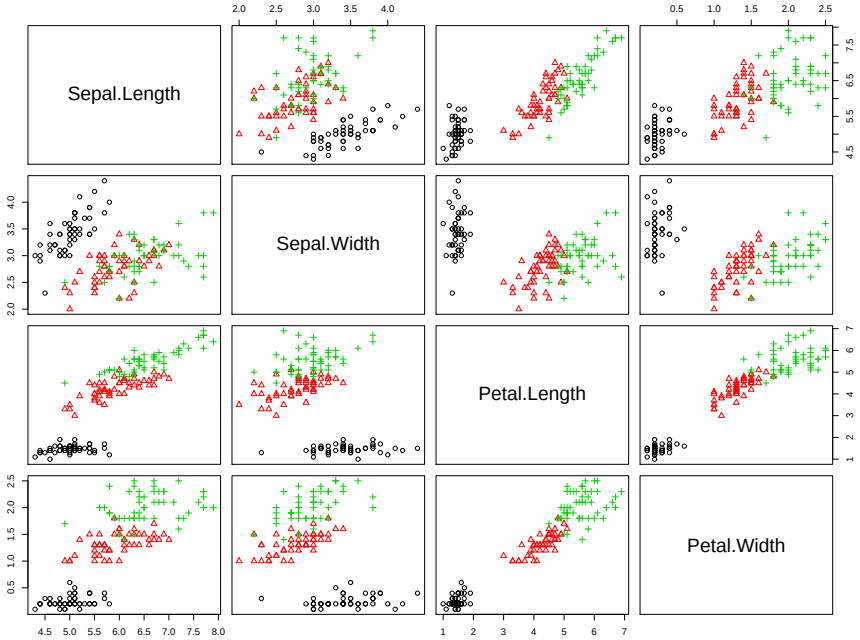


Figure 4.2: Iris data set: black circles represent setosa, red triangles represent versicolor, and green pluses represent virginica.

Table 4.1: Iris data set

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa
4.6	3.6	1.0	0.2	setosa
5.1	3.3	1.7	0.5	setosa
4.8	3.4	1.9	0.2	setosa
5.0	3.0	1.6	0.2	setosa
5.0	3.4	1.6	0.4	setosa
5.2	3.5	1.5	0.2	setosa

Table 4.1 – continued from previous page

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.2	3.4	1.4	0.2	setosa
4.7	3.2	1.6	0.2	setosa
4.8	3.1	1.6	0.2	setosa
5.4	3.4	1.5	0.4	setosa
5.2	4.1	1.5	0.1	setosa
5.5	4.2	1.4	0.2	setosa
4.9	3.1	1.5	0.2	setosa
5.0	3.2	1.2	0.2	setosa
5.5	3.5	1.3	0.2	setosa
4.9	3.6	1.4	0.1	setosa
4.4	3.0	1.3	0.2	setosa
5.1	3.4	1.5	0.2	setosa
5.0	3.5	1.3	0.3	setosa
4.5	2.3	1.3	0.3	setosa
4.4	3.2	1.3	0.2	setosa
5.0	3.5	1.6	0.6	setosa
5.1	3.8	1.9	0.4	setosa
4.8	3.0	1.4	0.3	setosa
5.1	3.8	1.6	0.2	setosa
4.6	3.2	1.4	0.2	setosa
5.3	3.7	1.5	0.2	setosa
5.0	3.3	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
5.5	2.3	4.0	1.3	versicolor
6.5	2.8	4.6	1.5	versicolor
5.7	2.8	4.5	1.3	versicolor
6.3	3.3	4.7	1.6	versicolor

Table 4.1 – continued from previous page

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
4.9	2.4	3.3	1.0	versicolor
6.6	2.9	4.6	1.3	versicolor
5.2	2.7	3.9	1.4	versicolor
5.0	2.0	3.5	1.0	versicolor
5.9	3.0	4.2	1.5	versicolor
6.0	2.2	4.0	1.0	versicolor
6.1	2.9	4.7	1.4	versicolor
5.6	2.9	3.6	1.3	versicolor
6.7	3.1	4.4	1.4	versicolor
5.6	3.0	4.5	1.5	versicolor
5.8	2.7	4.1	1.0	versicolor
6.2	2.2	4.5	1.5	versicolor
5.6	2.5	3.9	1.1	versicolor
5.9	3.2	4.8	1.8	versicolor
6.1	2.8	4.0	1.3	versicolor
6.3	2.5	4.9	1.5	versicolor
6.1	2.8	4.7	1.2	versicolor
6.4	2.9	4.3	1.3	versicolor
6.6	3.0	4.4	1.4	versicolor
6.8	2.8	4.8	1.4	versicolor
6.7	3.0	5.0	1.7	versicolor
6.0	2.9	4.5	1.5	versicolor
5.7	2.6	3.5	1.0	versicolor
5.5	2.4	3.8	1.1	versicolor
5.5	2.4	3.7	1.0	versicolor
5.8	2.7	3.9	1.2	versicolor
6.0	2.7	5.1	1.6	versicolor
5.4	3.0	4.5	1.5	versicolor
6.0	3.4	4.5	1.6	versicolor

Table 4.1 – continued from previous page

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
6.7	3.1	4.7	1.5	versicolor
6.3	2.3	4.4	1.3	versicolor
5.6	3.0	4.1	1.3	versicolor
5.5	2.5	4.0	1.3	versicolor
5.5	2.6	4.4	1.2	versicolor
6.1	3.0	4.6	1.4	versicolor
5.8	2.6	4.0	1.2	versicolor
5.0	2.3	3.3	1.0	versicolor
5.6	2.7	4.2	1.3	versicolor
5.7	3.0	4.2	1.2	versicolor
5.7	2.9	4.2	1.3	versicolor
6.2	2.9	4.3	1.3	versicolor
5.1	2.5	3.0	1.1	versicolor
5.7	2.8	4.1	1.3	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica
7.1	3.0	5.9	2.1	virginica
6.3	2.9	5.6	1.8	virginica
6.5	3.0	5.8	2.2	virginica
7.6	3.0	6.6	2.1	virginica
4.9	2.5	4.5	1.7	virginica
7.3	2.9	6.3	1.8	virginica
6.7	2.5	5.8	1.8	virginica
7.2	3.6	6.1	2.5	virginica
6.5	3.2	5.1	2.0	virginica
6.4	2.7	5.3	1.9	virginica
6.8	3.0	5.5	2.1	virginica
5.7	2.5	5.0	2.0	virginica
5.8	2.8	5.1	2.4	virginica

Table 4.1 – continued from previous page

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
6.4	3.2	5.3	2.3	virginica
6.5	3.0	5.5	1.8	virginica
7.7	3.8	6.7	2.2	virginica
7.7	2.6	6.9	2.3	virginica
6.0	2.2	5.0	1.5	virginica
6.9	3.2	5.7	2.3	virginica
5.6	2.8	4.9	2.0	virginica
7.7	2.8	6.7	2.0	virginica
6.3	2.7	4.9	1.8	virginica
6.7	3.3	5.7	2.1	virginica
7.2	3.2	6.0	1.8	virginica
6.2	2.8	4.8	1.8	virginica
6.1	3.0	4.9	1.8	virginica
6.4	2.8	5.6	2.1	virginica
7.2	3.0	5.8	1.6	virginica
7.4	2.8	6.1	1.9	virginica
7.9	3.8	6.4	2.0	virginica
6.4	2.8	5.6	2.2	virginica
6.3	2.8	5.1	1.5	virginica
6.1	2.6	5.6	1.4	virginica
7.7	3.0	6.1	2.3	virginica
6.3	3.4	5.6	2.4	virginica
6.4	3.1	5.5	1.8	virginica
6.0	3.0	4.8	1.8	virginica
6.9	3.1	5.4	2.1	virginica
6.7	3.1	5.6	2.4	virginica
6.9	3.1	5.1	2.3	virginica
5.8	2.7	5.1	1.9	virginica
6.8	3.2	5.9	2.3	virginica

Table 4.1 – continued from previous page

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
6.7	3.3	5.7	2.5	virginica
6.7	3.0	5.2	2.3	virginica
6.3	2.5	5.0	1.9	virginica
6.5	3.0	5.2	2.0	virginica
6.2	3.4	5.4	2.3	virginica
5.9	3.0	5.1	1.8	virginica

SVM-Relabeler was successfully used to separate the class of Setosa from the class of Versicolor and Verginica, using a Gaussian kernel.

