Numpy-Copy1

December 29, 2020

1 * Prerequisites

You should familiarize yourself with the `numpy.ndarray` class of python's `numpy` library.

You should be able to answer the following questions before starting this assignment. Let's assume `a` is a numpy array. * What is an array's shape (e.g., what is the meaning of `a.shape`)?
* What is numpy's reshaping operation? How much computational over-head would it induce?
* What is numpy's transpose operation, and how it is different from reshaping? Does it cause computation overhead? * What is the meaning of the commands `a.reshape(-1, 1)` and `a.reshape(-1)`? * Would happens to the variable `a` after we call `b = a.reshape(-1)`? Does any of `a`'s attributes change? * How do assignments in python and numpy work in general? * Does the `b=a` statement use copying by value? Or is it copying by reference? * Is the answer to the previous question change depending on whether `a` is a numpy array or a scalar value?

You can answer all of these questions by

```
1. Reading numpy's documentation from https://numpy.org/doc/stable/.
2. Making trials using dummy variables.
```

2 0. Data

2.1 0.1 Description

The UC Irvine's Machine Learning Data Repository Department hosts a Kaggle Competition with famous collection of data on whether a patient has diabetes (the Pima Indians dataset), originally owned by the National Institute of Diabetes and Digestive and Kidney Diseases and donated by Vincent Sigillito.

You can find this data at https://www.kaggle.com/uciml/pima-indians-diabetes-database/data. The Kaggle website offers valuable visualizations of the original data dimensions in its dashboard. It is quite insightful to take the time and make sense of the data using their dashboard before applying any method to the data.

2.2 0.2 Information Summary

- **Input/Output**: This data has a set of attributes of patients, and a categorical variable telling whether the patient is diabetic or not.

- **Missing Data**: For several attributes in this data set, a value of 0 may indicate a missing value of the variable.
- **Final Goal**: We want to build a classifier that can predict whether a patient has diabetes or not. To do this, we will train multiple kinds of models, and will be handing the missing data with different approaches for each method (i.e., some methods will ignore their existence, while others may do something about the missing data).

2.3 0.3 Loading

[1]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from utils import test_case_checker
```

[2]:
```
df = pd.read_csv('diabetes.csv')
df.head()
```

[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

3 1. Implementing the Simplest Classifier Ever!

If you look above there is a "Glucose" level variable in the data. Roughly speaking, a healthy person should never have blood Glucose levels of more than 140mg/liter, since a healthy pancreas should be able to control the glucose level by injecting proper amounts of insulin. Physicians use this fact to design the following simple diabetes diagnosis test, which is called the "Oral glucose tolerance test".

- Feed the patient an extremely large amount of fast-acting sugar extract.
- Test the blood glucose level after a couple of minutes.
- If the patient's blood glucose level was less than the 140 mg/liter threshold, then the patient is normal.
- If the patient's blood glucose level was in the 140-199 mg/liter range, then the patient is pre-diabetic.
- If the patient's blood glucose level was more than 200 mg/liter, then the patient is diabetic.

Of course the thresholds are up for debate and research, and this test is not 100% accurate; there are always exceptions and extreme cases. That's why there are many diabetes test types, and we're trying to also build our classifiers which hopefully should be more accurate than this simplistic test. However, this test provides a very simple and memorable way of diabetes assesment and diagnosis.

You can read about these tests at https://www.mayoclinic.org/diseases-conditions/diabetes/diagnosis-treatment/drc-20371451#:~:text=If%20it's%20126%20mg%2FdL,for%20the%20nex

Let's try and do our own investigation into how accurate this simplistic test can be. This can serve as a baseline for later comparison.

4 Task 1

Write a function `simple_pred_vec` that takes a 1-d array of glucose levels **g** and a threshold θ as input, and applies the following prediction rule for patient i: * Predict 1 (i.e., the patient is diabletic) if the patients glucose level g_i is equal or larger than the threshold (i.e. if $g_i \geq \theta$). * Otherwise predict 0 (i.e., the patient is non-diabetic).

For this task, the input is a 1-d numpy array g with a shape of $(N, 1)$ and a scalar value θ (where N is the number of patients). Write your function in a way that the output would have exactly the same shape as the g input including the dummy dimension of 1. The output data type should be boolean.

Note: You do not need to use any reshaping for this task. In other words, do not reshape the **g** input and do not try to force an $(N, 1)$ shape on the output array.

[3]:
```
def simple_pred_vec(g, theta):

    # your code here
    arr = []
    for i in range(len(g)):
        arr.append((g[i]>=theta))
    out = np.array(arr)
    return out
```

[4]:
```
assert (simple_pred_vec(g=np.array([100., 200., 140.]), theta=140.) == np.
 ↪array([False, True, True])).all()

