

Machine Learning

Textbook: Machine Learning Refined, by Watt, Borhani, and Katsaggelos, 2nd edition:
MLRefined.com

Previous Lecture

- ♦ K -fold cross-validation partitions the development data into disjoint folds and gives every observation an out-of-fold prediction.
- ♦ The pooled CV error is the average loss across all out-of-fold predictions; equal-size folds reduce this to the ordinary mean of the fold errors.
- ♦ Hyperparameters are chosen by comparing cross-validation performance, after which the selected model is refit using all development data.
- ♦ Every data-dependent preprocessing step must be fit separately inside each fold to prevent leakage.
- ♦ Larger K uses more observations for each fold fit but requires more model fits; LOOCV is the extreme case $K = P$.
- ♦ Fold construction must reflect the data structure: stratification, grouped folds, or time-respecting validation may be required.



§12.1 thru 12.6 - Kernel Methods

Earlier:

nonlinear feature maps
transformed

$\mathbf{x}$

into

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

This created nonlinear models
while keeping the model linear
in its trainable coefficients.

But:

the transformed feature vector
may contain thousands,
millions,
or even infinitely many coordinates.

Today:

sometimes we can avoid constructing
those coordinates entirely.

Feature-Space Inner Products

Suppose

$$\phi : \mathbb{R}^N \rightarrow \mathbb{R}^M.$$

Two transformed observations have inner product

$$\phi(\mathbf{x})^T \phi(\mathbf{z}).$$

For many algorithms:

transformed observations appear
only through inner products of this form.

If that inner product can be computed directly
from

$$\mathbf{x}$$

and

$$\mathbf{z},$$

we may never need to form

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

explicitly.

A Kernel

A function

$$K(\mathbf{x}, \mathbf{z})$$

is a kernel when it represents

an inner product in some feature space:

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z}).$$

Then:

instead of computing

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

and

$$\phi(\mathbf{z}),$$

compute

$$K(\mathbf{x}, \mathbf{z})$$

directly.

This substitution is called

the **kernel trick**.

The Linear Kernel

Choose the identity feature map:

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

Then:

$$K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}.$$

This is the

linear kernel.

So ordinary linear learning

already fits inside the kernel framework.

The interesting cases arise
when
 K
corresponds to a nonlinear feature map.

A Quadratic Polynomial Kernel

For
 $\mathbf{x}, \mathbf{z} \in \mathbb{R}^2$,
consider
 $K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^T \mathbf{z})^2$.

Write
 $\mathbf{x} = (x_1, x_2)^T$
and
 $\mathbf{z} = (z_1, z_2)^T$.

Then:
 K
 $= (1 + x_1 z_1 + x_2 z_2)^2$.

Expand it.

Expand the Kernel

We obtain
 $K(\mathbf{x}, \mathbf{z})$
 $= 1$
 $+ 2x_1 z_1$
 $+ 2x_2 z_2$
 $+ x_1^2 z_1^2$
 $+ 2x_1 x_2 z_1 z_2$
 $+ x_2^2 z_2^2$.

This can be written as

$\phi(\mathbf{x})^T \phi(\mathbf{z})$
using

$$\phi(\mathbf{x}) = \begin{bmatrix} 1 \\ \sqrt{2} x_1 \\ \sqrt{2} x_2 \\ x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{bmatrix}.$$

Thus one kernel evaluation
implicitly computes the inner product
of six nonlinear features.

Verify the Polynomial Kernel Numerically

Take

$$\mathbf{x} = (1, 2)^T$$

and

$$\mathbf{z} = (3, -1)^T.$$

Original inner product:

$$\begin{aligned} \mathbf{x}^T \mathbf{z} &= 3 - 2 \\ &= 1. \end{aligned}$$

Kernel:

$$\begin{aligned} (1 + 1)^2 &= 4. \end{aligned}$$

Using the explicit six-dimensional feature map:

$$\begin{aligned} \phi(\mathbf{x})^T \phi(\mathbf{z}) &= 4 \end{aligned}$$

as well.

