Learning to Detect Heavy Drinking Episodes Using Smartphone Accelerometer Data

Bartos-Elekes Miklós - 1900130792

Premissa

- Study of 2019
- Based only on available smartphone data
- “Movement” detection
- Binary classification problem
- Claimed to have 88% of accuracy achieved

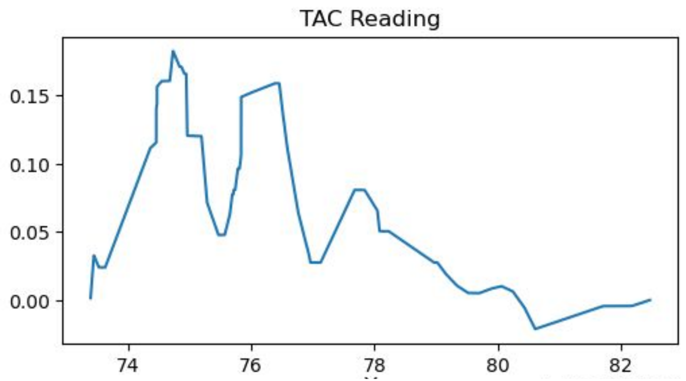
Parameters

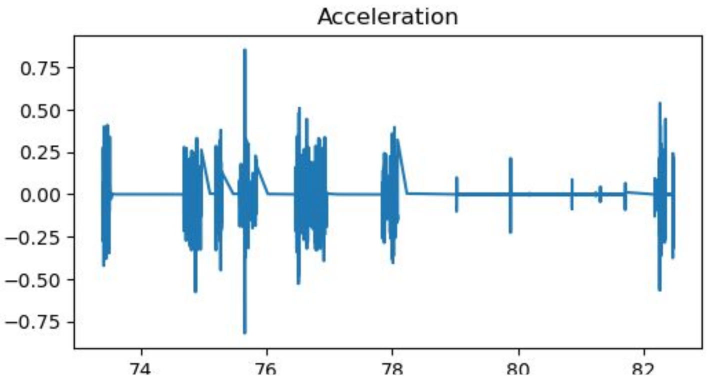
- Computer:
  - Economy laptop
  - 24GB RAM, 16 threads on 8 cores, no GPU
- Data
  - 14M records
  - Timestamp, acceleration 3D, TAC label, PID
  - PID: 13 distinctive persones

About the data

- Real-world data
  - Unreliable, noisy
  - Handling inaccurate sampling by different devices
  - Unmatching boundaries
  - Unexpected breaks during sampling for longer time
  - The data does not fit into the memory, it is hard to visualize
  - This we have statistical information, rather then
  - To be able to visualize
- By x-y-z the problem is not linearly

seperable



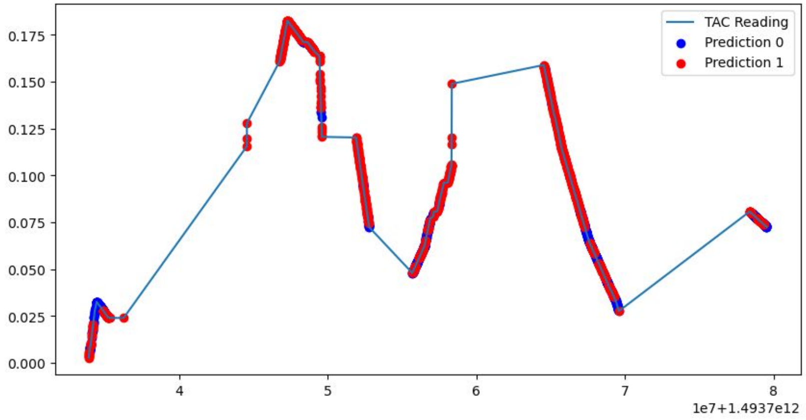


Preprocessing

- Aggregation - for faster calculation,
  - One pid - but it is built to handle multiple
  - Merge every 25 records, avg on values
  - By this, for training we only have 11k records
- Adjusting boundaries, merging tables
- Filling NaN values with assumed values
- Gaining statistical data about the dataframe
- Feature engineering
  - Velocity, distance values - not helpful
  - Stat data for the past 5 or 30 seconds - increased accuracy
  - 75 added feature
- Bias: 14% of the data was labeled

Decision-tree

- Based on the paper, random forest achieved the best results
- My first experiment was to build a decision tree
- Merging the prev 25 records (having 25*75 input) increased accuracy
- 78% accuracy
- With the right filter, it might be applicable
- Room for improvement



Neural network

- Simple multi layer NN
- Achieved more modest results
- 0.5M parameters
- 55-60%
  - Comparable to the papers results

```
class NN(nn.Module):
    def __init__(self):
        super(NN, self).__init__()
        self.fc1 = nn.Linear(1875, 256)
        self.fc2 = nn.Linear(256, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 32)
        self.fc5 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.flatten(x, 1)
        x = Fun.relu(self.fc1(x))
        x = Fun.relu(self.fc2(x))
        x = Fun.relu(self.fc3(x))
        x = Fun.relu(self.fc4(x))
        x = torch.sigmoid(self.fc5(x))
        return x
```

Convolutional approach

- Using a more modern approach
  - To avoid overlearning
- Slightly increase of accuracy
- Still, comparable to paper results

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        # Convolutional Block 1
        self.conv1 = nn.Conv2d(1, 32, kernel_size=(7, 7), padding=3)
        self.batchnorm1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=(7, 7), padding=3)
        self.batchnorm2 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d(kernel_size=(2, 2))  # Reduces (25,75) -> (12,37)

        # Convolutional Block 2
        self.conv3 = nn.Conv2d(32, 64, kernel_size=(7, 7), padding=3)
        self.batchnorm3 = nn.BatchNorm2d(64)
        self.conv4 = nn.Conv2d(64, 64, kernel_size=(7, 7), padding=3)
        self.batchnorm4 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d(kernel_size=(2, 2))  # Reduces (12,37) -> (6,18)

        # Convolutional Block 3
        self.conv5 = nn.Conv2d(64, 128, kernel_size=(7, 7), padding=3)
        self.batchnorm5 = nn.BatchNorm2d(128)
        self.conv6 = nn.Conv2d(128, 128, kernel_size=(7, 7), padding=3)
        self.batchnorm6 = nn.BatchNorm2d(128)
        self.pool3 = nn.MaxPool2d(kernel_size=(2, 2))  # Reduces (6,18) -> (3,9)

        # Global Average Pooling
        self.global_avg_pool = nn.AdaptiveAvgPool2d(1)  # Output shape -> [batch, 128, 1, 1]

        # Fully Connected Layers
        self.fc1 = nn.Linear(128, 64)
        self.fc2 = nn.Linear(64, 1)

        # Dropout for regularization
        self.dropout = nn.Dropout(0.3)
```

Results

- Helped to discover the dataset
- The dataset has several issues, which need to be fixed
  - Cleaning, biased, need for augmentation
- The models:
  - Reproduced the results of the cited paper
  - Data trained on one person is also applicable on an unknown person
    - Still better, if it is personalised

Spark Hadoop

- The library helped:
    - To manage a huge dataset
    - Build pipelines to preprocess
    - Gain statistical knowledge
    - Build windows, to partitioned data and focus on relevant records
    -