RNN_Overlap_Dataloader

December 6, 2021

1 Libraries

Import your libraries https://towardsdatascience.com/how-to-use-convolutional-neural-networks-for-time-series-classification-56b1b0a07a57

[1]:
```
import torch
import os
import math
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader, TensorDataset
from matplotlib import rc
from sklearn.utils import resample
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from torch import nn, optim
import torch.nn.functional as F
```

```
using gpu 4
```

2 Importing Data

Import the CSV file with Actions, Sum and Div as a Dataframe called df. Fill the empty values of Action with 0. Replace NaN values with 0. Delete first 100 rows.

[2]:
```
#load in df
Player = 15
Game = 2
Quarter = 2

df = pd.read_csv('Split_RNN_player15game2.csv')
df_Results = pd.read_csv('Player_15_Game2_Sprints_Q1234.csv')
```

```
#Clean and expand dataset
df = df.drop(columns = ['Unnamed: 0'])
df.head()
```

[2]:

	frAcc	frRoAcc	frDispl	frRoAng	frSpeed	timeLine	\
0	17.91200	2828.7	3394.9	81.515	0.23278	3581.0	
1	7.14760	2828.7	3395.7	81.397	0.41190	3581.0	
2	-0.89061	2828.8	3396.5	82.233	0.48338	3581.0	
3	4.17050	2828.8	3397.3	74.342	0.47447	3581.0	
4	1.15710	2828.8	3398.0	74.332	0.51618	3581.0	

	frameRotationalSpeedX	frameRotationalSpeedY	frameRotationalSpeedZ	\
0	7.3500	-8.9367	80.033	
1	4.0600	-8.1200	80.010	
2	-17.5700	-0.4200	81.340	
3	-1.6940	-10.5000	72.576	
4	-2.0922	-6.1600	73.111	

	wheelRotationalSpeedX	wheelRotationalSpeedY	wheelRotationalSpeedZ	\
0	-32.737	-39.114	45.033	
1	-1.050	-15.960	38.570	
2	10.780	-32.340	45.640	
3	16.485	-41.601	51.163	
4	23.847	-34.619	54.281	

	frRoSpeed	Filt_WheelX	Filt_FrameZ	Action	Quarter	\
0	-11.8030	67.674366	8.849375	0.0	1	
1	83.6110	65.140780	8.214781	0.0	1	
2	-789.0700	62.446959	7.721630	0.0	1	
3	-1.0099	59.592743	7.380241	0.0	1	
4	94.2010	56.579696	7.199759	0.0	1	

	wheelRotationalSpeedX_Diff
0	0.000
1	31.687
2	11.830
3	5.705
4	7.362

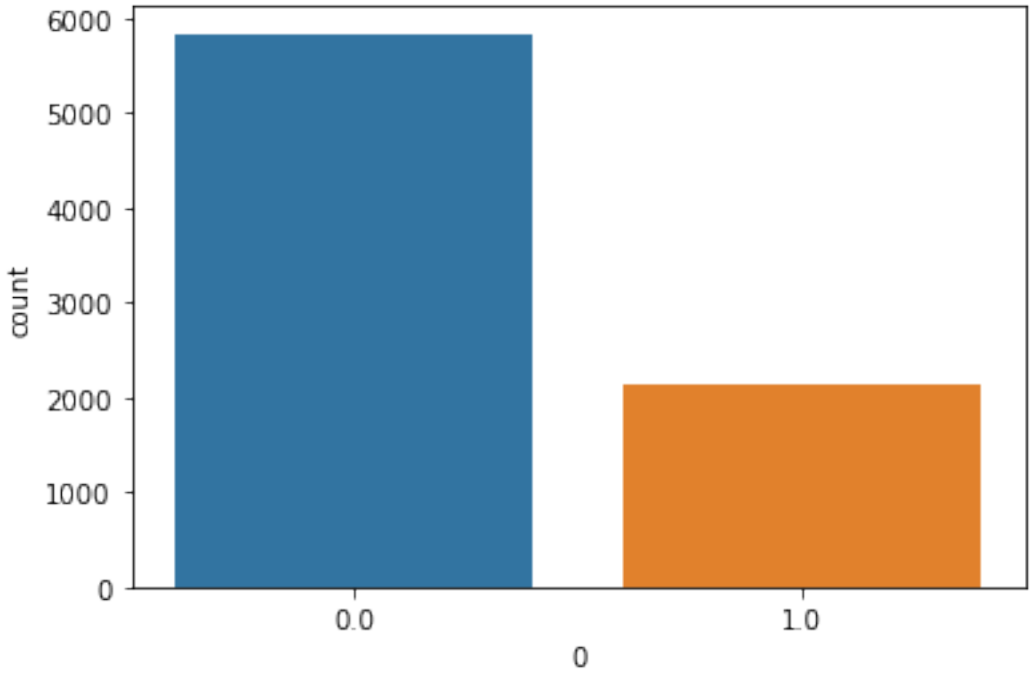
3 Pre Preprocessing

[3]:
```
#Split data set into train en test with the quarters of the game
train = pd.
 ↪read_csv('Split_Xtrain_balanced_RNN_player15game2_Quartor'+str(Quarter)+'.
 ↪csv')
```

```
test = df[df.Quarter == Quarter]

y_train_resample = pd.
 ↪read_csv('Split_ytrain_balanced_RNN_player15game2_Quartor'+str(Quarter)+'.
 ↪csv')
y_train_resample = y_train_resample.drop(columns = ['Unnamed: 0'])
y_test_resample = df_Results.Action[df_Results.Quarter == Quarter]
sns.countplot(y_train_resample['0'])
```

```
/opt/jupyterhub/anaconda/lib/python3.9/site-packages/seaborn/_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[3]: <AxesSubplot:xlabel='0', ylabel='count'>



4 Split Data

[4]:
```
def rounddown(x):
    return (int(math.ceil(x / 100.0)) * 100) - 100
```

[5]:
```
features =␣
↪['wheelRotationalSpeedX','frameRotationalSpeedY','frAcc','wheelRotationalSpeedX_Diff']

X_train = train[features]
X_train = X_train.iloc[0:rounddown(len(X_train))]

y_train = y_train_resample['0']

X_test = test[features]
X_test = X_test.iloc[0:rounddown(len(X_test))]

y_test = y_test_resample
```

5 Convert the X_train, X_test, y_train, y_test to Tensors

[6]:
```
X_train = torch.from_numpy(X_train.to_numpy()).float()
X_train = X_train.unsqueeze_(-1)
X_train = X_train.transpose(2, 0)
X_train = X_train.transpose(2,1)
X_train = torch.reshape(X_train, [-1,100,len(features)])

y_train = torch.squeeze(torch.from_numpy(y_train.to_numpy()).float())

X_test = torch.from_numpy(X_test.to_numpy()).float()
X_test = X_test.unsqueeze_(-1)
X_test = X_test.transpose(2,0)
X_test = X_test.transpose(2,1)
X_test = torch.reshape(X_test, [-1,100,len(features)])

y_test = torch.squeeze(torch.from_numpy(y_test.to_numpy()).float())
```

6 Load Data into Dataloader

[7]:
```
train_ds = TensorDataset(X_train, y_train)
train_dl = DataLoader(train_ds, batch_size=64, num_workers = 4, pin_memory =␣
↪True)
```

7 CNN def

Define the Convolutional Neural Network

[8]:
```
class LSTM(nn.Module):
    def __init__(self, hidden_state_size = 200, input_feature_size =␣
↪len(features)):
        super().__init__()
        self.hidden_state_size = hidden_state_size
        self.feature_size = input_feature_size
        self.lstm1 = nn.LSTM(input_feature_size, self.hidden_state_size,␣
↪batch_first=True)
        self.l2 = nn.Linear(self.hidden_state_size, 1)

    def forward(self, x):
        h, _ = self.lstm1(x)
        h = h[:,-1,:]
        y = self.l2(h)

        y = y + x[:,-1,-1:]

        return torch.sigmoid(y)

    def post_forward(self, y):
        return torch.round(y)

RNN = LSTM()
```

8 Training options

[9]:
```
criterion = nn.BCELoss()

optimizer = optim.Adam(RNN.parameters(), lr=0.001)
```

9 Training the RNN on the GPU

[10]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

X_test = X_test.to(device)
y_test = y_test.to(device)

RNN = RNN.to(device)
criterion = criterion.to(device)
```

