

Machine Learning

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

Previous Lecture

- ♦ k -means represents each observation by one of K centroid vectors and minimizes within-cluster squared distance.
- ♦ With centroids fixed, the optimal assignment sends each observation to its nearest centroid.
- ♦ With assignments fixed, the optimal centroid is the arithmetic mean of the observations assigned to that cluster.
- ♦ Alternating the assignment and centroid steps cannot increase the k -means objective, but it need not find the global minimum.
- ♦ Fixed centroids partition space into convex Voronoi cells separated by perpendicular-bisector hyperplanes.
- ♦ Initialization, feature scaling, outliers, and the choice of K can materially change the clustering.



§9.1, 9.3, 9.4 - Preparing Features

So far:

we usually treated the input features
as already ready for learning.

But an algorithm sees only numbers.

Their units,

scales,

coding,

and preprocessing

determine the geometry of the problem.

Today:

prepare the feature representation
before fitting the model.

The Feature Matrix

Store observations in the rows of

$$X = \begin{bmatrix} \text{---} \mathbf{x}_1^T \text{---} \\ \vdots \\ \text{---} \mathbf{x}_p^T \text{---} \end{bmatrix} \in \mathbb{R}^{P \times N}.$$

Each column is a feature.

Examples:

- height;
- income;
- temperature;
- age;
- a categorical indicator.

Changing a column of X
changes what the learning algorithm sees.

Two Different Reasons to Transform Features

Feature transformations can serve at least two distinct purposes.

Numerical purpose:

- improve conditioning
- and make optimization easier.

Modeling purpose:

- redefine what counts as
- a large or small difference between observations.

These are not the same.

For some models,

- scaling is mostly a reparameterization.

For others,

- scaling changes the actual fitted geometry.

We must know which situation we are in.

Centering a Feature

For feature j :

$$\mu_j = \frac{1}{P} \sum_{p=1}^P x_{pj}.$$

Centered value:

$$x_{pj}^c = x_{pj} - \mu_j.$$

Then

$$\frac{1}{P} \sum_{p=1}^P x_{pj}^c = 0.$$

Centering changes the origin
but not differences between observations:

$$\begin{aligned} &(x_a - \mu) - (x_b - \mu) \\ &= x_a - x_b. \end{aligned}$$

So centering alone preserves pairwise distances.

Standardization

Let

$$s_j > 0$$

measure the scale of feature j .

A common choice is

$$s_j = \sqrt{\frac{1}{P} \sum_{p=1}^P (x_{pj} - \mu_j)^2}.$$

Then define

$$z_{pj} = \frac{x_{pj} - \mu_j}{s_j}.$$

The standardized feature has:

mean 0;

variance 1

under the same denominator convention.

This produces dimensionless coordinates.

Standardization Is an Affine Change of Coordinates

For one feature:

$$z = \frac{x - \mu}{s}.$$

Invert:

$$x = \mu + sz.$$

Thus standardization is one-to-one

whenever

$$s > 0.$$

No information is lost

merely by changing coordinates.

But:

a learning algorithm may treat the new coordinate system differently.

That depends on its objective and regularization.

Linear Regression Under a Change of Scale

Suppose

$$\hat{y} = w_0 + w_1 x.$$

Write

$$x = \mu + sz.$$

Then

$$\begin{aligned}\hat{y} &= w_0 + w_1(\mu + sz) \\ &= (w_0 + w_1\mu) + (sw_1)z.\end{aligned}$$

So in standardized coordinates:

$$\tilde{w}_0 = w_0 + w_1\mu,$$

and

$$\tilde{w}_1 = sw_1.$$

Same predictions.

Different coefficients.

Several Features

Suppose

$$\hat{y} = w_0 + \sum_{j=1}^N w_j x_j.$$

With

$$\begin{aligned}x_j &= \mu_j + s_j z_j. \\ \hat{y} &= w_0 + \sum_j w_j \mu_j + \sum_j (s_j w_j) z_j.\end{aligned}$$

Therefore

$$\tilde{w}_0 = w_0 + \sum_j w_j \mu_j$$

and

$$\tilde{w}_j = s_j w_j.$$

For an unregularized linear model,
affine rescaling of features
does not change the set of predictions the model can represent.

It changes the parameterization.

Why Scaling Can Still Matter for Optimization

Even if the fitted prediction problem is equivalent,
gradient descent operates in parameter coordinates.

Badly scaled features
can create highly unequal curvature.

Example:
use two centered predictors

p	x_1	x_2
1	-1	-100
2	-1	100
3	1	-100
4	1	100

The columns are orthogonal,
but their scales differ by a factor of 100.

