ELL888 Indian Institute of Technology Delhi
Kernel Optimization for Minimal Complexity Machines (MCMs)
Scribed by: *Mridul Gupta (2021AIZ8322)*
Instructors: Sandeep Kumar and Jayadeva

Contents

1 Introduction **1**

2 Background **2**
2.1 Statistical Learning Theory 2
2.2 Minimal Complexity Machines [3] 3
2.2.1 Motivation 3
2.2.2 MCM formulations 3
2.2.3 Generalizing the MCM 6
2.3 Kernel Optimization 8
2.3.1 Motivation 8
2.3.2 Formulations 8

3 Kernel optimization using conformal maps for the minimal complexity machine [5] **12**

1 Introduction

Support Vector Machines (SVMs) are older than the state of Haryana (the linear variant at least), and the kernel version of SVMs were proposed by Vapnik *et al.* in 1992. Vapnik and Chernvonenkis along with others also developed theory key thoeretical concepts that constitute statistical learning theory. In what follows, we discuss the theoretical and practical advancements since then that led to the work titled “Kernel optimization using conformal maps for the minimal complexity machine” by Badge *et al.*

Since this is the congruence of two paths, one leading to MCMs and the other leading to kernel optimization in a data dependent way, in the following we first discuss MCMs, and then kernel optimization, and finally how the two fit together.

2 Background

The key concepts that led to MCMs were available in statistical learning theory long ago, but weren't applied until 2015. So I'll provide the background information necessry to understand MCMs. Then we discuss how a similar line of thinking naturally leads to kernel optimization, followed by a discussion about application of kernel optimization in MCMs.

2.1 Statistical Learning Theory

Let's starts by clearly stating the objective of Machine Learning. The model of learning from examples can be descibed using three components [2]:

1. a generator of random vectors x, drawn independently from a fixed but unknown distribution $P(x)$;
2. a supervisor that returns an output vector y for every input vector x, according to a conditional distribution function $P(y \mid x)$, also fixed but unknown;
3. a learning machine capable of implementing a set of functions $f(x, \theta), \theta \in \Theta$.

The problem of learning then is to choose the "right" function from the family of functions $f(x, \theta), \theta \in \Theta$. We want to choose this function so that it predicts the supervisor's response on seen as well as previously unseen data in the best possible way. The function choice has to be made based on a set of pairs $\{(x_i, y_i)\}_{i=1}^{M}$ called the training set. These samples are drawn from the distribution $P(x)P(y \mid x)$ which is the joint distribution $P(x, y)$. The M samples are assumed to be *independent and identically distributed (i.i.d.)*.

Next we need to define what "predicting the response in the best possible way means". For this we need to define how correct is $f(x, \theta)$ is compared with y. For this, one usually defines an error function (also called loss function or cost function) that assigns some real valued number to two objects (may be vectors, matrices, scalars, or general mathematical objects). This number represents the "cost" incurred by predicting $f(x, \theta)$ given y.

The next important quantity is the *risk functional* which is the expectation of loss. Since, x is a random vector, the output of $f(\cdot, \theta)$ is also random, so is the loss $\mathcal{L}$. Thus, it makes sense to take the expectation of it.

$$R(\theta) = \mathbb{E}_{(x,y)\sim P(x,y)}\Big[\mathcal{L}\big(y, f(x, \theta)\big)\Big] = \int \mathcal{L}\big(y, f(x, \theta)\big) dP(x, y) \tag{1}$$

The goal is then to find θ^* that minimizes the risk functional $R(\theta)$. The problem is that $P(x, y)$ is unknown, all that's known is the training set $\{(x_i, y_i)\}_{i=1}^{M}$.

What we do have, then, is:

$$R_{\text{emp}}(\theta) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}\big(y, f(x, \theta)\big) \tag{2}$$

R_{emp} is called the *empirical risk*. Given these quantities, we have the following result from statistical learning theory:

Theorem 2.1 *(Vapnik) If $0 \leq \mathcal{L}\big(y, f(x,\theta)\big) \leq B, \theta \in \Theta$, that is the loss is totally bounded, then with probability at least $1-\eta$ the inequality*

$$R(\theta) \leq R_{emp}(\theta) + \frac{B\varepsilon}{2}\left(1 + \sqrt{1 + \frac{4R_{emp}(\theta)}{B\varepsilon}}\right) \tag{3}$$

holds true simultaneously for all functions of the set $\mathcal{L}\big(y, f(x,\theta)\big)$. where

$$\varepsilon = 4\frac{h\left(\ln\frac{2M}{h}+1\right) - \ln\eta}{M} \tag{4}$$

and h is the VC-dimension of the set of functions.

On the RHS of equation 3, the term other than the empirical risk $R_{\text{emp}}(\theta)$ is called the *structural risk*. For a classification problem with zero-one loss, $B = 1$.

Another important result from Vapnik that shall be needed to continue our discussion of MCMs is:

Theorem 2.2 *(Vapnik) Let vectors $x \in X$ belong to a sphere of radius R. Then the set of Δ-margin separating hyperplanes has the VC-dimension h bounded by the inequality*

$$h \leq \min\left(\left[\frac{R}{\Delta}\right]^2, n\right) + 1. \tag{5}$$

where n is the feature dimension of the data. For more, refer [2], but for our discussion this is sufficient.

2.2 Minimal Complexity Machines [3]

2.2.1 Motivation

We want to minimize the actual **risk**, $R(\theta)$, in order to generalize prediction on future samples. We can do this by tightening the bound on the RHS of equation 3. We can optimize the empirical risk, and we can optimize the structural risk. Note that in structural risk, the only thing we can alter is ε. We need to minimize ε to tighten the bound on the risk. And in the expansion of ε we can alter only h assuming the training sample size is as large as we can get. $\varepsilon \propto h$. Thus to reduce ε, we want to minimize h.

Now let's look at figure 1. Imagine we have the data points in $\mathbb{R}^1$. The blue dot shows the optimal hyperplane that classifies the points with maximum margin. R is the radius of the hypersphere and Δ is the margin. According to equation 5, we'd like to find the hyperplane such that R is minimized while Δ is maximized simultaneously.

2.2.2 MCM formulations

The mathematical formulations of MCM [3] start by defining a mathematical quantity h_{MCM} as

$$h_{MCM} = \frac{\max_{i=1,2,\ldots,M}\|u^T x_i + v\|}{\min_{i=1,2,\ldots,M}\|u^T x_i + v\|} \tag{6}$$

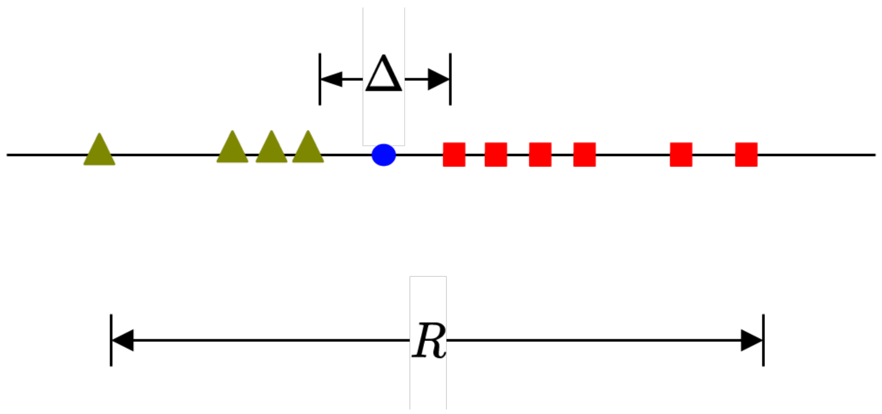


Figure 1: Data points $\in \mathbb{R}^1$

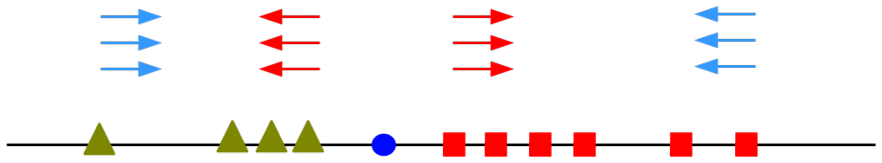


Figure 2: The red arrows are operations on margin and blue arrows on radius

where it is assumed that the separating hyperplane is $u^T x + v = 0$. It is assumed that the data are linearly separable. $x_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$

Augmenting the data vectors to have a feature whose value is always 1, and concatenating the weight vector, we have $\hat{x}_i \leftarrow \{x_i; 1\}$ and $\hat{u} \leftarrow \{u; v\}$. Then the hyperplane goes through the origin in the $\mathbb{R}^{(n+1)}$ space.

Then, the margin Δ is given by

$$\Delta = \min_{i=1,2,\ldots,M} \frac{\|\hat{u}^T \hat{x}_i\|}{\|\hat{u}\|} \tag{7}$$

and the radius is just $R = \max_{i=1,2,\ldots,M} \|\hat{x}_i\|$. So, the ratio of interest $\frac{R}{\Delta}$ is given by:

$$\frac{R}{\Delta} = \frac{\max_{i=1,2,\ldots,M} \|\hat{x}_i\|}{\min_{i=1,2,\ldots,M} \frac{\|\hat{u}^T \hat{x}_i\|}{\|\hat{u}\|}} = \frac{\max_{i=1,2,\ldots,M} \|\hat{u}\| \|\hat{x}_i\|}{\min_{i=1,2,\ldots,M} \|\hat{u}^T \hat{x}_i\|} \tag{8}$$

And using the Cauchy-Schwarz inequality ($\|a^T b\| \leq \|a\| \|b\|$)

$$\frac{R}{\Delta} \geq \frac{\max_{i=1,2,\ldots,M} \|\hat{u}^T \hat{x}_i\|}{\min_{i=1,2,\ldots,M} \|\hat{u}^T \hat{x}_i\|} = \frac{\max_{i=1,2,\ldots,M} \|u^T x_i + v\|}{\min_{i=1,2,\ldots,M} \|u^T x_i + v\|} \tag{9}$$

Thus

$$h_{MCM} \leq \frac{R}{\Delta} \tag{10}$$

$$\Rightarrow h_{MCM}^2 \leq \left(\frac{R}{\Delta}\right)^2 < 1 + \left(\frac{R}{\Delta}\right)^2 \tag{11}$$

From equation 5 we have for large dimensional data:

$$h \leq 1 + \left(\frac{R}{\Delta}\right)^2 \tag{12}$$

Thus $\exists \beta \in \mathbb{R}^+$, such that $h \leq \beta h_{MCM}^2$. Also since, $h_{MCM}^2 \geq 1$ and VC-dimension satisfies $h \geq 1$, $\exists \alpha \in \mathbb{R}, \alpha > 0$ such that $\alpha h_{MCM}^2 \leq h$. Combining the two we have $\exists \alpha, \beta > 0, \alpha, \beta \in \mathbb{R}$ such that

$$\alpha h_{MCM}^2 \leq h \leq \beta h_{MCM}^2 \tag{13}$$

That is h_{MCM}^2 is an exact bound on the VC-dimension h. And since the data are linearly separable, $u^T x_i + v \geq 0$ if $y_i = 1$ and $u^T x_i + v \leq 0$ if $y_i = -1$. Thus $\|u^T x_i + v\|$ can be written as $y_i(u^T x_i + v)$. Thus the machine capacity can be minimized by keeping h_{MCM}^2 as small as possible.

$$\underset{u,v}{\text{minimize}} \;\; h_{MCM} = \frac{\max_{i=1,\ldots,M} y_i(u^T x_i + v)}{\min_{i=1,\ldots,M} y_i(u^T x_i + v)} \tag{14}$$

Further the authors show that the above formulation can be simplified by writing:

$$h_{MCM} = \frac{g}{l} \tag{15}$$

$$\min_{u,v,g,l} \frac{g}{l} \quad \text{subject to} \tag{16}$$

$$g \geq y_i(u^T x_i + v), \quad i = 1, \ldots, M \tag{17}$$

$$l \leq y_i(u^T x_i + v), \quad i = 1, \ldots, M \tag{18}$$

Using Charnes-Cooper transformation, introducing $p = \frac{1}{l}$

$$\min_{u,v,g,l,p} g \cdot p \quad \text{subject to} \tag{19}$$

$$g \cdot p \geq y_i(p \cdot u^T x_i + p \cdot v), \quad i = 1, \ldots, M \tag{20}$$

$$l \cdot p \leq y_i(p \cdot u^T x_i + p \cdot v), \quad i = 1, \ldots, M \tag{21}$$

$$p \cdot l = 1 \tag{22}$$

Denoting $w \triangleq p \cdot u, b \triangleq p \cdot v$ and noting that $p \cdot l = 1$

$$\min_{w,b,h} h \quad \text{subject to} \tag{23}$$

$$h \geq y_i(w^T x_i + b), \quad i = 1, \ldots, M \tag{24}$$

$$1 \leq y_i(w^T x_i + b), \quad i = 1, \ldots, M \tag{25}$$

Equations 23-25 define the Minimal Complexity Machine (MCM). And it is trained by solving the Linear Programming Problem defined above.

2.2.3 Generalizing the MCM

The MCM above is generalized to allow for classification errors by introducing slack variables.

$$\begin{aligned}
&\min_{w,b,h,q} h + C \cdot \sum_{i=1}^{M} q_i \\
&h \geq y_i(w^T x_i + b) + q_i, \quad i = 1, \ldots, M \\
&1 \leq y_i(w^T x_i + b) + q_i, \quad i = 1, \ldots, M \\
&q_i \geq 0 \quad i = 1, \ldots, M
\end{aligned}$$

And for the non-linear case using Kernels

$$\min_{w,b,h,q} h + C \cdot \sum_{i=1}^{M} q_i \tag{26}$$

$$h \geq y_i(w^T \phi(x_i) + b) + q_i, \quad i = 1, \ldots, M \tag{27}$$

$$1 \leq y_i(w^T \phi(x_i) + b) + q_i, \quad i = 1, \ldots, M \tag{28}$$

$$q_i \geq 0 \quad i = 1, \ldots, M \tag{29}$$

where $\phi(\cdot)$ is the mapping function that maps input to a higher dimensional space, where the inputs are assumed to be linearly separable with some errors. As of yet, this doesn't use a kernel function $K(\cdot, \cdot)$. Assume that K is a kernel function corresponding to the map $\phi(\cdot)$; $\phi(x_j)^T \phi(x_k) = K(x_j, x_k)$. Also, since $\phi(x_i), i = 1, \ldots, M$ forms a basis for the feature space in which w lies, w can be written as a linear combination of the basis vectors

$$\begin{aligned}
&w = \sum_{j=1}^{M} \varsigma_j \phi(x_j) \\
&\Rightarrow w^T \phi(x_i) = \sum_{j=1}^{M} \varsigma_j \phi(x_j)^T \phi(x_i) = \sum_{j=1}^{M} \varsigma_j K(x_j, x_i)
\end{aligned}$$