# Checking against the pre-computed test database
test_results = test_case_checker(simple_pred_vec, task_id=1)
assert test_results['passed'], test_results['message']
```

5 Task 2

Using the `simple_pred_vec` function that you previously wrote, write a new function `simple_pred` function that takes a pandas dataframe `df` and threshold `theta` as input, and produces a prediction numpy array `pred`.

For this specific task, the `pred` variable will mostly take a shape of $(1, N)$. However, do not force this shape on the `pred` variable by reshaping it to have an exact shape of $(1, N)$; forcing such a shape on the `pred` variable may cause issues when using this function in later tasks.

The dataframe `df` has a column `Glucose` which indicates the blood glucose levels, and a column `Outcome` which indicates whether the patient is diabetic or not. You should extract the `Glucose` column from the dataframe and use it for thresholding and prediction.

- **Hint**: If you like to have the column `'des_col'` of a pandas dataframe `df` as a numpy array, then `df['des_col'].values` may be helpful.
- **Important Note**: The `df['des_col'].values` expression returns a numpy array that is one-dimensional (i.e., has a shape of $(N,)$). In order to maintain portability when possibly utilizing this function in later tasks, it is advised to reshape this `df['des_col'].values` array into having a shape of $(1, N)$ before using it in the `simple_pred` function.

[5]:
```
def simple_pred(df, theta):

    # your code here
    out = df['Glucose'].values.reshape(1,-1)
    pred = simple_pred_vec(out, theta)

    return pred
```

[6]:
```
assert np.array_equal(simple_pred(df, 120)[:,:5], np.array([[ True, False, ␣
↪True, False,  True]]))

# Checking against the pre-computed test database
test_results = test_case_checker(simple_pred, task_id=2)
assert test_results['passed'], test_results['message']
```

6 Task 3

6.1

6.2 Using the `simple_pred` function that you previously wrote, write a new function `simple_acc` function that takes a pandas dataframe `df` and threshold `theta` as input, predicts the `Outcome` label, and returns the accuracy `acc` of the predictor.

- In the most trivial case, `theta` can be scalar value (e.g., `theta=120`).

- `theta` can also be a column array (i.e., `theta.shape == (k,1)` where `k` could be any integer). In this case, `acc` should be a vector of accuracy values with the same number of elements as `theta`.
- `acc` should always be a 1-d numpy array (i.e., `acc.shape == (k,)`). Even if `theta` was a scalar value, `acc` should be a numpy array with the shape `(1,)`.
- You can use the exact same way of array extraction (with all the caveats and considerations) from the previous task to extract the `Outcome` column from the `df` dataframe.

Limitation 1 You should not be using any external libraries or functions for implementing this function. Only numpy functions should be used.

Limitation 2 You cannot use any loops, such as `for` and `while`, for implementing this function. You should learn how to implement such basic functionalities using numpy's matrix operations and functions.

- **Hint**: Assume that you have a prediction vector `pred` and a label vector `label` whose shapes are `(1,N)`.
  - Let's have `a= (pred==label)`. Think about what `a` means.
  - Can you express the prediction accuracy as a numpy function of `a`?
- **Hint**: Run the following snippet for yourself, and try and make sense of how each of the variables c, d, e, and f were generated given that a and b had different shapes.

```
import numpy as np
a = np.array([1,2,3,4]).reshape(1, -1) # "a" is a row vector
b = np.array([1,3,6]).reshape(-1, 1) # "b" is a column vector
c = (a == b)
d = (a * b)
e = (a + b)
f = (a > b)
print(f'c.shape is {c.shape}. d.shape is {d.shape}. e.shape is {e.shape}. f.shape is {f.shape}
print('----------')
print('c is ')
print(c)
print('----------')
print('d is ')
print(d)
print('----------')
print('e is ')
print(e)
print('----------')
print('f is ')
print(f)
print('----------')
```

[7]:
```
a = np.array([1,2,3,4]).reshape(1, -1) # "a" is a row vector
b = np.array([1,3,6,7]).reshape(-1, 1) # "b" is a column vector
c = (a == b)
```

```
d = (a * b)
e = (a + b)
f = (a > b)
g = b.T
h = (a == g)
print('a is ')
print(a)
print('----------')
print('b is ')
print(b)
print('----------')
print('g is ')
print(g)
print('----------')
print('h is ')
print(h)
print('----------')
print(f'a.shape is {a.shape}.b.shape is {b.shape}.c.shape is {c.shape}. d.shape␣
 ↪is {d.shape}. e.shape is {e.shape}. f.shape is {f.shape}.')
print('----------')
print('c is ')
print(c)
print('----------')
print('d is ')
print(d)
print('----------')
print('e is ')
print(e)
print('----------')
print('f is ')
print(f)
print('----------')
print(b.shape)
np.squeeze(b)
print(b.shape)
```

