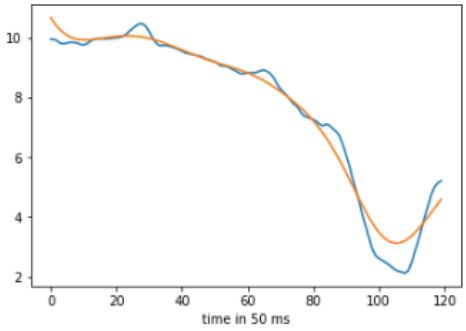
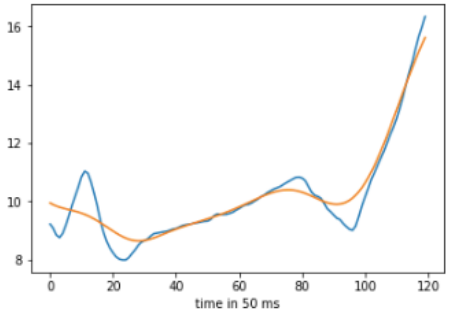
1. KINEXON Challenge

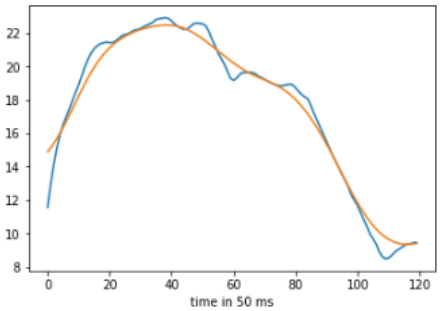
- The task was to develop a predictive model that is able to predict shots from a series of given labels
- Three data sets were provided with a time series of positional data and events where a shot was found

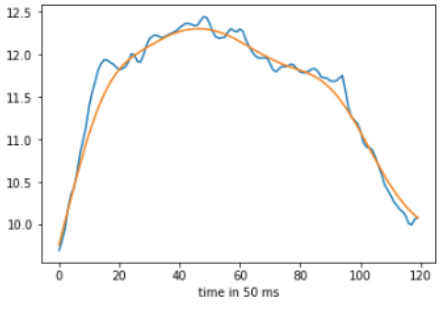
2. Data Preprocessing

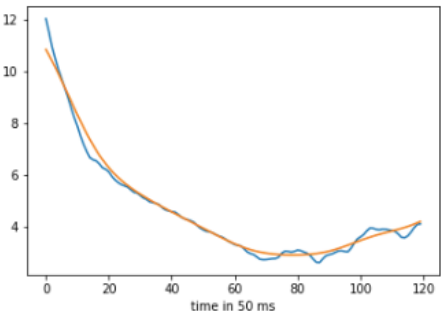
- The data was given in two files with inconsistent naming and thus, was merged and renamed to be consistent
- Further, different ball ids where given in data set 3
- Preprocessing
    - The third data set was split according to the ball id. As there were 3 ids this gave three datasets -> thus in the end 5 distinct datasets with following indexing
        - Data set 1: Corresponds to 180325_match1sthalf_*.csv
        - Data set 2: Corresponds to 180325_match2ndhalf_*.csv
        - Data set 3: Corresponds to 180428_match_*.csv. first ball id: ball2 ball
        - Data set 4: Corresponds to 180428_match_*.csv. second ball id: ball1 ball
        - Data set 5: Corresponds to 180428_match_*.csv. third ball id: ball ball3'
    - Columns were renamed and irrelevant columns removed giving columns
        - Columns positions: ts, xpos, ypos, idball
        - Columns evevents ts, distance, speed, idball
    - No resampling was required as the data is already sampled with frequency 50 ms
    - No column had a nan entry so no filling was required
    - Crop each timeseries 60 seconds before the first and after last shot event in the respective data set
- Computing Speed
    - Speed seems to be an interesting measure in this context as it contains information about if a ball sped up or slowed down which definitely makes sense to consider for shot detection
    - Computation was done as follows
        - First perform smoothing on xpos and ypos introduced on
            - *https://scipy-cookbook.readthedocs.io/items/SignalSmooth.html*
            - Description of this smoothing according to website: *"This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected window-length copies of the signal at both ends so that boundary effect are minimized in the beginning and end part of the output signal"*
            - I chose a window size of 41 for the convolution which corresponds to approx. two seconds as this seems to reduce noise such that good intuitive values result for the smoothed timeseries
            - The resulting time series was shifted back to fit the initial series as smoothing shifts it forward
            - Examples show that the fit is quire goos -> those examples of smoothed windows per data set for ypos are:
                - DS1:

- DS2:
  - 

- DS3:
  - 

- DS4:
  - 

- DS5:
  - 

  - 

- Extraction of turning points

- Apart from standard segmentation (=sliding windowing) I tried an alternative segmentation
- The idea: That when a shot occurs there is an increase in speed up to a maximum speed. From there the speed goes down again. Thus, the idea is to split the data at all maxima and use the window before each maxima
- Why not sliding window? Sliding window works well, but is potentially computationally expensive and compares windows at different points of a shot. The above approach performs a segmentation which includes multiple data points and thus, way less segments -> one question I asked myself was to check if this approach works and how well. However, also sliding window was used as the standard way to tackle this.