The Hessian Before Scaling

For least squares with these two centered predictors:

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

The Hessian is

$$H = \frac{2}{P} X^T X.$$

Here:

$$X^T X = \begin{bmatrix} 4 & 0 \\ 0 & 40000 \end{bmatrix}.$$

With

$$P = 4:$$

$$H = \begin{bmatrix} 2 & 0 \\ 0 & 20000 \end{bmatrix}.$$

Condition number:

$$\kappa(H) = 10000.$$

Gradient descent must accommodate
very different curvature scales.

After Standardizing the Second Feature

Define

$$z_2 = x_2/100.$$

Now both predictor columns contain only
 ± 1 .

The new design matrix satisfies

$$\begin{aligned} \tilde{X}^T \tilde{X} \\ = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}. \end{aligned}$$

Therefore

$$\tilde{H} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}.$$

Condition number:

$$\kappa(\tilde{H}) = 1.$$

The prediction problem is only reparameterized.

But the optimization geometry is dramatically improved.

Connection to Lesson 1.6

We previously studied

$$g(w_1, w_2) = w_1^2 + 100w_2^2.$$

Gradient descent zig-zagged

because curvature was very different
in different directions.

Feature scaling can create

or remove exactly this kind of anisotropy.

So preprocessing is connected directly to

condition number,
step size,
and convergence rate.

This is not merely cosmetic data cleaning.

Distance-Based Methods Are Different

For k -means:

$$\begin{aligned} & \| \mathbf{x} - \boldsymbol{\mu} \|^2 \\ &= \sum_j (x_j - \mu_j)^2. \end{aligned}$$

After standardization:

$$\begin{aligned} & \| \mathbf{z} - \boldsymbol{\nu} \|^2 \\ &= \sum_j \left(\frac{x_j - \mu_j}{s_j} \right)^2. \end{aligned}$$

Thus standardization is equivalent to

a weighted distance in the original coordinates:

$$\sum_j \frac{1}{s_j^2} (x_j - \mu_j)^2.$$

This is a different geometry.

It can change cluster assignments.

A Distance Example

Compare two observations:

$$\mathbf{x}_A = (0, 0)$$

and

$$\mathbf{x}_B = (1, 10).$$

Original squared distance:

$$\begin{aligned} &1^2 + 10^2 \\ &= 101. \end{aligned}$$

Now rescale feature 2 by

$$z_2 = x_2/10.$$

Then:

$$\begin{aligned} \mathbf{z}_A &= (0, 0), \\ \mathbf{z}_B &= (1, 1). \end{aligned}$$

New squared distance:

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

So scaling changes the meaning of closeness.

For k -means,
that changes the model itself.

PCA Is Also Scale Sensitive

PCA chooses directions
with large variance.

Multiplying feature j by
 a
multiplies its variance by
 a^2 .

Therefore a large-scale feature
can dominate PCA directions.

PCA on standardized variables
is closely related to PCA based on the
correlation matrix
rather than the covariance matrix.

Again:
this is a substantive change in geometry,
not merely a numerical convenience.

Units Matter

Suppose one feature is height.

Measure it in meters:

1.75.

Or centimeters:

175.

The physical information is identical.

But Euclidean squared distance changes by a factor of

100^2

in that coordinate.

A method should not accidentally depend
on an arbitrary choice of units
unless those units are intentionally part of the geometry.

This is one motivation for standardization.

Standardization Is Not Always Appropriate

Suppose feature 1 naturally varies
much more than feature 2
for scientifically meaningful reasons.

Standardizing them forces both
to have equal variance.

That may overemphasize
a feature whose small variation
is mostly measurement noise.

So the question is not:
“Should we always standardize?”

The correct question is:
“What geometry should differences in these features represent?”

Min–Max Scaling

Another transformation:

$$z = \frac{x - x_{\min}}{x_{\max} - x_{\min}}.$$

Training values are mapped approximately to
 $[0, 1]$.

Advantages:
bounded training range;

easy interpretation.

Limitations:
highly sensitive to extreme minima and maxima;

future observations may fall outside $[0, 1]$;

changes distance geometry just as standardization does.

No single scaling rule is universally best.

Outliers Can Distort Mean and Standard Deviation

Suppose a feature contains

1, 2, 2, 3, 100.

The large outlier changes
the mean and standard deviation substantially.

Possible robust alternatives:

center by the median;

scale by an interquartile-range quantity.

But:
robust scaling also changes the coordinate geometry.

The appropriate transformation depends on
the data-generating process
and the learning objective.

Categorical Features Are Not Ordinary Numbers

Suppose a feature records color:

{red, blue, green}.