Same transformed-space inner product.

No explicit expansion is necessary
if the kernel formula is available.

General Polynomial Kernel

A common family is

$$K(\mathbf{x}, \mathbf{z}) = (c + \mathbf{x}^T \mathbf{z})^d,$$

where typically

$$c \geq 0$$

and

$$d$$

is a positive integer.

The feature space contains
polynomial combinations
of the original variables
up to degree

$$d$$

depending on the offset

$$c.$$

The kernel computes their inner product
without explicitly listing every monomial.

The Gram Matrix

For training observations

$$\mathbf{x}_1, \dots, \mathbf{x}_P,$$

define

$$G_{pq} = K(\mathbf{x}_p, \mathbf{x}_q).$$

Then

$$G = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \cdots & K(\mathbf{x}_1, \mathbf{x}_P) \\ \vdots & \ddots & \vdots \\ K(\mathbf{x}_P, \mathbf{x}_1) & \cdots & K(\mathbf{x}_P, \mathbf{x}_P) \end{bmatrix}.$$

This is the

kernel matrix

or

Gram matrix.

Its size is

$$P \times P.$$

Why the Gram Matrix Is Symmetric

Because an inner product satisfies

$$\mathbf{a}^T \mathbf{b} = \mathbf{b}^T \mathbf{a}:$$

$$\begin{aligned} &K(\mathbf{x}_p, \mathbf{x}_q) \\ &= K(\mathbf{x}_q, \mathbf{x}_p). \end{aligned}$$

Therefore:

$$G = G^T.$$

A valid real-valued kernel
is symmetric.

But symmetry alone
is not enough.

The Gram matrix must also satisfy
a positivity property.

Positive Semidefiniteness

Let

$$\mathbf{a} \in \mathbb{R}^P.$$

Then:

$$\begin{aligned} &\mathbf{a}^T G \mathbf{a} \\ &= \sum_{p,q} a_p a_q K(\mathbf{x}_p, \mathbf{x}_q). \end{aligned}$$

If

$$\begin{aligned} &K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z}): \\ &\mathbf{a}^T G \mathbf{a} \\ &= \left\| \sum_{p=1}^P a_p \phi(\mathbf{x}_p) \right\|^2. \end{aligned}$$

Therefore:
 $\mathbf{a}^T G \mathbf{a} \geq 0.$

So every Gram matrix from a valid kernel
is positive semidefinite.

A Kernel Is More Than an Arbitrary Similarity Score

A function may look like
a reasonable measure of similarity
but fail to produce
positive-semidefinite Gram matrices.

Kernel methods rely on
inner-product geometry.

Thus a standard sufficient viewpoint is:
 $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z}).$

Equivalent kernel-validity conditions
can be stated directly
in terms of symmetric positive-semidefinite Gram matrices.

Do not replace a kernel
by an arbitrary similarity function
without checking validity.

A Gaussian / RBF Kernel

One of the most common nonlinear kernels is
 $K(\mathbf{x}, \mathbf{z}) = \exp(-\gamma \|\mathbf{x} - \mathbf{z}\|^2),$

with
 $\gamma > 0.$

This is the
Gaussian
or
radial basis function
kernel.

Properties:
 $K(\mathbf{x}, \mathbf{x}) = 1;$

nearby observations have kernel value near 1;

distant observations have kernel value near 0.

The Role of γ

RBF kernel:
$$K = e^{-\gamma \| \mathbf{x} - \mathbf{z} \|^2}.$$

Small
 γ :

similarity decays slowly;

each observation influences a broad region.

Large
 γ :

similarity decays rapidly;

influence becomes highly localized.

Thus
 γ
controls model flexibility.

It is a
hyperparameter.

Use validation / cross-validation
to choose it.