4.2 DNA Hairpin Data Set

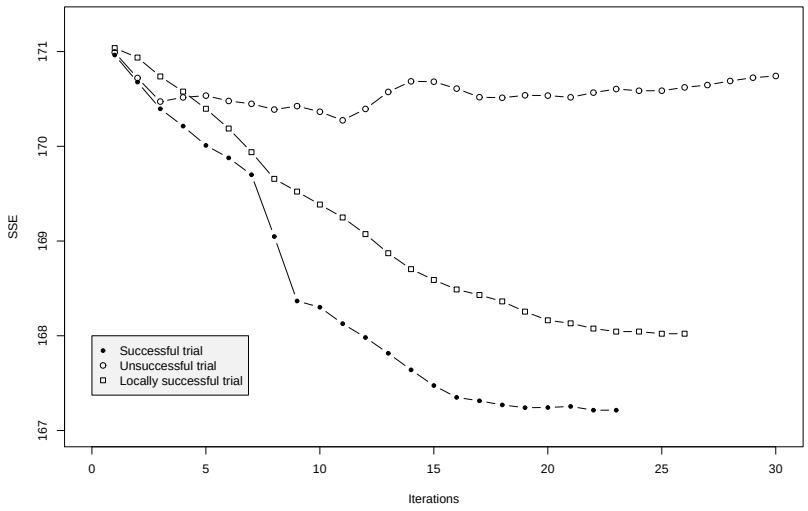
In [25, 26, 27, 28] it is shown that a nanometer-scale channel can be used to associate ionic current measurements with single-molecule channel blockades. In particular when α-hemolysin channel was used, the dimensionality allows for DNA/RNA measurements such that the ssDNA translocates while dsDNA does not. The entry aperture is 2.6 nm in diameter which is large enough to capture dsDNA. Furthermore, operation of the α-hemolysin nanopore detector demonstrates that it is possible to obtain at least Angstrom-level resolution of structural feature.

Using the observed structure (signal) a set of features representing the 8-base pair and 9-base pair were extracted (using methods of Hidden Markov Models) that in turn allowed the authors of [27] to accurately classify different DNA hairpins using Support Vector Machines.

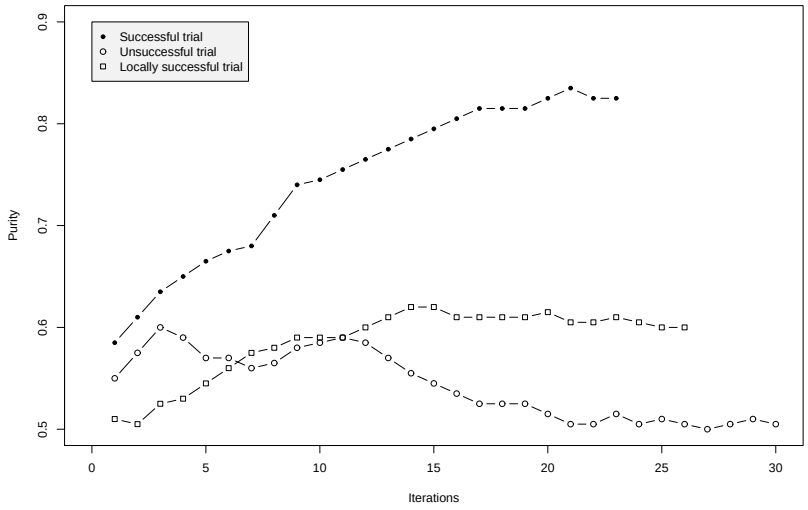
We have selected this data set to cluster a symmetric[1] sample of 200 8GC (8-base pair DNA ending in TAGC) and 9GC (9-base pair DNA ending in TAGC) feature vectors using the SVM-Relabeler algorithm. Each feature vector is 150 dimensional normalized to satisfy L1-norm = 1 constraint. For these experiments the following set of parameters where used: Absdiff kernel ($\gamma = 2.0$), $C = 1.5$, order of numerical approximation $\approx 10^{-}3$, relabeling factor (α) = 0.15.

Although convergence was always achieved, convergence to a global optimum was not guaranteed. Figure (4.3) illustrates the characteristic behavior of different possible solutions. At the end of a successful run of this algorithm it is expected that the generalization error (testing error) be very small. In this case and figure (4.5) a small value of Kernel-SSE (herein referred to as SSE) provides us with a reliable cluster validation measure.

[1]Symmetry implies that there are equal number of feature vectors in each class. However, this should not be an issue for a robust clustering algorithm. It is shown in the case of the Iris data set (section 4.1) this algorithm does not require a symmetric setting.

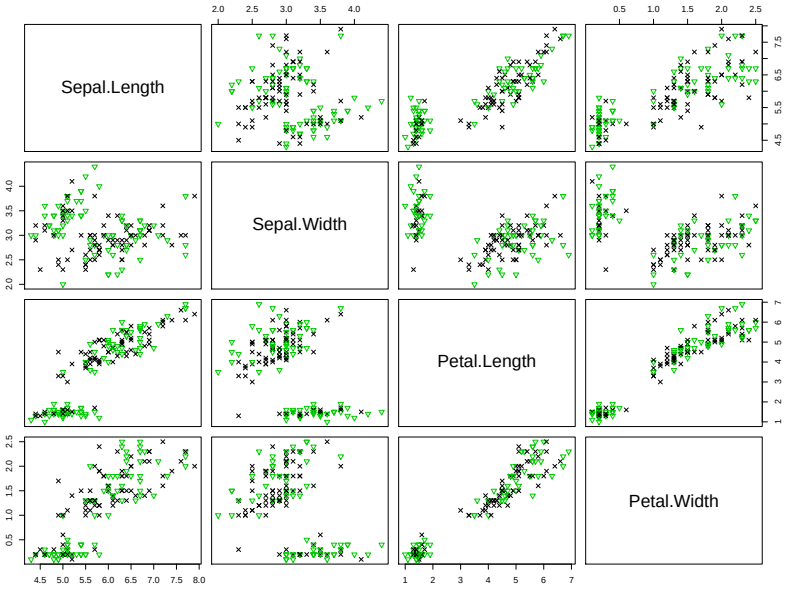


(a)