Encoding

red = 1,

blue = 2,

green = 3

creates artificial numerical structure.

It implies:

blue lies halfway between red and green;

green is three times red;

differences have numerical magnitude.

Usually none of these statements is intended.

We need a representation appropriate for categories.

One-Hot Encoding of a Categorical Feature

For three unordered categories:

Category	Red	Blue	Green
Red	1	0	0
Blue	0	1	0
Green	0	0	1

Now no category is numerically “between” the others.

This is one-hot encoding applied to an input feature.

Earlier:
we used one-hot encoding for multiclass targets.

Same representation idea.

Different role.

The Dummy-Variable Trap

Suppose a linear regression model includes:

an intercept column
 $\mathbf{1}$,

and all three indicator columns:
 R, B, G .

Every observation belongs to exactly one category.

Therefore
 $R + B + G = \mathbf{1}$.

The columns are linearly dependent.

Thus the design matrix is rank deficient.

This is the same multicollinearity issue we studied in least squares.

Reference-Category Coding

With an intercept,
omit one indicator column.

For example:
use only

B
and
 G .

Then red is the reference category.

Model:

$$\hat{y} = w_0 + w_B B + w_G G.$$

Predictions:

red:

$$\hat{y} = w_0;$$

blue:

$$\hat{y} = w_0 + w_B;$$

green:

$$\hat{y} = w_0 + w_G.$$

Thus coefficients represent differences
from the reference category.

Ordinal Categories Are Different

Some categories have a natural order.

Example:

small < medium < large.

An ordinal encoding may preserve the order.

But assigning

1, 2, 3

also assumes equal spacing.

That assumption may or may not be appropriate.

So we must distinguish:

nominal categories:

no natural order;

ordinal categories:

order exists,

but numerical spacing may still be unknown.

Missing Values Need an Explicit Policy

Suppose a feature value is missing.

Do not silently replace it with 0 unless 0 has the intended meaning.

Possible strategies include:

- remove observations or features;
- impute a value;
- use a model that handles missingness;
- include an indicator recording that a value was missing.

Every choice adds assumptions.

Mean imputation, for example, changes variances and pairwise distances.

There is no universally neutral treatment of missing data.

A Critical Rule: Fit Preprocessing on the Training Data

Suppose we split the data into:

- training set;
- validation set;
- test set.

Compute preprocessing quantities using the training set only.

Examples:

- training mean;
- training standard deviation;
- training minimum and maximum;
- categories observed during fitting;
- imputation values.

Then apply those fixed transformations to validation and test observations.

Why? Data Leakage

Suppose we use test observations to calculate the mean or scale before fitting the model.

Then information from the held-out data
has entered the training procedure.

This is called
data leakage.

The final test performance
is no longer a clean estimate
of performance on unseen data.

The test set must remain outside
all fitting decisions,
including preprocessing estimates.

A Simple Train/Test Standardization Example

Suppose the training feature has:

$$\mu_{\text{train}} = 10,$$

and

$$s_{\text{train}} = 2.$$

A test observation has

$$x_{\text{test}} = 14.$$

Transform it using the training quantities:

$$\begin{aligned} z_{\text{test}} &= \frac{14-10}{2} \\ &= 2. \end{aligned}$$

Do not recompute a new mean or scale
from the test set.

The test point must be mapped
into the coordinate system learned from training data.

Cross-Validation Requires the Same Discipline

During K -fold cross-validation:

each fold temporarily acts as validation data.

Therefore for each fold:

1. compute preprocessing parameters
from the fold's training portion;
2. transform that training portion;
3. transform the held-out fold
using the same fitted preprocessing;

4. fit and evaluate.

Preprocessing belongs inside
the cross-validation loop.

We will formalize cross-validation later.

Feature Preparation Can Affect Regularization

Suppose later we penalize

$$w_1^2 + w_2^2.$$

If one feature is rescaled,
the corresponding coefficient rescales inversely
to preserve predictions.

Therefore the numerical penalty on that coefficient changes.

So once regularization is introduced:
feature scaling is no longer
merely a harmless parameter re-labeling.

The penalty and scaling interact.

This is one reason standardized predictors
are common with coefficient penalties.

Next lesson:
regularization and feature selection.

Do Not Transform the Target Accidentally

In supervised learning:
predictors are stored in
 X .

The response is
 y .

A feature transformation
should not automatically be applied
to y .

Sometimes the target is deliberately transformed,
but that is a separate modeling decision
because it changes the loss interpretation
and prediction scale.

Keep predictor preprocessing
and target transformation conceptually distinct.

A Useful Preprocessing Checklist

For each feature ask:

What does the feature mean?

What are its units?