```
a is
[[1 2 3 4]]
----------
b is
[[1]
 [3]
 [6]
 [7]]
----------
g is
[[1 3 6 7]]
```

```
----------
h is
[[ True False False False]]
----------
a.shape is (1, 4).b.shape is (4, 1).c.shape is (4, 4). d.shape is (4, 4).
e.shape is (4, 4). f.shape is (4, 4).
----------
c is
[[ True False False False]
 [False False  True False]
 [False False False False]
 [False False False False]]
----------
d is
[[ 1  2  3  4]
 [ 3  6  9 12]
 [ 6 12 18 24]
 [ 7 14 21 28]]
----------
e is
[[ 2  3  4  5]
 [ 4  5  6  7]
 [ 7  8  9 10]
 [ 8  9 10 11]]
----------
f is
[[False  True  True  True]
 [False False False  True]
 [False False False False]
 [False False False False]]
----------
(4, 1)
(4, 1)
```

[8]:
```python
def simple_acc(df, theta):

    # your code here
    predict = simple_pred(df, theta)[0]
    actual = df['Outcome'].values.reshape(1,-1)

    correct = ((predict == (actual==1)).astype(int))

    acc = np.mean(correct,axis=1)

    return acc
```

[9]:
```
# Toy testing the shapes
assert simple_acc(df, theta=120).shape == (1,)
assert simple_acc(df, theta=np.array([50,100,300]).reshape(3,1)).shape == (3,)

# Toy testing the values
assert simple_acc(df, theta=120).round(3)==0.698
assert np.array_equal(simple_acc(df, theta=np.array([[50,100,20,40]]).T).
 ↪round(3), [0.352, 0.564, 0.35 , 0.35 ])


# Checking against the pre-computed test database
test_results = test_case_checker(simple_acc, task_id=3)
assert test_results['passed'], test_results['message']
```

7 Task 4

Write a function `best_theta_loopy` that takes a pandas dataframe `df`, and uses a `for` loop and the function `simple_acc` to select the `theta` threshold in the interval $[75, 200]$ yielding the highest accuracy. * Both 75 and 200 should be included in the test. * Only search for integer `theta` values.

You should produce `best_theta` and `best_accuracy`, where both of them are scalar values.

Note: In case multiple thresholds yielded the exact same highest accuracy, `best_theta` should be the smallest threshold among them. In other words, in case multiple optima for `theta` existed within the search range, the tie-breaking rule would be to pick the `theta` with the smallest value as `best_theta`.

[10]:
```
def best_theta_loopy(df):

    # your code here
    acc_ = []
    for i in range (75,201):
        acc_.append(simple_acc(df, i)[0])
    acc_ = np.array(acc_)
    best = np.argmax(acc_)
    best_theta = 75+best
    best_accuracy = simple_acc(df, best_theta)[0]

    return np.array([best_theta, best_accuracy])
```

[11]:
```
assert np.array_equal(best_theta_loopy(df.iloc[:10, :]), np.array((117, 0.9)))

# Checking against the pre-computed test database
test_results = test_case_checker(best_theta_loopy, task_id=4)
assert test_results['passed'], test_results['message']
```

Now that you have implemented this, let's see what the best threshold would be on the whole data set, and what the resulting accuracy would be.

We did not perform train/test splits, so the accuracy may be inflated a bit...

[12]:
```
best_theta, best_acc = tuple(best_theta_loopy(df))
best_theta, best_acc
```

[12]:
```
(144.0, 0.75)
```

8 Task 5

The use of a for loop in the `best_theta_loopy` function is unnecessary; you can re-write the function so that it has the same functionality without using any loops.

Write a function `best_theta` that takes a pandas dataframe `df`, and uses numpy operations to select the theta threshold in the range $[75, 200]$ yielding the highest accuracy. You should again produce both `best_theta` and `best_accuracy` scalars.

Note: In case multiple thresholds yielded the exact same highest accuracy, `best_theta` should be the smallest threshold among them. In other words, in case multiple optima for `theta` existed within the search range, the tie-breaking rule would be to pick the `theta` with the smallest value as `best_theta`.

Limitation 1 You should not be using any external libraries or functions for implementing this function. Only numpy functions should be used.

Limitation 2 You cannot use any loops, such as `for` and `while`, for implementing this function. You should learn how to implement such basic functionalities using numpy's matrix operations and functions.

[13]:
```
def best_theta(df):

    # your code here
    i = np.arange(75,201,1).reshape(-1,1)
    acc_ = simple_acc(df, i)
    best = np.argmax(acc_)
    best_theta = 75+best
    best_acc = acc_[best]
    return np.array((best_theta, best_acc))
```

[14]:
```
assert np.array_equal(best_theta(df.iloc[:10, :]), np.array((117, 0.9)))

