CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 16: Kernelization

Lecturer: C. V. Jawahar *Date: 7, Oct 2019*

16.99 Kernels and Kernelization

We know Kernels as a powerful tool to bring in "non-linearility" into the linear algorithms. With the help of kernels, we combine:

- the expressive power due to the non-linearity
- the elegance and robustness of the linear methods.

This led to bringing kernels into a wide variety of classical/popular linear algorithms, in the past. Today we see two such examples.

Objective of this lecture is to appreciate the notion of kernels, beyond SVMs.

16.100 Kernel Perceptron

Basic Perceptron

We know the perceotron algorithm from the past lectures, as a simple algorithm that can learn a separating hyperplane (linear function). Let us revisit this algorithm today, and see how to "kernalize" this algorithm.

Ver 1 We start with a version of the perceptron algorithm, that we know. It is possibly written down in a form that helps us in kernalizing. Here we assume that the weight vector is initialized as zero vector $\mathbf{0}$. Also, we assume that the learning rate is 1.0. We assume that the samples are linearly seperable.

Note that the representation $< \mathbf{w}, \mathbf{x_i} >$ is same as $\mathbf{w}^T\mathbf{x}_i$

Algorithm 1 PerceptronPrimal($\{(\mathbf{x}_i, y_i)\}$ for $i = 1, 2, \cdots, N$)

Require: $\exists \mathbf{w} \ni y_i = sign(\langle \mathbf{w}, \mathbf{x}_i \rangle)$ for $i = 1, 2, \cdots, N$

```
w = 0
repeat
    for i = 1 to N do
        if y_i ⟨w, x_i⟩ ≤ 0 then
            w ← w + y_i x_i
        end if
    end for
until no sample is misclassified
```

Figure 16.11: Our familiar perceptron algorithm

We can make a simple observation here. The final weight vector is a linear combination of the training data.

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$$

Note that we started with $\mathbf{w}$ as zero and it was changed only as $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$.

What is α_i here? It is related to the number of times the sample was used to modify the solution. (or number of times it entered the inside the condition).

- Q: What does it mean if α_i is zero?
- Q: Is α_i always non-negative?
- Q: Where is y_i absorbed in the final equation?
- Q: If we had a learning rate η in the original learning algorithm, does it change anything? Does the assumption of $\eta = 1.0$ constraining in any form?

Ver 2 If we appreciate the fact that $\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$, we can re-write $\mathbf{w}^T\mathbf{x}_j$ as

$$\sum_{i=1}^{N} \alpha_i \mathbf{x}_i^T \mathbf{x}_j = \sum_{i=1}^{N} \alpha_i G_{ij}$$

where G is Gram matrix, with elements defined as $G_{ij} = \mathbf{x}_i^T\mathbf{x}_j$

Algorithm 2 PerceptronDual($\{(\mathbf{x}_i, y_i)\}$ for $i = 1, 2, \cdots, N$)

Require: $\exists \alpha_j \ni y_i = sign(\langle \sum_{j=1}^{N} \alpha_j \mathbf{x}_j, \mathbf{x}_i \rangle)$ for $i, j \in \{1, 2, \cdots, N\}$

```
α_i = 0 for i = 1, 2, ··· , N
compute the Gram Matrix G_ij = ⟨x_i, x_j⟩
repeat
    for i = 1 to N do
        if y_i Σ_{j=1}^{N} α_j G_ij ≤ 0 then
            α_i ← α_i + y_i
        end if
    end for
until no sample is misclassified
```

Figure 16.12: A variant of the perceptron algorithm. Note the equivalence

Kernel Perceptron

We have now seen a linear algorithm that uses only dot products on the input data. This is the right time to think of Kernalizatin!!.

Ver 3: Kernel Perceptron Now that we have expressed the perceptron algorithm in terms of dot products, we can re-write the perceptron algorithm as "Kernel perceptron" with Gram matrix replaced by a Kernel matrix K, with (i, j) element defines as:

$$K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

Algorithm 3 KernelPerceptron($\{(\mathbf{x}_i, y_i)\}$ for $i = 1, 2, \cdots, N$, $\kappa(.,.)$)

$\alpha_i = 0$ for $i = 1, 2, \cdots, N$
compute the Kernel Matrix $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$
repeat
  for $i = 1$ to N **do**
    if $y_i \sum_{j=1}^{N} \alpha_j \mathbf{K}_{ij} \leq 0$ **then**
      $\alpha_i \leftarrow \alpha_i + y_i$
    end if
  end for
until no sample is misclassified

Figure 16.13: Kernel Perceptron Algorithm

Inference/Testing How do we evaluate the class/label of a new sample $\mathbf{x}$? In the classical case, we did it as

$$sign(\mathbf{w}^T \mathbf{x}).$$

In the kernelized version, we can do it as

$$sign(\sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}))$$

Discussions

- Can K-Perceptron be used for linearly non-separable data?
- What are the computational advantages/disadvantages of this algorithm?

16.101 Kernel PCA

Let us now kernalize another of our familiar algorithm – PCA. The trick for kernalizing the PCA algorithm is some what different from that of the perceptron algorithm. (It is not just re-writing the pseudo-codes!)

We make a simple observation that eigen vectors can be expressed as a linear combination of the input samples. This makes the PCA algorithm Kernalization friendly. Also note that we do not need the eigen vectors. We only need the dot product of any sample with the eigen vectors.

Notation: We use the following notations:

- M is the number of samples. (should have been N itself)
- C is the covariance matrix. We did not use Σ to avoid any potential confusion with the symbol for summation ($\sum$).
- $\mathbf{v}$ (and $\mathbf{V}$) is the eigen vector (EV).

16.101.1 Preliminaries

We start with the assumption that the data is centered, or the mean of the data is zero. This is not very restrictive. We also remove it at a later stage. Then we know that the covariance matrix is:

$$C = \frac{1}{M} \sum_{i=1}^{M} \mathbf{x_i} \mathbf{x_i^T} \text{ and } C\mathbf{v} = \lambda \mathbf{v}$$

EV is a linear combination of samples We know that $C\mathbf{v} = \lambda \mathbf{v}$. Or else,

$$\mathbf{v} = \frac{1}{\lambda M} \sum_{i=1}^{M} \mathbf{x_i} \mathbf{x_i^T} \mathbf{v} = \frac{1}{\lambda M} \sum_{i=1}^{M} \mathbf{x_i^T} \mathbf{v} \mathbf{x}_i$$
$$= \frac{1}{\lambda M} \sum_{i=1}^{M} \beta_i \mathbf{x}_i = \sum_{i=1}^{M} \alpha_i \mathbf{x}_i$$

Or in the simplest term, eigen vector (EV) is a linear combination of input samples.

Covariance Matrix in Feature Space In Kernel PCA, PCA is done in the feature space. For this, the data needs to be mapped into another space, i.e.,

$$\mathbf{x} \rightarrow \phi(\mathbf{x}).$$

Then assuming that data is centered, covariance matrix can be computed as:

$$C = \frac{1}{M} \sum_{j=1}^{M} \phi(\mathbf{x_j}) \phi(\mathbf{x_j})^T \tag{16.34}$$

16.101.2 EV as Linear Combination of Samples

For KPCA, we need to solve:

$$C\mathbf{V} = \lambda\mathbf{V} \tag{16.35}$$

- We do not want to work with $\phi(\cdot)$ explicitly. We can use the kernel $\kappa(\cdot,\cdot)$ and curcumvent this requriement.
- If $\phi(\cdot)$ maps to an infinite dimension, we will have to work with $\infty \times \infty$ matrices. That is impractical.
- Note that even in such cases, we will have ONLY $\leq M$ eigen vectors for C.

We can rewrite it as

$$\mathbf{V} = \frac{1}{\lambda}C\mathbf{V} = \frac{1}{\lambda M}\sum_{i=1}^{M}\phi(\mathbf{x_i})\phi(\mathbf{x_i})^{\mathbf{T}}\mathbf{V}$$

$$= \frac{1}{M}\sum_{i=1}^{M}(\phi(\mathbf{x_i})^{\mathbf{T}}\mathbf{V})\phi(\mathbf{x_i}) = \sum_i \alpha_i\phi(\mathbf{x_i})$$

Or

$$\mathbf{V} = \sum_{i=1}^{M}\alpha_i\phi(\mathbf{x_i})$$

Note that the upper case $\mathbf{V}$ is the eigen vector in the new feature space.

We do not want to compute the covariance matrix in the new feature space. We also do not want to compute this eigen vactor. We want only the projection of new samples into the new sub-spaces defined by the eigen vectors of the new covariance space.

16.101.3 Computing α

Taking a dot product with $\phi(\mathbf{x_k})$ on both sides of Equation 16.35: $(C\mathbf{V} = \lambda\mathbf{V})$

$$\phi(\mathbf{x_k}) \cdot C\mathbf{V} = \lambda\phi(\mathbf{x_k}) \cdot \mathbf{V} \;\; \forall k \tag{16.36}$$

Substituting for C, $\mathbf{V}$ and rearranging the summations,

$$\sum_{j=1}^{M}(\phi(\mathbf{x_k}) \cdot \phi(\mathbf{x_j}))\sum_{i=1}^{M}\alpha_i(\phi(\mathbf{x_j}) \cdot \phi(\mathbf{x_i})) \tag{16.37}$$

$$= \lambda M\sum_{i=1}^{M}\alpha_i(\phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_k}))\forall k \tag{16.38}$$

Note we can do it for all $\mathbf{x}_k$, leading to M similar equations. We can compactly represent these equations in matrix form with kernel matrix K (our familiar one).

Defining an $M \times M$ matrix K by

$$K_{ij} = (\phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_i})) = \kappa(\mathbf{x_i}, \mathbf{x_j})$$

We can compactly write these equations as:

$$M\lambda K\alpha = K^2\alpha$$
$$M\lambda\alpha = K\alpha$$

Thus the vectors α are the eigen vectors of K. Here, α is a vector of dimension M. K is a matrix of size $M \times M$, and it has M eigen vectors α. Let α^k be the eigen vector corresponding to λ_k.

The M eigen vectors of K, α^k lead to M eigen vectors of C.

The resulting set of eigenvectors $\mathbf{V}^k$ can now be computed as:

$$\mathbf{V^k} = \sum_{i=1}^{M}\alpha_i^k\phi(\mathbf{x_i}).$$

16.101.4 Projecting a New Sample

And the projection of a sample $\phi(\mathbf{x})$ onto this principal component can be evaluated as:

$$(\mathbf{V^k} \cdot \phi(\mathbf{x})) = \sum_{i=1}^{M}\alpha_i^k(\phi(\mathbf{x_i}) \cdot \phi(\mathbf{x}))$$
$$= \sum_{i=1}^{M}\alpha_i^k\kappa(\mathbf{x_i}, \mathbf{x})$$

16.101.5 Centering

We started with the assumption that the data is centered. In practice it is not. How do we center the data then?

Let $\hat{\phi}(\mathbf{x_i})$ be the centered version of $\phi(\mathbf{x_i})$. The (i,j) th element of the Kernel matrix (corresponding to centered data) is:

$$\hat{K}_{ij} = \hat{\phi}(\mathbf{x_i})^T\hat{\phi}(\mathbf{x_j}) \tag{16.39}$$

$$\hat{K}_{ij} = (\phi(\mathbf{x_i}) - \mu)^T(\phi(\mathbf{x_j}) - \mu)$$

Where $\mu = \frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x_n})$

$$\hat{K}_{ij} = (\phi(\mathbf{x_i}) - \mu)^T(\phi(\mathbf{x_j}) - \mu)$$
$$= (\phi(\mathbf{x_i}) - \frac{1}{N}\sum_{m=1}^{N}\phi(\mathbf{x_m}))^T(\phi(\mathbf{x_j}) - \frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x_n}))$$
$$= K(\mathbf{x_i}, \mathbf{x_j}) - \frac{1}{N}\sum_{n=1}^{N}K(\mathbf{x_n}, \mathbf{x_i}) - \frac{1}{N}\sum_{m=1}^{N}K(\mathbf{x_m}, \mathbf{x_j})$$
$$+ \frac{1}{N^2}\sum_{m,n=1}K(\mathbf{x_m}, \mathbf{x_n})$$
$$\hat{K}_{ij} = K_{ij} - \frac{1}{N}\sum_{n=1}^{N}K_{ni} - \frac{1}{N}\sum_{m=1}^{N}K_{mj} + \frac{1}{N^2}\sum_{mn}K_{mn}$$

16.101.6 Computational Procedure

1. First compute the kernel matrix K.
2. Then compute the kernel matrix $\hat{K}$ corresponding to the centered data.
3. Compute the eigen values and eigen vectors (α) of $\hat{K}$
4. Use α and the kernel function, evaluate the projection.

x

16.101.6.1 Kernel LDA and Beyond (*)

The kernel trick is not limited to PCA. You can also kernelaize algorithms like LDA. Please see:

- B. Scholkof, A Smola and K Muller, Nonlinear Component Analysis as a Kernel Eigenvalue Problem for KPCA
- S. Mika et al, Fisher Discriminant Analysis with Kernels for KLDA

16.102 Application: Back to Eigen Face

We had discussed PCA being a useful cue for representing faces in the form of eigen faces. The dimensionality reduction techniques like LDA (or Fisher discriminant) and KPCA are also used in very similar manner.

Q: Write pseudo codes for fisher and KPCA faces.

"Face recognition using kernel Methods" (NIPS 2001) is provided a nice comparison of these methods on two benchmarks. (see more details)

x

Method	Reduced Space	Error Rate (%)
ICA	40	6.25 (25/400)
Eigenface	30	2.75 (11/400)
Fisherface	14	1.50 (6/400)
Kernel Eigenface, d=2	50	2.50 (10/400)
Kernel Eigenface, d=3	50	2.00 (8/400)
Kernel Fisherface (P)	14	1.25 (5/400)
Kernel Fisherface (G)	14	1.25 (5/400)

Method	Reduced Space	Error Rate (%)
ICA	30	29.09 (48/165)
Eigenface	30	28.48 (47/165)
Fisherface	14	8.48 (14/165)
Kernel Eigenface, d=2	80	27.27 (45/165)
Kernel Eigenface, d=3	60	24.24 (40/165)
Kernel Fisherface (P)	14	6.67 (11/165)
Kernel Fisherface (G)	14	6.06 (10/165)