The RBF Kernel Hides Infinitely Many Features

In one dimension:

$$K(x,z) = e^{-\gamma (x-z)^2}.$$

Expand:
$$K(x,z) = e^{-\gamma x^2} e^{-\gamma z^2} e^{2\gamma xz}.$$

Now use
$$e^t = \sum_{m=0}^{\infty} t^m / m!.$$

Thus:
$$K(x,z) = e^{-\gamma x^2} e^{-\gamma z^2} \sum_{m=0}^{\infty} \frac{(2\gamma)^m}{m!} x^m z^m.$$

This is an inner product
involving infinitely many coordinates.

An Explicit Infinite Feature Map

Define
$$\phi_m(x) = e^{-\gamma x^2} \sqrt{\frac{(2\gamma)^m}{m!}} x^m$$

for
 $m = 0, 1, 2, \dots$

Then:
$$\sum_{m=0}^{\infty} \phi_m(x)\phi_m(z)$$
$$= e^{-\gamma(x-z)^2}.$$

Therefore the RBF kernel
corresponds to an infinite-dimensional feature representation.

Yet:
one kernel evaluation requires only
computing a squared distance
and an exponential.

This is the power of the kernel trick.

Kernel Ridge Regression

Return to ridge regression
in transformed feature space.

Let
 Φ
be the feature matrix.

For this derivation:
penalize all feature-space coefficients.

Objective:
$$g(\mathbf{w}) = \frac{1}{P} \|\Phi \mathbf{w} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2.$$

The usual ridge solution satisfies
$$(\Phi^T \Phi + P \lambda I) \mathbf{w} = \Phi^T \mathbf{y}.$$

But this still seems to require
explicit transformed features.

Why the Ridge Solution Lies in the Training Span

Let
$$S = \text{span}\{\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_P)\}.$$

Decompose any coefficient vector as
$$\mathbf{w} = \mathbf{w}_{\parallel} + \mathbf{w}_{\perp},$$

where
$$\mathbf{w}_{\parallel} \in S$$
and
$$\mathbf{w}_{\perp} \perp S.$$

For every training observation:

$$\phi(\mathbf{x}_p)^T \mathbf{w}_\perp = 0.$$

Therefore

$$\Phi \mathbf{w} = \Phi \mathbf{w}_\parallel.$$

The perpendicular component
does not change any training prediction.

But:

$$\|\mathbf{w}\|^2 = \|\mathbf{w}_\parallel\|^2 + \|\mathbf{w}_\perp\|^2.$$

So any nonzero

$$\mathbf{w}_\perp$$

increases the ridge penalty
without improving the fit.

Hence an optimum can be chosen with

$$\mathbf{w}_\perp = \mathbf{0}.$$

Thus the ridge solution lies in
S.

Write the Solution as a Combination of Training Features

Because

$$\mathbf{w} \in S,$$

there are coefficients

$$\boldsymbol{\alpha} \in \mathbb{R}^P$$

such that

$$\mathbf{w} = \Phi^T \boldsymbol{\alpha}.$$

Then training predictions are

$$\begin{aligned} &\Phi \mathbf{w} \\ &= \Phi \Phi^T \boldsymbol{\alpha}. \end{aligned}$$

But:

$$\Phi \Phi^T$$

is exactly the Gram matrix.

Thus:

$$\hat{\mathbf{y}} = G \boldsymbol{\alpha}.$$

Kernel Ridge Coefficients

Start with the primal normal equations:

$$(\Phi^T \Phi + P \lambda I) \mathbf{w} = \Phi^T \mathbf{y}.$$

Substitute

$$\mathbf{w} = \Phi^T \boldsymbol{\alpha}.$$

$$(\Phi^T \Phi + P\lambda I) \Phi^T \boldsymbol{\alpha} = \Phi^T \mathbf{y}.$$

Use

$$\begin{aligned} & (\Phi^T \Phi + P\lambda I) \Phi^T \\ &= \Phi^T (\Phi \Phi^T + P\lambda I). \end{aligned}$$

Therefore it is sufficient to choose

$$\begin{aligned} & \boldsymbol{\alpha} \\ \text{so that} \\ & (G + P\lambda I) \boldsymbol{\alpha} = \mathbf{y}. \end{aligned}$$

Because

$$G \succeq 0$$

and

$$\begin{aligned} & \lambda > 0: \\ & G + P\lambda I \end{aligned}$$

is positive definite
and invertible.