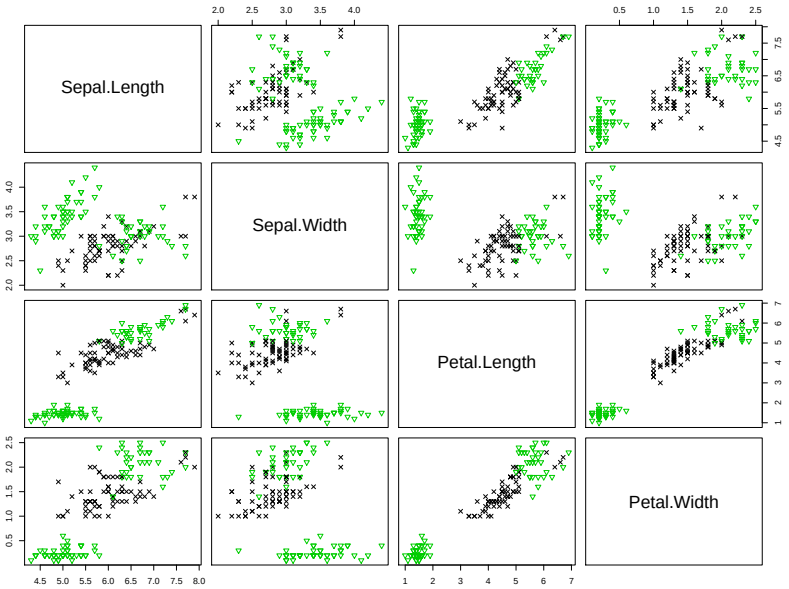


(b)

Figure 4.3: 3 different trials of SVM-Relabeler algorithm demonstrating the range of the possible solution space as measured by SSE ((a), Absdiff kernel $\gamma = 1.8$) and purity ((b), see appendix))

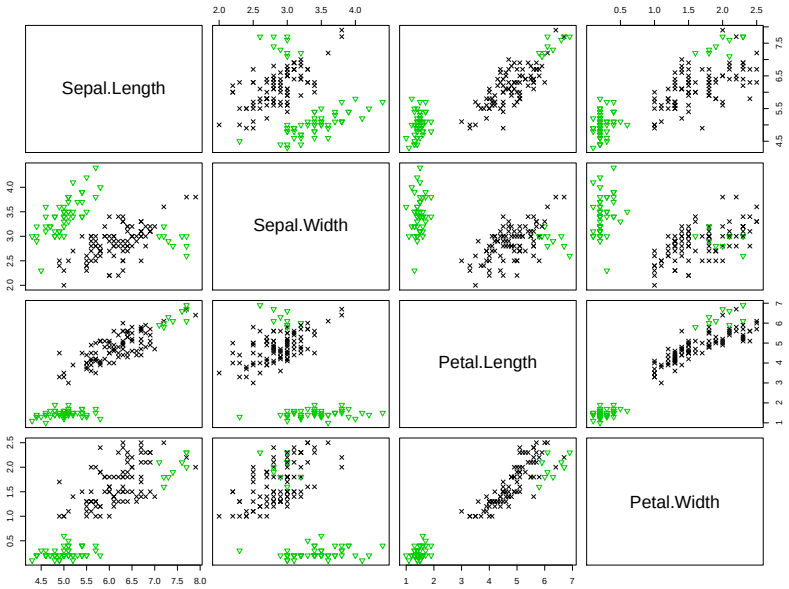


(a) Iteration 1

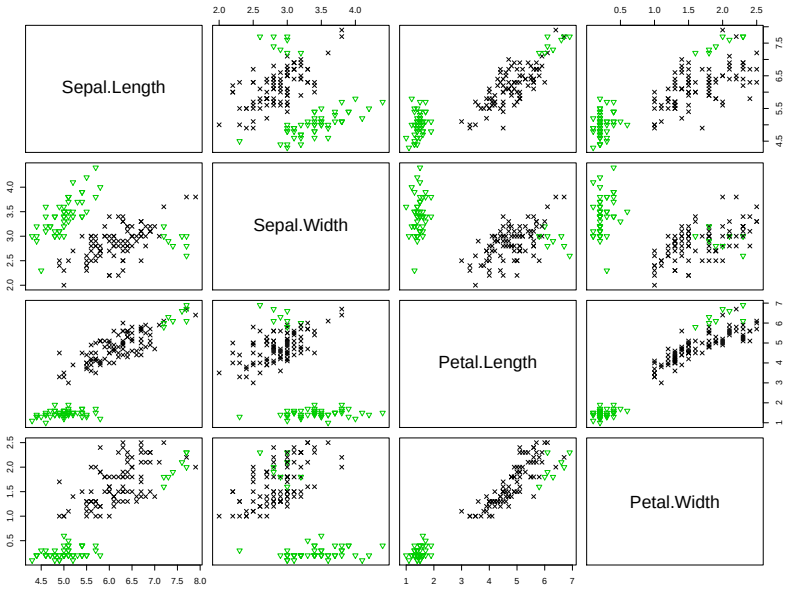


(b) Iteration 2

Figure 4.4: This figure shows the progress of the SVM-Relabeler algorithm as it clusters Setosa away from Versicolor and Verginica. Parameters: Gaussian kernel with $\gamma = 2.0$, $C = 1.5$, and $\alpha = 0.5$.

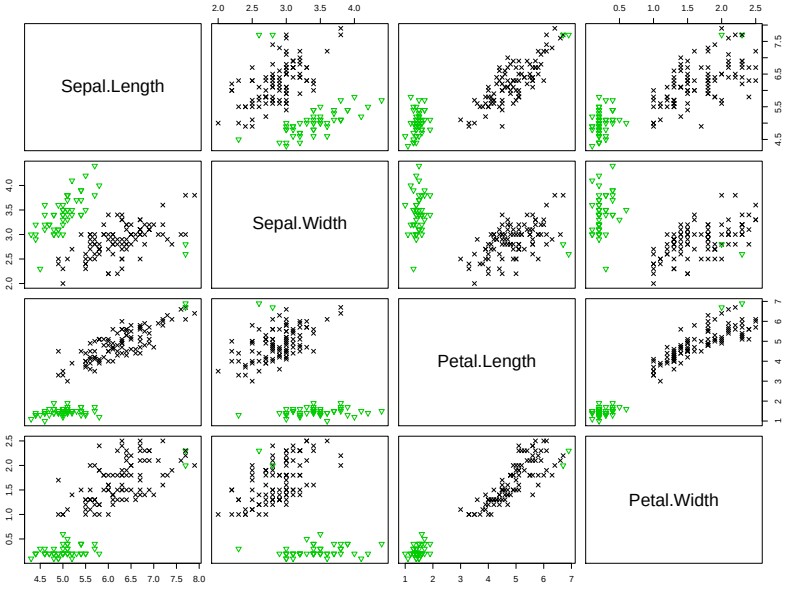


(c) Iteration 3

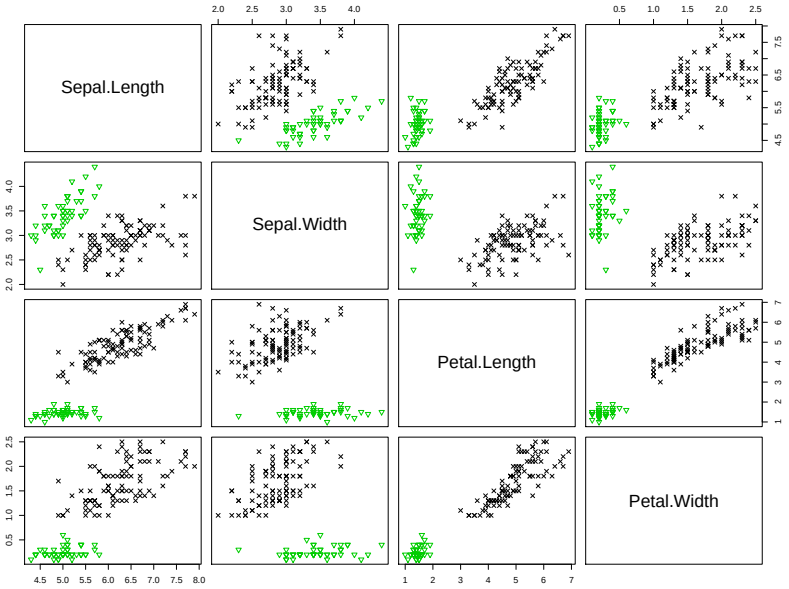


(d) Iteration 4

Figure 4.4: (continued)

