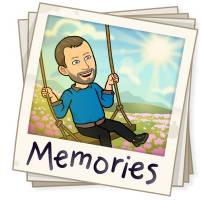


Machine Learning

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

Previous Lecture

- ♦ Regularization adds a coefficient penalty to the training objective and trades training fit against parameter control.
- ♦ Ridge uses an L_2 penalty and shrinks coefficients smoothly; its normal equations add $P\lambda I$ to the least-squares curvature matrix.
- ♦ Lasso uses an L_1 penalty; its nonsmooth corner at zero allows coefficients to become exactly zero.
- ♦ In one dimension, lasso produces the soft-threshold rule $\text{sign}(a) \max(|a| - \lambda/2, 0)$.
- ♦ Exact zero lasso coefficients provide embedded feature selection, but selected features need not be uniquely important or causal.
- ♦ Feature scaling changes coefficient penalties, so scaling and regularization must be designed together.



§10.1, 10.2, 10.4, 10.5 - Nonlinear Feature Maps

A linear model uses

$$f(\mathbf{x}) = w_0 + \mathbf{v}^T \mathbf{x}.$$

Its predictions are restricted by the original feature coordinates.

But:

we do not have to model the raw input directly.

Instead:

transform the input first.

Then fit a linear model

in the transformed features.

This is **nonlinear feature engineering**.

A Feature Map

Let

$$\mathbf{x} \in \mathbb{R}^N.$$

Define a transformation

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

Write

$$\phi(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{bmatrix}.$$

The functions

$$\phi_j$$

may be nonlinear in the original input.

Examples:

$$x^2,$$

$$x_1 x_2,$$

$$\sin x,$$

$$e^{-(x-c)^2}.$$

Linear in Parameters, Nonlinear in Input

After transforming the input:

$$f(\mathbf{x}; \mathbf{w}) = w_0 + \sum_{j=1}^M w_j \phi_j(\mathbf{x}).$$

This may be highly nonlinear

as a function of

$\mathbf{x}$.

But it is still linear

as a function of the coefficients

$\mathbf{w}$.

This distinction is fundamental:

nonlinear in input $\neq$ nonlinear in parameters.

Our old optimization machinery still applies.

The Augmented Feature Map

Include the intercept as a constant feature:

$$\tilde{\phi}(\mathbf{x}) = \begin{bmatrix} 1 \\ \phi_1(\mathbf{x}) \\ \vdots \\ \phi_M(\mathbf{x}) \end{bmatrix}.$$

Then

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

This looks exactly like

the linear model from Lesson 1.10.

Only the feature vector has changed.

From Data Matrix to Feature Matrix

For observations

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

build the transformed design matrix

$$\Phi = \begin{bmatrix} \text{---} \tilde{\phi}(\mathbf{x}_1)^T \text{---} \\ \vdots \\ \text{---} \tilde{\phi}(\mathbf{x}_P)^T \text{---} \end{bmatrix}.$$

Dimensions:

$$\Phi \in \mathbb{R}^{P \times (M+1)}.$$

Predictions:

$$\hat{\mathbf{y}} = \Phi \mathbf{w}.$$

Once

$$\Phi$$

has been constructed,

we can reuse the linear-learning formulas
without modification.

Nonlinear Regression

Suppose

$$x \in \mathbb{R}.$$

Choose the quadratic feature map:

$$\tilde{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}.$$

Then

$$\begin{aligned} f(x) \\ = w_0 + w_1x + w_2x^2. \end{aligned}$$

This is a quadratic curve
in the original input.

But:

$$f$$

is linear in

$$w_0, w_1, w_2.$$

Therefore least squares remains
a convex quadratic optimization problem.

A Dataset a Straight Line Cannot Fit Well

Consider

x	-2	-1	0	1	2
y	4	1	0	1	4

The data satisfy

$$y = x^2.$$

A straight-line model

$$w_0 + w_1x$$

cannot reproduce this curvature.

Because the data are symmetric:

the least-squares slope is

$$w_1 = 0.$$

The best intercept is the mean response:

$$w_0 = 2.$$

So the best straight line is

$$\hat{y} = 2.$$

Training Error of the Straight-Line Model

Predictions:

$$(2, 2, 2, 2, 2).$$

Residuals

$$\hat{y} - y:$$

$$(-2, 1, 2, 1, -2).$$

Squared residuals:

$$4, 1, 4, 1, 4.$$

Thus

$$\text{SSE} = 14.$$

And

$$\text{MSE} = \frac{14}{5} = 2.8.$$

The failure is not an optimization failure.

It is a representation failure.

A straight line simply cannot represent

$$x^2.$$

Add the Feature x^2

Use

$$\tilde{\phi}(x) = (1, x, x^2)^T.$$

Then

$$\Phi = \begin{bmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}.$$

Choose

$$\mathbf{w} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Then

$$\begin{aligned} \Phi \mathbf{w} &= (4, 1, 0, 1, 4)^T \\ &= \mathbf{y}. \end{aligned}$$

Therefore

$$\text{MSE} = 0.$$

The optimization method did not become more sophisticated.

The feature representation became more expressive.

Least Squares With Nonlinear Features

The squared-error objective is

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

Gradient:

$$\nabla g(\mathbf{w}) = \frac{2}{P} \Phi^T (\Phi \mathbf{w} - \mathbf{y}).$$

Normal equations:

$$\Phi^T \Phi \mathbf{w} = \Phi^T \mathbf{y}.$$

This is exactly the linear-regression formula

with

$$X$$

replaced by

$$\Phi.$$

Again:

nonlinear prediction

does not require nonlinear coefficient optimization.

Polynomial Features

For one variable:
degree- d polynomial features are
 $1, x, x^2, \dots, x^d$.

The model is
 $f(x) = w_0 + w_1x + \dots + w_dx^d$.

Increasing
 d
increases the range of shapes
the model can represent.

But:
greater expressiveness
also increases model complexity.

A sufficiently flexible polynomial
can fit training noise
as well as genuine pattern.

We will study that tradeoff next.

Interactions Between Features

Suppose
 $\mathbf{x} = (x_1, x_2)^T$.

A linear model contains:
 x_1
and
 x_2 .

A quadratic feature map can include:
 $1, x_1, x_2, x_1^2, x_1x_2, x_2^2$.

The term
 x_1x_2
is an
interaction feature.

It allows the effect of one variable
to depend on the value of another.

Meaning of an Interaction

Consider
 $f(x_1, x_2)$
 $= w_0 + w_1x_1 + w_2x_2 + w_{12}x_1x_2$.

Change x_1
by a small amount.

The partial derivative is $\frac{\partial f}{\partial x_1} = w_1 + w_{12}x_2$.

Thus the effect of x_1
depends on x_2 .

Without the interaction:
 $\partial f / \partial x_1 = w_1$
is constant.

Interactions encode joint effects
that additive linear models cannot express.

A Nonlinear Classification Boundary

Suppose $\mathbf{x} = (x_1, x_2)^T$.

Use the feature $\phi(\mathbf{x}) = x_1^2 + x_2^2$.

Define score $f(\mathbf{x}) = -1 + x_1^2 + x_2^2$.

Predict:

class +1
if $f(\mathbf{x}) > 0$,

class -1
if $f(\mathbf{x}) < 0$.

The decision boundary satisfies $x_1^2 + x_2^2 = 1$.

That boundary is a circle.

The Boundary Is Linear in Feature Space

Define $z = x_1^2 + x_2^2$.

Then the same score is simply

$$f = -1 + z.$$

In the transformed one-dimensional feature space:

the boundary is

$$z = 1.$$

Linear boundary in feature space.

Curved boundary in the original input space.

This is the central geometric idea

behind nonlinear feature maps.

An XOR-Style Interaction Example

Consider four points:

	x_1	x_2	y
1	-1	-1	-1
2	-1	1	1
3	1	-1	1
4	1	1	-1

No single straight line

separates these labels.

But define

$$\phi(\mathbf{x}) = x_1x_2.$$

Then:

$$\phi = 1$$

for the negative class,

and

$$\phi = -1$$

for the positive class.

The score

$$f(\mathbf{x}) = -x_1x_2$$

classifies all four points correctly.

Logistic Regression With Nonlinear Features

For binary labels

$$y_p \in \{0, 1\}:$$

score:

$$f_p = \tilde{\phi}(\mathbf{x}_p)^T \mathbf{w}.$$

Probability:

$$p_p = \sigma(f_p).$$

With feature matrix

Φ :

$$\mathbf{f} = \Phi \mathbf{w}.$$

Cross-entropy gradient:

$$\nabla g(\mathbf{w}) = \frac{1}{P} \Phi^T (\mathbf{p} - \mathbf{y}).$$

Same logistic-regression gradient as before.

Only the design matrix changed.

Multiclass Models Work the Same Way

For

K

classes:

class scores:

$$f_k(\mathbf{x}) = \mathbf{w}_k^T \tilde{\boldsymbol{\phi}}(\mathbf{x}).$$

Softmax:

$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}}.$$

Prediction:

$$\hat{y} = \arg \max_k f_k.$$

Pairwise boundary:

$$(\mathbf{w}_j - \mathbf{w}_k)^T \tilde{\boldsymbol{\phi}}(\mathbf{x}) = 0.$$

Linear in transformed coordinates.

Potentially nonlinear in original coordinates.

Fixed Features Versus Learned Features

So far:

we choose

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

before optimization.

Examples:

$$x^2,$$

$$x_1 x_2,$$

$$\sin x.$$

These are

fixed feature maps.

Later:

the transformation itself
will contain trainable parameters.

Then we will

learn the features.

Neural networks are a major example.

For now:

feature map fixed,
coefficients learned.

Basis Functions

A collection

$$\phi_1, \dots, \phi_M$$

is often called a set of

basis functions.

Examples:

polynomials:

$$1, x, x^2, \dots;$$

trigonometric:

$$\sin x, \cos x, \sin 2x, \cos 2x, \dots;$$

localized functions:

$$e^{-(x-c_j)^2/\gamma^2}.$$

The fitted model forms a weighted combination:

$$f(x) = \sum_j w_j \phi_j(x).$$

Choosing the basis determines

the family of shapes the model can express.

A Localized Feature

Consider

$$\phi_c(x) = e^{-(x-c)^2/\gamma^2}.$$

This feature is largest near

$$x = c.$$

Far from

c :

the feature becomes small.

Several centers

$$c_1, \dots, c_M$$

create a collection of localized responses.

A linear combination of such functions
can produce flexible nonlinear curves.

This idea will reappear later
in kernel methods.

For now:
the centers and widths are fixed by us.

Feature Count Can Grow Quickly

For
 N
original variables,

all polynomial terms of total degree at most
 d
produce
 $\binom{N+d}{d}$
features including the constant.

Example:
 $N = 10,$
 $d = 2.$

Number of features:
 $\binom{12}{2}$
 $= 66.$

With
 $N = 20,$
 $d = 3:$
 $\binom{23}{3}$
 $= 1771.$

Nonlinear feature expansion
can create high-dimensional models very quickly.

Why Feature Expansion Can Become Dangerous

More features can reduce training error.

But they also create:

more parameters;

greater sensitivity to noise;

stronger multicollinearity;

larger computational cost;

greater opportunity to fit accidental structure.

Thus:

more expressive $\neq$ automatically better.

Regularization helps control large feature sets.

Validation helps decide how much complexity
is actually useful.

Scaling Polynomial Features

Suppose

$$x$$

is numerically large.

Then

$$x^2, x^3, \dots$$

may become extremely large.

Example:

$$x = 100.$$

Then

$$x^2 = 10,000,$$

$$x^3 = 1,000,000.$$

This can create poor numerical conditioning.

A common strategy:

center and scale the original inputs
before creating high-degree polynomial features.

Feature preparation and nonlinear feature engineering
should be designed together.

A Subtle Point About Interpretation

Suppose a quadratic model contains

$$x$$

and

$$x^2.$$

The coefficient of

$$x$$

is no longer simply

“the change in prediction for a one-unit increase in x .”

Instead:

$$f(x) = w_0 + w_1x + w_2x^2.$$

Then

$$f'(x) = w_1 + 2w_2x.$$

The local effect of

x

depends on

x itself.

Nonlinear features change coefficient interpretation

as well as predictive shape.

A Feature Map Is a Modeling Assumption

Choosing

$$\phi(x) = x^2$$

asserts that quadratic structure

is worth considering.

Choosing

$$\phi(x) = \sin x$$

asserts that periodic structure

may be useful.

Choosing

$$x_1x_2$$

asserts that interaction

may matter.

Feature engineering is therefore

not merely algebraic manipulation.

It encodes hypotheses

about useful structure in the data.

Do Not Use the Test Set to Design Features

Suppose we inspect the final test set

and notice an unexpected curved pattern.

Then we add

$$x^2$$

because it improves test performance.

The test set has now influenced model design.

It is no longer a clean final evaluation set.

Feature-map choices,
polynomial degree,
interaction choices,
and regularization strength
should be selected
using training / validation procedures.

The final test set remains untouched.

Polynomial Features in Python

For one-dimensional data:

```
import numpy as np

x = np.array([-2., -1., 0., 1., 2.])

Phi = np.column_stack([
    np.ones(len(x)),
    x,
    x**2
])
```

New Python idea:

np
combines one-dimensional arrays
as columns of a matrix.

Here:

Φ
contains
1,
 x ,

and

x^2 .

Fit the Quadratic Model in Python

```
y = np.array([4., 1., 0., 1., 4.])

w = np.linalg.lstsq(
    Phi,
    y,
    rcond=None
)[0]

print(w)
```

Output:

```
[0. 0. 1.]
```

Predictions:

```
y_hat = Phi @ w

mse = np.mean((y_hat - y)**2)
```

Result:
 $mse = 0$
up to floating-point roundoff.
Same least-squares solver.
Richer design matrix.

Build Quadratic Features in Two Variables

Suppose
 $\mathbf{x}$
has two columns:
 x_1
and
 x_2 .

```

x1 = X[:, 0]
x2 = X[:, 1]

Phi = np.column_stack([
    np.ones(len(X)),
    x1,
    x2,
    x1**2,
    x1*x2,
    x2**2
])

```

The feature matrix contains
all monomials
of total degree at most 2.
A linear coefficient vector applied to
 Φ
produces a quadratic surface
in the original variables.

An XOR Feature Map in Python

```

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

x1 = X[:, 0]
x2 = X[:, 1]

interaction = x1 * x2

score = -interaction

pred = np.where(
    score > 0,
    1,
    -1
)

```

```
)  
  
print(pred)
```

Output:

```
[-1  1  1 -1]
```

One nonlinear interaction feature

transforms a non-linearly-separable pattern
into a trivial one-dimensional separation.

A Reusable Polynomial Feature Function

```
def quadratic_features(X):  
    x1 = X[:, 0]  
    x2 = X[:, 1]  
  
    return np.column_stack([  
        np.ones(len(X)),  
        x1,  
        x2,  
        x1**2,  
        x1*x2,  
        x2**2  
    ])
```

Then:

```
Phi_train = quadratic_features(X_train)  
Phi_test  = quadratic_features(X_test)
```

Important:

use the same feature-map definition
for training and future data.

The model's learned coefficients

correspond to a specific transformed coordinate system.

The Nonlinear-Feature Pipeline

Raw input:

$\mathbf{x}$.

Feature transformation:

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

Linear score:

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

Then:

regression:

use score as prediction;

binary classification:

pass score through sigmoid;

multiclass classification:

build one score per class and use softmax.

The same transformed features

can support many learning tasks.

What Nonlinear Features Do, and Do Not, Solve

They can solve:

insufficient expressiveness of a raw linear model.

They do not automatically solve:

overfitting;

poor feature scaling;

data leakage;

missing data;

inappropriate loss functions;

bad validation procedures.

Expressiveness is one part

of a complete learning system.

What did we learn?

- ◆ A nonlinear feature map transforms $\mathbf{x}$ into a new vector $\phi(\mathbf{x})$ before fitting the model.
- ◆ A model can be nonlinear in the original input while remaining linear in its trainable coefficients.
- ◆ Once the transformed design matrix Φ is built, least-squares and logistic-regression formulas are unchanged except that X is replaced by Φ .
- ◆ Polynomial and interaction features create curved regression surfaces and nonlinear classification boundaries in the original input space.
- ◆ Feature expansion can grow combinatorially, increasing conditioning problems and the risk of fitting accidental training structure.
- ◆ Fixed nonlinear features increase expressiveness; choosing how much expressiveness is useful requires validation and a study of generalization.



Next

- Nonlinear feature maps can make a model arbitrarily more expressive.
- Training error therefore tends to decrease as the feature family becomes richer.
- But better training fit does not guarantee better prediction on new observations.
- Next we study underfitting, overfitting, and the mathematical idea of generalization.

- **1.29 - Model Complexity and Generalization**

Prepared by Dr. Jodin Morey.

Materials for Other Courses Found at MathTalker.org