10 Define functions for calculating Accuracy of model

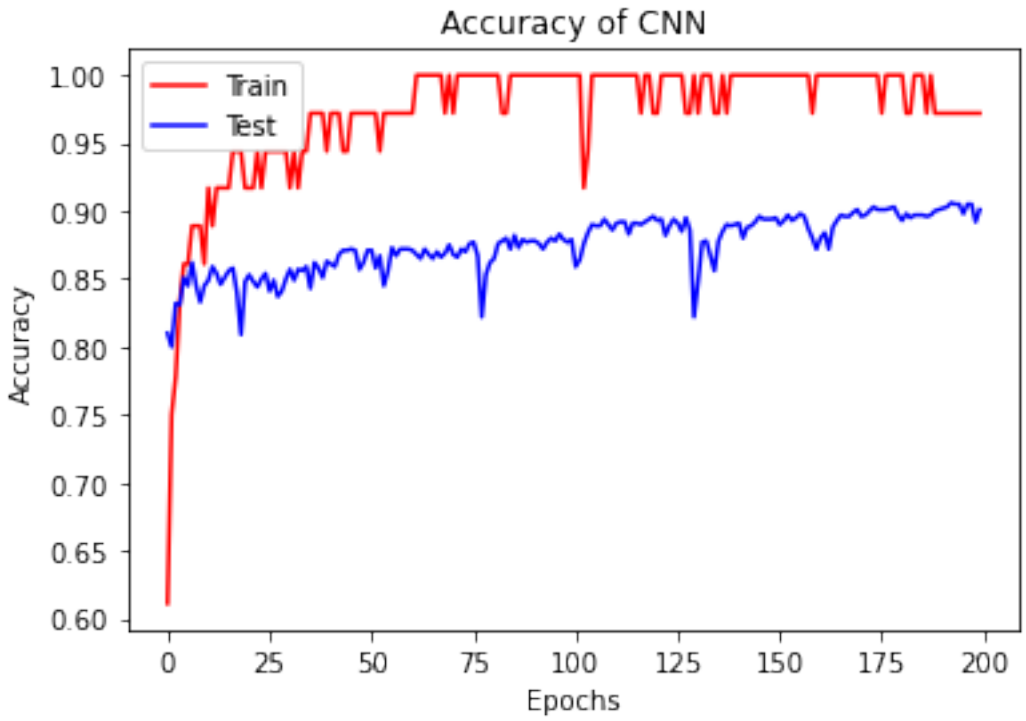
[11]:
```
def calculate_accuracy(y_true, y_pred):
    return (y_true == y_pred).sum().float() / len(y_true)
```

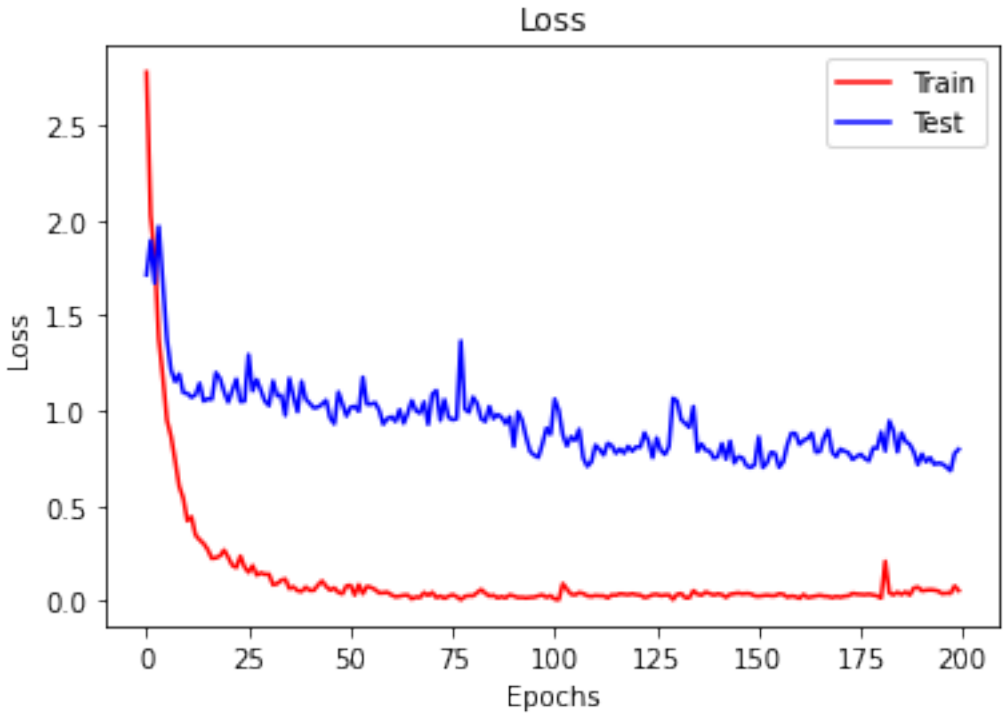
11 For loop through different epochs

[12]:
```
def round_tensor(t, decimal_places=3):
    return round(t.item(), decimal_places)

Results_acc = pd.DataFrame(columns = ['Epoch', 'Acc_train', 'Acc_test'])
Results_loss = pd.DataFrame(columns = ['Epoch', 'Loss_train', 'Loss_test'])
Results_recall = pd.DataFrame(columns = ['Epoch', 'Recall_test'])
Results_prec = pd.DataFrame(columns = ['Epoch', 'Prec_test'])

for epoch in range(200):

    for x, y in train_dl:

        x, y = x.to(device), y.to(device)

        optimizer.zero_grad()
        y_pred = RNN(x)
        y_pred = y_pred.squeeze(axis = 1)
        train_loss = criterion(y_pred, y)
        train_loss.backward()
        optimizer.step()

    if epoch % 1 == 0:

        train_acc = calculate_accuracy(y, RNN.post_forward(y_pred))
        train_loss = criterion(y_pred,y)

        y_test_pred = RNN(X_test)
        y_test_pred = torch.squeeze(y_test_pred)
        test_loss = criterion(y_test_pred, y_test)
        test_acc = calculate_accuracy(y_test, RNN.post_forward(y_test_pred))

        Confusion = confusion_matrix(y_test.cpu(),y_test_pred.ge(.5).view(-1).
↪cpu())
        test_recall = Confusion[1][1]/(Confusion[1][1] + Confusion[1][0])
        test_prec = Confusion[1][1]/(Confusion[0][1] + Confusion[1][1])
```

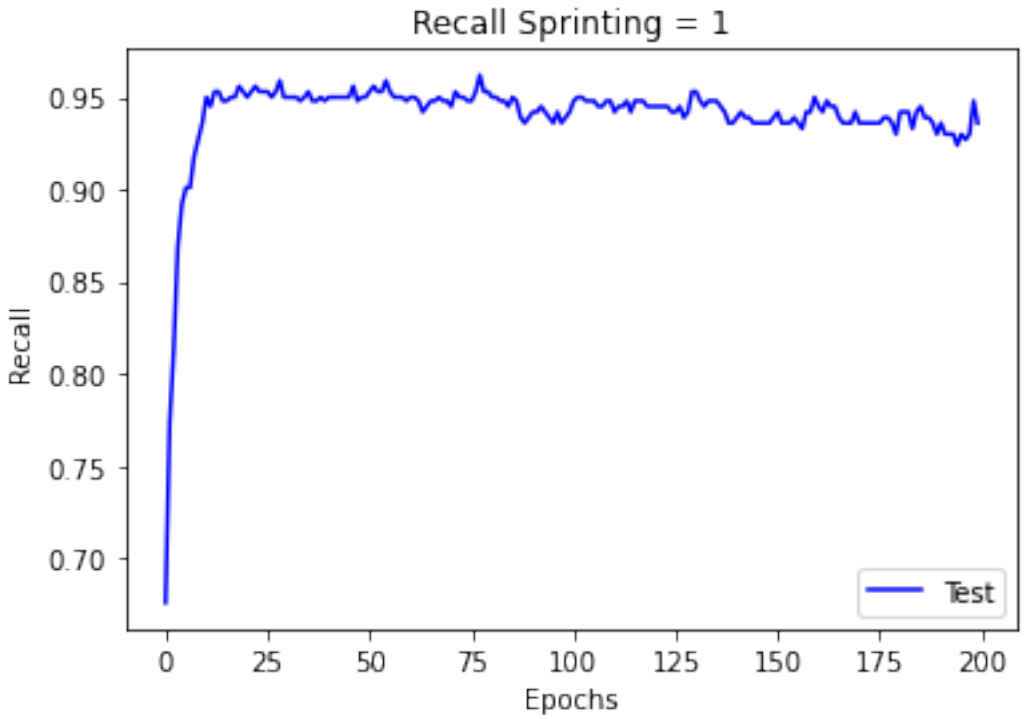
```
        Acc = {'Epoch': epoch, 'Acc_train': round_tensor(train_acc), 'Acc_test':
↪ round_tensor(test_acc)}
        Loss = {'Epoch': epoch, 'Loss_train': round_tensor(train_loss),␣
↪'Loss_test': round_tensor(test_loss)}
        Recall = {'Epoch': epoch, 'Recall_test': round_tensor(test_recall)}
        Prec = {'Epoch': epoch,'Prec_test': round_tensor(test_prec)}

        Results_acc = Results_acc.append(Acc, ignore_index=True)
        Results_loss = Results_loss.append(Loss, ignore_index=True)
        Results_recall = Results_recall.append(Recall, ignore_index=True)
        Results_prec = Results_prec.append(Prec, ignore_index=True)
```

