**From the CIS520 Machine Learning**

# Lectures: Kernels

On this page... (hide)

## Dual Representations and Kernels

Nearest neighbor methods and Kernel regression shift focus away from features and their parameters to examples and their weights. In many machine learning methods there is a duality between feature weights and example weights. This duality allows us to use the kernel trick to expand the representational power of a model without (much) computational expense. Before we tackle more complex applications of this idea, let's consider standard MAP (a.k.a. ridge) regression from the dual perspective.

### Linear Regression: A dual view

Recall that the model is

$$P(y \mid \mathbf{x}, \mathbf{w}, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\{\frac{-(\mathbf{w}^\top \mathbf{x} - y)^2}{2\sigma^2}\}$$

and the prior is a zero-mean diagonal Gaussian

$$P(\mathbf{w} \mid \lambda) = \prod_j \frac{1}{\lambda\sqrt{2\pi}} \exp\{\frac{-w_j^2}{2\lambda^2}\}$$

The MAP estimate is:

$$\hat{\mathbf{w}}_{MAP} = \arg\max_{\mathbf{w}} \left(\log P(D \mid \mathbf{w}, \sigma) + \log P(\mathbf{w} \mid \lambda)\right) = \arg\min_{\mathbf{w}} \left(\frac{1}{2\sigma^2} \sum_i (\mathbf{w}^\top \mathbf{x}_i - y_i)^2 + \frac{1}{2\lambda^2} \mathbf{w}^\top \mathbf{w}\right)$$

Taking the derivative with respect to $\mathbf{w}$ and setting it to zero, we get:

$$\hat{\mathbf{w}}_{MAP} = \frac{\lambda^2}{\sigma^2} \sum_i (y_i - \hat{\mathbf{w}}_{MAP}^\top \mathbf{x}_i)\mathbf{x}_i = \sum_i \alpha_i \mathbf{x}_i = \mathbf{X}^\top \alpha$$

where

$$\alpha_i = \frac{\lambda^2}{\sigma^2}(y_i - \hat{\mathbf{w}}_{MAP}^\top \mathbf{x}_i).$$

Expressing $\hat{\mathbf{w}}_{MAP}$ as a linear combination of examples is called dual representation. In fact MAP parameters for other linear models with Gaussian priors (i.e. quadratic regularization) can be expressed this way (this can be proved rigorously in very general settings using the Representer Theorem).

## Logistic regression: A dual view

We can express the MAP solution to logistic regression in a similar way. Recall that the model is (assuming $y \in \pm 1$):

$$P(y \mid \mathbf{x}, \mathbf{w}) = \frac{1}{1+\exp\{-y\mathbf{w}^\top \mathbf{x}\}}$$

and the prior is a zero-mean diagonal Gaussian:

$$P(\mathbf{w} \mid \lambda) = \prod_j \frac{1}{\lambda\sqrt{2\pi}} \exp\{\frac{-w_j^2}{2\lambda^2}\}$$

The MAP estimate is:

$$\hat{\mathbf{w}}_{MAP} = \arg\min_{\mathbf{w}} \left(\sum_i \log(1+\exp\{-y_i \mathbf{w}^\top \mathbf{x}_i\}) + \frac{1}{2\lambda^2}\mathbf{w}^\top \mathbf{w}\right)$$

Taking the derivative with respect to $\mathbf{w}$ and setting it to zero, we get:

$$\hat{\mathbf{w}}_{MAP} \; = \; \lambda^2 \sum_i \frac{y_i \mathbf{x}_i \exp\{-y_i \hat{\mathbf{w}}_{MAP}^\top \mathbf{x}_i\}}{1+\exp\{-y_i \hat{\mathbf{w}}_{MAP}^\top \mathbf{x}_i\}} \; = \; \lambda^2 \sum_i y_i \mathbf{x}_i (1 - P(y_i | \mathbf{x}_i, \hat{\mathbf{w}}_{MAP})) \; = \; \sum_i \alpha_i \mathbf{x}_i = \mathbf{X}^\top \alpha$$

where

$$\alpha_i = \lambda^2 y_i (1 - P(y_i|\mathbf{x}_i, \hat{\mathbf{w}}_{MAP})).$$

Again, the MAP solution is a linear combination of examples.

