Chapter 7: Kernel methods

Volodymyr Vovk

v.vovk@rhul.ac.uk
Office Bedford 2-20

CS3920/CS5920 Machine Learning
Last edited: November 12, 2022

Version with solutions on slides 15–16, 31–32, and 34–35

Plan

Idea

- Kernel methods are another way to turn linear methods into non-linear ones.
- There are many methods in machine learning whose predictions depend only on the dot products between the samples.
- For example, the predictions computed by Nearest Neighbours only depend on (squared) distances

$$\|x - x'\|^2 = x \cdot x - 2x \cdot x' + x' \cdot x'.$$

- Now suppose that every time a dot product $x \cdot x'$ appears somewhere, we replace it with a generalization of the dot product of the form $K(x, x')$, where K is some function that we will refer to as a kernel (the precise definition is to follow).

Linear kernel

- Intuition: a kernel is a function that quantifies the similarity between two samples (details: later).
- For example, we could simply take

$$K(x, x') = x \cdot x' = \sum_{j=0}^{p-1} x[j]x'[j],$$

  which would just give us back the original method.
- This is known as the linear kernel, because our method stays linear.
- The linear kernel essentially quantifies the similarity of a pair of samples using correlation (assuming the samples are standardized by dividing by their standard deviation, as explained later).

Polynomial kernels

- But one could instead choose another form for the kernel, such as

$$K(x, x') = (1 + x \cdot x')^d,$$

  where d is a positive integer.
- This is known as a polynomial kernel of degree d.
- Using such a kernel with $d > 1$, instead of the standard linear kernel, leads to much more flexible models.
- It amounts to fitting the original method in a higher-dimensional space involving polynomials of degree d, rather than in the original sample space.

Radial kernel

- Another popular choice is the radial kernel

$$K(x, x') = \exp\left(-\gamma \left\|x - x'\right\|^2\right)$$
$$= \exp\left(-\gamma \sum_{j=0}^{p-1} (x[j] - x'[j])^2\right),$$

where γ is a positive constant.

Plan

Formal definition of kernels

- The definition of a kernel: we take a feature mapping $F : \mathbf{X} \to H$ of the sample space $\mathbf{X}$ into a feature space H equipped with dot product (formally, a "Hilbert space"). This gives us the kernel

$$K(x, x') = F(x) \cdot F(x').$$

- In general, a kernel is any function that can be obtained this way.
- Any such kernel can be used in the "kernel trick".

General scheme

- a feature mapping F can be combined with nearly any algorithm
- if the original algorithm contains the training and test samples only in dot products, we can write a kernel version of this algorithm
- kernel trick:
  - write the algorithm in such a way that all xs only appear in dot products
  - replace all dot products with kernels

An advantage of using kernels

- We do not need to use F explicitly!
- For many simple-looking kernels (such as radial kernels), the corresponding feature spaces can even be infinite-dimensional.
- There is a simple criterion for K being a kernel: it should be symmetric and positive definite. (Next slide.)

Criterion (1)

- A function $K(x, x')$, where $x, x' \in \mathbf{X}$, is symmetric if it is always true that
$$K(x, x') = K(x', x).$$
- K is positive definite (or nonnegative definite) if, for any n, any samples $x_1, \ldots, x_n$, and any numbers $a_1, \ldots, a_n$,
$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j K(x_i, x_j) \geq 0.$$

Criterion (2)

Theorem

A continuous function $K : \mathbf{X}^2 \to \mathbb{R}$ is a kernel if and only if it is symmetric and positive definite.

- If $\mathbf{X}$ is a discrete space, then K is continuous by definition.
- In one direction the statement is obvious (see the next slide), in the other much less so (not in this module).
- The polynomial and radial kernels are symmetric and positive definite (although the latter is not easy to check).

Criterion (3)

- Any kernel is symmetric:

$$K(x, x') = F(x) \cdot F(x') = F(x') \cdot F(x) = K(x', x).$$

- Any kernel is positive definite:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j K(x_i, x_j) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j F(x_i) \cdot F(x_j)$$

$$= \left(\sum_{i=1}^{n} a_i F(x_i) \right) \cdot \left(\sum_{j=1}^{n} a_j F(x_j) \right) = \left\| \sum_{i=1}^{n} a_i F(x_i) \right\|^2 \geq 0.$$

Combining kernels (1)

The theorem immediately implies (assuming the kernels are continuous):

- The sum of two kernels is a kernel: if K_1 and K_2 are kernels, so is $K_1 + K_2$.
- Kernels can be scaled: if K is a kernel and $w > 0$, then wK is a kernel as well.