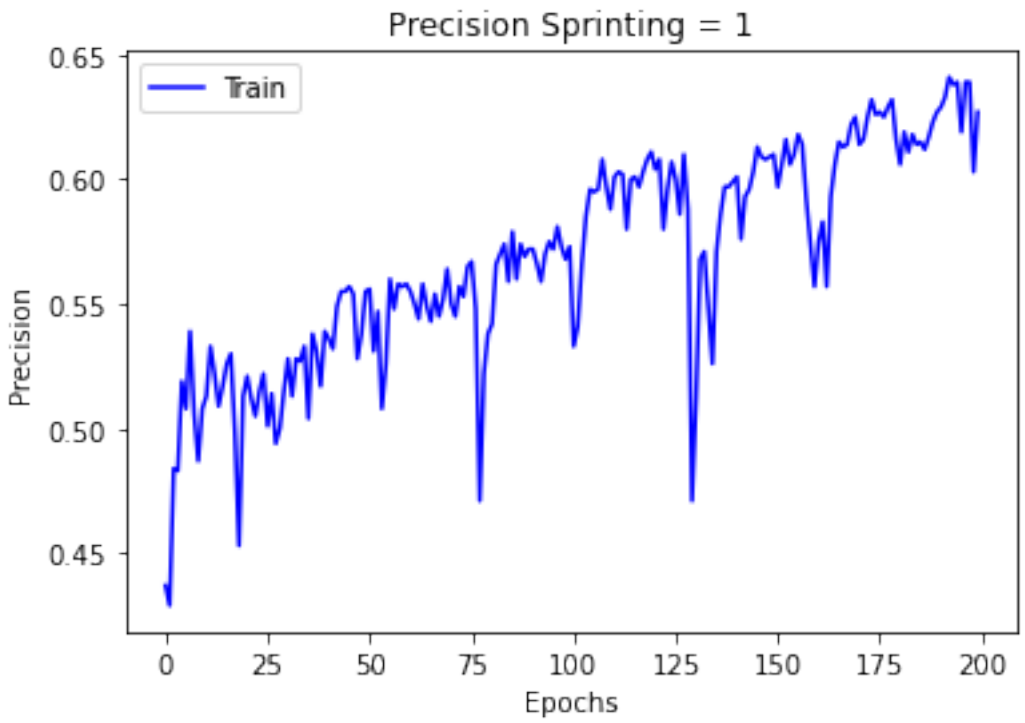
[13]:
```
plt.plot(Results_acc.Epoch,Results_acc.Acc_train,'r',Results_acc.
↪Epoch,Results_acc.Acc_test,'b')
plt.legend(['Train','Test'])
plt.title('Accuracy of CNN')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.show()

plt.plot(Results_loss.Epoch,Results_loss.Loss_train,'r',Results_loss.
↪Epoch,Results_loss.Loss_test,'b')
plt.legend(['Train','Test'])
plt.title('Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.show()
```


Accuracy of CNN
Train
Test
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
25
50
75
100
125
150
175
200
Epochs


Loss
Train
Test
Loss
2.5
2.0
1.5
1.0
0.5
0.0
0
25
50
75
100
125
150
175
200
Epochs

[14]:
```
plt.plot(Results_recall.Epoch , Results_recall.Recall_test,'b')
plt.legend(['Test'])
plt.title('Recall Sprinting = 1')
plt.ylabel('Recall')
plt.xlabel('Epochs')
plt.show()

plt.plot(Results_prec.Epoch,Results_prec.Prec_test,'b')
plt.legend(['Train'])
plt.title('Precision Sprinting = 1')
plt.ylabel('Precision')
plt.xlabel('Epochs')
plt.show()
```





