

Machine Learning

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

Previous Lecture

- ♦ Weighted least squares gives different observations different influence through positive weights β_p .
- ♦ The weighted normal equations are $X^T W X \mathbf{w} = X^T W \mathbf{y}$.
- ♦ Weighted least squares is ordinary least squares after multiplying each row by the square root of its weight.
- ♦ Independent Gaussian noise with common variance leads to ordinary least squares by maximum likelihood.
- ♦ Unequal known Gaussian variances lead to weights $\beta_p = 1/\sigma_p^2$.
- ♦ Probabilistic regression describes both a predicted center and uncertainty around that prediction.



§6.1 - From Regression to Classification

Regression:

predict a numerical response.

Now change the question.

Suppose the output is a category:

spam / not spam,
disease / no disease,
pass / fail.

This is **classification**.

Binary Classification

Start with two classes.

We will encode them as

$$y_p \in \{-1, +1\}.$$

The symbols -1 and $+1$

are labels,
not numerical quantities we are trying to approximate.

Another common encoding is

0 and 1.

For now:

$\{-1, +1\}$ makes the algebra especially clean.

We Still Begin With a Linear Function

For one input:

$$f(x; \mathbf{w}) = w_0 + w_1x.$$

For several inputs:

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1x_1 + \cdots + w_Nx_N.$$

But now:

f is not itself the class label.

It is a **score**.

Positive score:

evidence for class $+1$.

Negative score:

evidence for class -1 .

Turn the Score Into a Class

Use the decision rule

$$\hat{y} = \text{sign}(f(\mathbf{x}; \mathbf{w})).$$

Thus

$$f(\mathbf{x}; \mathbf{w}) > 0 \Rightarrow \hat{y} = +1,$$

and

$$f(\mathbf{x}; \mathbf{w}) < 0 \Rightarrow \hat{y} = -1.$$

At

$$f(\mathbf{x}; \mathbf{w}) = 0,$$

the point lies exactly on the decision boundary.

A tie convention can assign either class.

A One-Dimensional Example

Suppose

x	1	2	3	4	5	6
y	-1	-1	-1	+1	+1	+1

Choose the score

$$f(x) = x - 3.5.$$

Then

x	1	2	3	4	5	6
$f(x)$	-2.5	-1.5	-0.5	0.5	1.5	2.5
$\hat{y}$	-1	-1	-1	+1	+1	+1

All six observations are classified correctly.

The Decision Boundary

The class changes when

$$f(x) = 0.$$

For

$$f(x) = x - 3.5:$$

$$x - 3.5 = 0.$$

Therefore

$$x = 3.5.$$

The score is a line.

The classifier uses only which side of zero the score lies on.

Three Different Objects

Keep these separate.

Score:

$$f(\mathbf{x}; \mathbf{w}).$$

Predicted class:

$$\text{sign}(f).$$

Decision boundary:

$$f(\mathbf{x}; \mathbf{w}) = 0.$$

For regression:

the model output itself was the numerical prediction.

For classification:

the score is an intermediate quantity.

Several Inputs

Write

$$f(\mathbf{x}; \mathbf{w}) = w_0 + \mathbf{v}^T \mathbf{x},$$

where

$$\mathbf{v} = \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix}.$$

The decision boundary is

$$w_0 + \mathbf{v}^T \mathbf{x} = 0.$$

In two dimensions:

a line.

In three dimensions:

a plane.

In N dimensions:

a hyperplane.

A Two-Dimensional Boundary

Let

$$f(x_1, x_2) = -3 + x_1 + x_2.$$

Boundary:

$$-3 + x_1 + x_2 = 0.$$

So

$$x_2 = 3 - x_1.$$

Point (1, 1):

$$f = -1,$$

class -1 .

Point (2, 2):

$$f = 1,$$

class $+1$.

The line separates the two prediction regions.

The Weight Vector Is Normal to the Boundary

For

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

the boundary consists of all $\mathbf{x}$ satisfying

$$\mathbf{v}^T \mathbf{x} = -w_0.$$

If $\mathbf{x}_A$ and $\mathbf{x}_B$

are both on the boundary:

$$\mathbf{v}^T \mathbf{x}_A = \mathbf{v}^T \mathbf{x}_B.$$

Therefore

$$\mathbf{v}^T (\mathbf{x}_A - \mathbf{x}_B) = 0.$$

So $\mathbf{v} \perp$ decision boundary.

The coefficients determine the orientation of the separating hyperplane.

When Is an Observation Classified Correctly?

Suppose $y_p \in \{-1, +1\}$.

Correct classification means:

if $y_p = +1$, then $f_p > 0$;

if $y_p = -1$, then $f_p < 0$.

Both cases can be written as

$$y_p f(\mathbf{x}_p; \mathbf{w}) > 0.$$

Wrong classification:

$$y_p f(\mathbf{x}_p; \mathbf{w}) < 0.$$

One formula handles both classes.

The Signed Score

Define $m_p = y_p f(\mathbf{x}_p; \mathbf{w})$.

This is often called a **functional margin**.

$m_p > 0$:
correct side of the boundary.

$m_p < 0$:
wrong side.

Large positive m_p :
score is strongly on the correct side.

For our one-dimensional example:

x	1	2	3	4	5	6
m	2.5	1.5	0.5	0.5	1.5	2.5

The closest observations to the boundary are $x = 3$ and $x = 4$.

Scale Matters for the Functional Margin

The score

$$f(x) = x - 3.5$$

and the score

$$\tilde{f}(x) = 10(x - 3.5)$$

produce exactly the same classifications.

They also have exactly the same boundary.

But every functional margin is multiplied by 10.

Therefore

$$y_p f_p$$

is not itself a geometric distance.

To measure signed distance to the hyperplane:

$$\gamma_p = \frac{y_p(w_0 + \mathbf{v}^T \mathbf{x}_p)}{\|\mathbf{v}\|}.$$

This quantity is unchanged if all coefficients are multiplied by the same positive constant.

Classification Error

The most direct question is:
was the class correct?

Define the zero-one loss

$$\ell_{0/1}(y, f) = \begin{cases} 0, & yf > 0, \\ 1, & yf \leq 0. \end{cases}.$$

Then classification error rate is

$$E(\mathbf{w}) = \frac{1}{P} \sum_{p=1}^P \ell_{0/1}(y_p, f(\mathbf{x}_p; \mathbf{w})).$$

This directly counts the fraction misclassified.

Why Not Just Minimize Classification Error?

Zero-one loss matches the final goal well.

But as a function of the parameters:

- it is flat over large regions;
- it jumps when an observation crosses the boundary;
- it is not differentiable at those jumps.

Gradient descent has little useful information.

Newton's method is not appropriate.

We want a smoother objective
that still encourages correct classification.

Could We Use Least Squares on the Labels?

We could pretend the labels are numerical targets:

$$y_p \in \{-1,+1\},$$

and minimize

$$\sum_p (f_p - y_p)^2.$$

Mathematically this is possible.

But the loss asks the score to be numerically close to -1 or $+1$.

Classification only needs the score to have the correct sign.

Those are different goals.

A Correct Prediction Can Have Large Squared Error

Suppose

$$y = +1.$$

Compare three scores:

Score f	0.2	1	4
Predicted class	+1	+1	+1
Squared loss $(f - y)^2$	0.64	0	9

All three classifications are correct.

But squared loss heavily penalizes the score 4.

Why?

Because regression loss cares about closeness to the number 1.

Classification cares primarily about which side of zero we are on.

Least-Squares Classification on Our Example

Fit a regression line to

$$x = (1,2,3,4,5,6),$$

$$y = (-1,-1,-1,+1,+1,+1).$$

Least squares gives

$$f(x) = -1.8 + \frac{18}{35}x$$

since

$$\frac{18}{35} \approx 0.5143.$$

Its zero crossing is

$$-1.8 + \frac{18}{35}x = 0,$$

so

$$x = 3.5.$$

Here least squares happens to produce a perfect classification boundary.

But it did so by solving a regression problem,
not by using a classification-specific loss.

What Else Do We Want From a Classifier?

A hard prediction gives only

$$\hat{y} = -1$$

or

$$\hat{y} = +1.$$

But often we also want:
how certain is the prediction?

A score such as

$$f = 7$$

is larger than

$$f = 0.1.$$

But neither is automatically a probability.

Scores are unbounded:

$$-\infty < f < \infty.$$

Probabilities must satisfy:

$$0 \leq p \leq 1.$$

We need a transformation.

What Should a Probability Transformation Do?

We want a function

$$\sigma(f)$$

such that

$$-\infty < f < \infty$$

maps to

$$0 < \sigma(f) < 1.$$

Also:

very negative score $\Rightarrow$ probability near 0;

score 0 $\Rightarrow$ probability 1/2;

very positive score $\Rightarrow$ probability near 1.

Next lesson:
the logistic sigmoid.

Classification in Python

Enter the binary data:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1., 2., 3., 4., 5., 6.])
y = np.array([-1, -1, -1, 1, 1, 1])
```

Define a score:

```
def score(x):
    return x - 3.5
```

Turn scores into classes:

```
scores = score(x)

y_pred = np.where(scores >= 0, 1, -1)
```

New Python idea:

np
chooses values according to a condition.

Compute Classification Error in Python

Compare predictions with labels:

```
wrong = (y_pred != y)
wrong
```

Output:

```
[False False False False False False]
```

Then

```
error_rate = np.mean(wrong)
```

gives

0.

New Python idea:

comparisons create Boolean arrays.

In NumPy:

```
False=0,
True=1
```

when averaged numerically.

So the mean of the Boolean array
is the fraction misclassified.

Compute Margins in Python

Functional margins:

```
margins = y * scores
margins
```

Output:

```
[2.5 1.5 0.5 0.5 1.5 2.5]
```

Positive margin:

correct classification.

Negative margin:

misclassification.

This multiplication by the label

will appear repeatedly in classification losses.

Plot the One-Dimensional Boundary

```
plt.scatter(x, y)
plt.axvline(3.5)

plt.xlabel("x")
plt.ylabel("Class label")
plt.show()
```

New Python idea:

```
plt
```

draws a vertical decision boundary.

The classes are discrete,

but the boundary still lives in the input space.

Regression Versus Classification

	Regression	Classification
Output	numerical	categorical
Linear quantity	prediction	score
Final rule	use f	use sign / probability
Simple loss	squared error	classification-specific loss
Boundary	not central	$f(\mathbf{x}) = 0$

The linear algebra is familiar.

The interpretation of the model output has changed.

What did we learn?

- ♦ Binary classification predicts one of two class labels rather than a numerical response.
- ♦ A linear classifier first produces a real-valued score $f(\mathbf{x}; \mathbf{w})$.
- ♦ The sign of the score gives the predicted class, and $f(\mathbf{x}; \mathbf{w}) = 0$ defines the decision boundary.
- ♦ The product $y_P f_P$ is positive for a correctly classified observation and negative for a misclassified one.
- ♦ Zero-one loss directly counts classification mistakes but is difficult to optimize with gradient methods.
- ♦ Least-squares loss is not naturally aligned with classification, motivating a new smooth classification loss.



Next

- A linear classifier produces an unbounded real-valued score.
- We want to convert that score into a probability between 0 and 1.
- The logistic sigmoid provides exactly such a transformation.
- **1.15 - Logistic Regression**