Machine Learning

Pas cu pas partea 2

Așa arată setul de date

După ce am făcut modificările

```
Int64Index: 714 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  714 non-null    int64
 1   Survived     714 non-null    int64
 2   Pclass       714 non-null    int64
 3   Name         714 non-null    object
 4   Sex          714 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        714 non-null    int64
 7   Parch        714 non-null    int64
 8   Ticket       714 non-null    object
 9   Fare         714 non-null    float64
 10  Cabin        185 non-null    object
 11  Embarked     714 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 72.5+ KB
```

Maparea datelor

Pentru a utiliza datele, ele toate trebuie să fie numere.

Trebuie să transformăm coloanele Sex și Embarked în numere.

Aceasta se poate face cu mapare.

Maparea crează indici pentru valori non-numerice.

Exemplu de mapare:
Male -> 0, Female -> 1

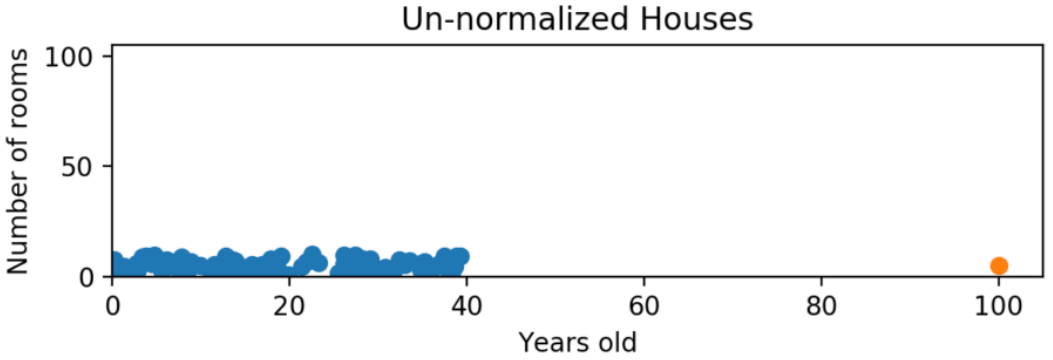
```
Int64Index: 714 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  714 non-null    int64
 1   Survived     714 non-null    int64
 2   Pclass       714 non-null    int64
 3   Name         714 non-null    object
 4   Sex          714 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        714 non-null    int64
 7   Parch        714 non-null    int64
 8   Ticket       714 non-null    object
 9   Fare         714 non-null    float64
 10  Cabin        185 non-null    object
 11  Embarked     714 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 72.5+ KB
```

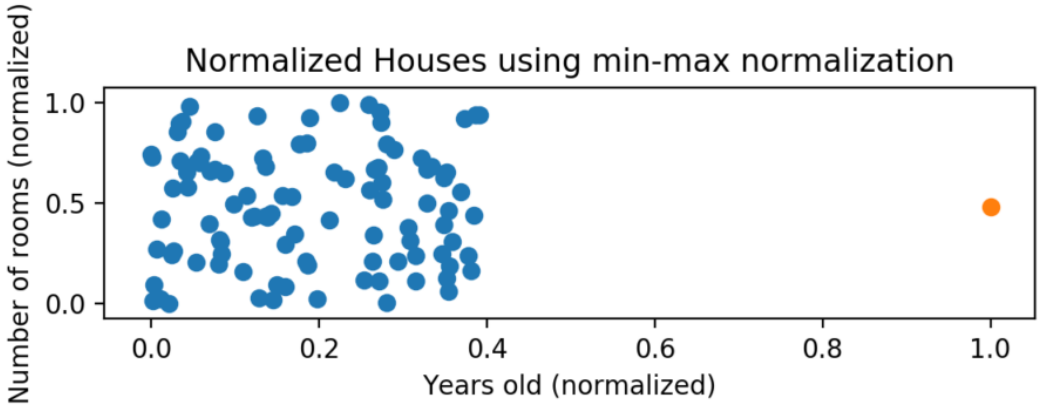
Normalizarea datelor

Rețelele naturale nu lucrează optimal cu valori în afara din intervalul numerelor de la 0 la 1.

Din cauza asta, trebuie să normalizăm toate valorile numerice, să fie în intervalul de numere de la 0 la 1.

Vom utiliza min-max normalizare, unde valoarea maximă va fi 1, iar valoarea minimă 0, și tot ce se află între ele vor fi între 0 și 1.





Crearea setului X și Y

Setul X va fi setul de intrare, iar setul Y va fi setul de ieșire.

Rețeaua neurală va încerca să prezică setul Y cu valorile din setul X.

data este setul de date cu valorile null eliminate.

În funcția map_data se face maparea și normalizarea datelor dintr-un set de date, și returnează un tensor care poate fi utilizat de rețeaua neurală.

Tensor este un array (matrice) cu mai multe dimensiuni.

```python
def map_data(dataset):
    p_class = dataset['Pclass']  #Contained data values: 1, 2, 3.  Map 1 to 0, 2 to 0.5 and 3 to 1
    x_p_class = list((p_class - p_class.min()) / (p_class.max() - p_class.min()))
    sex = dataset['Sex']  #Contained data values: 'male', 'female'. Map male to 0 and female to 1
    x_sex = list(sex.rank(method='dense', ascending=False).astype(int) - 1)
    age = dataset['Age']  #Contained data are integers between 0 and 100. Map from 0 to 100 to 0 to 1
    x_age = list((age - age.min()) / (age.max() - age.min()))
    sibsp = dataset['SibSp']  #Contained data are integers between 0 and 10. Map from 0 to 10 to 0 to 1
    x_sibsp = list((sibsp - sibsp.min()) / (sibsp.max() - sibsp.min()))
    parch = dataset['Parch']  #Same as sibsp
    x_parch = list((parch - parch.min()) / (parch.max() - parch.min()))
    # ticket = dataset['Ticket']  #Unused
    fare = dataset['Fare']  #Contains data from 0 to 512. Map from 0 to 512 to 0 to 1
    x_fare = list((fare - fare.min()) / (fare.max() - fare.min()))
    # cabin = dataset['Cabin'] #Unused
    embarked = dataset['Embarked']
    x_embarked = list(embarked.rank(method='dense', ascending=False).astype(int) - 1) # Map according to frequency
    return torch.Tensor([[x_p_class[i], x_sex[i], x_age[i], x_sibsp[i], x_parch[i], x_fare[i], x_embarked[i]] for i in range(len(dataset))])
```

```python
x_train = map_data(data)
y_train = torch.LongTensor([entry for entry in data['Survived']])
```

Cum arată datele de intrare și ieșire

Datele de intrare este un vector de dimensiunea 7.

Datele de ieșire este un vector de dimensiunea 1, adică un scalar.

În acest caz, rețeaua trebuie să prezică că această persoană nu a supraviețuit.

```
1 print(x_train[0])
```

```
tensor([1.0000, 0.0000, 0.2712, 0.2000, 0.0000, 0.0142, 0.0000])
```

```
1 print(y_train[0])
```

```
tensor(0)
```

Crearea modelului

Avem nevoie de:
Un model simplu care poate primi vectori 7 dimensionali ca intrare.

Care clasifică două stări:
Survived, Died

Fc1 funcționează ca intrare

Fc2 are o marime aleatorie

Fc3 funcționează ca ieșire, și are activație pentru clasificare - softmax logaritmic.

Variabila net reprezintă modelul creat.

```
1  import torch.nn as nn
2  import torch
3  import torch.nn.functional as F
4
5
6  class Net(nn.Module):
7      def __init__(self):
8          super().__init__()
9          self.fc1 = nn.Linear(7, 128, bias=True)
10         self.fc2 = nn.Linear(128, 128, bias=True)
11         self.fc3 = nn.Linear(128, 2, bias=True)
12
13     def forward(self, x):
14         x = F.relu(self.fc1(x))
15         x = F.relu(self.fc2(x))
16         x = F.log_softmax(self.fc3(x), dim=0)
17         return x
18
19
20 net = Net()
21 print(net)
```

Antrenarea modelului

Antrenarea în pytorch este cam complexă deoarece se face manual.

În esență o epocă este o parcurgere pe tot setul de antrenare, iar pentru fiecare intrare in setul de date, se calculează prezicerea ”output = net(x)” și se calculează valoarea funcției de pierdere ”loss = F.nll_loss(output,y)”.