Is its scale arbitrary or meaningful?

Is it numeric,
categorical,
or ordinal?

Are there missing values?

Are there extreme outliers?

Will the model use distances,
coefficient penalties,
or gradient optimization?

Can every preprocessing parameter
be estimated using training data only?

Feature preparation should be tied
to the mathematics of the model.

Center and Scale in Python

```
import numpy as np

X_train = np.array([[1., 100.],
                    [2., 120.],
                    [3., 140.],
                    [4., 160.]])

mu = np.mean(X_train, axis=0)

scale = np.sqrt(
    np.mean(
        (X_train - mu)**2,
        axis=0
    )
)

Z_train = (X_train - mu) / scale
```

The arrays
mu
and
scale
are fitted preprocessing parameters.

Store them.

Do not recompute them on test data.

Transform New Data With the Training Parameters

```
X_test = np.array([[5., 180.],
                   [6., 200.]])

Z_test = (X_test - mu) / scale
```

Same
mu.

Same
scale.

This places training and test observations
in the same feature coordinate system.

A preprocessing transform should therefore have
a **fit** stage
and a **transform** stage.

Check the Standardized Training Data

```
print(np.mean(Z_train, axis=0))

print(
    np.mean(
        Z_train**2,
        axis=0
    )
)
```

The first result is approximately
[0. 0.]

The second is approximately
[1. 1.]

under our denominator-*P* convention.

So the code matches the mathematical definition.

A Reusable Python Pattern

```
def fit_standardizer(X):
    mu = np.mean(X, axis=0)

    scale = np.sqrt(
        np.mean(
            (X - mu)**2,
            axis=0
        )
    )
```

```

    )

    return mu, scale

def transform_standardizer(X, mu, scale):
    return (X - mu) / scale

```

Use:

```

mu, scale = fit_standardizer(X_train)

Z_train = transform_standardizer(
    X_train,
    mu,
    scale
)

Z_test = transform_standardizer(
    X_test,
    mu,
    scale
)

```

This coding pattern helps prevent leakage.

Constant Features Need Special Handling

Suppose one feature is constant
on the training data.

Then

$$s_j = 0.$$

Standardization would divide by zero.

A constant feature contains no variation
within the training sample.

For many models:

it contributes no useful distinction among observations.

Typical action:

remove it
or handle it explicitly.

Do not silently divide by zero.

One-Hot Features in Python

Suppose categories are coded as
0,1,2.

Use the identity matrix:

```
category = np.array([0, 2, 1, 0])
```

```
one_hot = np.eye(3)[category]

print(one_hot)
```

Output:

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

For a regression design matrix with an intercept,
one indicator column would typically be omitted
to avoid exact linear dependence.

Scaling and Conditioning in Python

Use the badly scaled predictor matrix:

```
X = np.array([[-1., -100.],
              [-1.,  100.],
              [ 1., -100.],
              [ 1.,  100.]])

H = 2 * X.T @ X / len(X)

print(H)
print(np.linalg.cond(H))
```

Output:

```
[[2.e+00  0.e+00]
 [0.e+00  2.e+04]]

10000.0
```

After dividing the second column by 100:

```
Z = X.copy()
Z[:, 1] = Z[:, 1] / 100

H_scaled = 2 * Z.T @ Z / len(Z)

print(np.linalg.cond(H_scaled))
```

Output:

```
1.0
```

Feature scaling can radically improve optimization conditioning.

What We Are Not Doing Yet

So far:

- transformations mostly changed
- centering,
- scale,
- and coding.

Soon we will create

genuinely new predictive features:

polynomials;

interactions;

nonlinear basis functions.

Those transformations enlarge
the class of functions the model can represent.

That is different from simple rescaling.

We will treat nonlinear feature maps
in a later lesson.

What did we learn?

- ◆ Centering changes the origin; standardization also rescales each feature by a chosen measure of spread.
- ◆ For an unregularized linear model, affine feature scaling can be absorbed into the coefficients without changing the model.
- ◆ Scaling can dramatically improve gradient-descent conditioning even when it does not change the underlying unreduced data.
- ◆ For distance-based methods such as k -means and PCA, scaling changes the geometry and can change the fitted model.
- ◆ Unordered categorical features require a representation such as one-hot encoding; with an intercept, redundant in the model.
- ◆ All fitted preprocessing quantities must be estimated from training data only and then reused unchanged on validation data.

Next

- Preparing features changes the coordinate system in which a model is trained.
- The next question is which features or coefficients the model should be allowed to use strongly.
- Penalties can stabilize models, control complexity, and sometimes drive coefficients toward zero.
- **1.27 - Regularization and Feature Selection**