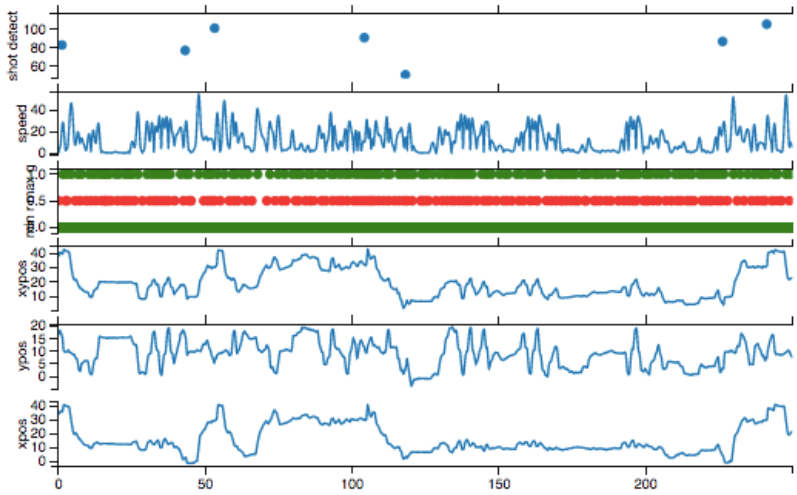
- Further pre-processing was performed during feature extraction which is described further below with the approaches

3. Data Exploration

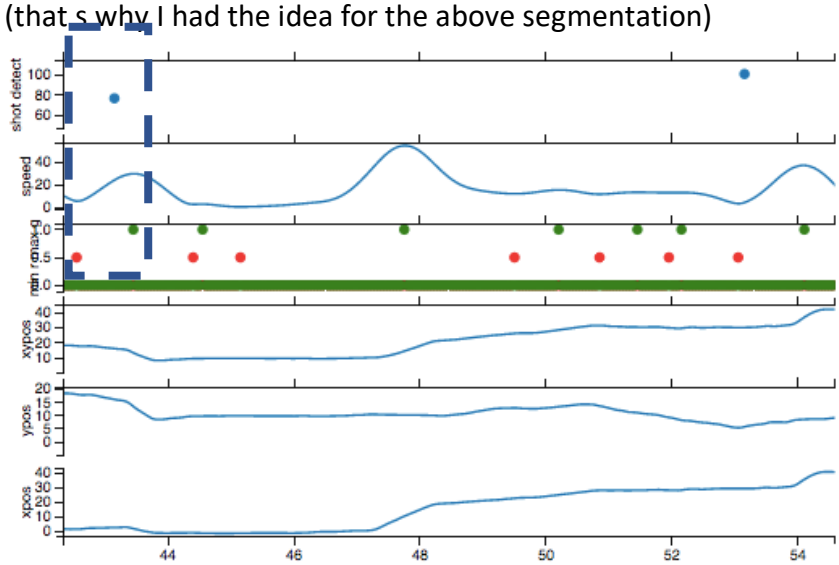
- For data exploration I used jupyter notebook. Please refer to the file *Classification.ipynb* to see the exploration I performed, as well as the above pre-processing that was used
- Explored statistics include the following
  - Per Dataset extract Statistics
    - Notes on running jupyter:
      - If plotting shows error:
      - ndarray not JSONSerializable -→ use python -m pip install --user "git+https://github.com/javadba/mpld3@display_fix"
      - also you might run jupyter with extended data rate
        - jupyter notebook --NotebookApp.iopub_data_rate_limit=1e10
    - Descriptions Examle:

```
--------------------
Dataset 0
--------------------
--------------------> Events

                 ts   distance       speed  idball
count  5.600000e+01  56.000000   56.000000    56.0
mean   1.521991e+12   8.107679   85.588036  1357.0
std    6.223290e+05   1.471382   17.033872     0.0
min    1.521990e+12   4.880000   50.100000  1357.0
25%    1.521990e+12   6.805000   76.380000  1357.0
50%    1.521991e+12   8.430000   86.990000  1357.0
75%    1.521991e+12   9.460000   97.485000  1357.0
max    1.521992e+12  10.210000  113.690000  1357.0

--------------------> Positions

                 ts          xpos          ypos   idball        xspeed  \
count  4.080700e+04  40807.000000  40807.000000  40807.0  40807.000000
mean   1.521991e+12     18.968134     10.055867   1357.0      0.005280
std    5.890055e+05     10.858453      4.776372      0.0     10.638984
min    1.521990e+12     -2.411000     -3.658000   1357.0    -54.368659
25%    1.521990e+12      9.929500      7.658000   1357.0     -4.484483
50%    1.521991e+12     19.375000      9.858000   1357.0     -0.019136
75%    1.521991e+12     29.320500     12.938000   1357.0      3.845820
max    1.521992e+12     43.189000     20.637000   1357.0     58.074069

              yspeed         speed         xypos       xsmooth       ysmooth
count  40807.000000  40807.000000  40807.000000  40807.000000  40807.000000
mean      -0.012279     11.785434     22.652625     18.968152     10.055874
std       11.638680     10.476028      9.406785     10.790431      4.575839
min      -40.628849      0.000000      1.604666     -1.908969     -2.605561
25%       -4.202186      3.141509     14.869033      9.979394      7.593278
50%        0.008755      8.868705     21.678598     19.326054      9.886231
75%        3.991653     18.272862     30.867437     29.335123     12.990044
max       42.939602     58.088481     45.188829     41.656623     20.362325
```