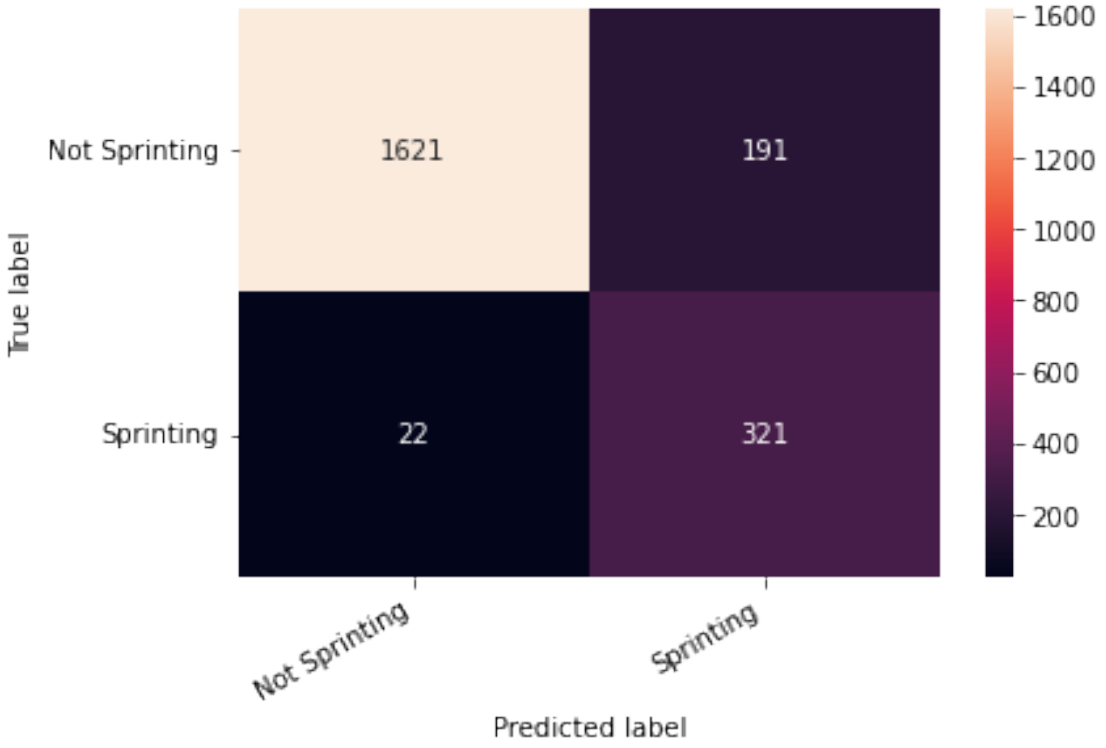
12 Validate/Tune Model

Validate results of the model (Precision/Recall). Tune the parameters of the model to achieve better results

[15]:
```
classes = ['Not Sprinting', 'Sprinting']
y_pred = RNN(X_test)
y_pred = y_pred.ge(.5).view(-1).cpu()
y_test = y_test.cpu()
print(classification_report(y_test, y_pred, target_names=classes))
```

```
              precision    recall  f1-score   support

Not Sprinting       0.99      0.89      0.94      1812
    Sprinting       0.63      0.94      0.75       343

     accuracy                           0.90      2155
    macro avg       0.81      0.92      0.84      2155
 weighted avg       0.93      0.90      0.91      2155
```

[16]:
```
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=classes, columns=classes)
hmap = sns.heatmap(df_cm, annot=True, fmt="d")
hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(), rotation=0, ha='right')
hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(), rotation=30, ha='right')
plt.ylabel('True label')
plt.xlabel('Predicted label');
```



13 Export results

[17]:
```
df_results = pd.DataFrame(y_pred)
df_results.to_csv('Predictions_15x2_Quartor'+str(Quarter)+'_Overlap.csv')
df_results
```

[17]:

	0
0	False
1	False
2	False
3	False
4	False

```
…     …
2150  False
2151  False
2152  False
2153  False
2154  False

[2155 rows x 1 columns]
```

14 Close Kernel

[ ]:
```
%%javascript
Jupyter.notebook.session.delete();
```

```
<IPython.core.display.Javascript object>
```