Thus

$$\boldsymbol{\alpha} = (G + P\lambda I)^{-1} \mathbf{y}.$$

Numerically:

$$\begin{aligned} & \text{solve} \\ & (G + P\lambda I) \boldsymbol{\alpha} = \mathbf{y}. \end{aligned}$$

Do not explicitly compute the inverse.

The fit now requires only
the kernel matrix
 G .

No explicit feature coordinates appear.

Predict a New Observation

For a new

$$\mathbf{x},$$

feature-space prediction is

$$f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^T \mathbf{w}.$$

Using

$$\begin{aligned} \mathbf{w} &= \Phi^T \boldsymbol{\alpha} \\ f(\mathbf{x}) \\ &= \sum_{p=1}^P \alpha_p \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}_p). \end{aligned}$$

Therefore:

$$f(\mathbf{x}) = \sum_{p=1}^P \alpha_p K(\mathbf{x}, \mathbf{x}_p).$$

A kernel predictor compares

the new observation
with the training observations.

A Small Kernel Ridge Example

Take

$$x = (-1, 0, 1)$$

and

$$y = (1, 0, 1).$$

Use

$$K(x, z) = (1 + xz)^2.$$

The Gram matrix is

$$G = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 4 \end{bmatrix}.$$

Choose

$$\lambda = 1/3.$$

Since

$$P = 3:$$

$$P\lambda = 1.$$

Solve

$$(G + I)\alpha = y.$$

Solve for α

The system is

$$\begin{bmatrix} 5 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 5 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

By symmetry:

$$\alpha_1 = \alpha_3.$$

The solution is

$$\alpha = \begin{bmatrix} 1/4 \\ -1/4 \\ 1/4 \end{bmatrix}.$$

Training predictions:

$$G\alpha = \begin{bmatrix} 3/4 \\ 1/4 \\ 3/4 \end{bmatrix}.$$

Ridge shrinkage prevents exact interpolation.

Predict at a New Point

Take

$$x_* = 2.$$

Kernel comparisons with training inputs:

$$K(2,-1) = (1 - 2)^2 = 1,$$

$$K(2,0) = 1,$$

$$K(2,1) = (1 + 2)^2 = 9.$$

Therefore:

$$\begin{aligned} f(2) &= 1\left(\frac{1}{4}\right) + 1\left(-\frac{1}{4}\right) + 9\left(\frac{1}{4}\right) \\ &= \frac{9}{4} = 2.25. \end{aligned}$$

Prediction used only
kernel evaluations
and the fitted
 α_p .

Connection to Support Vector Machines

In Lesson 1.17:
we postponed nonlinear SVMs.

The kernel trick supplies them.

A kernel SVM decision function has the form

$$f(\mathbf{x}) = b + \sum_{p=1}^P \alpha_p y_p K(\mathbf{x}_p, \mathbf{x}).$$

Only observations with
 $\alpha_p \neq 0$
affect the decision function.

These are the
support vectors.

A nonlinear kernel
creates a nonlinear boundary
in the original input space.

Where the Kernel Enters the SVM Optimization

For the hard-margin SVM dual:

$$\underset{\alpha}{\text{maximize}} \quad \sum_p \alpha_p - \frac{1}{2} \sum_{p,q} \alpha_p \alpha_q y_p y_q K(\mathbf{x}_p, \mathbf{x}_q)$$

subject to
 $\alpha_p \geq 0$
and
 $\sum_p \alpha_p y_p = 0$.

