National Technical University of Ukraine
“Igor Sikorsky Kyiv Polytechnic Institute”
Institute of Physics and Technology

Lecture 15

Objects and Systems Identification Methods. Kernels

Dmytro Progonov,
PhD, Associate Professor,
Department Of Physics and Information Security Systems

Content

- Problem statement;
- Kernel functions:
  - Mercer (positive defined) kernels;
  - Kernels derived from probabilistic generative models;
  - Kernel for building generative models;
- Kernel machines;
- The kernel trick;
- Support vector machines.

Problem statement

So far *we have been assuming that each object* that we wish to classify or cluster or process in anyway *can be represented as a fixed-size feature vector*, typically of the form $\mathbf{x}_i \in \mathbb{R}^D$. However, for certain kinds of objects, *it is not clear how to best represent them as fixed-size feature vectors*.

One approach to such problem is to define a generative model for the data, and use the inferred latent representation and/or the parameters of the models as features, and then to plug-in theirs to standard methods.

Another approach is to assume that we have some way of measuring the similarity between objects, that does not require preprocessing them into feature vector format – we will call its as ***kernel function***.

Kernel functions examples

The ***squared exponential kernel*** or ***Gaussian kernel***:

$$\kappa(\mathbf{x}, \acute{\mathbf{x}}) = \exp\left[-\frac{1}{2}(\mathbf{x} - \acute{\mathbf{x}})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \acute{\mathbf{x}})\right]$$

If $\boldsymbol{\Sigma}$ is spherical, we get the isotropic kernel (***radial basis function***):

$$\kappa(\mathbf{x}, \acute{\mathbf{x}}) = \exp\left[-\frac{\|\mathbf{x} - \acute{\mathbf{x}}\|^2}{2\sigma^2}\right]$$

Cosine similarity for comparing documents:

$$\kappa(\mathbf{x}_i, \acute{\mathbf{x}}_i) = \frac{\mathbf{x}_i{}^T \acute{\mathbf{x}}_i}{\|\mathbf{x}_i\|_2 \|\acute{\mathbf{x}}_i\|_2}$$

Linear kernel:

$$\kappa(\mathbf{x}, \acute{\mathbf{x}}) = \mathbf{x}^T \acute{\mathbf{x}}$$

Matern kernel:

$$\kappa(r) = \frac{2^{1-v}}{\Gamma(v)}\left(\frac{r\sqrt{2v}}{l}\right)^v K_v\left(\frac{r\sqrt{2v}}{l}\right), r = \|\mathbf{x} - \acute{\mathbf{x}}\|, v, l > 0$$

Mercer (positive defined) kernels (1/2)

Some data processing methods require that the kernel function satisfy requirement that the ***Gram matrix***, defined by

$$\mathbf{K} = \begin{pmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

be positive defined for any set of inputs $\{\mathbf{x}_i\}_{i=1}^{N}$. Such kernels are ***Mercer (positive defined) kernels***.

Mercer (positive defined) kernels (2/2)

Mercer's theorem: if the Gram matrix is positive defined, we can compute an eigenvector decomposition of it as follows:

$$\mathbf{K} = \mathbf{U}^T \mathbf{\Lambda} \mathbf{U}$$

where $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues $\lambda_i > 0$. Consider an elements of $\mathbf{K}$:

$$k_{ij} = \left(\mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:,i}\right)^T \left(\mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:,j}\right)$$

Let us define $\phi(\mathbf{x}_i) = \mathbf{\Lambda}^{\frac{1}{2}} \mathbf{U}_{:,i}$. Then we can write:

$$k_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

In general, if the kernel is Mercer, then there exist a function ϕ mapping $\mathbf{x} \in \mathcal{X}$ to $\mathbb{R}^D$ such that

$$\kappa(\mathbf{x}, \acute{\mathbf{x}}) = \phi(\mathbf{x})^T \phi(\acute{\mathbf{x}})$$

Kernels derived from probabilistic generative models (1/2)

The ***probability product kernel*** is defined as follows:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \int p(\mathbf{x}|\mathbf{x}_i)^{\rho} p(\mathbf{x}|\mathbf{x}_j)^{\rho} d\mathbf{x}, \rho > 0$$

where $p(\mathbf{x}|\mathbf{x}_i)$ is often approximated by $p\left(\mathbf{x}|\widehat{\boldsymbol{\theta}}(\mathbf{x}_i)\right)$, $\widehat{\boldsymbol{\theta}}(\mathbf{x}_i)$ is a parameter estimate computed using a single data vector.

For example, suppose $p(\mathbf{x}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \sigma^2\mathbf{I})$ ($\sigma^2 = const$). If $\rho = 1$, and we use $\widehat{\boldsymbol{\mu}}(\mathbf{x}_i) = \mathbf{x}_i$ and $\widehat{\boldsymbol{\mu}}(\mathbf{x}_j) = \mathbf{x}_j$, we find that:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{(4\pi\sigma^2)^{D/2}} \exp\left[-\frac{1}{4\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right]$$

Kernels derived from probabilistic generative models (2/2)

A more efficient way to use generative models to define kernels is to use a ***Fisher kernel*** which is defined as follows:

$$\kappa(\mathbf{x}, \hat{\mathbf{x}}) = \mathbf{s}(\mathbf{x})^T \mathbf{I}^{-1} \mathbf{s}(\hat{\mathbf{x}})$$

where $\mathbf{s}$ is the gradient of the log likelihood, or score function, evaluated at the MLE $\widehat{\boldsymbol{\theta}}$:

$$\mathbf{s}(\mathbf{x}) \triangleq \nabla_{\boldsymbol{\theta}} \log[p(\mathbf{x}|\boldsymbol{\theta})|_{\widehat{\boldsymbol{\theta}}}]$$

and $\mathbf{I}$ is the Fisher information matrix:

$$\mathbf{I} = -\nabla^2 \log[p(\mathbf{x}|\boldsymbol{\theta})|_{\widehat{\boldsymbol{\theta}}}]$$

