A study on Deep Multi-layer Kernel Machines

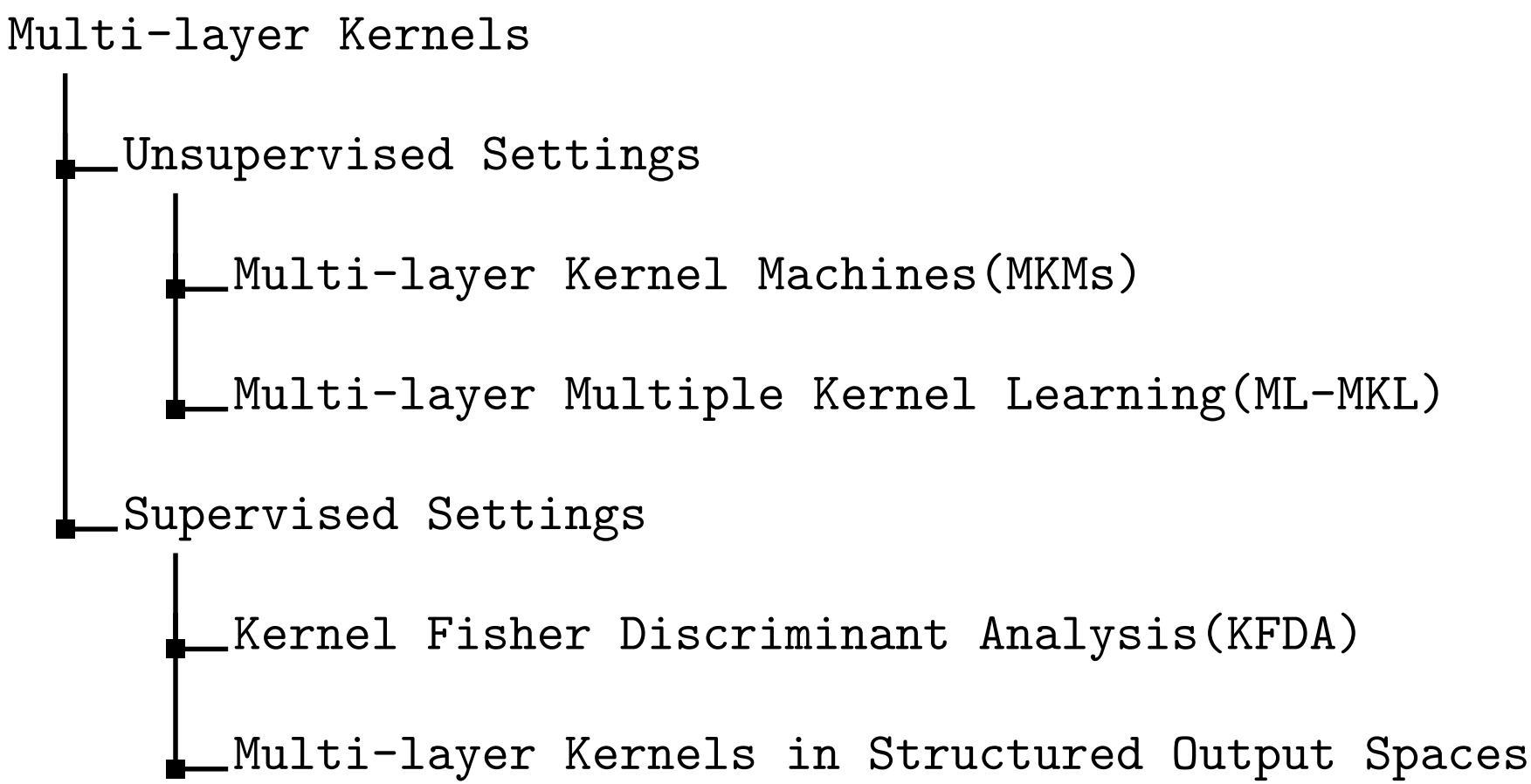
Akhil P M

Supervisor : Sumitra S.

June 6, 2016

Overview

Multi-layer Kernels



Representation Learning

Representation learning : a set of **preprocessing** of raw data, to produce a good representation of the data itself, which simplifies extracting useful information from the data when building classifiers or other predictors.

Eg: dimensionality reduction, feature extraction, de-noising etc.

An effective mechanism for transferring the **prior knowledge** of humans to the machine learning system.

Eg: Sparse auto-encoders are formulated with the assumption of sparse representation of data in human brain.

Representation Learning

ML algorithms are broadly classified into two

- Shallow architectures(eg: SVM, single hidden layer neural net, perceptron etc)
- Deep Architectures(eq: deep learning algorithms like CNN, DBN etc.)

Deep Learning

Deep learning algorithms makes use of depth as a key criteria for producing good representations(hence they belong to deep architectures).

Deep Learning is a set of algorithms in machine learning that attempts to model high-level abstractions in data by using model architectures composed of multiple non-linear transformations.

Deep Learning

Deep Learning methods are preferred over shallow ones in many complex learning tasks such as computer vision, speech recognition etc. due to:

- the wide range of functions that can be parameterized by composing weakly nonlinear transformations.
- the broad range of prior knowledge they can transfer to the learning system.
- the invariance being modelled by such systems against local changes in the input.
- their ability to learn more abstract concepts in an hierarchical fashion.

Deep Learning

Challenges:

- the optimization process often gets trapped in local minima which results in poor generalization capacity.
- training the networks with more than 2 or 3 layers was a big computational challenge.

Deep Learning

Solutions:

- Hinton et al. proposed *greedy layerwise unsupervised pre-training* method which is used to initialize the weights of the network.
- Rise in computing power made computational challenges less problematic(with multi-core CPUs and GPUs, highs-speed clusters)

Deep Learning

Most of the deep learning algorithms are developed on top of neural network based models. Cho et al. explored the concept of kernel methods based Deep Learning by proposing **Multi-layer Kernel Machines**(MKMs).

The architecture of MKMs consists of multiple layers, with each layer performing unsupervised feature extraction using kernel PCA followed by supervised feature selection.

Multi-layer Kernel Machines

MKMs are extended in different forms:

- by using Multiple Kernel Learning(MKL) in each layer.
- by using other methods for feature extraction in place of kernel PCA.