For the standard soft-margin version:
add upper bounds
 $0 \leq \alpha_p \leq C$.

The transformed feature vectors
occur only through their inner products.

Replace those inner products
by a kernel.

Kernel SVM Geometry

In feature space:
the SVM boundary is still a hyperplane.

But the inverse image
of that hyperplane
under
 ϕ
can be curved or highly complex
in the original input space.

Thus:
linear separator in feature space $\Rightarrow$ nonlinear separator in input space.

This is exactly the geometric principle
from Lesson 1.28,
now without explicit feature construction.

Kernel Hyperparameters Control Complexity

Polynomial kernel:
 $(c + \mathbf{x}^T \mathbf{z})^d$.

Hyperparameters include:
degree
 d
and possibly offset
 c .

RBF kernel:
 $e^{-\gamma \|\mathbf{x} - \mathbf{z}\|^2}$.

Hyperparameter:
 γ .

Regularized kernel methods also contain:
ridge
 λ
or SVM
 C .

These choices interact.

Use cross-validation
to choose them.

Feature Scaling Matters for Kernels

Consider the RBF kernel:

$$K = e^{-\gamma \| \mathbf{x} - \mathbf{z} \|^2}.$$

If one feature is measured
on a much larger numerical scale:

that feature can dominate
the squared distance.

Then the kernel similarity
is effectively controlled
by that coordinate.

Thus standardization is often important
for distance-based kernels.

But:
as always,
scaling changes the geometry
and must make substantive sense.

Cross-Validation Must Include Kernel Preprocessing

Suppose we compare
RBF values of
 γ .

Inside each fold:

fit the standardizer
using only the fold-training data;

transform fold-training data;

transform fold-validation data
using the same fitted standardizer;

build kernel matrices;

fit the candidate kernel model;

evaluate on the held-out fold.

The same leakage rule from Lesson 1.30 applies.

Kernels Trade Feature Dimension for Sample Dimension

Explicit feature approach:

store

$$\Phi \in \mathbb{R}^{P \times M}.$$

Kernel approach:

store

$$G \in \mathbb{R}^{P \times P}.$$

This is attractive when:

M

is enormous

but

P

is moderate.

But if

P

is very large:

the kernel matrix itself

can become expensive.

Memory grows like

$$P^2.$$

Dense linear-system solves

can approach cubic cost in

$$P.$$

Kernel Methods Are Not Automatically Faster

The kernel trick avoids

explicit high-dimensional features.

But it may require:

all pairwise training similarities;

a dense $P \times P$ matrix;

many kernel evaluations for prediction.

Thus:

implicit features $\neq$ free computation.

Kernel methods can be excellent
for moderate datasets
with rich nonlinear structure.

For extremely large datasets:
explicit approximate features
or neural feature learning
may be more scalable.

Kernel Matrix in Python

Polynomial kernel:

```
import numpy as np

def polynomial_kernel(x, z, degree=2, c=1.0):
    return (c + x @ z)**degree
```

For a dataset:

```
def gram_matrix(X, kernel):
    P = len(X)

    G = np.zeros((P, P))

    for p in range(P):
        for q in range(P):
            G[p, q] = kernel(
                X[p],
                X[q]
            )

    return G
```

The output is symmetric
for a symmetric kernel.

Verify the Polynomial Feature Map in Python

```
x = np.array([1., 2.])
z = np.array([3., -1.])

K = (1 + x @ z)**2

phi_x = np.array([
    1,
    np.sqrt(2)*x[0],
    np.sqrt(2)*x[1],
    x[0]**2,
    np.sqrt(2)*x[0]*x[1],
    x[1]**2
])

phi_z = np.array([
    1,
    np.sqrt(2)*z[0],
    np.sqrt(2)*z[1],
    z[0]**2,
    np.sqrt(2)*z[0]*z[1],
    z[1]**2
])
```

```

])

print(K)
print(phi_x @ phi_z)

```

Output:

```

4.0
4.0

```

The kernel value equals
the explicit feature-space inner product.