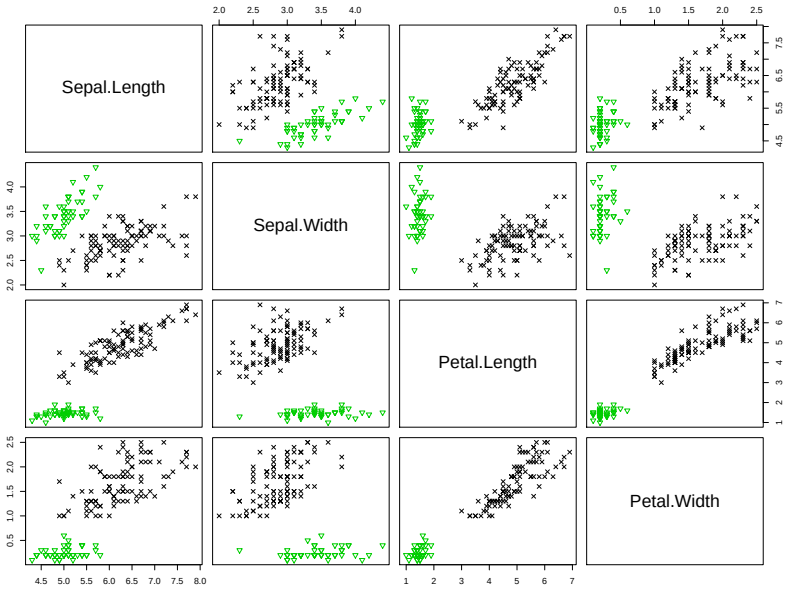


(e) Iteration 5



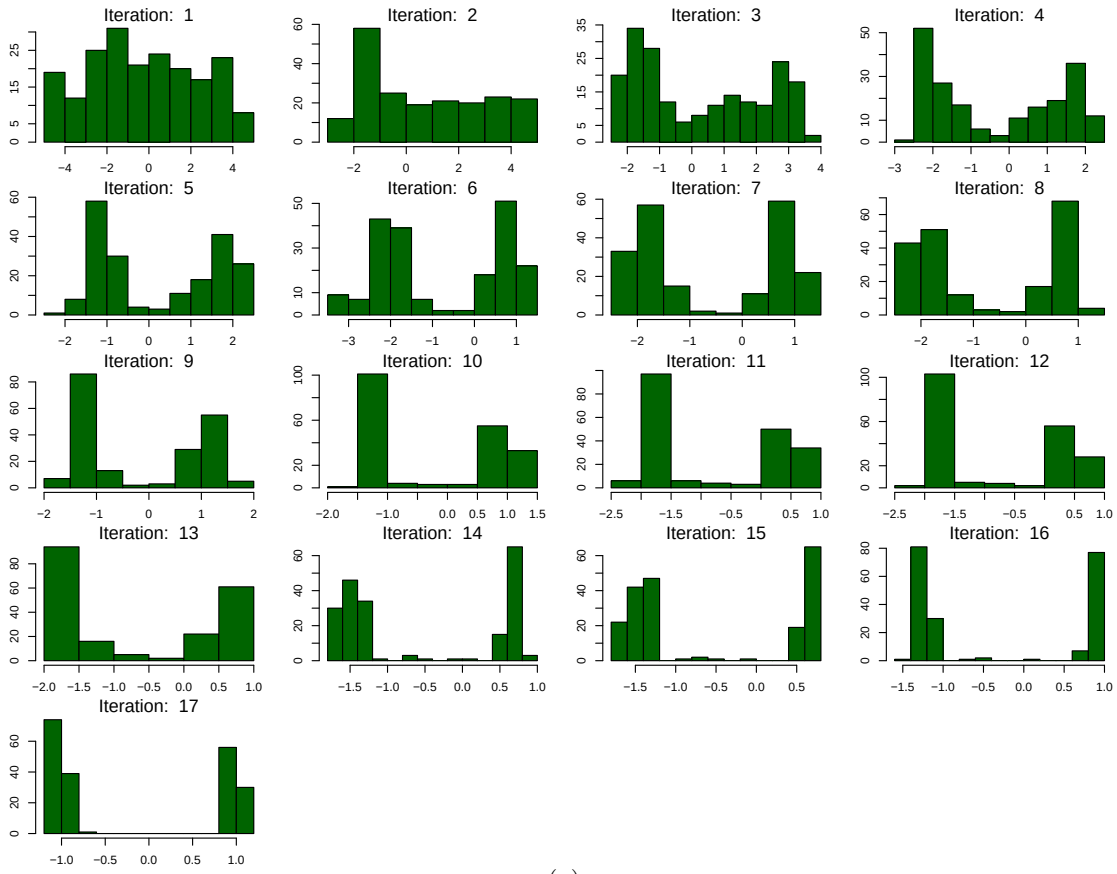
(f) Iteration 6

Figure 4.4: (continued)



(g) Iteration 7

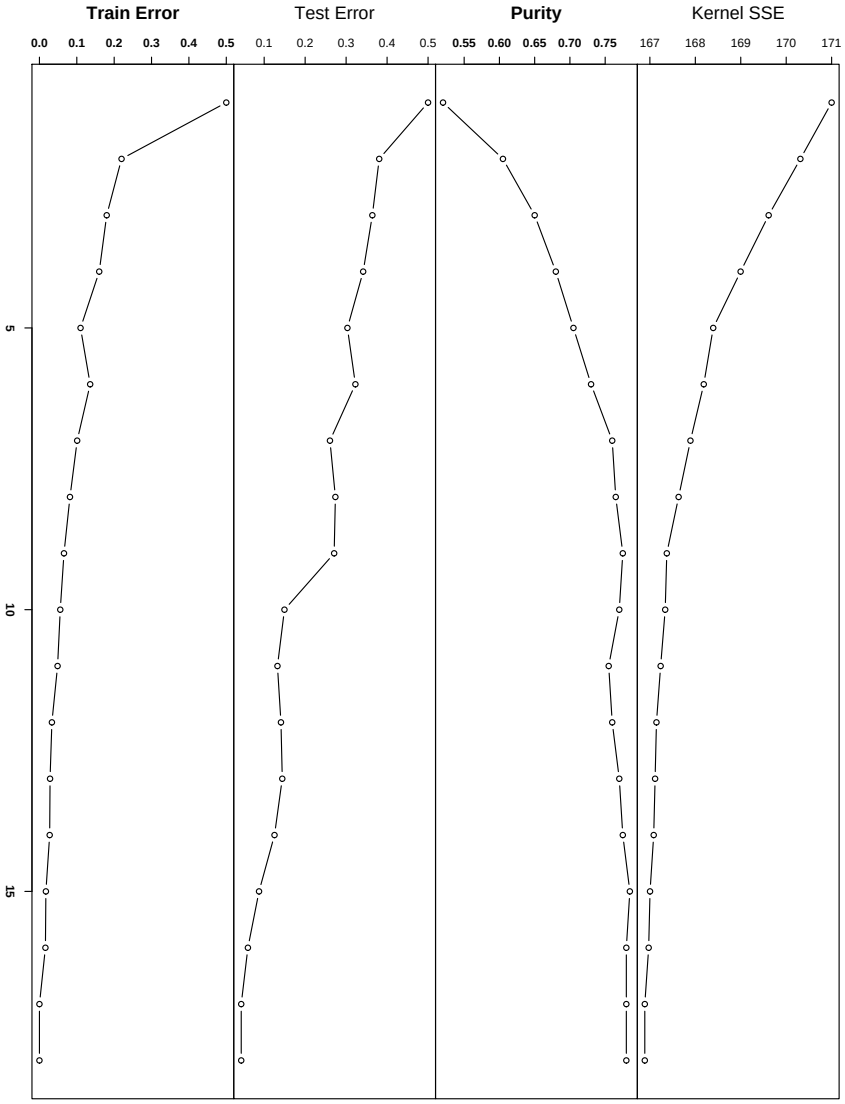
Figure 4.4: (continued)



(a)

Figure 4.5: (a) is a histogram of the projection of feature vectors on ω, a unit vector orthogonal to the decision hyperplane. Therefore, features are classified based on the sign of this projection, and it is clear from this figure that by iteration 17 the data set has been clearly separated. Figure (b) shows, iteration by iteration for the same experiment as in (a), a decrease in Kernel SSE value, test and training errors and simultanous increase in the purity.

As mentioned in section (3.6) annealing function is the property of the entire system, while perturbation function is a local property. Naturally, one would expect that the system to be more acceptable to increase in SSE in the early iterations and less so as the temperature of the system drops. However, As shown in Figure (4.6, top-left) it happens so that constant perturbation , with $p_{pert} = 0.10$ in this case, could result in a local-optimum solution that could be otherwise avoided using a perturbation function depending on the number of iterations of unchanged SSE (Figure



(b)

Figure 4.5: (continued)

(4.6, top-right)). These results were produced using an exponential cooling function, $T_{k+1} = \beta^k T_K$, with $\beta = 0.96$ and $T_0 = 10$. The initial temperature, T_0 should be large enough to be comparable with the change of SSE, ΔSSE, and therefore increase the randomness by making the Boltzman factor $e^{-\frac{\Delta SSE}{T}} \approx e^0$, while $\beta(< 1)$ should be large enough to speed up the cooling effect.

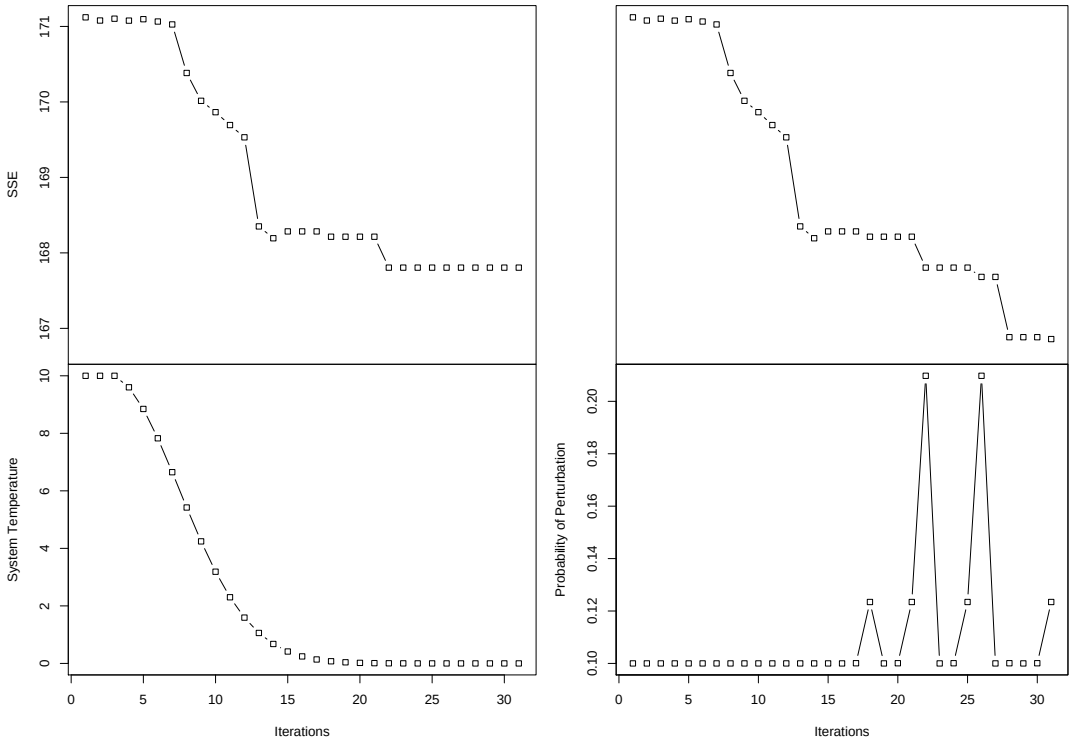


Figure 4.6: This figure compares the behavior of Algorithm 9 using constant and variable perturbation functions. Top-left demonstrates the case where constant 10% perturbation is applied at every iteration, while in addition to that, the top-right figure increases the probability of perturbation based on the number of iterations during which SSE remained constant (bottom-right). The bottom-left figure shows the annealing process.

Chapter 5

Conclusion

5.1 Discussions and Future Work

5.1.1 Regularized Divergence Kernels

Regularized distance kernels introduced in section (3.2.2) (e.g. Gaussian, Laplace and Absdiff) are provably positive-definite [see semigroup book] and therefore satisfy Mercer's condition. Regularized divergence kernels, and particularly the *Sentropic* kernel, on unrestricted domain would violate Mercer's condition, since the square root of the polarization term fails to be metric (see proof in appendix A). Extensive numerical tests, however, show that for various real random vectors C and a Sentropic kernel matrix, K, built from DNA hairpin data, $C^T K C$ is consistently positive.

To ensure positive definitivity, however, one may employ the technique of **Iterated Kernels** ([29]). We define the iteration using,

$$\begin{aligned} K^{(p)}(\boldsymbol{x},\boldsymbol{y}) &= \int K^{(n-1)}(\boldsymbol{x},\boldsymbol{z})K(\boldsymbol{z},\boldsymbol{y})d\boldsymbol{z} \\ &= \int \dots \int K(\boldsymbol{x},\boldsymbol{z}_1)K(\boldsymbol{z}_1,\boldsymbol{z}_2)\dots K(\boldsymbol{z}_{n-1},\boldsymbol{y})d\boldsymbol{z}_1\dots d\boldsymbol{z}_{n-1} \end{aligned} \quad (5.1)$$

