Discussion Forums

Week 2

SUBFORUMS

All

Assignment: Python Basics with numpy (optional)

Discussing important concepts

Assignment: Logistic Regression with a Neural Network mindset

Assignment: Practice Questions

Week 2

E

any hints on how to vectorize Y_prediction in the predict function??

e Week 2 · 2 months ago

hi,

any hints on how to vectorize the Y_prediction assignment instead of an if/else?

if/else works fine but i assume a vectorized version is faster/less lines?

thanks,

***

Convert the entries of a into 0 (if activation <= 0.5) or 1 (if activation > 0.5), stores the predictions in a vector Y_prediction. If you wish, you can use an if/else statement in a for loop (though there is also a way to vectorize this)

****

0 Upvotes Reply Follow this discussion

Earliest **Top** **Most Recent**

Ruben Sebastiaan Kooijman · a month ago · Edited

Nice thread, I found a (not so simple) solution, but still wanted to share it here. By defining a lambda and 'vectorize' that function using np.vectorize. Now you could call the vectorized lambda on A[0,i]

```
x = np.random.randn(5); f = lambda v: 1 if v > 0.5 else 0; vf = np.vectorize(f); y = vf(x)
```

0 Upvotes Reply

CM

Chakrapani Mishra · a month ago · Edited

On a million size Vector, (A > 0.5) is at least 14 times faster than np.rint(A). Tried it multiple times on a random vector of size 1 million and 10 million. Every time, RINT was slower by a factor of at least 14. Even 1*(A > 0.5) is 4-5 times faster than RINT.

So, even though results are same, (A > 0.5) is a better vectorized implementation it seems. BUT, it gives values in terms of Booleans. After converting from Boolean to INT using "astype(int)", the factor comes back to around 4-5. So 1*(A>0.5) is equally good.

1 Upvote Hide 1 Reply

M

Paul T Mielke Mentor · a month ago

But it turns out the grader is fine with Boolean outputs, so the plain (A > 0.5) will work and is the best performance.

Thanks for doing these measurements and publishing the results!

0 Upvotes

ZZ

Reply

Reply

MM

Matthew McClellan · 2 months ago

I went with np.ceil(A - 0.5). Anything less than or equal to 0 would below 0 and the ceil would therefore be 0, anything greater would have a ceiling of 1

2 Upvotes Hide 1 Reply

M

Paul T Mielke Mentor · a month ago · Edited

That's a clever solution! It would be interesting to run Chakrapani's comparison on that versus the straight Boolean expression. On general principles, you'd expect it to have performance equivalent to that of *np.rint*, since it's doing essentially the same type of calculation and packaged in the same way (as a direct numpy function call).

0 Upvotes

ZZ

Reply

Reply

Fred Fischer Mentor · 2 months ago · Edited

I ended up using Y_prediction = np.ceil(2*A)-1 which is pretty obscure... Y_prediction = 1 * (A > 0.5) is much better and more readable. At first I used np.round, but that sends 0.5 to 1 which is wrong... the rule is that A <= 0.5 should go to 0.

0 Upvotes Reply

YH Yechen Huang · 2 months ago

There is a built-in function numpy.rint which round elements of the array to the nearest integer.
https://docs.scipy.org/doc/numpy/reference/generated/numpy.rint.html

3 Upvotes Hide 1 Reply

HC He Chaoyi · 2 months ago

Good suggestion! Thx!

0 Upvotes

ZZ

Reply

Reply

E e · 2 months ago

great suggestions, but is using a list comprehension type of argument just sending the loop inside a function like "np.where" instead of actually using some numerical method for more efficiency?

0 Upvotes Reply

CY Charles Yang · 2 months ago

Y_prediction = 1 * (A >= 0.5)

4 Upvotes Hide 6 Replies

Zacchaeus Williams · 2 months ago

How is this right? Can you explain.

0 Upvotes

GJ Gabriel Juarez · 2 months ago · Edited

test the idea:

A = np.arange(4)

print A

Y_prediction = 1 * (A >= 2)

print Y_prediction

[0 1 2 3]

[0 0 1 1]

under the hood, for every value in A, we test >=0.5, makes a True/False, which works as a 1/0...

We end up multiplying 1 * 1 or 1* 0, depending on the result of the >= 0.5 test

0 Upvotes

Pieter Nel · 2 months ago

That can't be right if you consider that Y_prediction should be 0 for all values <=0.5 0.5 itself should be translate to a 0, not a 1 as per your example.

0 Upvotes

Philippe Stessel · 2 months ago

Yes, I think this nice bit of code should be 1 * (A > 0.5) to conform to assignment.

1 Upvote

Paul T Mielke Mentor · 2 months ago

@Philippe: That's a good point! Dunno if the grader is subtle enough to catch that boundary case, but better safe than sorry ;^) ...

For what it's worth, I tried the straightforward strategy in *predict* of just using the boolean vector expression directly and the grader accepted it. The equivalent of:

v = (A > 0.5)

The result is boolean values, but the grader is fine with that, even though it no longer exactly matches the "expected output" shown in the notebook.

1 Upvote

Philippe Stessel · 2 months ago

@Paul: Even simpler!

0 Upvotes

ZZ

Reply

Reply

MV

Michael Vigoda · 2 months ago

np.where is also a possibility, as Peter mentioned

```
>>> a = np.array([[12, 4, 3, 9, 10]])
>>> a
array([[12,  4,  3,  9, 10]])
>>> np.where(a>4,1,0)
array([[1, 0, 0, 1, 1]])
```

3 Upvotes Reply

Lum Ramabaja · 2 months ago

you can also use np.vectorize in combination with python's lambda expression.

0 Upvotes Reply

Gustavo Monge · 2 months ago · Edited

Thanks. Very good tip !

Why np.int instead of just int?

i.e. print((vec > 1).astype(**int**))

0 Upvotes Reply

PS

Peter Shook · 2 months ago

see np.where

0 Upvotes Reply

VK

Vikram Kalabi · 2 months ago · Edited

Trick is to treat the vector/matrix as if it were scalar. Please see following links for clarity.

Vector/Matrix - Scalar comparison

Typecasting boolean array

Here is a sample code:

```
vec = np.random.randn(5)
print(vec)
print(vec > 1)
print((vec > 1).astype(np.int))
```

Output:

[1.05725876 0.51886838 1.39668718 0.17516998 1.61066233]

[True False True False True]

[1 0 1 0 1]

3 Upvotes Reply

1

ZZ

Reply

Reply