Titanic

[ ] ↳ *Скрыто 11 ячеек.*

The formatting

- One-hot encode: 'Sex', 'Embarked'
- Remove: 'Name', 'Ticket', 'Cabin'
- Fill null values with the mean of the associated column.

[ ] ↳ *Скрыто 2 ячейки.*

Split on train and test

[ ] ↳ *Скрыто 3 ячейки.*

Prepare inputs for model

```
# Format the data into PyTorch tensors
X_train = torch.FloatTensor(X_train.values)
X_test = torch.FloatTensor(X_test.values)
y_train = torch.LongTensor(y_train.values)
y_test = torch.LongTensor(y_test.values)


import torch
import torch.nn as nn
import torch.nn.functional as F

class Model(torch.nn.Module):

    def __init__(self, input_features):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(input_features, 270)
        self.bn1 = nn.BatchNorm1d(270)
        self.fc2 = nn.Linear(270, 50)
        self.bn2 = nn.BatchNorm1d(50)
        self.fc3 = nn.Linear(50, 2)

    def forward(self, x):
        x = self.fc1(x)
        x = self.bn1(x)
        x = F.dropout(x, p=0.1)
        x = F.relu(x)
        x = self.fc2(x)
        x = self.bn2(x)
        x = F.dropout(x, p=0.1)
        x = F.relu(x)
```

```
        x = self.fc3(x)
        x = torch.sigmoid(x)
        return x



device = 'cuda' if torch.cuda.is_available() else 'cpu'
print('Using {} device'.format(device))
```

```
Using cpu device
```

```
model = Model(X_train.shape[1]).to(device)
print(model)
optimizer = torch.optim.Adam(model.parameters(), lr=0.1, betas=(0.9, 0.99))
criterion = nn.CrossEntropyLoss()
```

```
Model(
  (fc1): Linear(in_features=11, out_features=270, bias=True)
  (bn1): BatchNorm1d(270, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
  (fc2): Linear(in_features=270, out_features=50, bias=True)
  (bn2): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
  (fc3): Linear(in_features=50, out_features=2, bias=True)
)
```

```
# При данных зачениях (и те, что закомментированы), значение accuracy равняется 0.83
#batch_size = 25
#num_epochs = 20
#learning_rate = 0.02
#batch_size = 39
#num_epochs = 20
#learning_rate = 0.05
batch_size = 90
num_epochs = 90
learning_rate = 0.1
batch_no = len(X_train) // batch_size
print(batch_no)
```

```
7
```

```
train_loss = np.zeros((num_epochs*batch_no,))
train_accuracy = np.zeros((num_epochs*batch_no,))
valid_loss = np.zeros((num_epochs*batch_no,))
valid_accuracy = np.zeros((num_epochs*batch_no,))
```

```
import torch.nn as nn
loss_fn   = nn.CrossEntropyLoss()
```

```
p=0

for epoch in range(num_epochs):
    if epoch % 5 == 0:
        print('Epoch {}'.format(epoch+1))
```

```
        # x_train, y_train = shuffle(X_train, y_train)
        x_train = X_train.to(device) # needs assignment
        y_train = y_train.to(device) # needs assignment
        # Mini batch learning
        for i in range(batch_no):
            start = i * batch_size
            end = start + batch_size
            x_var = x_train[start:end]
            y_var = y_train[start:end]
            #Backward + Optimize
            optimizer.zero_grad()
            pred = model(x_var)
            loss =criterion(pred, y_var)

            train_loss[p] = loss.item()
            train_correct = (torch.argmax(pred, dim=1) == y_var).type(torch.FloatTensor)
            train_accuracy[p] = train_correct.mean()

            loss.backward()
            optimizer.step()
            p+=1
            with torch.no_grad():
              y_pred = model(X_test)
              loss = loss_fn(y_pred, y_test)
              valid_loss[epoch] = loss.item()
              correct = (torch.argmax(y_pred, dim=1) == y_test).type(torch.FloatTensor)
              valid_accuracy[epoch] = correct.mean()
```

```
Epoch 1
Epoch 6
Epoch 11
Epoch 16
Epoch 21
Epoch 26
Epoch 31
Epoch 36
Epoch 41
Epoch 46
Epoch 51
Epoch 56
Epoch 61
Epoch 66
Epoch 71
Epoch 76
Epoch 81
Epoch 86
```

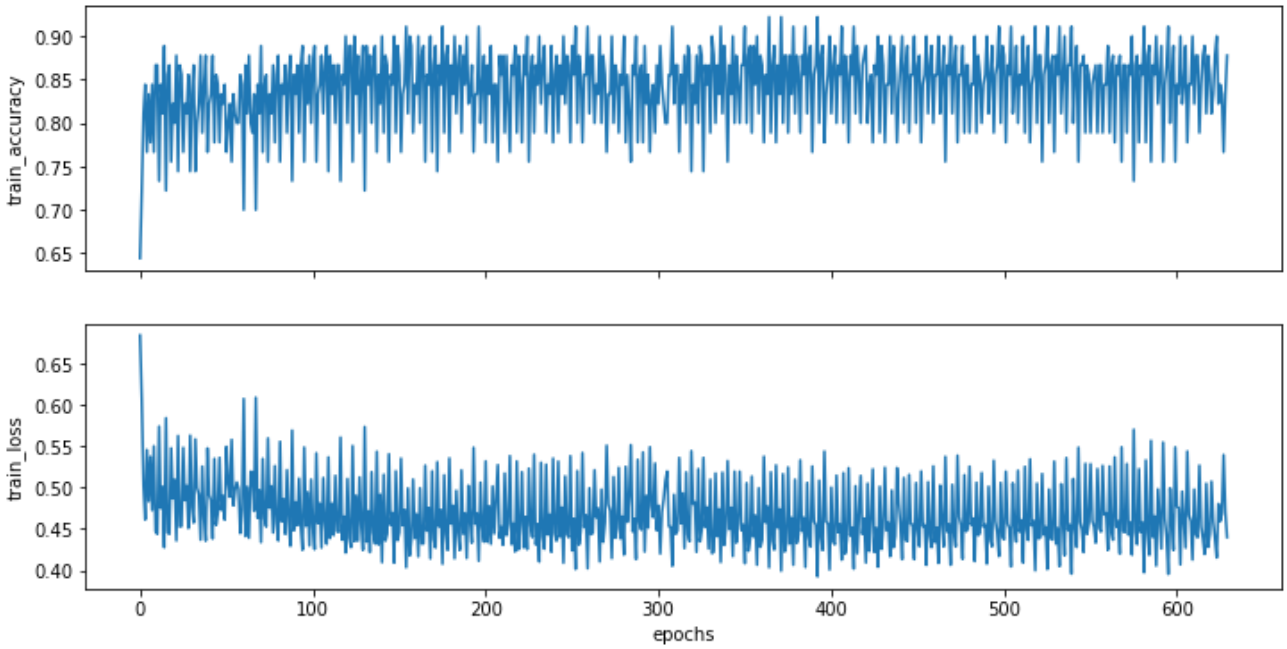
```
# Evaluate the model
test_var =  X_test.to(device) # needs assignment
with torch.no_grad():
    result = model(test_var)
values, labels = torch.max(result, 1)
num_right = np.sum(labels.data.cpu().numpy() == y_test.cpu().numpy())
print('Accuracy {:.2f}'.format(num_right / len(y_test)))
```

```
      Accuracy 0.84
```

```
fig, [ax1, ax2] = plt.subplots(2, figsize=[12, 6], sharex=True)

ax1.plot(train_accuracy)
ax1.set_ylabel('train_accuracy')
ax2.plot(train_loss)
ax2.set_ylabel('train_loss')
ax2.set_xlabel("epochs")
```

```
Text(0.5, 0, 'epochs')
```




```
fig, (ax1, ax2) = plt.subplots(2, figsize=(12, 6), sharex=True)

ax1.plot(valid_accuracy)
ax1.set_ylabel("valid_accuracy")
ax2.plot(valid_loss)
ax2.set_ylabel("valid_loss")
ax2.set_xlabel("epochs");
```


```
y_pred = model(X_test)
y_hat=torch.argmax(y_pred, dim=1)
len(y_test)
```

```
179
```

```
0.4 |
```

```
from sklearn.metrics import plot_confusion_matrix
# confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
from sklearn.metrics import confusion_matrix
conf=confusion_matrix(y_test, y_hat)
conf
```

```
array([[106,   4],
       [ 28,  41]])
```

```
# Precision and recall
from sklearn.metrics import precision_score, recall_score,f1_score
print(f"precision: {precision_score(y_test, y_hat, average='weighted')}")
print(f"recall: {recall_score(y_test, y_hat, average='weighted')}")
print(f"f1 score: {f1_score(y_test, y_hat, average='weighted')}")
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('\nAccuracy: {:.2f}\n'.format(accuracy_score(y_test, y_hat,)))

print('Micro Precision: {:.2f}'.format(precision_score(y_test, y_hat,  average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(y_test, y_hat, average='micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(y_test, y_hat, average='micro')))

print('Macro Precision: {:.2f}'.format(precision_score(y_test, y_hat, average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(y_test, y_hat, average='macro')))
print('Macro F1-score: {:.2f}\n'.format(f1_score(y_test, y_hat, average='macro')))

print('Weighted Precision: {:.2f}'.format(precision_score(y_test, y_hat,average='weighted'
print('Weighted Recall: {:.2f}'.format(recall_score(y_test, y_hat, average='weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(y_test, y_hat, average='weighted')))

from sklearn.metrics import classification_report
print('\nClassification Report\n')
print(classification_report(y_test, y_hat))
```

```
precision: 0.8373273298312904
recall: 0.8212290502793296
f1 score: 0.8112030694925794

Accuracy: 0.82
```

```
Micro Precision: 0.82
Micro Recall: 0.82
Micro F1-score: 0.82

Macro Precision: 0.85
Macro Recall: 0.78
Macro F1-score: 0.79

Weighted Precision: 0.84
Weighted Recall: 0.82
Weighted F1-score: 0.81

Classification Report

              precision    recall  f1-score   support

           0       0.79      0.96      0.87       110
           1       0.91      0.59      0.72        69

    accuracy                           0.82       179
   macro avg       0.85      0.78      0.79       179
weighted avg       0.84      0.82      0.81       179
```

✓ 0 сек. выполнено в 09:28