Multi-layer Kernel Machines

Typical kernel functions are single layer in nature.

$$k(x, y) = \phi(x) \cdot \phi(y)$$

where $x, y \in \mathcal{X}$, a normed space.

Iteratively applying the mapping $\phi(\cdot)$, we get a multi-layer kernel function

$$k^{(L)}(x, y) = \underbrace{\phi(\phi(\ldots\phi(x)))}_{\text{L times}} \cdot \underbrace{\phi(\phi(\ldots\phi(y)))}_{\text{L times}}$$

$L =$ number of layers

Multi-layer Polynomial Kernel

Single layer polynomial kernel is computed as

$$k(x, y) = (x \cdot y)^d$$

Two layer composition of polynomial kernel

$$\begin{aligned}\phi(\phi(x)) \cdot \phi(\phi(y)) &= \Big(\phi(x) \cdot \phi(y)\Big)^d \\ &= (x \cdot y)^{d\,d} = (x \cdot y)^{d^2}\end{aligned}$$

these are simply polynomials of higher degree.

Multi-layer Gaussian Kernel

Single layer gaussian kernel

$$k(x, y) = e^{-\lambda \|x-y\|^2}$$

Two layer composition of Gaussian kernel

$$\begin{aligned} \phi(\phi(x)) \cdot \phi(\phi(y)) &= e^{-\lambda \left\| \phi(x)-\phi(y) \right\|^2} \\ &= e^{-2\lambda(1-k(x,y))} \end{aligned}$$

This is also not doing any meaningful computation.

Multi-layer Arc-cosine Kernel

Cho et al. proposed a new kernel, on which multi-layer composition is meaningful.
Consider a single layer neural network with weights W_{ij} connects the j^{th} input unit to the i^{th} output unit. The network maps input x to output $f(x)$ by applying a non-linear map

$$f(x) = g(W \cdot x)$$

$g(\cdot)$ induces the non-linearity(activation function)

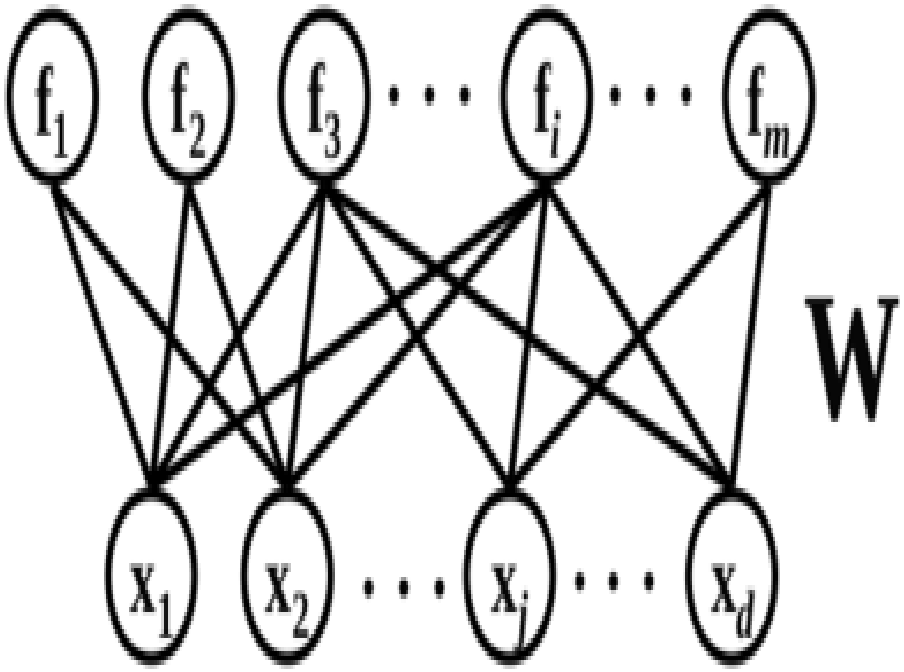
$$g_n(z) = \Theta(z)z^n$$

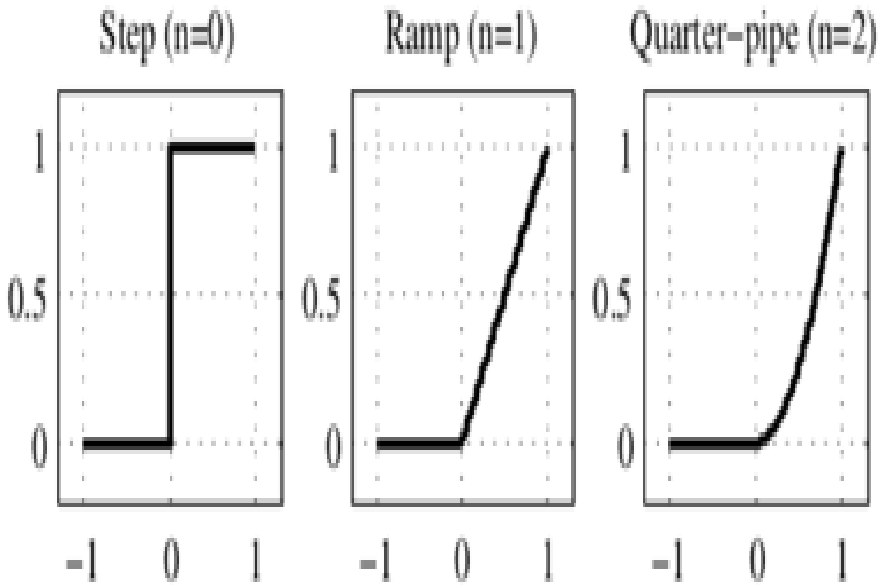
with

$$\Theta(z) = \frac{1}{2}(1 + sign(z))$$

$g_n(z)$ is called **one-sided polynomial activation function**.

Multi-layer Arc-cosine Kernel





Multi-layer Arc-cosine Kernel

Let $f(x)$ and $f(y)$ be the outputs corresponding to inputs x and y. Then

$$f(x) \cdot f(y) = \sum_{i=1}^{m} \Theta(w_i \cdot x)\Theta(w_i \cdot y)(w_i \cdot x)^n (w_i \cdot y)^n$$

- w_i = the i^{th} row of weight matrix W.
- m = the number of output units.

Assume W_{ij} are Gaussian distributed with zero mean and unit variance and the network has an infinite number of output units. Then

$$lim_{m \to \infty} \frac{2}{m} f(x) \cdot f(y) = k_n(x, y)$$

$$k_n(x, y) = 2 \int dw \frac{e^{-\frac{\|w^2\|}{2}}}{(2\pi)^{d/2}} \Theta(w \cdot x)\Theta(w \cdot y)(w \cdot x)^n (w \cdot y)^n$$

Multi-layer Arc-cosine Kernel

It has an alternate form

$$k_n(x, y) = \frac{1}{\pi} \|x\|^n \|y\|^n J_n(\theta)$$

where

$$J_n(\theta) = (-1)^n (sin\theta)^{2n+1} \left(\frac{1}{sin\theta} \frac{\partial}{\partial \theta} \right)^n \left(\frac{\pi - \theta}{sin\theta} \right)$$

n is called the **degree of the kernel** and

$$\theta = cos^{-1} \left(\frac{x \cdot y}{\|x\| \|y\|} \right)$$

Multi-layer Arc-cosine Kernel

$J_n(\theta)$ for n=0,1,2 is computed as shown below.

$$J_0(\theta) = \pi - \theta$$

$$J_1(\theta) = sin\theta + (\pi - \theta)cos\theta$$

$$J_2(\theta) = 3sin\theta cos\theta + (\pi - \theta)(1 + 2cos^2\theta)$$

for n=0, it takes the simple form

$$k_0(x, y) = 1 - \frac{1}{\pi} cos^{-1}\left(\frac{x \cdot y}{\|x\| \, \|y\|}\right)$$

hence the name Arc-cosine kernel is given!

Multi-layer Arc-cosine Kernel

Multi-layer composition is computed recursively in arc-cosine kernel

$$k_n^{(L+1)}(x,y) = \frac{1}{\pi}\left[k^{(L)}(x,x)\ k^{(L)}(y,y)\right]^{\frac{n}{2}} J_n(\theta_n^{(L)})$$

$\theta_n^{(L)}$ = the angle between images of x and y in the feature space after L layer composition

$$\theta_n^{(L)} = \cos^{-1}\left(k^{(L)}(x,y)\left[k^{(L)}(x,x)\ k^{(L)}(y,y)\right]^{\frac{-1}{2}}\right)$$

Multi-layer Arc-cosine Kernel

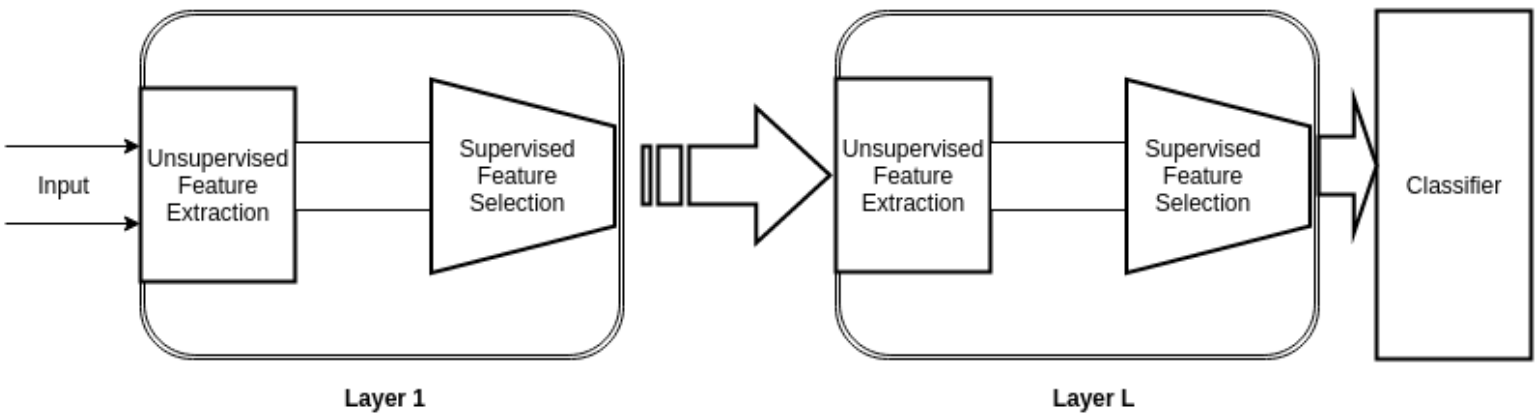
Intuition

if the base kernel $k(x, y) = \phi(x) \cdot \phi(y)$ can mimic the computation of single-layer network, then the iterated mapping in $k^{(L)}(x, y)$ can mimic the computation of multi-layer network.

Multi-layer Kernel Machines

MKMs are a new family of deep learning algorithm with *kernel methods*. In each layer we perform

- unsupervised feature extraction with KPCA.
- a supervised feature selection.



Multi-layer Kernel Machines

Dataset	Loss in Percentage						
	$\mathrm{SVM}_{\mathrm{RBF}}$	$\mathrm{SVM}_{\mathrm{Poly}}$	NNet	DBN-3	SAA-3	DBN-1	MKMs
back-rand	14.58	16.62	20.04	**6.73**	11.28	9.80	10.55
back-image	22.61	24.01	27.41	16.31	23.00	**16.15**	21.39
rot-back-image	55.18	56.41	62.16	**47.39**	51.93	52.21	51.61
rect-image	24.04	24.05	33.20	23.69	24.05	**22.50**	23.01

Table : Experimental Results of MKMs with Arc-cosine Kernels

Multi-layer Kernel Machines

Compared to DBN:

- MKMs are non-parametric, whereas DBN is parametric. DBN requires to learn millions of parameters to learn and hence computationally expensive.
- MKMs have no difficult optimization as compared to DBN.

Multi-layer Kernel Machines

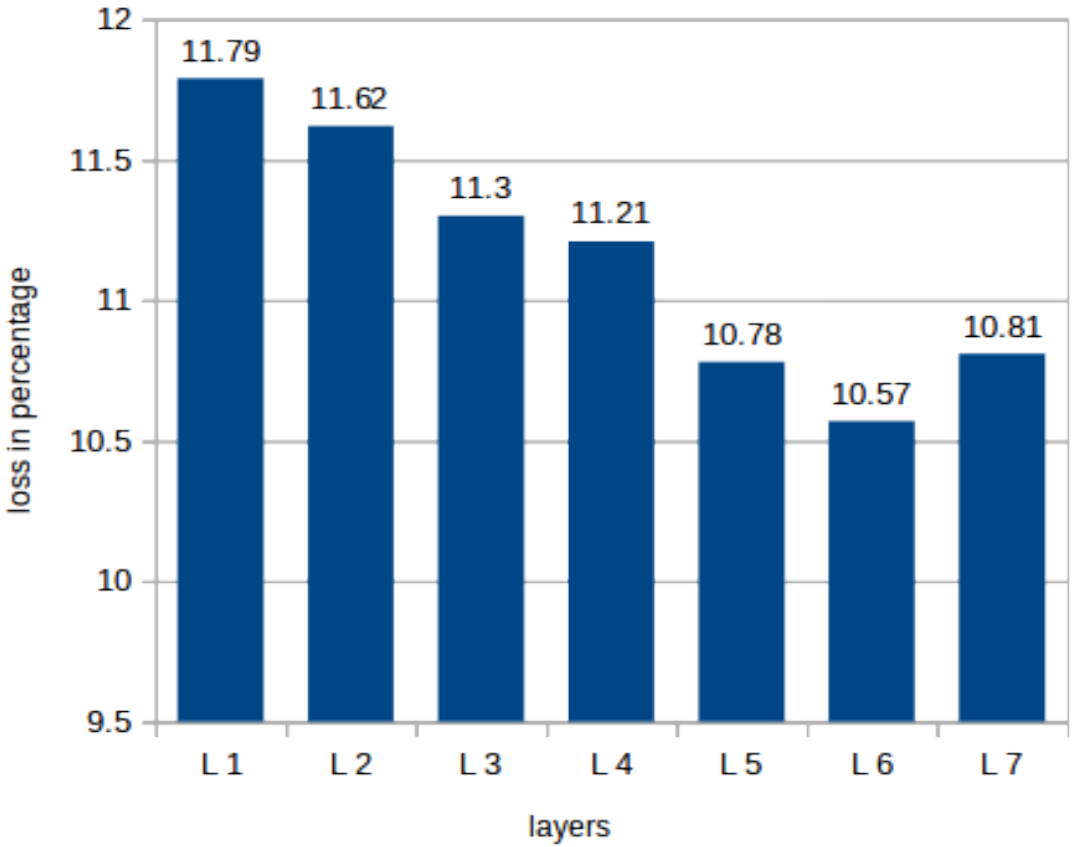


Figure : classifier performance on *mnist-back-rand* dataset.

Multi-layer Kernel Machines

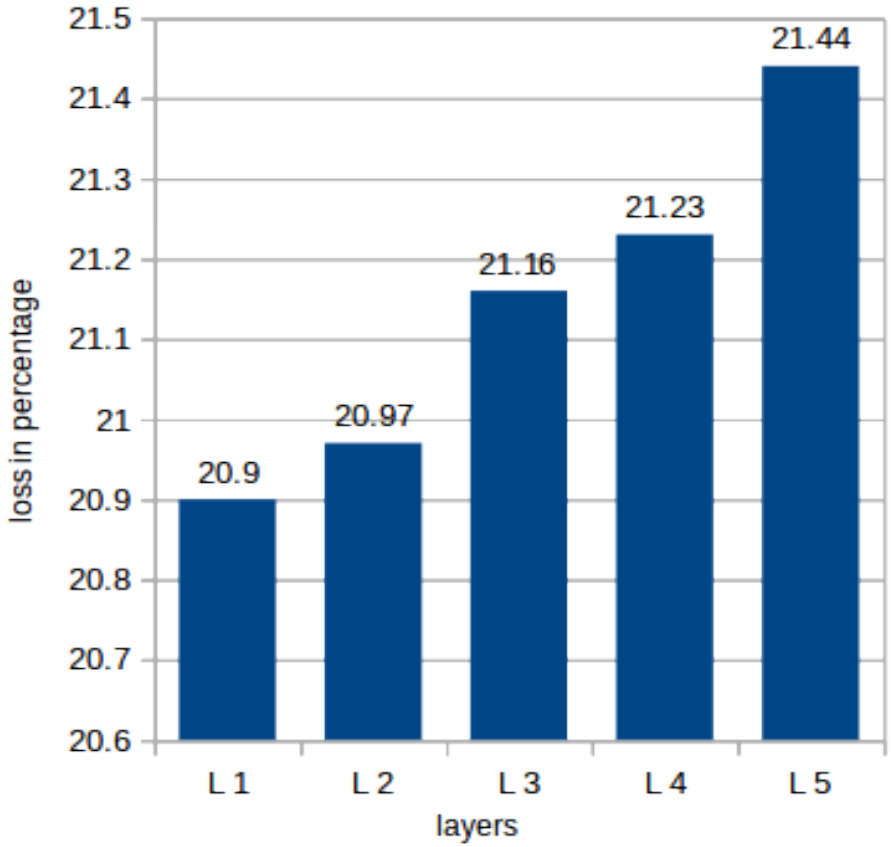


Figure : classifier performance on *mnist-back-image* dataset.

Multi-layer Mixed Kernel Machines

The architecture of the system is the same as MKMs, but different kernels are used in different layers.

Empirically, mixing a gaussian/polynomial kernel in between arc-cosine kernels gives the best result.

Multi-layer Mixed Kernel Machines

Dataset	Loss in Percentage						
	$\mathrm{SVM}_{\mathrm{RBF}}$	$\mathrm{SVM}_{\mathrm{Poly}}$	NNet	DBN-3	SAA-3	DBN-1	MKMs(mix)
back-rand	14.58	16.62	20.04	**6.73**	11.28	9.80	9.54
back-image	22.61	24.01	27.41	16.31	23.00	**16.15**	20.94
rot-back-image	55.18	56.41	62.16	**47.39**	51.93	52.21	54.03
rect-image	24.04	24.05	33.20	23.69	24.05	**22.50**	25.04

Table : Experimental Results of MKMs with Mixed Kernels

Multi-layer Multiple Kernel Learning

Multiple Kernel Learning(MKL) : automate the process of choosing optimum kernel for the learning task.

$$\min_{k \in \mathcal{K}} \min_{f \in \mathcal{H}_k} \lambda \|f\|^2_{\mathcal{H}_k} + \sum_{i=1}^{n} l(y_i, f(x_i))$$

$l(\cdot) \rightarrow$ the loss function
$\mathcal{H}_k \rightarrow$ the RKHS corresponding to the kernel $k \in \mathcal{K}$.

$$\mathcal{K} = \left\{ k(\cdot\ ,\ \cdot) = \sum_{t=1}^{m} \mu_t k_t(\cdot\ ,\ \cdot) : \ \sum_{t=1}^{m} \mu_t = 1, \mu_t \geq 0 \right\}$$

Unsupervised MKL

In MKMs we took a combination of kernels in each layer with unsupervised MKL.
For finding optimum kernel two quality criteria are used.

- A good kernel should enable each training instances to be well reconstructed from the localized bases weighted by the kernel values. ie, for each x_i, $\left\| x_i - \sum_j k_{ij} x_j \right\|^2$, where $k_{ij} = k(x_i, x_j)$, should be minimum.
- A good kernel should induce kernel values that are coincided with the local geometry of the training data. That is equivalent to minimizing the distortion over all training data $\sum_{i,j} k_{ij} \left\| x_i - x_j \right\|^2$.

Unsupervised MKL

The cost function

$$\min_{k\in\mathcal{K}} \frac{1}{2}\sum_{i=1}^{n}\left\|x_i - \sum_{x_j\in B_i} k_{ij}x_j\right\|^2 + \gamma * \sum_{i=1}^{n}\sum_{x_j\in B_i} k_{ij}\left\|x_i - x_j\right\|^2$$

γ is a tuning parameter($\gamma > 0$).

Unsupervised MKL

The cost function can be converted into a convex QP problem w.r.to kernel weitghts μ.

$$J(\mu) = \mu^T \left(\sum_{t=1}^{m} \sum_{i=1}^{n} k_{t,i} k_{t,i}^T \circ d_i d_i^T \circ P \right)^T \mu + z^T \mu$$

where $[z]_t = \sum_{i=1}^{n} (2\gamma v_i \circ d_i - 2p_i \circ d_i)^T k_{t,i}$, $P = X^T X$, and $k_{t,i} = \left[k^t(x_i, x_1), \ldots, k^t(x_i, x_n) \right]^T$ is the i^{th} column of the t^{th} kernel matrix. p and v are columns of P and M corresponding to x_i respectively. M is defined as

$$[M]_{ij} = x_i^T x_i + x_j^T x_j - 2x_i^T x_j$$

Unsupervised MKL

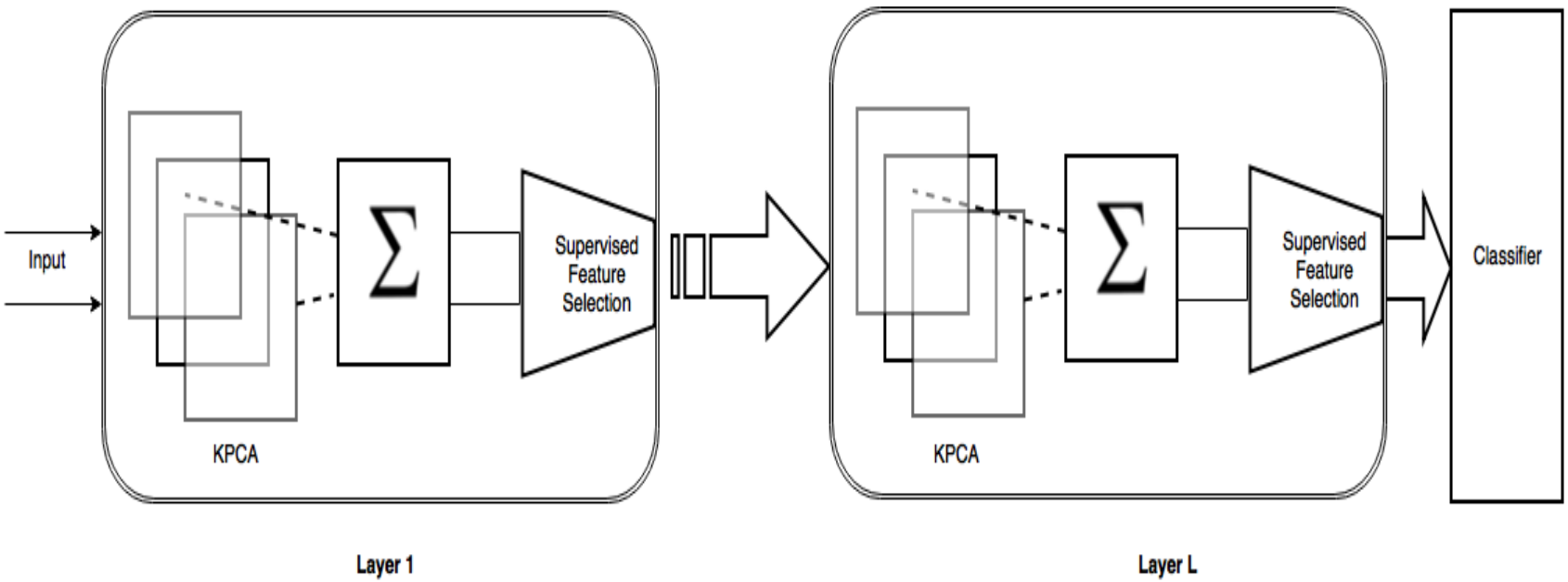


Figure : ML-MKL architecture

Unsupervised MKL

Algorithm 1: ML-MKL Algorithm

Input: data X, true labels y, number of layers L, base kernels for each layer $K_{base}^{(l)} = \{k_1^{(l)}, k_2^{(l)}, k_m^{(l)}\}$, N_B, γ;