## Solving Regression in the Dual

Plugging in $\mathbf{w} = \mathbf{X}^\top \alpha$ into the linear regression objective, we get

$$\left(\frac{1}{2\sigma^2}(\mathbf{X}\mathbf{w}-\mathbf{y})^\top(\mathbf{X}\mathbf{w}-\mathbf{y}) + \frac{1}{2\lambda^2}\mathbf{w}^\top\mathbf{w}\right) = \left(\frac{1}{2\sigma^2}(\mathbf{X}\mathbf{X}^\top\alpha-\mathbf{y})^\top(\mathbf{X}\mathbf{X}^\top\alpha-\mathbf{y}) + \frac{1}{2\lambda^2}\alpha^\top\mathbf{X}\mathbf{X}^\top\alpha\right)$$

Let’s define the *Gram matrix* $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$ and plug it in above to get:

$$\left(\frac{1}{2\sigma^2}(\mathbf{K}\alpha-\mathbf{y})^\top(\mathbf{K}\alpha-\mathbf{y}) + \frac{1}{2\lambda^2}\alpha^\top\mathbf{K}\alpha\right)$$

Notice that the entire objective is a now function of $\mathbf{y}$, $\mathbf{K}$ and dual weights $\alpha$. All the information about features went into defining $\mathbf{K}$, the Gram matrix, also called the kernel matrix. The kernel matrix is $n$ by $n$ and is defined by the dot products between feature vectors. $\mathbf{K}_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$ The kernel function, defined as the $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$ is what generates the matrix given a set of examples.

Minimizing over $\alpha$, we obtain the solution in terms of the kernel $\mathbf{K}$ and outcomes $\mathbf{y}$:

$$\hat{\alpha}_{MAP} = (\mathbf{K} + \frac{\sigma^2}{\lambda^2}\mathbf{I})^{-1}\mathbf{y}$$

Plugging this solution back into our model prediction for a new instance, $\mathbf{x}$, we get:

$$y_{MAP}(\mathbf{x}) = \mathbf{x}^\top \hat{\mathbf{w}}_{MAP} = \mathbf{x}^\top \mathbf{X}^\top \hat{\alpha}_{MAP} = \mathbf{k}(\mathbf{x})^\top (\mathbf{K} + \frac{\sigma^2}{\lambda^2}\mathbf{I})^{-1}\mathbf{y}$$

where

$$\mathbf{k}(\mathbf{x}) = \mathbf{X}\mathbf{x}$$

is a vector with i'th component equal to $\mathbf{x}_i^\top \mathbf{x}$. Once again, the remarkable fact is that the prediction is a linear combination of training set values $y_i$, where the weighting is only a function of the dot products between feature vectors of a new instance and training instances $\mathbf{k}_i(\mathbf{x}) = \mathbf{x}^\top \mathbf{x}_i$ as well as the dot products between the training instances $\mathbf{K}_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$.

## The Kernel Trick

The dual representation of the prediction function, which does not refer to the features directly, but only through dot-products, allows us to expand the number of features (as long as we can compute the dot-product between them efficiently). By defining a feature mapping $\phi(\mathbf{x}) : \mathbf{R}^m \mapsto \mathbf{R}^M$ we can redefine the kernel function as the dot product in the new expanded feature space of dimension M. $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}') = \sum_{j=1}^M \phi_j(\mathbf{x})\phi_j(\mathbf{x}')$ With this mapping, our weight vector is of dimension M $\hat{\mathbf{w}}_{MAP} = \sum_i \alpha_i \phi(\mathbf{x}_i)$ but the importance of the kernel trick is that we never need to explicitly construct $\hat{\mathbf{w}}_{MAP}$ or $\phi(\mathbf{x}_i)$ for any example, just evaluate the dot product. Luckily, for many feature mappings, the dot product above can be evaluated much more efficiently than O(M). Consider an original feature space of dimension $m = 2$, where $\mathbf{x} = (x_1, x_2)$ and a kernel function defined as:

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^2 = (x_1 x'_1 + x_2 x'_2)^2 = x_1^2 x'^2_1 + 2x_1 x'_1 x_2 x'_2 + x_2^2 x'^2_2 = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)(x'^2_1, \sqrt{2}x'_1 x'_2, x'^2_2)^\top$$

This kernel function can be expressed as a dot product of a feature mapping:

$$\phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1 x_2, x_2^2)$$

If the dimension m of $\mathbf{x}$ is larger than 2, the mapping will create m choose 2 new features which are all the second order terms, so $M = O(m^2)$, but computing $k(\mathbf{x}, \mathbf{x}')$ is $O(m)$. If we consider a higher-order polynomial kernel, $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^p$, we will get a feature mapping with $M = O(m^p)$, while computing the kernel function is again $O(m)$.

## Constructing Kernels

Often, kernels are constructed by trying to express a similarity function between instances and then making sure that indeed it corresponds to some feature mapping $\phi$. To show that a kernel corresponds to a feature mapping it is enough to check that

$k(\mathbf{x}, \mathbf{x}')$ is positive semi-definite, which means that for any set of points $\{\mathbf{x}_i\}_{i=1}^n$ and any n, the Gram matrix K is *positive semi-definite* [1]. Recall that an n by n real symmetric matrix $\mathbf{K}$ is positive semi-definite ($\mathbf{K} \succeq 0$) if for any vector $\mathbf{z}$ of dimension n, $\mathbf{z}^\top \mathbf{K} \mathbf{z} \geq 0$

Intuitively, positive semi-definiteness is a matrix analog of non-negativity for scalars. Using eigen-decomposition, a positive semi-definite matrix can always be written as:

$$\mathbf{K} = \sum_{i=1}^n \lambda_i \mathbf{z}_i \mathbf{z}_i^\top; \text{ so } \mathbf{z}^\top \mathbf{K} \mathbf{z} = \sum_{i=1}^n \lambda_i (\mathbf{z}_i^\top \mathbf{z})^2$$

where $\mathbf{z}_i$ are real eigenvectors and $\lambda_i$ are non-negative and real eigenvalues.

One of the most common way of constructing kernels is by building more complex valid kernels ones by composing other valid kernels.

Here are some construction rules (we assume below that c>0, $k_1(\cdot,\cdot)$ and $k_2(\cdot,\cdot)$ are valid kernels, $q(\cdot)$ is a polynomial with positive coefficients, $\mathbf{A} \succeq 0$, and $f(\cdot)$ is any function.

1. $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$
2. $k(\mathbf{x}, \mathbf{x}') = c k_1(\mathbf{x}, \mathbf{x}')$
3. $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$
4. $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}')$
5. $k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}'))$
6. $k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}')$
7. $k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}'))$
8. $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{A} \mathbf{x}'$
9. ...

Using the rules above, it is easy to show that the Gaussian kernel is valid:

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2) = \exp(-\mathbf{x}^\top \mathbf{x} / 2\sigma^2) \exp(\mathbf{x}^\top \mathbf{x}' / \sigma^2) \exp(-\mathbf{x}'^\top \mathbf{x}' / 2\sigma^2) = f(\mathbf{x}) \exp(\mathbf{x}^\top \mathbf{x}' / \sigma^2) f(\mathbf{x}')$$

You will show in your homework that the Gaussian kernel corresponds to an infinite-dimensional feature mapping function.

The great advantage of kernels is that they can also be constructed when $\mathbf{x}$ is not a pre-defined vector of features, but instead some complex object, like an image or a set of items, or a sequence or a graph. For example, if $S_1$ and $S_2$ are sets, then a valid kernel on sets is the size of their intersection, exponentiated:

$$k(S_1, S_2) = \exp(|S_1 \cap S_2|)$$

Kernels have been popularized in the machine learning community through Support Vector Machines, which have nice sparsity properties ($\alpha$ has many zeros) that make them particularly well suited to using kernels. Here is a *little preview of kernel SVMs* [2].

## Links

1. **en.wikipedia.org/wiki/Positive-definite_matrix**
2. **www.csie.ntu.edu.tw/~cjlin/libsvm**