It is shown in [17] that $K^{(2)}$ satisfies Mercer's condition and is positive definite even when K does not. However we generalize and show that $K^{(n=2p)}$, $\forall p \in \mathbb{N}^+$, is positive

definite even when K is not. This follows from Mercer's theorem [see Mercer],

$$\begin{aligned}
&\int K^{(n)}(\boldsymbol{x},\boldsymbol{y})\Phi(\boldsymbol{x})\Phi(\boldsymbol{y})d\boldsymbol{x}d\boldsymbol{y} \\
&= \int \left(\int \dots \int K(\boldsymbol{x},\boldsymbol{z}_1)K(\boldsymbol{z}_1,\boldsymbol{z}_2)\dots K(\boldsymbol{z}_{n-1},\boldsymbol{y})d\boldsymbol{z}_1\dots d\boldsymbol{z}_{n-1} \right) \Phi(\boldsymbol{x})\Phi(\boldsymbol{y})d\boldsymbol{x}d\boldsymbol{y} \\
&= \left(\int\int K(\boldsymbol{x},\boldsymbol{z})\Phi(\boldsymbol{x})d\boldsymbol{x}d\boldsymbol{z} \right)^2 \left(\int\int K(\boldsymbol{z}_i,\boldsymbol{z}_j)d\boldsymbol{z}_i d\boldsymbol{z}_j \right)^{2(p-1)} \geq 0
\end{aligned}$$

5.1.2 Cluster Validators

The fitness of the cluster identified by the SVM-Relabeler algorithm is modeled using a supervised measure, *purity* (see section 3.3) and an unsupervised measure, *Kernel SSE*, which is the cohesion of the clusters. In other words, we assume that the fitness of a cluster can be tracked using the compactness of that cluster as the algorithm progresses. It is necessary to note that i) the notion of compactness as a way to evaluate clusters favours spherical clusters over clusters spread over a linear region, and ii) the known classes may not properly correspond to the geometric structure of the clusters. However, this limitation does not normally affect our methodology, since this limitation is imposed over the feature space, and not the input space. That is, with a choice of proper kernels and parameters, feature space is more likely to have higher variation in the lifted dimensions.

If such a problem is known to exist, however, one can use pre-processing (see [30]) to scale and transform the data set. The simplest method of transformation is PCA and KPCA analysis (sections 2.4.1 and 2.4.2) to reduce the dimensionality by constructing new dimensions that carry the most variation. One of the differences between PCA and KPCA is that KPCA can extract a set of principal components that may be larger than input dimensionality. For an m dimensional feature set, PCA can find maximum of m nonzero covariance eigenvalues. However, since KPCA performs eigenvalue decomposition on an $n \times n$ kernel matrix K (for n is the size of feature set), there could be n non-zero eigenvalues. It is interesting to note that if K is calculated using dot product kernel, there can only be m non-zero eigenvalues that are identical to those of the covariance matrix. Hence, KPCA reduces to PCA when

dot product kernel is used.

Another method for scaling in feature space was proposed in [17], to replace the kernel function $K(\boldsymbol{x}, \boldsymbol{y})$ with $K(\boldsymbol{x}, D\boldsymbol{y})$, where D is a diagonal matrix with nonnegative diagonal elements $d_1, \ldots, d_N$. The newly created kernel effectively scales direction i of input space by $\sqrt{d_i}$.

Scaling and transformation requires knowledge about the geometry of the problem at hand. This information is often unavailable to data analysts. SVM-Relabeler can be improved to take advantage of the discriminant function provided by the SVM solution. In figure (4.5) we have shown the histogram of the decision space and demonstrated that the method is effectively separating the clusters. Let m_1, m_2, s_1 and s_2 be the means and standard deviation of the decision space projection shown in figure (4.5). The function,

$$J(\omega) = \frac{|m_1 - m_2|}{s_1^2 + s_2^2}$$

for ω is the derived weight factor, can be used to validate the clusters. Generally, larger $J(\omega)$ corresponds to better separation, while taking into account the compactness of the clusters. Note that, $J(\omega)$ is used as the objective function of Fisher's discriminants (see [30]).

5.1.3 Tuning of SVM-Relabeler

There is a plethora of literature surrounding the problem of Kernel selection (see for example, [31, 32, 33, 34]). The problem remains to be highly data set dependant and so far accurate methods for choosing kernels are not feasible. This problem is partially addressed in [34] by an empirical observation for the popular Gaussian kernel, where the optimal values of the width hyper-parameter σ are shown to lie in between the 0.1 and 0.9 quantile of the $||\boldsymbol{x} - \boldsymbol{y}||^2$ statistics.

The SVM-Relabeler results produced in this work assume the value of 1.5 for the SVM regularization constant, C. Larger values of C tend to reduce the accuracy by sacrificing the generalization error for higher training error – a case of over-fitting. This is because C penalizes misclassifiction required by SVM-Relabeler

algorithm. Similarly, smaller values of C destabalize SVM classifiers by allowing very large margines that are highly under-fitted.

The choices of the SVM regularization constant, C, has a profound distinction when used in the context of SVM-Relabeler. The choice of C in the supervised SVM controls the trade off between the training error and the generalisation error, and therefore, it effectively controls the soft margin. Since this margin is not known to us when clustering, the value of C has to be chosen emperically. It may prove beneficial to consider ν-svm ([35] and [36]), a re-formulation of SVMs to replace the soft margin C, which can take any positive value, with another one, ν, that lies in the range of zero and one. The significance of the parameter ν is that it provides an upper bound on the fraction of margin errors and the lower bound on the fraction of support vectors.

The relabeling parameter, α, which controls the amount of label flipping performed by the algorithm is observed to be stable given a good choice of kernel and parameters. That is, when everything else is tuned properly, small changes in α do not alter the accuracy significantly. It was observed that, although, in general larger α provides faster convergence, smaller α does not necessarily provide better accuracy. The optimal choice of α remains to be subject of future research.

5.2 Concluding Remarks

In this work, a new method of clustering is introduced by providing an algorithm based on Support Vector Machines that does not explicitly depend on an objective function. The solution arrived to by the heuristics of this algorithm can be later analysed using a criterion function, but it remains external to the algorithm. The hope is that this work provides a fresh and new prespective on unsupervised learning processes that are often labeled by practitioners, and theoreticians alike, as subjective.

Bibliography

[1] J. Kleinberg. An impossibility theorem for clustering. *Proc. of the 16th conference on Neural Information Processing Systems*, (12), 2002.

[2] M.Bern and D.Eppstein. *Approximation algorithms for geometric problems*, pages 296–345. Approximation algorithms for NP-hard problems. PWS Publishing Co, 1996.

[3] C.J.C.Burgest. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):121–167, 1998.

[4] V.N.Vapik. *The Nature of Statistical Learning Theory.* New York: Springer-Verlag, 1995.

[5] T.Poggio and F.Girosi. Regularization algorithms for learning that there are equivalent to multilayer networks. *Science*, 247:978–982, 1990.

[6] V.N.Vapik. *Statistical Learning Theory.* New York: Wiley, 1998.

[7] R.C.Williamson, A.J.Smola, and B.Scholkopf. Regularization algorithms for learning that there are equivalent to multilayer networks. *Science*, 247:978–982, 1990.

[8] S.Geman, E.Bienenstock, and R.Doursat. Neural network and bias/variance dilemma. *Neural Computation*, 4(2):1–58, 1992.

[9] C.M.Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[10] Y.Freund and R.E.Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[11] R.Fisher. On the mathematical foundations of theoretical statistics. *Phil. Trans. R. Soc. Lond.*, (222):309–368, 1922.

[12] D.P.Bertsekas. *Nonlinear Programming*. Belmont, MA: Athena Scientific, 1995.

[13] J.Platt. *Fast Training of Support Vector Machines Using Sequential Minimal Optimization*, pages 185–208. Advances in Kernel Methods – Support Vector Learning. Cambrige, MA: MIT Press, 1999.

[14] V.N.Vapik. *Estimation of Dependences Based on Empirical Data*. Berlin: Springer-Verlag, 1982.

[15] C.Saunders, M.O.Stitson, J.Weston, L.Bottou, B.Scholkopf, and A.Smola. Support vector machine reference manual. Technical Report CSD-TR-98-03, Royal Holloway Univ. Tech Rep., 1998.

[16] S.S.Keerthi, S.K.Shevade, C.Bhattacharyya, and K.R.K.Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(1):637–649, 2001.

[17] B.Scholkopf, E.Smola, L. Bottou, C.Burges, H.Bultho, K.Gegenfurtner, and P.Ha Ner. Nonlinear component analysis as a kernel eigenvalue problem. *Journal of Neural Computation*, 10(10):1299–1319, 1998.

[18] A.Ben-Hur, D.Horn, H.T.Siegelmann, and V.N.Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

[19] S.Winters-Hilt, A.Yelundur, C.McChesney, and M.Landry. Support vector machine implementations for classification and clustering. *Proceeding, BMC Bioinformatics*, (Supp2), 2006.

[20] P.Tan, M.Steinbach, and V.Kumar. *Introduction to data mining*. Pearson Education, Inc., 2006.

[21] M.Burset and R.Guigo. Evaluation of gene structure prediction programs. *Genomics*, 3(34):353–367, 1996.

[22] S.Winters-Hilt and S.Merat. Svm clustering. *Proceeding, BMC Bioinformatics*, 8(S18):Supp7, 2007.

[23] R.A.Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[24] E.Anderson. The irises of the gasp peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.

[25] W.Vercoutere, S.Winters-Hilt, H.Olsen, D.W.Deamer, D.Haussler, and M.Akeson. Rapid discrimination amoung individual dna hairpin molecules at single-nucleotide resolution using an ion channel. *Nat. Biotechnol.*, 19:248–252, 2001.

[26] W.Vercoutere, S.Winters-Hilt, D.W.Deamer V.S.Deguzman, J.T.Rodgers S.Ridino, A.Marziali H.Olsen, and M.Akeson. Discrimination among individial watson-crick base-pair at the termini of single dna hairpin molecules. *Nuccleic Acids. Res.*, 31:1311–1318, 2003.

[27] S.Winters-Hilt, W.Vercoutere, V.S.Deguzman, D.W.Deamer, M.Akeson, and D.Haussler. Highly accurate classification of watson-crick base-pair on termini of single dna molecules. *Biophysics Journal*, 84:1–10, 2003.

[28] S.Winters-Hilt and M.Akeson. Nanopore cheminformatics. *DNA Cell Biology*, 23:675–683, October 2004.

[29] R.Courant and D.Hilbert. *Methods of Mathematical Physics*, volume 1. N.Y.: Interscience Publications, Inc., June 1970.

[30] R.O.Duda, P.E.Hart, and D.G.Stork. *Pattern Classification*. New York: John Wiley Sons, 2001.

[31] S.J.Kim, A.Magnani, and S.Boyd. Optimal kernel selection in kernel fisher discriminant analysis. *ACM International Conference Proceeding Series*, 148:465–472, 2006.

[32] R.Debnath and H.Takahashi. Kernel selection for the support vector machine. *IEICE TRANSACTIONS on Information and Systems*, E87-D(12):2903–2904, 2004.

[33] A.Argyriou, R.Hauser, C.A.Micchelli, and M.Pontil. A dc-programming algorithm for kernel selection. *ACM International Conference Proceeding Series*, 148:41–48, 2006.

[34] B.Caputo, K.Sim, F.Furesjo, and A.Smola. Appearance-based object recognition using svms: which kernel should i use? *Whistler: Proc of NIPS workshop on Statitsical methods for computational experiments in visual processing and computer vision*, 2002.

[35] B.Scholkopf, A.Smola, R.Williamson, and P.Bartlett. New support vector algorithms. Technical Report nc2-tr-1998-031, GMD First and Australian National University, 1998.

[36] D.J.Crisp and C.J.C.Burges. A geometric interpretation of nu-svm classifiers. *NIPS*, 12:244–250, 2000.

Appendix A

Proof of Non-Positive-Definitivity of the Sentropic Kernel

Definition 14 *The symmetric Kullback–Leibler divergence, $S_{sentropic}$, of two feature vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined: by:*

$$S_{sentropic}(\boldsymbol{x}, \boldsymbol{y}) = D(\boldsymbol{x}||\boldsymbol{y}) + D(\boldsymbol{x}||\boldsymbol{y}) = \sum_i (x_i - y_i) \ln (x_i/y_i)$$

Theorem 15 *The square root of the symmetric Kullback–Leibler divergence, is not a metric.*

Proof An example can be given to counter the claim that $\sqrt{S_{sentropic}}$ satisfies the triangle inequality and hence is not a metric. Suppose, three feature vectors defined below,

$$\begin{aligned} \boldsymbol{x} =& (1-\delta, \delta)^T \\ \boldsymbol{y} =& (1-2\delta, 2\delta)^T \\ \boldsymbol{z} =& (1-3\delta, 3\delta)^T \end{aligned} \tag{A.1}$$

We rewrite $S_{sentropic}(\boldsymbol{x}, \boldsymbol{z})$ in terms of $S_{sentropic}(\boldsymbol{x}, \boldsymbol{y})$ and $S_{sentropic}(\boldsymbol{y}, \boldsymbol{z})$. Note,

$S_{sentropic}(\boldsymbol{x}, \boldsymbol{y})$ and $S_{sentropic}(\boldsymbol{y}, \boldsymbol{z})$ are simplified as following,

$$\begin{aligned} S_{sentropic}(\boldsymbol{x}, \boldsymbol{y}) &= (x_1 - y_1) \ln \frac{x_1}{y_1} + (x_2 - y_2) \ln \frac{x_2}{y_2} \\ &= \delta \ln \frac{1-\delta}{1-2\delta} - \delta \ln \frac{1}{2} \\ &= \delta \ln \frac{2(1-\delta)}{1-2\delta} \qquad \text{(A.2)} \\ S_{sentropic}(\boldsymbol{y}, \boldsymbol{z}) &= \delta \ln \frac{1-2\delta}{1-3\delta} - \delta \ln \frac{2}{3} \\ &= \delta \ln \frac{3(1-2\delta)}{2(1-3\delta)} \qquad \text{(A.3)} \end{aligned}$$

Similarly, $S_{sentropic}(\boldsymbol{x}, \boldsymbol{z})$ can be written as,

$$\begin{aligned} S_{sentropic}(\boldsymbol{x}, \boldsymbol{z}) &= 2\delta \ln \frac{1-\delta}{1-3\delta} - 2\delta \ln \frac{1}{3} \\ &= 2\delta \ln \frac{3(1-\delta)}{1-3\delta} \\ &= 2\delta \ln \left[\frac{3(1-2\delta)}{2(1-3\delta)} \cdot \frac{2(1-\delta)}{1-2\delta} \right] \\ &= 2 \left[\delta \ln \frac{3(1-2\delta)}{2(1-3\delta)} \cdot \delta \ln \frac{2(1-\delta)}{1-2\delta} \right] \\ &= 2 \left[S_{sentropic}(\boldsymbol{y}, \boldsymbol{z}) + S_{sentropic}(\boldsymbol{x}, \boldsymbol{y}) \right] \qquad \text{(A.4)} \end{aligned}$$

But, $\sqrt{S_{sentropic}(\cdot, \cdot)}$ is strictly concave and for a strictly concave function, f, we have,

$$f(\frac{x+y}{2}) > \frac{f(x) + f(y)}{2}. \qquad \text{(A.5)}$$

Using this property and equation (A.4) we get,

$$\begin{aligned} \sqrt{S_{sentropic}(\boldsymbol{x}, \boldsymbol{z})} &= 2\sqrt{\frac{S_{sentropic}(\boldsymbol{x}, \boldsymbol{y}) + S_{sentropic}(\boldsymbol{y}, \boldsymbol{z})}{2}} \qquad \text{(A.6)} \\ &> \sqrt{\frac{S_{sentropic}(\boldsymbol{x}, \boldsymbol{y})}{2}} + \sqrt{\frac{S_{sentropic}(\boldsymbol{y}, \boldsymbol{z})}{2}} \qquad \text{(A.7)} \end{aligned}$$

showing that $\sqrt{S_{sentropic}}$ fails to satisfy the triangle inequality and hence is not a metric. ∎

Finally we can show that the sentropic kernel (equation 3.2) is not a positive definite kernel.