Exercise: prove these implications.

Not in this module: the product of two kernels is also a kernel.

Exercise: proofs of the implications (1)

- *Let K_1 and K_2 be kernels, and set $K := K_1 + K_2$. First we check that K is symmetric:*

$$K(x, x') = K_1(x, x') + K_2(x, x') \\ = K_1(x', x) + K_2(x', x) = K(x', x)$$

(we know that K_1 and K_2 are symmetric). Now let us check that K is positive definite

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(x_i, x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j (K_1(x_i, x_j) + K_2(x_i, x_j))$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K_1(x_i, x_j) + \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K_2(x_i, x_j) \geq 0 + 0 = 0$$

(we know that K_1 and K_2 are positive definite). Therefore, K is a kernel.

Exercise: proofs of the implications (2)

- *Let K be a kernel and $w > 0$; set $K' := wK$. First we check that K' is symmetric:*

$$K'(x, x') = wK(x, x') = wK(x', x) = K'(x', x)$$

(we know that K is symmetric). Now let us check that K' is positive definite

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K'(x_i, x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j wK(x_i, x_j)$$

$$= w\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(x_i, x_j) \geq 0$$

(we know that K is positive definite). Therefore, K' is a kernel.

Combining kernels (2)

- Suppose we have several kernels, say K_1, K_2, K_3, and do not know which one to use.
- We can use their sum $K_1 + K_2 + K_3$ instead.
- More generally, we can use $w_1 K_1 + w_2 K_2 + w_3 K_3$ for any positive weights w_1, w_2, w_3.

Normalizing kernels (1)

- Kernels are often interpreted as measures of similarity: $K(x, x')$ shows how similar x and x' are.
- For the linear kernel, this interpretation makes sense only if x and x' have the same length.
- Therefore, the feature mapping F is often normalized, i.e., replaced by
$$\tilde{F}(x) = F(x) / \|F(x)\| .$$
- The corresponding kernel is the normalized kernel
$$\tilde{K}(x, x') = \tilde{F}(x) \cdot \tilde{F}(x') = \frac{F(x) \cdot F(x')}{\|F(x)\| \, \|F(x')\|} = \frac{K(x, x')}{\sqrt{K(x, x)}\sqrt{K(x', x')}}.$$

Normalizing kernels (2)

- There is a danger: what if $K(x, x) = 0$ or $K(x', x') = 0$? (i.e., $F(x) = 0$ or $F(x') = 0$?)
- In this case you should set, say, $\tilde{K}(x, x') = 0$ instead of trying to apply the general formula (which would lead to a runtime error in Python).

Introduction

- This subsection: a kernel between two text documents (or "strings").
- Idea: to compare them by means of the substrings (of a given length c) they contain.
  - the more substrings in common, the more similar they are
- It is important that such substrings do not need to be contiguous, and the degree of contiguity of one such substring in a document determines how much weight it will have in the comparison.

Decay factor λ

- For example: the substring "c-a-r" is present both in "card" and in "custard", but with different weighting.
- For each such substring there is a dimension of the feature space, and the value of such coordinate depends on how frequently and how compactly the substring is embedded in the text.
- To deal with non-contiguous substrings ("subsequences"), it is necessary to introduce a decay factor $\lambda \in (0, 1]$ that can be used to weigh the gaps.

Example (1)

- Consider the strings (mini-documents) "cat", "car", "bat", and "bar". For $c = 2$ (the length of the subsequences taken into account), we obtain an 8-dimensional feature space (ignoring the 0 features), where the words are mapped as follows:

	a-r	a-t	b-a	b-r	b-t	c-a	c-r	c-t
$F(\text{cat})$	0	λ^2	0	0	0	λ^2	0	λ^3
$F(\text{car})$	λ^2	0	0	0	0	λ^2	λ^3	0
$F(\text{bat})$	0	λ^2	λ^2	0	λ^3	0	0	0
$F(\text{bar})$	λ^2	0	λ^2	λ^3	0	0	0	0

- The (unnormalized) kernel between "car" and "cat" is

$$K(\text{car}, \text{cat}) = \lambda^4.$$

Example (2)

- The cell in column u and row $F(s)$ is 0 if u is not a subsequence of s.
- If u is a subsequence of s, the cell is λ^k, where k is the length of the embedding of u into s (for now assume the embedding is unique).
- When computing the length of the embedding, you should also count the letters that need to be added to u to get a contiguous substring of s.