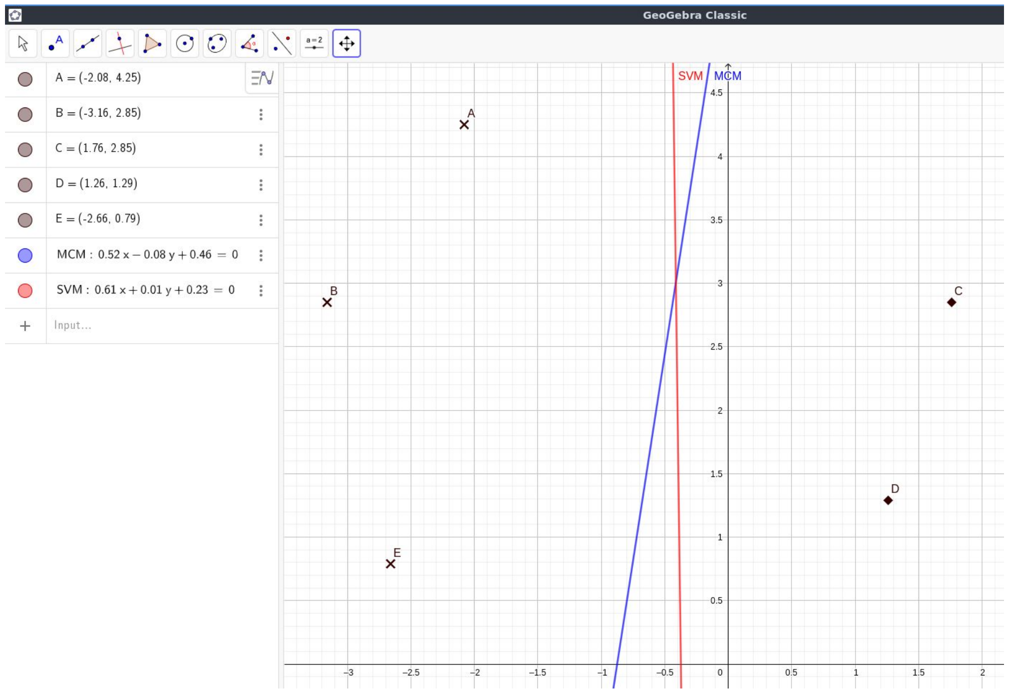
Now, the MCM formulation can be rewritten using $K(\cdot, \cdot)$ as

$$\min_{\varsigma,b,h,q} h + C \cdot \sum_{i=1}^{M} q_i \quad \text{subject to} \tag{30}$$

$$h \geq y_i \left(\sum_{j=1}^{M} \varsigma_j K(x_j, x_i) + b \right) + q_i, \quad i = 1, \ldots, M \tag{31}$$

$$1 \leq y_i \left(\sum_{j=1}^{M} \varsigma_j K(x_j, x_i) + b \right) + q_i, \quad i = 1, \ldots, M \tag{32}$$

$$q_i \geq 0 \quad i = 1, \ldots, M \tag{33}$$



A = (-2.08, 4.25)

B = (-3.16, 2.85)

C = (1.76, 2.85)

D = (1.26, 1.29)

E = (-2.66, 0.79)

MCM : 0.52 x − 0.08 y + 0.46 = 0

SVM : 0.61 x + 0.01 y + 0.23 = 0

Figure 3: Separating hyperplanes learned by MCM and SVM

After optimizing, those vectors x_i for which the corresponding ς_i is non zero form the support vectors that support w. $SV = \{i \mid \varsigma_i \neq 0\}$. One last thing that needs to be defined explicitly is how to predict on new data once the optimization is complete. The class is assigned as:

$$y_{\text{pred}} = \text{sign}\left(\sum_{i \in SV} \varsigma_i K(x_i, x_{\text{test}}) + b\right)$$

This completes the MCM formulation. One thing to note is Vapnik's SVM formulation was also motivated from the fact that constraining the VC-dimension will improve generalization, but Vapnik does this by only trying to maximize the margin Δ, whereas in MCM the ratio R/Δ is constrained to control the complexity of the machine. This results in an extra constraint bounding the maximum distance from the hyperplane from above (eq. 31), thus providing actual theoretical guarantees. It has also been experimentally showed that the number of support vectors (and thus the size itself of MCM) is much smaller, allowing these machines to be used in edge devices where neural networks can't be used. In figure 3, I take a very simple two dimensional dataset, and solve a linear MCM without slack using an online linear programming solver. To compare with SVM, I use `scikit-learn`'s `LinearSVC` module with a very high value of C (so basically without slack) and plot the decision boundaries for the two. It's not at all apparent that one of them has VC-dimension bound by a known number as a side effect of optimization, whereas the other doesn't. And the difference in performance on test dataset can only be understood by looking at the numbers.

Note: Since MCM is linear optimization, the kernel is not required to be positive definite from the optimization perspective. From the SVM theory perspective, it'll then not lie in the familiar

Reproducing Kernel Hilbert Space. It'll instead lie in a *Krein space* and the inner product will have to be adjusted accordingly. We only consider Mercer Kernels.

2.3 Kernel Optimization

The key ideas for kernel optimization comes from [1]. When the data are non-linearly separable, it is assumed that when projected to a space of high enough dimension, they'll be separable. But this is only true when the mapping function is suitably chosen. No kernel defeats all others. If the kernel is "wrong" for the current task, the class separability in the feature space/image space may be poorer than it was in input space.

2.3.1 Motivation

Again let's consider the data shown in figure 1. Even though the data are already linearly separable, let's see what we can do to improve the class separability. Consider figures 4-6. We can see that if we magnify around the feature space around the class boundary (near hyperplane) while compressing the space around the edges, we can improve class separability thus improving generalization.

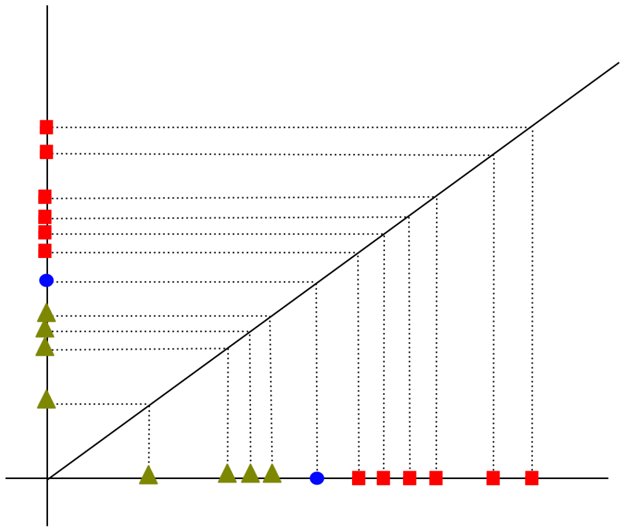


Figure 4: Identity map, to compare with

2.3.2 Formulations

This is approximately what Amari *et al.* do. They do not compress around the edges, but just focus on magnifying on the confusion region, the boundary while leaving the radius intact. That is they try to increase the margin even further by having an appropriate distortion. Since the boundary is not generally known, magnification is done close to support vectors, since they are close to the boundary. These are called "empirical cores" in the context of kernel optimization (since after optimization the actual support vectors will be different).

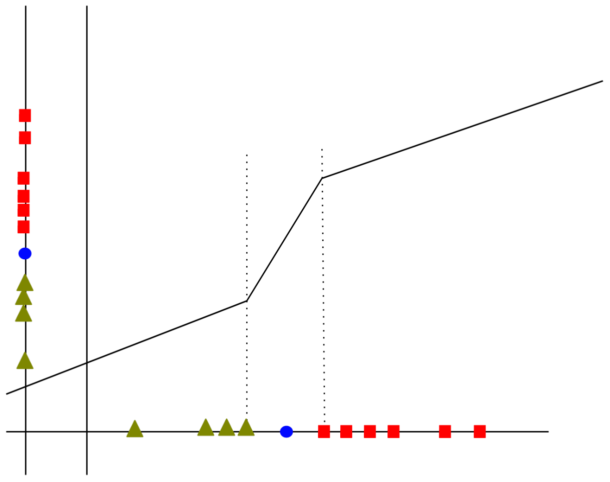


Figure 5: A non-linear map that is steeper between the classes, shallower within (data dependent)

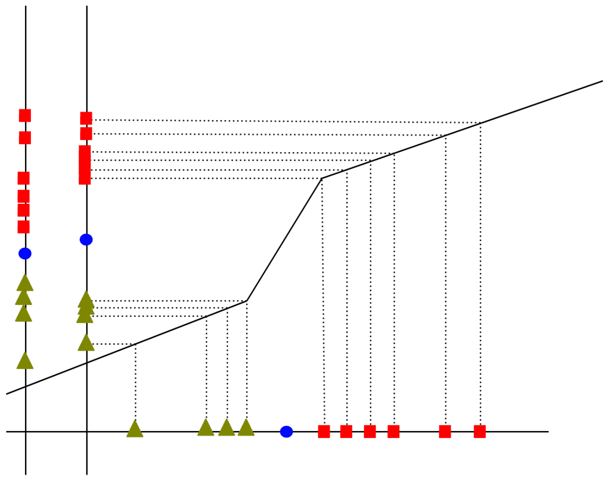


Figure 6: This improves class separability, and also compresses the data reducing VC-dimension

The "magnification" is done by applying a *conformal transformation* to the kernel. Conformal transformation of a space preserves angles but may change lengths locally, which is just what we need, to magnify without altering angular structures. Conformal transformation of the kernel *by a factor* $c(x)$ is given by:

$$\tilde{K}(x, x') = c(x)c(x')K(x, x')$$

This transformation function is constructed in a data dependent way to be maximal around support vectors. For more detail about how to arrive at this, and the underlying Riemannian metric tensor based theory, refer [1].

This work is taken further by Xiong *et al.* [4]. Until now we have been talking about "class separability" subjectively, but Xiong *et al.* solidify this by giving an actual mathematical formulation of class separability that can be used to guide the kernel optimization. But first define the factor function they use

$$k(x, y) = c(x)c(y)k_0(x, y)$$
$$c(x) = \alpha_0 + \sum_{i=1}^{n_{ec}} \alpha_i k_1(x, a_i)$$

$k_0(\cdot, \cdot)$ is a basic kernel, like Gaussian. $k_1(x, a_i) = \exp(-\gamma\|x - a_i\|^2)$ and a_i are "empirical cores". We can see that $k_1(x, a_i)$ is maximum near a_i, and thus most magnification is around the empirical cores. Also, to get an intuitive sense of "magnification", consider that the $k_0(\cdot, \cdot)$ is some sort of distance or similarity measure (that's what the Riemannian metric implies, measure of distances). α_i's are called combination coefficients. It is easy to write the matrix notation of these by defining K and K_0 as the matrix corresponding to $k(\cdot, \cdot)$ and $k_0(\cdot, \cdot)$ respectively.

$$K = [c(x_i)c(x_j)k_0(x_i, x_j)]_{M\times M} = CK_0C$$

where $C = diag(c(x_1), c(x_2), \ldots, c(x_M))$. Furthermore

$$\mathbf{c} = \begin{pmatrix} 1 & k_1(x_1, a_1) & \ldots & k_1(x_1, a_{n_{ec}}) \\ 1 & k_2(x_2, a_1) & \ldots & k_1(x_2, a_{n_{ec}}) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & k_M(x_M, a_1) & \ldots & k_1(x_M, a_{n_{ec}}) \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n_{ec}} \end{pmatrix} \triangleq K_1\alpha \tag{34}$$

where K_1 is an $M \times (n_{ec} + 1)$ matrix.

Next, they use a measure of class separability called *Fisher scalar* to define their own Kernel based class separability metric. The Fisher scalar is defined as

$$J = \frac{\operatorname{tr} S_b}{\operatorname{tr} S_w} \tag{35}$$

$$S_b = \frac{1}{M}\sum_{i=1}^{2} M_i(\bar{z}_i - \bar{z})(\bar{z}_i - \bar{z})^T \tag{36}$$

$$S_w = \frac{1}{M}\sum_{i=1}^{2}\sum_{j=1}^{M_i}(z_j^i - \bar{z}_i)(z_j^i - \bar{z}_i)^T \tag{37}$$

where i indexes through the two classes, M_i is the number of samples in the i^{th} class. $\{z_j\}_{j=1}^{M}$ are the images of the training data in the empirical feature space. $M_1 + M_2 = M$. $\bar{z}, \bar{z}_1, \bar{z}_2$ are the centers of the entire training data and those of class 1 and 2 resepctively in the empirical feature space. z_j^i is the j^{th} data sample in i^{th} class. S_b and S_w are the between-class scatter and within-class scatter. Reducing the within-class scatter reduces the neighborhood size, thus reducing the radius. While increasing the between-class scatter increases the distances between the samples of different classes. Thus the chances of misclassification in a dense neighborhood go down (or structural risk goes down). This is why we want maximize J. See figure 7 where Xiong *et al.* show how the two