Theorem 16 *Sentropic kernel is not positive definite.*

Proof Using theorems (12) and (15) it is clear that $S_{sentropic}$ is not a negative definite kernel. Furthermore, using theorem (11) it is proven that if $S_{sentropic}$ is not a negative definite kernel, $K_{sentropic}$ in equation (3.2), is not a positive definite kernel. ∎

Appendix B

kernlib Framework

kernlib is a Kernel based machine learning framework developed by the author using Java with hopes of it being an opensource contribution in the near future. This package facilitates implementation of kernel-based supervised and unsupervised learning algorithms. Currently, an SMO based SVM, Kernel k-means, KPCA and various tools to report, log, compare and varify results are implemented.

B.1 Configuration

Java property files are used by the user to provide input for different algorithms. In the property file illusterated in figure (B.1) the constants "SMO", "ABSDIFF", and etc., are various java factory classes defined separately in the "mapping" property file shown in figure (B.1). These factory classes are in charge of parsing the property file according to respective algorithm parametrization.

B.2 SVM engine

In this illusteration property file in figure (B.1) is passed as a command line argument to an application wanting to use the SVM engine. The SVM engine can be initialized as demonstrated in figure (B.2). Note that the training data format and SVM parameter inputs are parsed using the corresponding factory objects.

```
# SVM engine configuration
engine = SMO
cval = 1.5
epsilon = 0.001
tolerance = 0.001
maxiter = 1000
random.seed = 1213654164793

# Data configuration
data.test.type = SINGLE_DNA_TRAIN
data.test.file = /home/sammerat/workspace/SWSVM/data/9bphp/8GC9CG
    .train
data.test.type = SINGLE_DNA_TEST
data.test.file = /home/sammerat/workspace/SWSVM/data/9bphp/8GC9CG
    .test

# Kernel configuration
class = ABSDIFF
sigma = 2

# SVM-Relabeler parameters
relabeler.proc = SYM_PERC_REL_PROC
relabeler.maxiter = 30
relabeler.relfactor=0.15
relabeler.sa.temp.proc = EXP_COOL
relabeler.sa.temp.init = 6.0
relabeler.sa.maxiter = 10.0
```

Figure B.1: Example property file to configure data, train and test using an SMO SVM engine, and cluster using the SVM-Relabeler engine

```
SMO = edu.uno.cs.bioinformatics.svm.engine.SMOParameters
ABSDIFF = edu.uno.cs.bioinformatics.kernel.AbsdiffKernel
GAUSSIAN = edu.uno.cs.bioinformatics.kernel.GaussianKernel
SINGLE_DNA_TRAIN = edu.uno.cs.bioinformatics.data.
    SingleFileDNATestingDataFactory
SYM_PERC_REL_PROC = edu.uno.cs.bioinformatics.svm.cluster.
    relabeler.SymmetricPercentileProcedureFactory
EXP_COOL = edu.uno.cs.bioinformatics.svm.cluster.relabeler.sa.
    ExponentialCoolingFactory
```

Figure B.2: Example java class mapping file.

```
PropertiesConfiguration propConf = null;

try {
        propConf = new PropertiesConfiguration(args[0]);
}catch (ConfigurationException ce) {
        logger.error(args[0] + " property file not found.",ce);
}
TrainingData trainingData = null;
SVMParameters svmParams = null;
try  {
        trainingData = TrainingDataFactory.
                getTrainTrainingDataFactory(propConf).
                    getTrainingData();
        svmParams = SVMParametersBuilder.
                            getSVMParameters(propConf);
}catch(PropertyException pce) {
        logger.error(pce.getMessage(),pce);
        return;
}
```

Figure B.3: Simple initialization of the SVM engine.

Learning and prediction using SVM is then demonstrated in figures (B.2) and (B.2).

```
SVMLearner svmLearner = svmParams.getSVMLearner();
SVMModel svmModel = svmLearner.learnSVM(trainingData, svmParams);
```

Figure B.4: Learning using the SVM engine.

```
SVMPredictionData pred = new SVMPredictor(svmModel,testData).
    predict();
```

Figure B.5: Prediction using the SVM engine.

B.3 SVM-Relabeler

I figure B.3 an SVM-Relabeler is constructed by first instantiating the "RelabelerIter" class, which encapsulates a run of the SVM-Relabeler algorithm. Further runs, until the point of termination, is performed and in each step various information regarding the convergence of the algorithm (such as cross-validation error, SSE and training error) is obtained.

```
RelabelerIter relIter = new RelabelerIter(data, relpar);
for (int i = 0; relIter.hasNext() && i < relPar.getMaxIter(); i
   ++) {
        relIter.iterate();
        /* get CV error*/
        /* get training error */
        /* ... */
        /* get purity */
        SupClustVal scv = new SupClustVal(knownLabel, relIter.
           getClusterData().getLabels());
        /* get Kernel SSE */
        DoubleMatrix1D sse = TrainingDataUtils.sse(relIter.
           getRelParams().getSVMParams().getKernelf(), relIter.
           getClusterData());
        sseList.add(sse.zSum());
}
```

Figure B.6: Example SVM-Relabeler usage.

Appendix C

Johnson's S_B Distribution

Johnson's S_B distribution is the bounded form of 4-parameter lognormal distribution developed by Johnson in (see johnson). In this distribution the random variable, x, is transformed using:

$$z = \frac{x - \xi}{\lambda} \tag{C.1}$$

where ξ and $\xi + \lambda$ are the minimum and maximum values for x. Furthermore, a unit normal variable y is defined as:

$$y = \gamma + \delta ln\left(\frac{z}{1-z}\right) = \gamma + \delta ln\left(\frac{x-\xi}{\xi+\lambda-x}\right) \tag{C.2}$$

Note that the mean and variance of $ln(z/(1-z))$ are related by:

$$\mu = -\frac{\gamma}{\delta} \text{ and } \sigma^2 = \frac{1}{\delta^2}. \tag{C.3}$$

Finally, with the above substitutions into a normal distribution, the S_B distribution is written as:

$$f(x;\gamma,\delta,\lambda,\xi) = \frac{\delta}{\lambda\sqrt{2\pi}z(1-z)} exp\left(-\frac{1}{2}\left[\gamma + \delta ln\left(\frac{z}{1-z}\right)\right]^2\right) \tag{C.4}$$

Vita

Sepehr Merat was born in Tehran, Iran on August 17th, 1982. In 1997 he immigrated to Vancouver, British Columbia. He received his Bachelor's degree in Science with concentration in Computing Science from Simon Fraser University in 2006.