RBF Kernel in Python

```

def rbf_kernel(x, z, gamma=1.0):
    diff = x - z

    return np.exp(
        -gamma * (diff @ diff)
    )

```

Example:

```

x = np.array([0.])
z = np.array([1.])

print(
    rbf_kernel(
        x,
        z,
        gamma=0.5
    )
)

```

Output:

```

0.6065306597

```

This equals

$e^{-1/2}$.

Kernel Ridge in Python

Use

```

x = (-1,0,1).
X = np.array([[ -1.],
               [  0.],
               [  1.]])

y = np.array([1., 0., 1.])

def kernel(x, z):
    return (1 + x @ z)**2

G = gram_matrix(X, kernel)

lam = 1/3
P = len(X)

alpha = np.linalg.solve(

```

```

        G + P*lam*np.eye(P),
        Y
    )

    print(alpha)

```

Output:

```
[ 0.25 -0.25  0.25]
```

Exactly the hand calculation.

Predict With the Kernel Model in Python

```

def predict_kernel_ridge(
    X_train,
    alpha,
    x_new,
    kernel
):
    k = np.array([
        kernel(xp, x_new)
        for xp in X_train
    ])

    return k @ alpha

```

For

```
x_new = np.array([2.])
```

the prediction is

2.25.

The new point is represented

only through its kernel similarities
with training observations.

Check Positive Semidefiniteness Numerically

For a symmetric Gram matrix:

```

eigenvalues = np.linalg.eigvalsh(G)

print(eigenvalues)

```

New Python idea:

`np`

computes eigenvalues

of a real symmetric matrix.

For a valid kernel:

the Gram eigenvalues
should be nonnegative
up to small floating-point error.

This is a useful numerical check

for a custom kernel.

The Kernel-Method Pipeline

Input observations:

$$\mathbf{x}_1, \dots, \mathbf{x}_P.$$

Choose kernel:

$$K(\mathbf{x}, \mathbf{z}).$$

Build pairwise Gram matrix:

$$G.$$

Fit dual / kernel coefficients:

$$\alpha.$$

Predict new observation:

$$f(\mathbf{x}) = \sum_p \alpha_p K(\mathbf{x}_p, \mathbf{x})$$

or, for SVMs,
include labels and intercept.

Feature-space coordinates
may never appear explicitly.

What Kernels Accomplish

Explicit nonlinear features:

choose
 $\phi(\mathbf{x}),$

build
 $\Phi,$

fit a linear model in those features.

Kernel method:

choose
 $K(\mathbf{x}, \mathbf{z}),$

compute feature-space inner products directly,

fit an algorithm expressible through those inner products.

Thus:
kernel methods are implicit nonlinear feature methods.

What did we learn?

- ♦ A kernel computes an inner product $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$ without requiring the transformed coordinates explicitly.
- ♦ The kernel / Gram matrix contains all pairwise training-set inner products in feature space and must be symmetric positive semidefinite for a valid kernel.
- ♦ Polynomial kernels implicitly generate polynomial and interaction features; the RBF kernel corresponds to an infinite-dimensional feature representation.
- ♦ Kernel ridge regression solves $(G + P\lambda I)\alpha = \mathbf{y}$ and predicts through weighted kernel similarities to the training observations.
- ♦ Kernel SVMs remain linear separators in feature space while producing nonlinear boundaries in the original input space.
- ♦ Kernel hyperparameters and preprocessing must be selected with leakage-safe cross-validation, and kernel computation becomes expensive when the number of observations is very large.



Next

- Kernel methods use fixed implicit feature maps chosen in advance through the kernel.
- A more flexible alternative is to make the feature representation itself trainable.
- Neural networks build layers of parameterized transformations and learn those transformations jointly with the final predictor.
- We begin with the architecture and algebra of a feedforward neural network.
- **1.32 - Neural-Network Architecture**