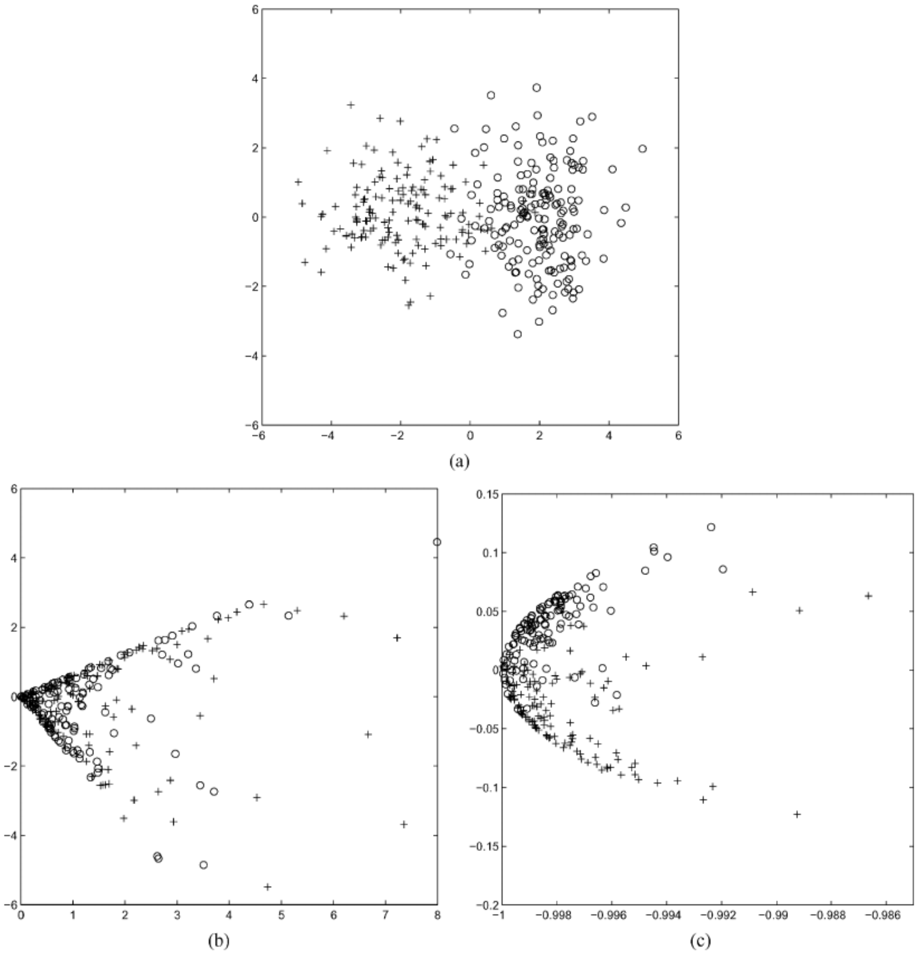


Figure 7: This picture is taken from Xiong *et al.* [4]. It shows how a kernel can harm class separability. (a) Input. (b) Two dimensional projection of empirical feature space for second order polynomial kernel. (c) Two dimensional projection of Gaussian kernel.

popular kernels harm class separability. The transformed space has lower J value.

This can be further simplified by ordering the data points such that the first M_1 points belong to class 1, $y_i = -1, i \leq M_1$ and the remaining belong to class 2. Then kernel matrix can be written in a block form as:

$$K = \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix}$$

K_{11} is an $M_1 \times M_1$ submatrix of K that corresponds to class 1. Similarly K_{22} is an $M_2 \times M_2$ sized matrix corresponding to class 2. K_{12} and K_{21} are $M_1 \times M_2$ and $M_2 \times M_1$ respectively. Next define "between-class" and "within-class" kernel scatter matrices B and W as:

$$B = \begin{pmatrix} \frac{1}{M_1}K_{11} & 0 \\ 0 & \frac{1}{M_2}K_{22} \end{pmatrix} - \begin{pmatrix} \frac{1}{M}K_{11} & \frac{1}{M}K_{12} \\ \frac{1}{M}K_{21} & \frac{1}{M}K_{22} \end{pmatrix} \tag{38}$$

$$W = \begin{pmatrix} k_{11} & 0 & \dots & 0 \\ 0 & k_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & k_{MM} \end{pmatrix} - \begin{pmatrix} \frac{1}{M_1}K_{11} & 0 \\ 0 & \frac{1}{M_2}K_{22} \end{pmatrix} \tag{39}$$

Also, denote B_0 and W_0 as the between-class and within-class kernel scatter matrices corresponding to the basic kernel K_0.

Theorem 2.3 *(Xiong* et al.*) Let 1_k be the k-dimensional vector whose entries are all equal to one. Then*

$$J = \frac{1_M^T B 1_M}{1_M^T W 1_M} = \frac{\mathbf{c}^T B_0 \mathbf{c}}{\mathbf{c}^T W_0 \mathbf{c}} \tag{40}$$

Proof: refer [4] Xiong *et al.* then optimize the kernel such that the class separability metric J is maximized using a gradient based approach. Here we diverge to the approach used by Badge *et al.* in [5]. Instead of solving as a gradient based optimization, they identify the Rayleigh Quotient in equation 40 and naturally propose it as a Generalized Eigenvalue Problem and try to solve simultaneously for λ and $\mathbf{c}$ in

$$B_0\mathbf{c} = \lambda W_0 \mathbf{c}$$

where λ is the set of eigenvalues. This is further rewritten using equation 34 as $[K_1^T B_0 K_1]\alpha = \lambda[K_1^T(W_0 + DI)K_1]\alpha$. D is a small constant added to the diagonal values for stability, I is identity matrix. Let $K_1^T B_0 K_1 \triangleq P, [K_1^T(W_0 + DI)K_1] \triangleq Q$. So we have

$$P\alpha = \lambda Q \alpha \tag{41}$$

From the solution we can get α (corresponding to maximum eigenvalue λ), which in turn gives $\mathbf{c}$, which in turn gives the conformal map transformed kernel K. And, finally we have enough theory to discuss Kernel optimization for MCMs. Note, all the previous theory regarding kernel optimization was in context of support vector machines (SVMs).

3 Kernel optimization using conformal maps for the minimal complexity machine [5]

We have basically every ingredient necessary to talk about this: we have the motivation to use MCMs, we have the motivation and the method to perform kernel optimization, we have the statistical learning theory backing up our intuition that separating out classes should improve generalization. Now let's talk about how kernel optimization is done in the context of MCMs. It's just a simple algorithm:

1. Transform the dataset to zero mean and unit variance.
2. Solve the optimization defined in equations 30-33 using the basic kernel to get support vectors for which $\lambda_i \neq 0$ (or to account for computational limitations, $|\lambda_i| > \varepsilon$ where ε is a small constant.
3. Compute W_0, B_0 (eq 39,38)
4. Solve the generalized eigenvalue problem in equation 41 to find α corresponding to largest eigenvalue.
5. Use α to compute the scaling factor $\mathbf{c}$
6. Use the scaling factor to compute the optimized kernel $k(x_i, x_j) = c(x_i)c(x_j)k_0(x_i, x_j)$
7. Use the optimized kernel matrix $K(\cdot, \cdot)$ corresponding to kernel function $k(\cdot, \cdot)$ to train a new MCM (that is again solve equations 30-33.)
8. Use this new MCM on test data.

Results in [5] shows that MCM with optimized kernels perform better when measured by accuracy by a big margin for almost all of the benchmark datasets as compared with vanilla MCM or SVM. The maximum number of samples in any of these datasets was 1000. When applied on datasets of larger size, the MCM's do not perform much better, which is expected from equation 4. As M, the training sample size plays a much larger role in reducing ε and in turn reducing the structural risk, it doesn't help a lot in big data setting to focus on structural risk minimization. But MCMs still provide a sparser model to be deployed on power constrained devices.

Lastly, I'd like to show you some visualization I made by running an implementation of the algorithm in [5]. Figure 8 shows the data, I generate two classes from multivariate normal distributions. There are 100 data points, 50 in each class. And figure 9 shows the projected class for an unoptimized and an optimized Gaussian RBF kernel. This is implemented in `python` using `scipy`'s linear optimization library for solving MCM and Lanczos implementation (`eigsh`) for solving generalized eigenvalaue problem. The MCM solution gives 85 out of the 100 data points as support vectors, so 15% of them (that is 12 of them) are randomly chosen as empirical cores. This was decided empirically. Basically distorting the space everywhere is a really bad choice. The accuracy on unseen test data was 65%. After optimization, the number of support vectors went down to 75, while the accuracy rose to 75%. It can be seen in the figure 9 that optimization doesn't really improve class separation by a big margin since the data is already pretty well separated in the input space. Also, we can see that the within class scatter reduces for one of the classes. What's happening with the other class is not yet fully understood to me.

For other visualizations, results and more please see the papers referred.

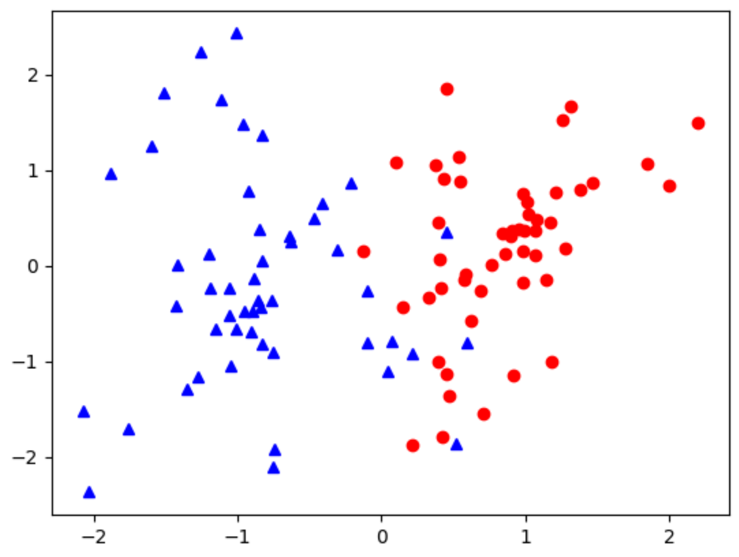
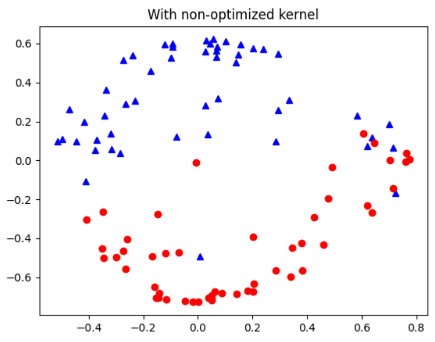
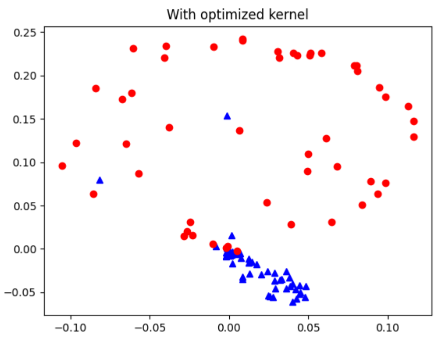


Figure 8: Training Data



(a) Unoptimized RBF Kernel



(b) Optimized Kernel

Figure 9: Training Data mapped through kernels and then projected to two dimensional Euclidean space through kernel PCA.

References

[1] Amari SI, Wu S. *Improving support vector machine classifiers by modifying kernel functions.*. Neural Networks. 1999 Jul 1;12(6):783-9.

[2] Vapnik VN. *An overview of statistical learning theory.* IEEE transactions on neural networks. 1999 Sep;10(5):988-99.

[3] Jayadeva. *Learning a hyperplane classifier by minimizing an exact bound on the VC dimension.* Neurocomputing. 2015 Feb 3;149:683-9.

[4] Xiong H, Swamy MN, Ahmad MO. *Optimizing the kernel in the empirical feature space.*IEEE transactions on neural networks. 2005 Mar 7;16(2):460-74.

[5] Badge S, Soman S, Chandra S, Jayadeva. *Kernel optimization using conformal maps for the minimal complexity machine.*Engineering Applications of Artificial Intelligence. 2021 Nov 1;106:104493.