După care se cheamă optimizatorul, care schimbă rețeaua neurală.

```
import torch.optim as optim
optimizer = optim.Adam(net.parameters(), lr=0.0001)

EPOCHS = 1
BATCHES = 10

loss = torch.Tensor(1)
correct = 0
total = 0
loss_graph = []
accuracy_graph = []
for epoch in range(EPOCHS):
    for i in range(len(x_train)):
        x = x_train[i]
        y = y_train[i]
        if i % BATCHES == 0:
            loss_graph.append(loss.item())
            accuracy_graph.append(correct / total if total != 0 else 0.5)
            net.zero_grad()
        output = net(x)
        loss = F.nll_loss(output, y)
        loss.backward()
        optimizer.step()
        correct += 1 if y == torch.argmax(output) else 0
        total += 1

    print(f"Epoch {epoch} loss:", loss)
    print(f"Accuracy:{correct / total}")
```

Acuratețea modelului

După cum vedem, valoarea funcției de pierdere are un trend de scădere, iar acuratețea are un trend de creștere.

La sfârșitul antrenarii, pe setul de date de antrenare modelul are acuratețe de 80%.

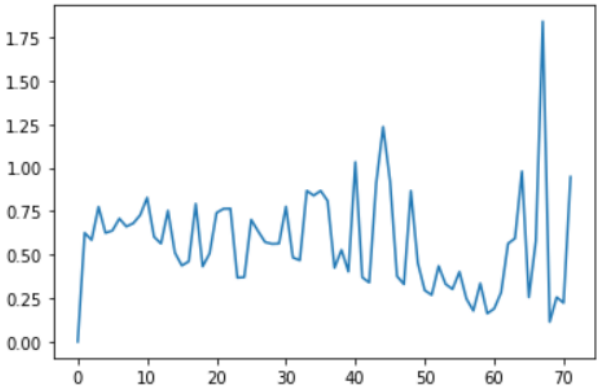
Este normal ca în antrenare să parcurgem de mai multe ori tot setul de date de antrenare. (Mai multe epoci)

Dar din cauză că acest model este mic, acesta face rapid overfitting.

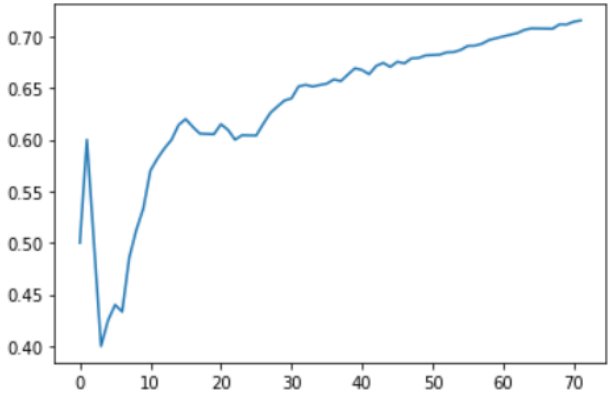
```
import matplotlib.pyplot as plt

plt.plot([i for i in range(len(loss_graph))], loss_graph, label="loss")
plt.show()
plt.plot([i for i in range(len(accuracy_graph))], accuracy_graph, label="accuracy")
plt.show()
```

Valoarea funcției de pierdere



Acuratețea



Rezultate

Când este alegere aleatorie, acuratețea este de 50% în medie. Când alegem doar ”Did not survive” avem acurateț̦e de 60%.

Pentru o epocă de antrenare, modelul de are la sfârșit 80% acurateț̦e pe setul de date de antrenare, iar pe setul de date de testare acesta are acuratețea de 75.358%

După trei epoci de antrenare, la antrenare acurateț̦e ~85%, la testare acurateț̦e 76.315%.

Un rezultat acceptabil.

```python
# Random guess accuracy
import random

random_correct = 0
for row in data['Survived']:
    if round(random.random()) == row:
        random_correct += 1

random_accuracy = float(random_correct) / float(data['Survived'].size)
print("Random accuracy:",random_accuracy)
```

```
Random accuracy: 0.4943977591036415
```

```
showcase accuracy: 0.8
```

Public Score
0.75358

```
showcase accuracy: 0.85
```

Public Score
0.76315

Ce putem îmbunătăți?

Multe. Dar problema e în setul de date.
Se poate de identificat anomaliile, de normalizat valorile mai bine și multe altele.

Aceasta este tot.

Referinte

https://www.kaggle.com/c/titanic

https://cyberhoot.com/cybrary/data-normalization/

https://towardsdatascience.com/how-to-code-a-simple-neural-network-in-pytorch-for-absolute-beginners-8f5209c50fdd