Course 3 Week 3 lecture notebook 01

Permutation Method

In this lecture notebook, we'll demonstrate the permutation method on a dataset you worked on in Week 1 of Course 2. By going through this short exercise, you'll hopefully develop some intuition on how to do the related task in this week's programming assignment.

Setup

First, let's import the packages we'll be using:

In [1]:
```
import pandas as pd
import pickle
from util import cindex
```

```
Using TensorFlow backend.
```

Next, we load the model we trained during the said assignment in Course 2. This accepts a patient's age, systolic blood pressure, diastolic blood pressure, and cholesterol to output a risk score (i.e. probability of an event).

In [2]:
```
model_X = pickle.load(open('C2M1_model.model', 'rb'))
```

We will also use the same test dataset from that programming assignment:

In [3]:
```
# patient data (test set in the previous assignment)
X_baseline = pd.read_csv('./lecture_nb_permutation_data/X_data_norm
alized.csv',index_col=0)

# corresponding patient outcome
y = pd.read_csv('./lecture_nb_permutation_data/y_data_test.csv',ind
ex_col=0)
```

Let's print the first 10 cells to peek into the dataset. Take note that these are already standardized and normalized as you did previously.

In [4]:
```
X_baseline.head(10)
```

Out[4]:

	Age	Systolic_BP	Diastolic_BP	Cholesterol
4320	0.747640	-1.211321	-1.003548	-1.072461
2006	-0.758669	-1.476605	-1.541427	-0.434487
5689	0.067833	0.399011	0.904060	0.694137
472	-0.257881	0.655984	1.157183	0.700589
1370	0.800621	-0.664058	0.228552	1.366445
1457	-0.988683	0.935349	0.752583	-0.709739
5783	0.145403	0.759190	0.155382	1.790223
3484	0.149429	-0.852439	0.671971	0.477288
1145	-0.250499	-0.173969	0.006586	-0.360930
130	-0.757791	-0.781030	0.431331	-0.617740

C-index and Baseline Performance

Now let's get the c-index of the trained model using our test dataset. Just to refresh:

- The c-index measures the discriminatory power of a risk score.
- Intuitively, a higher c-index indicates that the model's prediction is in agreement with the actual outcomes of a pair of patients.
- The formula for the c-index is

$$\text{cindex} = \frac{\text{concordant} + 0.5 \times \text{ties}}{\text{permissible}}$$

- A permissible pair is a pair of patients who have different outcomes.
- A concordant pair is a permissible pair in which the patient with the higher risk score also has the worse outcome.
- A tie is a permissible pair where the patients have the same risk score.

Computing this for our test dataset will give us our baseline performance. For efficiency, we will be using the `concordance_index()` method from the lifelines (https://lifelines.readthedocs.io/en/latest/lifelines.utils.html) package instead of the homemade c-index function we developed in the Course 2 assignment. This is wrapped in a `cindex()` function in the `util` library we imported earlier.

In [5]:
```
# get the patient's risk scores by feeding the baseline data to the trained model
scores = model_X.predict_proba(X_baseline)[:, 1]

# measure the model's cindex given the patient outcome and computed risk scores above
c_index = cindex(y.values, scores)

print(f"baseline c-index is {c_index:.4f}")
```

```
baseline c-index is 0.8182
```

This should look familiar as this is also the result you got in the Course 2 assignment.

Feature Importance and the Permutation Method

As discussed in the lecture videos, one way we can compute the importance of a feature in a given model is by shuffling the values of a particular column in our test set. Evaluating the performance of the model after this shuffling will allow us to quantify the feature's importance. For this exercise, we'll choose to shuffle the `Age` column. You will do this programmatically in this week's assignment but for now, you'll use a dataset that is shuffled beforehand.

In [6]:
```
# patient data permuted at the Age column
X_permuted_1 = pd.read_csv('./lecture_nb_permutation_data/X_data_normalized_perm1.csv',index_col=0)

# Let's print the first 10 cells and compare with the baseline table above.
# All are in the same place except for the values in the Age column
X_permuted_1.head(10)
```

Out[6]:

	Age	Systolic_BP	Diastolic_BP	Cholesterol
4320	-0.494544	-1.211321	-1.003548	-1.072461
2006	-0.289638	-1.476605	-1.541427	-0.434487
5689	0.425693	0.399011	0.904060	0.694137
472	-2.244324	0.655984	1.157183	0.700589
1370	-0.078010	-0.664058	0.228552	1.366445
1457	-1.475322	0.935349	0.752583	-0.709739
5783	0.694167	0.759190	0.155382	1.790223
3484	-0.695555	-0.852439	0.671971	0.477288
1145	-0.761429	-0.173969	0.006586	-0.360930
130	0.021294	-0.781030	0.431331	-0.617740

Now, let's see how our model performs with this shuffled input:

In [7]:
```
# get the patient's risk scores by feeding the baseline data with shuffled Age column
scores = model_X.predict_proba(X_permuted_1)[:, 1]

# measure the c-index given the patient outcome and newly computed risk scores
c_index = cindex(y.values, scores)
print(f"c-index for 1st permutation is {c_index:.4f}")
```

```
c-index for 1st permutation is 0.6749
```

We see a big drop in the performance. To quantify the importance of this feature, we simply subtract this drop from the baseline performance. This results in `0.8182 - 0.6749 = 0.1433`

Additional Permutations

The result above (i.e. `0.1433`) describes the feature importance of `Age` in our model given the shuffled dataset. However, you might deduce that the predicted risk scores are greatly affected by how the Age column is rearranged. If we arrange it differently, then the model will predict different values and this might lead to a different c-index, and consequently a different value of the feature importance. To illustrate, let's import two more datasets with different permutations of the `Age` column. Since there is a random element when doing the permutation, the resulting order of the elements in the `Age` column will be different for each shuffle. You can see that in the two datasets below.

In [8]:
```
# 2nd permutation
X_permuted_2 = pd.read_csv('./lecture_nb_permutation_data/X_data_normalized_perm2.csv',index_col=0)

X_permuted_2.head(10)
```

Out[8]:

	Age	Systolic_BP	Diastolic_BP	Cholesterol
4320	0.728099	-1.211321	-1.003548	-1.072461
2006	0.008176	-1.476605	-1.541427	-0.434487
5689	0.604035	0.399011	0.904060	0.694137
472	-1.179030	0.655984	1.157183	0.700589
1370	0.835747	-0.664058	0.228552	1.366445
1457	0.872547	0.935349	0.752583	-0.709739
5783	0.498609	0.759190	0.155382	1.790223
3484	0.324675	-0.852439	0.671971	0.477288
1145	0.671958	-0.173969	0.006586	-0.360930
130	-0.925788	-0.781030	0.431331	-0.617740

In [9]:
```
# 3rd permutation
X_permuted_3 = pd.read_csv('./lecture_nb_permutation_data/X_data_normalized_perm3.csv',index_col=0)

X_permuted_3.head(10)
```

Out[9]:

	Age	Systolic_BP	Diastolic_BP	Cholesterol
4320	-0.593886	-1.211321	-1.003548	-1.072461
2006	0.303244	-1.476605	-1.541427	-0.434487
5689	-1.755198	0.399011	0.904060	0.694137
472	-0.500154	0.655984	1.157183	0.700589
1370	-0.109112	-0.664058	0.228552	1.366445
1457	0.439787	0.935349	0.752583	-0.709739
5783	1.335356	0.759190	0.155382	1.790223
3484	-0.428345	-0.852439	0.671971	0.477288
1145	0.199869	-0.173969	0.006586	-0.360930
130	0.146330	-0.781030	0.431331	-0.617740

Now let's evaluate the performance of the model with these new datasets:

In [10]:
```
# get the patient's risk scores by feeding a dataset with a different permutation of the Age column
scores = model_X.predict_proba(X_permuted_2)[:, 1]

# measure c-index
c_index = cindex(y.values, scores)

print(f"c-index for 2nd permutation: {c_index:.4f}")
```

```
c-index for 2nd permutation: 0.6554
```

In [11]:
```
# get the patient's risk scores by feeding a dataset with a different permutation of the Age column
scores = model_X.predict_proba(X_permuted_3)[:, 1]

# measure c-index
c_index = cindex(y.values, scores)

print(f"c-index for 3rd permutation: {c_index:.4f}")
```

```
c-index for 3rd permutation: 0.6428
```

As you can see, there are slight differences in the resulting c-index depending on how the values are shuffled. This will then result in different values of the feature importance as well. Let's summarize our results in a table for clarity:

Dataset	**C-index**	**Importance**
Baseline	0.8182	n/a
1st permutation	0.6749	0.1433
2nd permutation	0.6554	0.1628
3rd permutation	0.6428	0.1754

To take these variations into account, we can take the mean of these different results to have a single value of the feature importance. More formally:

$$I_x = |perf - perf_x|$$

where I_x is the importance of feature x and

$$perf_x = \frac{1}{n} \cdot \sum_{i=1}^{n} perf_i^{sx}$$

where $perf_i^{sx}$ is the performance with the feature x shuffled in the ith permutation.

Applying this to our results, that would be `abs(0.8182 - mean([0.6749, 0.6554, 0.6428]) = 0.1605`. We would want to get more permutations to get a more accurate value of the feature importance. You will get the chance to work this out in the week's programming assignment.

That's it for this lecture on the permutation method! We can now measure the global importance of a feature in the predictions of a given model. In the next section of the course, you will learn how to determine the importance of a feature for an individual patient in the dataset.