Output: kernel weights μ^l for each layer, predicted labels;

1.Initialize $[M]_{ij} = x_i^T x_i + x_j^T x_j - 2x_i^T x_j$, $\mathbf{D} = d_1, d_2, \ldots, d_n$ as row vectors, where $d_i = \{1 \text{ if } x_j \in B_i \text{ else } 0 \forall x_j \in X\}$, $\mu = \frac{1}{m}$, $\mathbf{P} = X^T X$;

2.**for** *each layer l* **do**

a. $\mathbf{W} = \sum_{t=1}^{m} \sum_{i=1}^{n} k_{t,i}^{(l)} k_{t,i}^{(l)^T} \circ d_i d_i^T \circ P$

b. $[\mathbf{z}]_t^l = \sum_{i=1}^{n} (2\gamma v_i \circ d_i - 2p_i \circ d_i)^T k_{t,i}^{(l)}$

c. $\boldsymbol{\mu}^{*^l} = \mu^{l^T} W \mu^l + z^{l^T} \mu^l$

d. $\mathbf{K}_{new} = \sum_{t=1}^{m} \mu_t^l * K_t^{(l)}$

e. extract principal components with $\mathbf{K}_{new}$

f. select most informative features for layer $l(X_{new})$

g. $\mathbf{P} = X_{new}^T X_{new}$

end

3.Give the final set of features to any classifier;

Experimental Results

Dataset	**Loss in Percentage**						
	$\mathrm{SVM}_{\mathrm{RBF}}$	$\mathrm{SVM}_{\mathrm{Poly}}$	NNet	DBN-3	SAA-3	DBN-1	**ML-MKL**
back-rand	14.58	16.62	20.04	**6.73**	11.28	9.80	8.43±0.088
back-image	22.61	24.01	27.41	16.31	23.00	**16.15**	20.92±0.092
rot-back-image	55.18	56.41	62.16	**47.39**	51.93	52.21	51.21±0.811
rect-image	24.04	24.05	33.20	23.69	24.05	**22.50**	22.88±0.124

Table : Experimental Results of ML-MKL.

Experimental Results

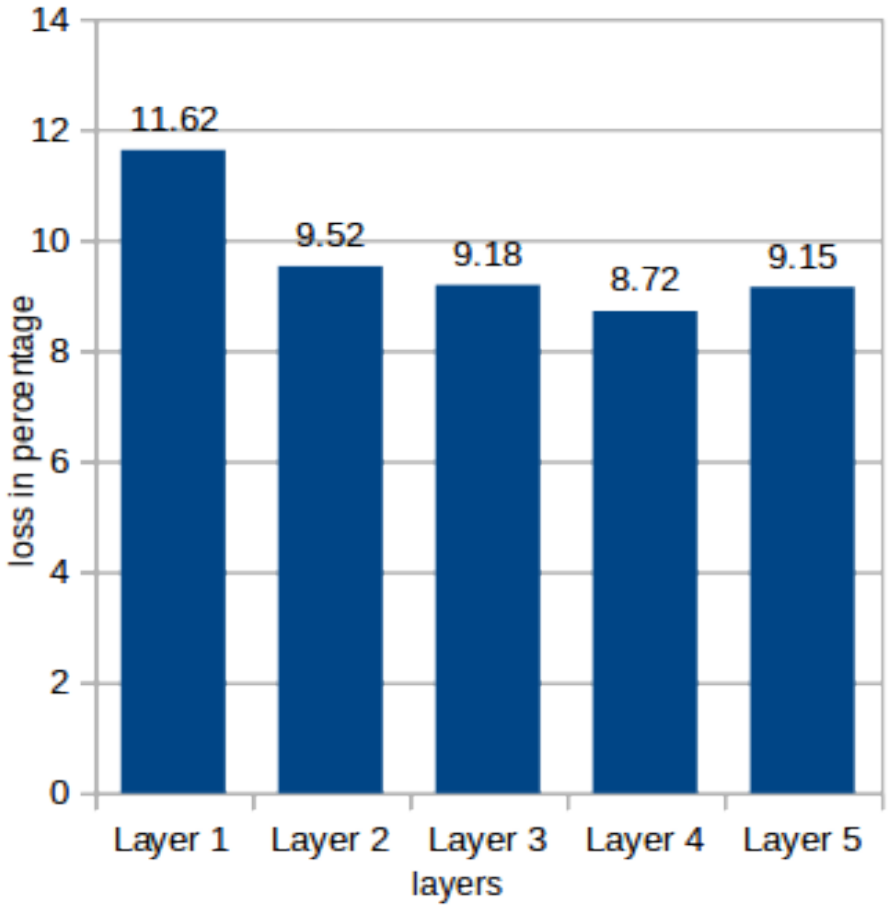


Figure : classifier performance on *mnist-back-rand* dataset in each layer.

Experimental Results

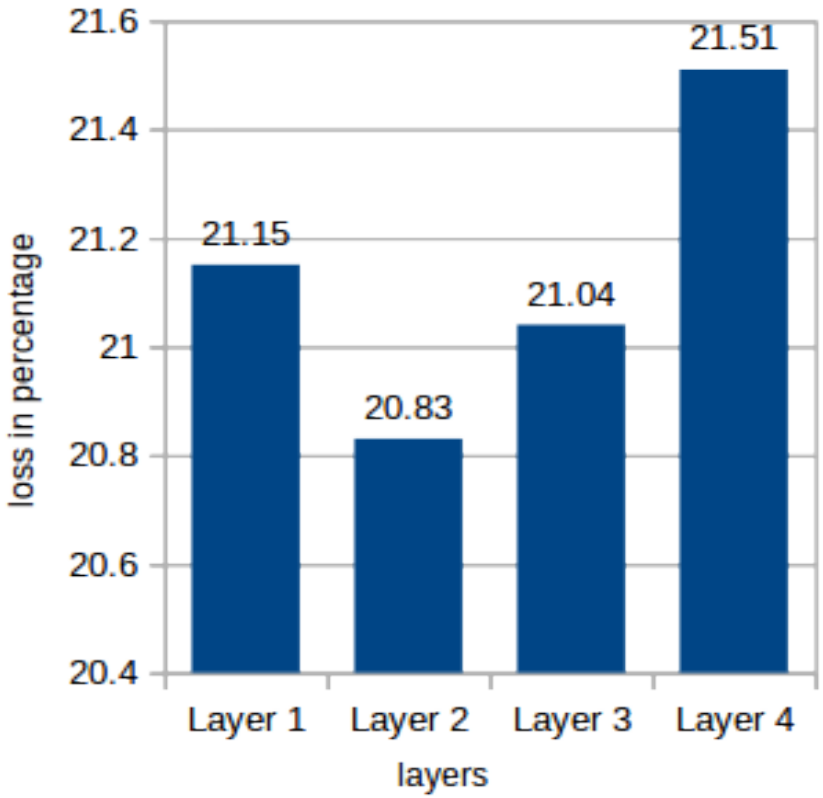


Figure : classifier performance on *mnist-back-image* dataset in each layer.

Experimental Results

Layers	**Loss in Percentage**							
	k_1	k_2	k_3	k_4	k_5	k_6	k_7	K_{conv}
Layer 1	11.46	9.92	10.06	10.04	10.1	9.87	10.06	11.62
Layer 2	10.36	9.87	9.90	9.89	9.92	9.82	9.81	9.52
Layer 3	9.46	11.07	10.33	9.85	9.70	9.56	9.01	9.18
Layer 4	9.26	9.06	8.95	8.96	8.93	8.72	9.00	**8.72**
Layer 5	9.18	9.07	9.02	9.20	9.22	9.36	9.47	9.15

Table : Individual kernels performance evaluation for *mnist-back-rand* dataset.

Experimental Results

Layers	**Loss in Percentage**					
	k_1	k_2	k_3	k_4	k_5	K_{conv}
Layer 1	20.91	21.82	21.82	21.83	21.77	21.15
Layer 2	21.27	21.00	21.09	21.04	21.04	**20.83**
Layer 3	20.95	21.18	21.02	20.99	21.05	21.03
Layer 4	21.06	21.36	21.42	21.64	21.66	21.51

Table : Individual kernels performance evaluation for *mnist-back-image* dataset.

Kernel Fisher Discriminant Analysis

FDA: Aims to find a set of features that discriminates the classes very well(works in input space).
KFDA: non-linear generalization of FDA(works in RKHS).

Let $X_1 = \{x_1^1, \ldots, x_{n_1}^1\}$ and $X_2 = \{x_1^2, \ldots, x_{n_1}^2\}$ be data samples from two classes (class 1 and class 2). KFDA find the directions f which maximizes the cost function

$$\mathcal{J}(f) = \frac{f^T S_B^\phi f}{f^T S_W^\phi f}$$

Kernel Fisher Discriminant Analysis

S_B^ϕ and S_W^ϕ are the between and within class scatter matrices respectively

$$S_B^\phi = (m_1^\phi - m_2^\phi)(m_1^\phi - m_2^\phi)^T$$

$$S_W^\phi = \sum_{i=1,2} \sum_{x \in X_i} (\phi(x) - m_i^\phi)(\phi(x) - m_i^\phi)^T$$

where $m_i^\phi = \frac{1}{n_i} \sum_{j=1}^{n_i} \phi(x_j^i)$.

Intuition

maximizing $\mathcal{J}(f)$ is equivalent to finding a direction f which maximizes the separation of the two classes while minimizing the within class variance.

Kernel Fisher Discriminant Analysis

we have

$$f = \sum_{i=1}^{n} \alpha_i \phi(x_i)$$

applying it in $\mathcal{J}(f)$, we get

$$f^T S_B^\phi f = \alpha^T M \alpha$$

$$f^T S_W^\phi f = \alpha^T N \alpha$$

where $M = (M_1 - M_2)(M_1 - M_2)^T$ and $(M_i)_j = \frac{1}{n_i} \sum_{k=1}^{n_i} k(x_j, x_k^i)$.

Kernel Fisher Discriminant Analysis

$N = \sum_{i=1,2} K_i(I - \mathbf{1}_{n_i})K_i^T$, K_i is an $n \times n_i$ matrix with entries $(K_i)_{nm} = k(x_n, x_m^i)$(kernel matrix for class i)
I is the identity matrix.
$\mathbf{1}_{n_i}$ is the matrix with with all entries $\frac{1}{n_i}$

Then the cost function can be formulated in terms of α

$$\mathcal{J}(\alpha) = \frac{\alpha^T M \alpha}{\alpha^T N \alpha}$$

Kernel Fisher Discriminant Analysis

Dataset	**Loss in Percentage**						
	$\mathrm{SVM}_{\mathrm{RBF}}$	$\mathrm{SVM}_{\mathrm{Poly}}$	NNet	DBN-3	SAA-3	DBN-1	**KFDA**
rect-image	24.04	24.05	33.20	23.69	24.05	22.50	**21.96**
convex	19.13	19.82	32.25	19.92	**18.41**	18.63	19.02

Table : Experimental Results of KFDA with multi-layer kernels.

Multi-layer Kernel on Structured Learning Problems

Objective : learning a function

$$h : \mathcal{X} \longrightarrow \mathcal{Y}$$

where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space(structured). To learn h, we assume that a training sample of input-output pairs

$$S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$$

is available and drawn i.i.d from a distribution P(X,Y).

We find an $h \in \mathcal{H}$ that minimizes the empirical risk(ERM)

$$R_s^{\triangle} = \frac{1}{m} \sum_{i=1}^{m} \triangle(y_i, h(x_i))$$

StructSVM : Problem Formulation

Here $\triangle(y, \overline{y})$ denotes the **loss** associated with predicting $\overline{y}$ when y is the correct output. We assume $\triangle(y, y) = 0$ and

$$\triangle(y, \overline{y}) \geq 0 \text{ for } y \neq \overline{y}$$

Method

StructSVM selects an $h \in \mathcal{H}$ that minimizes a regularized empirical risk on S. The general idea here is to learn a discriminant function $f : \mathcal{X} \times \mathcal{Y} \longrightarrow \mathbb{R}$ over input/output pairs from which one derives a prediction by maximizing f over all $y \in \mathcal{Y}$ for a given input x.

$$h_w(x) = \arg\max_{y \in \mathcal{Y}} f_w(x, y)$$

We assume $f_w(x, y)$ takes a linear form

$$f_w(x, y) = \langle w, \Psi(x, y) \rangle$$

StructSVM : Problem Formulation

Here $w \in \mathbb{R}^N$ is a parameter vector and $\Psi(x, y)$ is a **combined feature space** relating input x and output y.

Inuitive picture for $f_w(x, y)$

We can think of $f_w(x, y)$ as a compatibility function that measures how well the output y matches the given input x.

$\Psi(x, y)$ has to be defined specific to problems(Natural Language Parsing(NLP), Protein Sequence Alignment, Image Segmentation etc).

StructSVM : Problem Formulation

1. **Margin Rescaling**(MR)

In MR the position of the hinge is adapted keeping the slope fixed.

$$\triangle_{MR}(y, h_w(x)) = \max_{\overline{y} \in \mathcal{Y}} \{\triangle(y, \overline{y}) - \langle w, \Psi(x, y) \rangle + \langle w, \Psi(x, \overline{y}) \rangle\}$$

$$\geq \triangle(y, h_w(x))$$

1. **Slack Rescaling**(SR)

In SR the slope is adjusted while keeping the position of the hinge fixed.

$$\triangle_{SR}(y, h_w(x)) = \max_{\overline{y} \in \mathcal{Y}} \{\triangle(y, \overline{y})(1 - \langle w, \Psi(x, y) \rangle + \langle w, \Psi(x, \overline{y}) \rangle)\}$$

$$\geq \triangle(y, h_w(x))$$

StructSVM : Problem Formulation

n-Slack StructSVM with Margin Rescaling

$$\min_{w,\xi \geq 0} \quad \frac{1}{2}\|w\|^2 + \frac{C}{m}\sum_{i=1}^{m}\xi_i$$

$$\text{s.t } \forall \overline{y_1} \in \mathcal{Y} : \; w^T[\Psi(x_1, y_1) - \Psi(x_1, \overline{y_1})] \geq \triangle(y_1, \overline{y_1}) - \xi_1$$

$$\vdots$$

$$\text{s.t } \forall \overline{y_m} \in \mathcal{Y} : \; w^T[\Psi(x_m, y_m) - \Psi(x_m, \overline{y_m})] \geq \triangle(y_m, \overline{y_m}) - \xi_m$$

StructSVM : Problem Formulation

n-Slack StructSVM with Slack Rescaling

$$\min_{w,\xi \geq 0} \quad \frac{1}{2}\|w\|^2 + \frac{C}{m}\sum_{i=1}^{m}\xi_i$$

$$\text{s.t } \forall \overline{y_1} \in \mathcal{Y} : \; w^T[\Psi(x_1, y_1) - \Psi(x_1, \overline{y_1})] \geq 1 - \frac{\xi_1}{\triangle(y_1, \overline{y_1})}$$

$$\vdots$$

$$\text{s.t } \forall \overline{y_m} \in \mathcal{Y} : \; w^T[\Psi(x_m, y_m) - \Psi(x_m, \overline{y_m})] \geq 1 - \frac{\xi_m}{\triangle(y_m, \overline{y_m})}$$

The constraints states that for each training example, (x_i, y_i) the score $w^T[\Psi(x_i, y_i)$ must be greater than the score $w^T[\Psi(x_i, \overline{y})$ of all outputs $\overline{y} \in \mathcal{Y}$ by a required margin. This margin is 1 in SR and $\triangle(y, \overline{y})$ in MR.

StructSVM : Experimental Results

Dataset	**Arc-Cosine Kernel**	**Other Kernel(best)**
Scene Segentation (multilabel - 6 class)	30.35	30.60
Vehicle Dataset (multiclass - 4 class)	26.48	24.90
Iris Dataset (multiclass - 3 class)	1.67	3.33
Breast Cancer Wiscosin (binary)	0.98	0.98
Synthetic Data (multiclass - 7 class)	33.85	32.55

Conclusion

- Performance of MKMs are comparable with popular deep learning algorithms like DBN, SAA etc.
- Using (unsupervised) MKL in MKMs improves the classifier performance.
- KFDA performs well when using, either a highly non-linear arc-cosine kernel(degree > 1) or a multi-layer arc-cosine kernel with very large number of layers (above 10).
- Multi-layer kernels performs competetive with single layer kernel machines on structured output spaces.

Future Plans

- A thoretical study on the optimal structure of ML-MKL model(number of layers required, number of kernels in each layer).
- Fine-tuning mechanisms for ML-MKL model.
- Multi-layer kernel can be tested on complex structured output prediction problems like parse tree prediction, protein segment alignment prediction etc.

Important References


Y. Cho, L.K. Saul, Kernel Methods for Deep Learning
Advances in Neural Information Processing Systems(NIPS) 22, 342 – 350, 2009.


J. Zhuang, Jialei Wang, Steven C.H Hoi, Xiangyang Lan, Unsupervised Multiple Kernel Learning.
Journal of Machine Learning Research(JMLR) Volume 20, 129–144, 2011.


Sebastian Mika, Gunnar Rätsch, Jason Weston, Bernhard Schölkopf, and Klaus-Robert Müller, Fisher Discriminant Analysis With Kernels.
Journel of Neural Networks for Signal Processing, pp 41–48, 1999.


Yoshua Bengio and Yann LeCun, Scaling learning algorithms towards AI. in Bottou, L. and Chapelle, O. and DeCoste, D. and Weston, J. (Eds)
Large-Scale Kernel Machines, MIT Press, 2007.


Yoshua Bengio, Aaron Courville and Pascal Vincent, Representation Learning: A Review and New Perspectives.
IEEE Transactions on Pattern Analysis and Machine Intelligence(TPAMI) Volume 35, 1798–1828, August 2013.


T. Joachims, T. Finely, C. John Nu, Cutting-Plane Training on Structural SVMs
Journal of Machine Learning 77(1), 27 – 59, 2009.

List of papers based on thesis

1. Akhil P M, Asharaf S, Sumitra S, *Unsupervisd MKL in Multi-layer Kernel Machines*(paper under preparation).


Thank
You