Example (3)

- The normalized kernel is:

$$\tilde{K}(\text{car}, \text{cat}) = \frac{\lambda^4}{2\lambda^4 + \lambda^6} = \frac{1}{2 + \lambda^2},$$

since

$$K(\text{car}, \text{car}) = K(\text{cat}, \text{cat}) = 2\lambda^4 + \lambda^6.$$

- In general, the document will contain more than one word, but we make it into one string by concatenating all the words and the spaces ␣ (ignoring the punctuation and capitalization).
- In principle, the table on on slide 22 should contain $(26 + 1)^2 = 729$ columns ("a-a", "a-b",..., "␣-␣"), but only 8 of them are non-zero.

Example with non-unique embeddings

- Consider the string "london". For $c = 2$, we obtain an 11-dimensional feature vector (if we ignore the 0 components):

d-n	d-o	l-d	l-n	l-o
λ^3	λ^2	λ^4	$\lambda^3 + \lambda^6$	$\lambda^2 + \lambda^5$

n-d	n-n	n-o	o-d	o-n	o-o
λ^2	λ^4	λ^3	λ^3	$2\lambda^2 + \lambda^5$	λ^4

- The string "london" maps to the feature vector

$$\left(\lambda^3, \lambda^2, \lambda^4, \lambda^3 + \lambda^6, \lambda^2 + \lambda^5, \lambda^2, \lambda^4, \lambda^3, \lambda^3, 2\lambda^2 + \lambda^5, \lambda^4\right).$$

- Each cell is the sum of λ^k over all embeddings.

Putting everything together

- Once we have created a kernel, it is natural to normalize to remove any bias introduced by document length.
- We create a new embedding $\tilde{F}(s) = F(s)/\|F(s)\|$, which gives rise to the kernel

$$\tilde{K}_c(s,t) = \frac{K_c(s,t)}{\sqrt{K_c(s,s)}\sqrt{K_c(t,t)}}$$

(remember c is the length of the subsequences taken into account).

- We can create a kernel $K(s,t)$ that combines different $\tilde{K}_c(s,t)$ giving different (positive) weights to several c (as explained on slide 17).

Efficient computation of string kernels

- String kernels can be computed efficiently.
- Not in this module: it is possible to speed up the computation of $K_c(s,t)$ to $O(c\,|s|\,|t|)$ (where $|\cdot|$ is the length of a string, `len` in Python).

Plan

Kernels with Nearest Neighbours

Very easy to kernelize:

- In the Nearest Neighbours algorithm the predicted label for a test sample x^* is taken from the nearest training samples.
- Let us map all samples into a feature space; the squared distance becomes

$$\begin{aligned}\left[d(F(x),F(x'))\right]^2 &= \|F(x)-F(x')\|^2 \\ &= (F(x)-F(x'))\cdot(F(x)-F(x')) \\ &= F(x)\cdot F(x)+F(x')\cdot F(x')-2F(x)\cdot F(x') \\ &= K(x,x)+K(x',x')-2K(x,x').\end{aligned}$$

- Kernelized version of Nearest Neighbours: measure the distance by

$$d_K(x,x')=\sqrt{K(x,x)+K(x',x')-2K(x,x')}.$$

Exercise

- Consider the degree 1 polynomial kernel $K(x, x') = 1 + xx'$.
- Consider the regression problem with the training set
  - $x_1 = -2$ and $y_1 = 2$,
  - $x_2 = 4$ and $y_2 = 3$

  and a test sample $x^* = 1$.
- Predict its label using the Nearest Neighbour algorithm.

