

Machine Learning

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

Previous Lecture

- ♦ A binary confusion matrix records true positives, false positives, false negatives, and true negatives.
- ♦ Accuracy can be misleading when the classes are strongly imbalanced.
- ♦ Recall measures the fraction of actual positives detected; specificity measures the fraction of actual negatives correctly rejected.
- ♦ Precision measures the fraction of predicted positives that are actually positive.
- ♦ F_1 combines precision and recall, while balanced accuracy combines recall and specificity.
- ♦ Changing the classification threshold trades false positives against false negatives; ROC and precision–recall curves summarize this behavior across thresholds.



§7.1 thru 7.4 - Multiclass Classification

Binary classification:

two possible labels.

Many problems have more:

handwritten digit 0, ..., 9;

species A, B, or C;

image: cat, dog, bird, or horse.

Today:

extend linear classification to

$K \geq 3$ classes.

Class Labels

Let

$$y_p \in \{1, 2, \dots, K\}.$$

For each input $\mathbf{x}_p$,

we want one predicted class $\hat{y}_p$.

Binary classification needed one score

because the sign separated two classes.

With K classes:

one sign is no longer enough.

We will assign one score to each class.

One Linear Score Per Class

For class k define

$$f_k(\mathbf{x}) = w_{k0} + \mathbf{v}_k^T \mathbf{x},$$

for $k = 1, \dots, K$.

Given one input,

the model produces K real numbers:

$$f_1(\mathbf{x}), \dots, f_K(\mathbf{x}).$$

Interpretation:

larger score $\Rightarrow$ stronger preference for that class.

The scores are not yet probabilities.

Winner-Take-All Prediction

Predict the class with the largest score:

$$\hat{y} = \operatorname{argmax}_{k=1, \dots, K} f_k(\mathbf{x}).$$

Example:

Class	1	2	3
Score	0.4	2.1	-0.7

Largest score: 2.1.

Therefore $\hat{y} = 2$.

We care about the ordering of the scores.

Score Vector

Collect the K scores:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_K(\mathbf{x}) \end{bmatrix}.$$

Using augmented input

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix},$$

let each row of W contain one class parameter vector.

Then

$$\mathbf{f}(\mathbf{x}) = W\tilde{\mathbf{x}}.$$

One matrix multiplication produces every class score.

A Three-Class Example

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

Define

$$\begin{aligned}f_1 &= 2 - x_1 - x_2, \\f_2 &= x_1, \\f_3 &= x_2.\end{aligned}$$

Thus

$$W = \begin{bmatrix} 2 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

with

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}.$$

Classify Three Points

At $(0,0)$:

$$\begin{aligned}\mathbf{f} &= (2, 0, 0)^T, \\ \text{so } \hat{y} &= 1.\end{aligned}$$

At $(3,0)$:

$$\begin{aligned}\mathbf{f} &= (-1, 3, 0)^T, \\ \text{so } \hat{y} &= 2.\end{aligned}$$

At $(0,3)$:

$$\begin{aligned}\mathbf{f} &= (-1, 0, 3)^T, \\ \text{so } \hat{y} &= 3.\end{aligned}$$

Each class wins in a different region of the plane.

Where Is the Boundary Between Two Classes?

The boundary between classes j and k
occurs when their scores tie:

$$f_j(\mathbf{x}) = f_k(\mathbf{x}).$$

Equivalently:

$$f_j(\mathbf{x}) - f_k(\mathbf{x}) = 0.$$

Since both scores are linear,
their difference is linear.

Therefore pairwise multiclass decision boundaries are hyperplanes.

Pairwise Boundaries in the Example

Class 1 versus class 2:

$$2 - x_1 - x_2 = x_1, \\ \text{so } 2x_1 + x_2 = 2.$$

Class 1 versus class 3:

$$2 - x_1 - x_2 = x_2, \\ \text{so } x_1 + 2x_2 = 2.$$

Class 2 versus class 3:

$$x_1 = x_2.$$

All three boundaries meet at

$$\left(\frac{2}{3}, \frac{2}{3}\right), \\ \text{where } f_1 = f_2 = f_3 = 2/3.$$

A Class Region Is an Intersection of Inequalities

Class 1 is predicted when

$$f_1 \geq f_2 \\ \text{and} \\ f_1 \geq f_3.$$

For our example:

$$2x_1 + x_2 \leq 2, \\ \text{and} \\ x_1 + 2x_2 \leq 2.$$

Thus the class-1 region is the intersection of two half-planes.

For linear scores:

each class decision region is convex.

A Different Route: One Versus Rest

A simple way to reuse binary classifiers:

train one classifier for each class.

Classifier k asks:

“class k ” versus “not class k .”

For K classes:

train K binary models.

At prediction time:

choose the class with the largest score.

This is called **one-versus-rest**

or **one-versus-all**.

One Versus Rest: Advantage and Limitation

Advantage:

reuses binary methods we already know.

We could train K logistic classifiers,
perceptrons,
or SVMs.

But:

the K models are trained separately.

Their raw scores need not be directly comparable in a probabilistic sense.

We may prefer to learn all classes jointly.

The score-vector formulation does that naturally.

Correct Classification in Score Language

Suppose the true class is y .

Correct classification means:

$$f_y(\mathbf{x}) > f_k(\mathbf{x}) \quad \text{for every } k \neq y.$$

Equivalently:

$$f_y(\mathbf{x}) - \max_{k \neq y} f_k(\mathbf{x}) > 0.$$

This suggests a multiclass margin.

Multiclass Margin

Define

$$m = f_y(\mathbf{x}) - \max_{k \neq y} f_k(\mathbf{x}).$$

Then:

$m > 0$ means correct classification.

$m < 0$ means some wrong class has a larger score.

Large positive margin:

true-class score comfortably exceeds every competitor.

This is the multiclass analogue of yf
in binary classification.

A Multiclass Margin Example

Suppose the true class is $y = 2$,

and

$$\mathbf{f} = \begin{bmatrix} 1.4 \\ 2.0 \\ 0.5 \end{bmatrix}.$$

Best competing score:

$$\max(1.4, 0.5) = 1.4.$$

Therefore

$$m = 2.0 - 1.4 = 0.6.$$

Correct classification.

If instead

$$\mathbf{f} = (2.3, 2.0, 0.5)^T,$$

then $m = 2.0 - 2.3 = -0.3$.

Class 1 wins.

Multiclass Perceptron Loss

Binary perceptron loss was

$$\max(0, -m).$$

Use the same idea:

$$\ell_P = \max(0, \max_{k \neq y} f_k - f_y).$$

Equivalent form:

$$\ell_P = \max(0, -m).$$

Correct class has the largest score:

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

Wrong class wins:

loss equals the score advantage of the strongest competitor.

Which Wrong Class Matters?

Let

$$j^* = \operatorname{argmax}_{k \neq y} f_k(\mathbf{x}).$$

This is the strongest competing class.

If $f_{j^*} > f_y$,

then

$$\ell_P = f_{j^*} - f_y.$$

Only two class parameter vectors appear in this local loss:
the true class and the strongest wrong class.

That gives a simple update.

Multiclass Perceptron Update

Suppose class j^* beats the true class y .

For augmented input $\tilde{\mathbf{x}}$:
increase the true-class parameters:
 $\mathbf{w}_y \leftarrow \mathbf{w}_y + \alpha \tilde{\mathbf{x}}$.

Decrease the winning wrong-class parameters:
 $\mathbf{w}_{j^*} \leftarrow \mathbf{w}_{j^*} - \alpha \tilde{\mathbf{x}}$.

All other class parameter vectors remain unchanged.

The update raises the true score
and lowers the strongest incorrect score.

A Numerical Multiclass Update

Suppose
$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ 2 \end{bmatrix},$$

true class $y = 2$,
and scores are $(2.5, 1.5, 0.2)$.

Class 1 incorrectly wins.

Let $\alpha = 0.1$.

Then
$$\mathbf{w}_2 \leftarrow \mathbf{w}_2 + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix},$$

and
$$\mathbf{w}_1 \leftarrow \mathbf{w}_1 - \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}.$$

The score ordering is pushed toward the correct class.

Score Differences Matter More Than Score Levels

Suppose the scores are
 $(1, 3, 2)$.

Prediction:

class 2.

Add 100 to every score:

(101, 103, 102).

Prediction:

still class 2.

More generally:

$$\operatorname{argmax}_k f_k = \operatorname{argmax}_k (f_k + c).$$

Only relative score differences affect the predicted class.

This invariance will reappear in softmax probabilities.

Positive Rescaling Also Preserves the Winner

If $a > 0$,

$$\operatorname{argmax}_k f_k = \operatorname{argmax}_k a f_k.$$

Thus hard predictions do not determine the absolute scale of the scores.

But the size of score differences can matter to a training loss.

And if we want probabilities,

we need a principled normalization of the scores.

Why Not Apply a Sigmoid to Every Score?

We might try

$$p_k = \sigma(f_k)$$

for each class.

Then every number lies between 0 and 1.

But in general:

$$\sum_{k=1}^K \sigma(f_k) \neq 1.$$

Example:

if all three scores equal 0,

then each sigmoid equals 0.5.

Total:

1.5.

Those cannot be probabilities of mutually exclusive classes.

We need a joint normalization.

Requirements for Multiclass Probabilities

We want probabilities $p_1, \dots, p_K$ such that

$p_k > 0$
for every class,

and

$$\sum_{k=1}^K p_k = 1.$$

We also want:

larger score $\Rightarrow$ larger probability.

Next lesson:

the **softmax** function.

Scores in Python

Use the three-class example:

```
import numpy as np

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

x = np.array([1., 3., 0.])

scores = W @ x
print(scores)
```

Output:

```
[-1.  3.  0.]
```

Here $\mathbf{x}$ is already augmented:

$(1, x_1, x_2)$.

Choose the Winning Class in Python

Use

```
winner = np.argmax(scores)
print(winner)
```

Output:

```
1
```

New Python idea:

`np` returns the index of the largest entry.

Python indices begin at 0.

So index 1 corresponds to mathematical class 2.

If we use labels $1, \dots, K$:

```
predicted_class = np.argmax(scores) + 1
```

Classify Many Observations at Once

Put augmented observations in the rows of X :

```
X = np.array([[1., 0., 0.],
              [1., 3., 0.],
              [1., 0., 3.],
              [1., 1., 1.]])
```

```
scores = X @ W.T
```

Then **scores** has one row per observation
and one column per class.

Predictions:

```
y_pred = np.argmax(scores, axis=1) + 1
```

Output:

```
[1 2 3 2]
```

New Python idea:

axis=1 asks for the maximum across each row.

At (1,1) there is a tie between classes 2 and 3;
NumPy returns the first maximum.

Multiclass Margin in Python

Suppose

```
scores = np.array([1.4, 2.0, 0.5])
true_class = 2
```

Convert the class label to a Python index:

```
y_index = true_class - 1
true_score = scores[y_index]
```

Ignore the true class when finding the strongest competitor:

```
competitors = scores.copy()
competitors[y_index] = -np.inf

best_wrong = np.max(competitors)
margin = true_score - best_wrong
```

Result:

0.6.

Multiclass Perceptron Loss in Python

```
def multiclass_perceptron_loss(scores, true_class):
    y = true_class - 1

    competitors = scores.copy()
    competitors[y] = -np.inf

    best_wrong = np.max(competitors)
```

```
return max(0, best_wrong - scores[y])
```

The loss is zero when the true score wins.

Otherwise it measures how far the strongest wrong score exceeds the true score.

From Binary to Multiclass

Binary classification:

- one score;
- sign chooses one of two classes.

Multiclass classification:

- K scores;
- largest score chooses one of K classes.

Binary margin:

$$yf.$$

Multiclass margin:

$$f_y - \max_{k \neq y} f_k.$$

The central idea is unchanged:

- the correct class should outrank its competitors.

What did we learn?

- ♦ A K -class linear classifier assigns one linear score $f_k(\mathbf{x})$ to each class.
- ♦ Prediction uses the winner-take-all rule $\hat{y} = \arg \max_k f_k(\mathbf{x})$.
- ♦ The boundary between classes j and k satisfies $f_j = f_k$ and is therefore a hyperplane for linear scores.
- ♦ The multiclass margin is the true-class score minus the largest competing score.
- ♦ A multiclass perceptron update raises the true-class score and lowers the strongest incorrect score.
- ♦ Independent sigmoids do not generally produce class probabilities that sum to one, motivating softmax.



Next

- Multiclass prediction chooses the largest of K scores.
- Those scores are not yet probabilities.
- We need a transformation that makes all class probabilities positive and makes them sum to one.
- Next: softmax and the multiclass analogue of binary cross-entropy.
- **1.20 - Softmax and Categorical Cross-Entropy**