# Checking against the pre-computed test database
test_results = test_case_checker(best_theta, task_id=5)
assert test_results['passed'], test_results['message']
```

9 Task 6

Using the `simple_pred` function that you previously wrote, write a new function `simple_confusion` function that takes a pandas dataframe `df` and threshold `theta` as input, and produces the confusion matrix M, where the rows correspond to the predicted values and the columns correspond to the actual values.

- `theta` is a scalar value.
- M should have a shape of $(2, 2)$ since there are two classes, and the entries should be integer values.

Limitation 1 You should not be using any external libraries or functions for implementing this function. Only numpy functions should be used.

Limitation 2 You cannot use any loops, such as `for` and `while`, for implementing this function. You should learn how to implement such basic functionalities using numpy's matrix operations and functions.

- **Hint**: Assume that you have a prediction vector `pred` and a label vector `label` whose shapes are `(1,N)`.
  - Think about what the matrix `a=(np.array([0,1]).reshape(2,1) == pred).astype(np.int)` represents. What is its shape?
  - Think about what the matrix `b=(label.reshape(-1,1) == np.array([0, 1]).reshape(1,2)).astype(np.int)` represents. What is its shape?
  - Think about how you can derive the confusion matrix as a function of `a` and `b`. What matrix operation can be helpful?

[15]:
```
np.array([0,1]).reshape(2,1)
```

[15]:
```
array([[0],
       [1]])
```

[16]:
```
def simple_confusion(df, theta):

    # your code here
    a = (np.array([0,1]).reshape(2,1) == simple_pred(df, theta)).astype(np.int)␣
 ↪ #(1, N)
    b = (df['Outcome'].values.reshape(-1,1) == np.array([0, 1]).reshape(1,2)).
↪astype(np.int)
    M = a.dot(b)

    return M
```

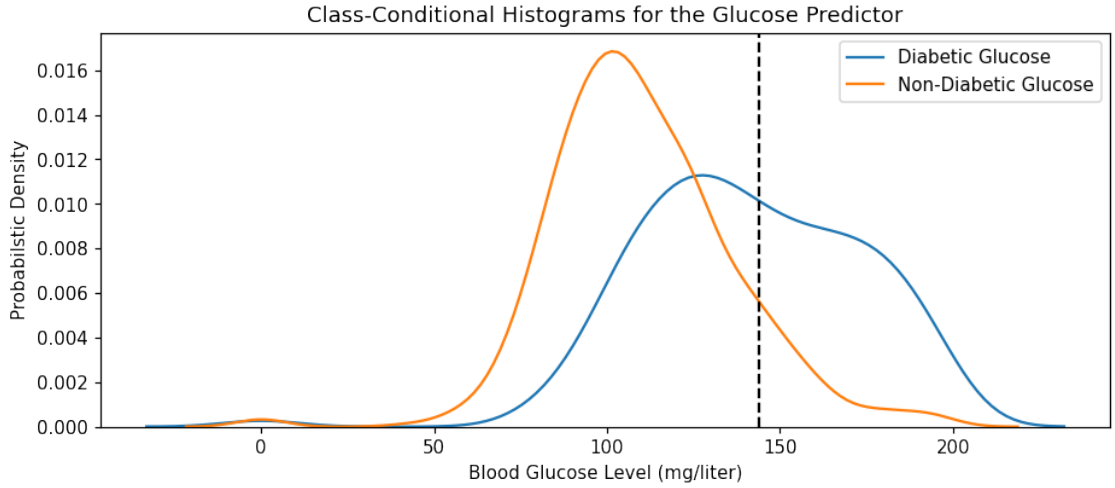
[17]:
```
assert np.array_equal(simple_confusion(df.iloc[:100, :], theta=144), np.
↪array([[55, 24], [ 8, 13]]))

# Checking against the pre-computed test database
test_results = test_case_checker(simple_confusion, task_id=6)
assert test_results['passed'], test_results['message']
```

Let's obtain the confusion matrix and plot the class conditional histograms.

[18]:
```
cm = simple_confusion(df, theta=144)
cm
```

[18]:
```
array([[450, 142],
       [ 50, 126]])
```

[19]:
```
fig, ax = plt.subplots(figsize=(10,4), dpi=108)
sns.kdeplot(df['Glucose'][df['Outcome']==1].values, ax=ax, label='Diabetic␣
↪Glucose')
sns.kdeplot(df['Glucose'][df['Outcome']==0].values, ax=ax, label='Non-Diabetic␣
↪Glucose')
ax.set_xlabel('Blood Glucose Level (mg/liter)')
ax.set_ylabel('Probabilstic Density')
ax.set_title('Class-Conditional Histograms for the Glucose Predictor')
_=ax.axvline(x=144, c='black', ls='--')
```



One question to think about is "why didn't the dashed black separator get placed right at the intersection of the Blue and Orange histograms?".

[ ]: