

Introduction to Machine Learning

Day 9: From Logistic Regression to a Neuron

Joanna Bieri DATA301

Get out your handwritten notes!

Mini Review - Logistic Regression One More Time

REVIEW

- Up to now every model we used was written by somebody else. Starting today we build models ourselves, one piece at a time.
- Same wine data as Day 5. A wine is **good** if its quality score is 7 or 8.

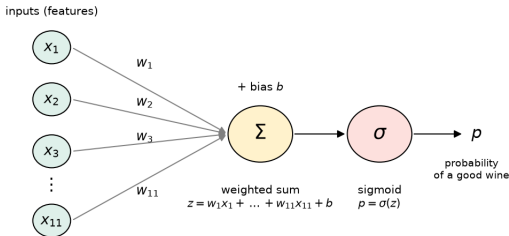
`LogisticRegression(C=np.inf)`

validation accuracy: 0.892

- The model learned eleven **weights**, one per feature, and one more number called the **intercept** or **bias**.

Mini Review - Logistic Regression is a Neuron

REVIEW



- **Step 1: a weighted sum.** $z = w_1x_1 + w_2x_2 + \dots + w_{11}x_{11} + b$
- **Step 2: the sigmoid.** $p = \sigma(z) = \frac{1}{1+e^{-z}}$
- This picture is an **artificial neuron**.

Mini Review - Tensors

REVIEW

- A **tensor** is PyTorch's version of a numpy array. Ask for float32 every time.
- @ is **matrix multiplication**. That is one weighted sum per row!

```
X_train_t = torch.tensor(X_train_scaled,  
                           dtype=torch.float32)
```

```
z = X_train_t @ w + b      # Step 1: the weighted sum  
p = torch.sigmoid(z)      # Step 2: the sigmoid
```

Two lines of PyTorch did everything predict_proba does!

Mini Review - Autograd

REVIEW

Autograd (automatic gradients) is the part of PyTorch that computes derivatives for us.

```
# requires_grad=True: keep track of x
x = torch.tensor(3.0, requires_grad=True)
```

```
f = x ** 2          # PyTorch remembers how f was made
f.backward()        # do the calculus
```

```
print(x.grad)       # tensor(6.)
```

For $f(x) = x^2$, calculus says $f'(x) = 2x$, so at $x = 3$ the slope is 6.

Mini Review - How Wrong Is the Neuron? The Loss

REVIEW

- The **loss** is one number that says **how wrong** the neuron is.
- Logistic regression uses the **log loss** (PyTorch calls it **binary cross entropy**).
 - If the wine is good ($y = 1$), the loss is $-\log(p)$.
 - If the wine is not good ($y = 0$), the loss is $-\log(1 - p)$.
- With every weight at zero the loss is 0.6931. With scikit-learn's weights it is 0.2739.

Mini Review - Teaching the Neuron: Gradient Descent

REVIEW

$$w = w - \eta \frac{\partial L}{\partial w} \quad \eta \text{ (eta) is the **learning rate**}$$

```
for step in range(n_steps):  
    p = torch.sigmoid(X_train_t @ w + b)           # 1. Predict  
    loss = F.binary_cross_entropy(p, y_train_t)    # 2. Loss  
    loss.backward()                                 # 3. Backward  
    with torch.no_grad():                           # 4. Update  
        w -= learning_rate * w.grad  
        b -= learning_rate * b.grad  
    w.grad.zero_()                                  # 5. Clear  
    b.grad.zero_()
```

Mini Review - Did It Work?

REVIEW

	our neuron	scikit-learn
training log loss	0.274	0.2739
validation accuracy	0.892	0.892
wines where they agree	all of them	

We built logistic regression from scratch, and it matches scikit-learn on every validation wine.

Logistic Regression is a Neuron

ASK

	feature 1	feature 2	bias
x	2	1	
w	1	-2	0.5

What is z ? Is the probability above or below 0.5?

Hint. $z = w_1x_1 + w_2x_2 + b$. The sigmoid of 0 is exactly 0.5.

Logistic Regression is a Neuron

ANSWER

$x = (2, 1)$, $w = (1, -2)$, $b = 0.5$. What is z ? Above or below 0.5?

$$z = 1*2 + (-2)*1 + 0.5 = 0.5$$

$$p = \text{sigmoid}(0.5) = 0.6225$$

z is positive, so p is above 0.5 and the neuron says yes.

Tensors and Autograd

ASK

`X_train_t` is 1199 wines by 11 features.

`w` has 11 weights.

What does `X_train_t @ w` give us?

Hint. Each row times the weights, number by number, then added.

Tensors and Autograd

ANSWER

What does $X_{train_t} @ w$ give us, in words?

- 1199 numbers: one weighted sum z per training wine.
- Add the bias and take the sigmoid, and you have 1199 probabilities.
- In the notes these matched `predict_proba` to 7 decimal places.

Tensors and Autograd

ASK

Why do we clear `.grad`
at the end of every step?

Hint. In the notes, x^2 at 3 has slope 6. After two `backward()` calls, `x.grad` said 12.

Tensors and Autograd

ANSWER

Why do we clear `.grad` at the end of every step?

- Autograd **adds** each new derivative to what is already in `.grad`.
- Without clearing, every step would add its gradient on top of all the old ones, and the steps would go the wrong way.
- That is why the loop ends with `w.grad.zero_()`.

How Wrong Is the Neuron?

ASK

Why do we train with the log loss
and not the accuracy?

Hint. Change one weight by a tiny amount. What happens to the accuracy? Is it continuous? Does it have a slope?

How Wrong Is the Neuron?

ANSWER

Why do we train with the log loss and not the accuracy?

- Accuracy only changes when a wine crosses 0.5. A tiny change usually moves no wine across, so the accuracy stays the same.
- The accuracy is flat with jumps, so its slope is zero. Gradient descent needs a slope.
- The log loss changes smoothly with every weight, so it has a slope.

Gradient Descent

ASK

Why is there a minus sign in

$$w = w - \eta \frac{\partial L}{\partial w}?$$

Hint. Which way does the derivative point? Which way should the loss go?

Gradient Descent

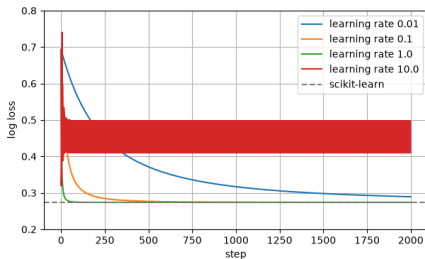
ANSWER

Why is there a minus sign in $w = w - \eta \partial L / \partial w$?

- The derivative points uphill, toward a bigger loss.
- We want a smaller loss, so we step the opposite way.
- With a plus sign the loss would go up at every step.

Gradient Descent

ASK



Which learning rate would you use? What is the red band?

Hint. How fast does each curve reach the dashed line? Does it stay?

Gradient Descent

ANSWER

Four learning rates. Which would you use? What is the red band?

- 0.01 is too slow: after 2000 steps the loss is still 0.2894.
- 0.1 and 1.0 both reach scikit-learn's loss, 1.0 faster.
- 10 is too big. Each step jumps over the bottom, and the loss bounces between about 0.41 and 0.50. That is the red band.

Did It Work?

ASK

We compared to

`LogisticRegression(C=np.inf).`

Why `C=np.inf` and not the default?

Hint. C is like $1/\alpha$. Does our neuron have a penalty?

Did It Work?

ANSWER

Why compare to `LogisticRegression(C=np.inf)` and not the default?

- A huge C means no regularization, so scikit-learn and our neuron solve the same problem.
- With the default $C = 1$, scikit-learn adds a penalty and our neuron does not, so the weights would differ.
- On Thursday we connect many neurons into a network and use PyTorch's tools for the loop.

Questions?

Questions?