Kernel for building generative models

A ***smoothing kernel*** satisfies the following properties:

$$\int \kappa(x)dx = 1; \int x\kappa(x)dx = 0; \int x^2\kappa(x)dx > 0$$

A kernel with compact support is the ***Epanechnikov kernel***:

$$\kappa(x) \triangleq \frac{3}{4}(1 - x^2)\mathbb{I}(|x| < 1)$$

The Epanechnikov kernel is not differentiable at the boundary of its support. An alternative is the ***tri-cube kernel***:

$$\kappa(x) \triangleq \frac{70}{81}(1 - x^3)^3\mathbb{I}(|x| < 1)$$

Kernel machines

We define a kernel machine to be a GLM where the input feature vector has the form:

$$\phi(\mathbf{x}) = [\kappa(\mathbf{x}, \boldsymbol{\mu}_1), \kappa(\mathbf{x}, \boldsymbol{\mu}_2), \cdots, \kappa(\mathbf{x}, \boldsymbol{\mu}_K)]$$

where $\boldsymbol{\mu}_k \in \mathcal{X}$ are a set of K ***centroids***. This input feature vector also are called a ***kernelized feature vector***. Note that if κ is an RBF kernel, this is called an ***RBF network***.

We can use the kernelized feature vector for logistic regression by defining $p(y|\mathbf{x}, \boldsymbol{\theta}) = Ber(\mathbf{w}^T \phi(\mathbf{x}))$. This provides a simple way to define a non-linear decision boundary.

The kernel trick

Rather than defining our feature vector in terms of kernels, $\phi(\mathbf{x}) = [\kappa(\mathbf{x}, \boldsymbol{\mu}_1), \kappa(\mathbf{x}, \boldsymbol{\mu}_2), \cdots, \kappa(\mathbf{x}, \boldsymbol{\mu}_K)]$, we can instead work with the original feature vector $\mathbf{x}$, but modify the algorithm so that it replaces all inner products of the form $\langle \mathbf{x}, \hat{\mathbf{x}} \rangle$ with a call to the kernel function $\kappa(\mathbf{x}, \hat{\mathbf{x}})$. This is called the ***kernel trick***.

Recall that in 1NN (nearest neighbors) classifier, we just need to compute the Euclidean distance of a test vector to all the training points, find the closest one, and look up its label. This can be kernelized by observing that

$$\|\mathbf{x}_i - \acute{\mathbf{x}}_i\|_2^2 = \langle \mathbf{x}_i, \mathbf{x}_i \rangle + \langle \acute{\mathbf{x}}_i, \acute{\mathbf{x}}_i \rangle - 2\langle \mathbf{x}_i, \acute{\mathbf{x}}_i \rangle$$

This allows us to apply the nearest neighbor classifier to structured data objects

Support vector machines (1/3)

Support vector machine (SVM) classifier involves ***three key ingredients***:

The kernel trick is necessary to prevent underfitting, i.e. to ensure that the feature vector is sufficiently rich that a linear classifier can separate the data. If the original features are already high dimensional, it suffices to use a *linear kernel*, which is equivalent to working with the original features.

The ***sparsity*** and ***large margin principles*** are necessary to prevent overfitting, i.e. to ensure that we do not use all basis functions.

Support vector machines (2/3)

Our goal is to derive a discriminant function $f(\mathbf{x})$ which will be linear in the feature space implied by the choice of kernel. Consider a point $\mathbf{x}$ in this induced space:

$$\mathbf{x} = \mathbf{x}_\perp + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

where r is the distance of $\mathbf{x}$ from the decision boundary whose normal vector is $\mathbf{w}$, and $\mathbf{x}_\perp$ is the orthogonal projection of $\mathbf{x}$ onto this boundary. Hence

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = (\mathbf{w}^T\mathbf{x}_\perp + w_0) + r\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|} = (\mathbf{w}^T\mathbf{x}_\perp + w_0) + r\|\mathbf{w}\|$$

Now $0 = f(\mathbf{x}_\perp) = \mathbf{w}^T\mathbf{x}_\perp + w_0$ so $f(\mathbf{x}) = r\|\mathbf{w}\|$ and $r = f(\mathbf{x})/\|\mathbf{w}\|$.

Support vector machines (3/3)

We would like to make this distance $r = f(\mathbf{x})/\|\mathbf{w}\|$ as large as possible. Intuitively, the best line to pick is the one that *maximizes the margin*, i.e. the perpendicular distance to the closest point. In addition, we want to ensure each point is on the correct side of the boundary ($f(\mathbf{x}_i)y_i > 0$). So our objectives become

$$\max_{\mathbf{w},w_0} \min_{i=1:N} \left(\frac{y_i(\mathbf{w}^T\mathbf{x}_i + w_0)}{\|\mathbf{w}\|} \right)$$

Let us define the scale factor such that $f(\mathbf{x}_i)y_i = 1$ for the points that is closest to the decision boundary. Hence we require $f(\mathbf{x}_i)y_i \geq 1$. Maximizing $1/\|\mathbf{w}\|$ is equivalent to minimizing $\|\mathbf{w}\|^2$. Thus we get the new objective

$$\min_{\mathbf{w},w_0} \frac{1}{2}\|\mathbf{w}\|^2, s.t. y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1, i = 1:N$$

Conclusion

- Definition and properties of Kernel functions were considered;
- The kernel trick was shown. Application of kernel trick or building a Kernel machines were presented;
- Core principles of Support vector machines were shown.