Solution to the exercise

- *Answer: It's a tie! No answer. (Alternatively, say 2.5, or toss a coin.)*
- *The intermediate results are*

$$
\begin{aligned}
K(x_1, x_1) &= 5 \\
K(x_2, x_2) &= 17 \\
K(x^*, x^*) &= 2 \\
K(x_1, x^*) &= -1 \\
K(x_2, x^*) &= 5 \\
d^2(x^*, x_1) &= 2 + 5 - 2 \times (-1) = 9 \\
d^2(x^*, x_2) &= 2 + 17 - 2 \times 5 = 9.
\end{aligned}
$$

Answer to a question (*)

- *This slide is very optional and answers a question asked during the lecture.*
- *The question was: are we doing Nearest Neighbour in an infinite-dimensional space (implicitly via the use of the kernel)?*
- *The answer is: not in this case. For example, we can use* $F(x) := (x, 1)$ *as feature mapping. This feature mapping can be represented geometrically as shifting the real line up by 1 unit. The feature space is 2D (and we are only using a 1D part of it).*
- *It is clear that Nearest Neighbour in the feature and original spaces produces identical predictions (so we have another solution, which, however, works only in this special case, while the previous solution was general).*

Another exercise

- Consider the string kernel with parameter $c = 2$ (as on slide 22). Use the decay factor $\lambda = 1$. (By default, the string kernel is unnormalized.)
- Consider the regression problem with the training set
  - $x_1 =$ `'cat'` and $y_1 = 0$,
  - $x_2 =$ `'car'` and $y_2 = 2$

  and the test sample $x^* =$ `'bat'`.
- Predict its label using the Nearest Neighbour algorithm.

Solution to the exercise (1)

- *Answer: 0.*
- *The nearest neighbour to x^* is x_1 (which is intuitively obvious).*

Solution to the exercise (2)

These are the calculations:

$$
\begin{aligned}
K(x_1, x_1) &= 2\lambda^4 + \lambda^6 = 3 \\
K(x_2, x_2) &= 2\lambda^4 + \lambda^6 = 3 \\
K(x^*, x^*) &= 2\lambda^4 + \lambda^6 = 3 \\
K(x_1, x^*) &= \lambda^4 = 1 \\
K(x_2, x^*) &= 0 \\
d^2(x^*, x_1) &= 3 + 3 - 2 \times 1 = 4 \\
d^2(x^*, x_2) &= 3 + 3 - 2 \times 0 = 6.
\end{aligned}
$$

Kernelizing other prediction algorithms

- The prediction for Ridge Regression (and in particular, Least Squares) can be written in the form involving the samples only via their pairwise dot products (the details not in this module).
- Replacing the dot products $x \cdot x'$ by the kernels $K(x, x')$ gives Kernel Ridge Regression.
- Maximum margin classifiers (a linear method discussed in the next chapter) can also be written in this form.
- Therefore, they can also be kernelized (giving support vector machines).
- There has been a kernelization industry in machine learning; all kind of linear algorithms have been kernelized.

Some applications of kernels (1)

- Kernels were a big breakthrough when they appeared in the 1990s (in the form of support vector machines, the topic of the next chapter).
- Historically, the first area where kernel methods were used was handwritten digit recognition; they turned out to be competitive with tangent distance (and neural networks).

Some applications of kernels (2)

- String kernels are used in natural language processing (obviously).
- They are also widely used in bioinformatics.
- For example, string kernels can model, to some degree, the evolutionary process of insertion and deletion of DNA fragments.
- Amazingly, they make it possible to apply linear methods to discrete samples (such as texts, DNA sequences, and proteins).
- They have been used in automated phone helplines (e.g., by AT&T in the USA).

Further details (1)

John Shawe-Taylor and Nello Cristianini,
Kernel Methods and Pattern Analysis
2004
Section 2.2 (primal and dual ridge regression), Chapter 3 (properties of kernels), in particular Theorem 3.11 (characterization of kernels) and Proposition 3.22 (combining kernels); Chapters 10 and 11: kernels for text and strings (especially Section 11.5)

V05, Chapter 1: explicit conformal prediction based on kernel ridge regression

Further details (2)

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels, Journal of Machine Learning Research **2** (2002) 419–444.

This original paper is perhaps still the best source on string kernels. It gives an $O(c\,|s|\,|t|^2)$ algorithm for computing $K_c(s,t)$. The Shawe-Taylor and Cristianini book (Section 11.5) gives $O(c\,|s|^2\,|t|^2)$ and $O(c\,|s|\,|t|)$ algorithms. The paper also has an $O(c\,|s|\,|t|)$ algorithm.

Changes since first posted

On 10 November (as mentioned in class):

- improved the wording on slide 22, making it clear that the number of features is more than 8; made a similar change on slide 25;
- on slide 24, mentioned the number of features in the table on slide 22.