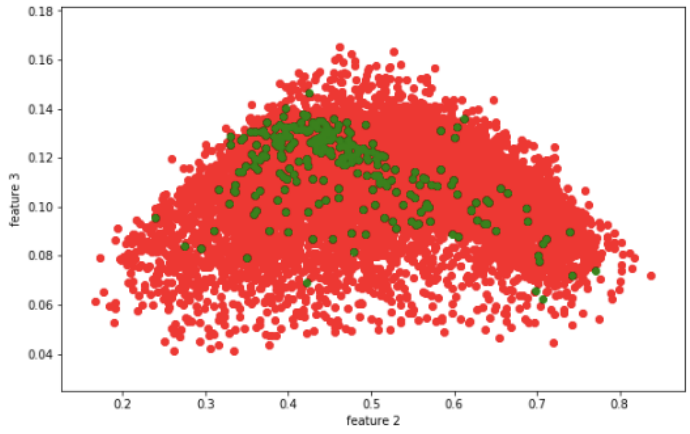
- Plot all relevant data to interactively check it out using mpld3 (shows subrange only)

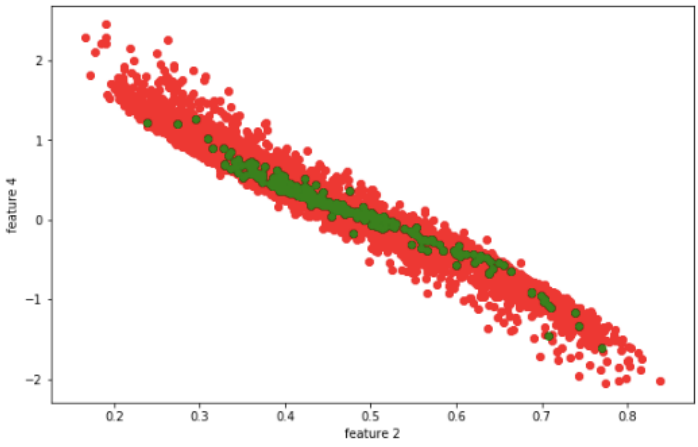


- With rows
  - Shots with its speed
  - Speed
  - Turning points upper green=Maxima, red= Minima lower green irrelevant
  - Xypos = Manitude of xpos and ypos
  - ypos
  - xpos
- Zooming in see that shot seems to be often between minima and maxima (that s why I had the idea for the above segmentation)



- The feature based approach gave features per window (see below), those can be plotted pairwise to see if visually discrimination is found between shot and not shot windows – just to get an impression of the data distribution
  - Examples:

- 

- 

4. Build Classifier to detect Shots using ML

- Next, a predictive model is built. I tried two approaches. This is given in main.py
- A coarse overview of the steps:
    - First
        - Use sliding windows and the above described maxima segmentation to find windows
        - Assign a label with shot if a shot was found in the window and a label no shot if no shot was found in the window
        - Extract standard features from it mean, max, skew, etc.
        - Select the best features using a validation dataset
        - Train a classifier -> Random Forest worked best
        - Test the classifier using the test data set
    - Second:
        - Use sliding windows approach to find windows of fixed length
        - Assign a label per window
        - Train a Deep Learning classifier LSTM using training and validation set
        - Evaluate model with test data set

- This is detailed in the following

a. Feature Based Approach

- All parameters for this approach can be set in main.py (I will describe the procedure in detail in the following)

- First, do the segmentation (together with above preprocessing)
  - In main.py:
    - Set resegment=True
    - Choose segmentation to use:
      - Sliding window:

```
segmentation = SEGMENTATION.SLIDING_WINDOW # Different segmentation techniques
resegment = True # if true perform segmentation - else run ML
```

      - Maxima Segmentation:

```
segmentation = SEGMENTATION.MAX_SPEED_1_SEC #
resegment = True # if true perform segmentati
```

  - 1st way Sliding window:
    - In this approach with a step size of 3 a window of 30 steps (=1.5s) is used
    - Per window a set of features is extracted resulting in a feature vector per window
    - Per window also a label is stored 1 if a shot was in this window and 0 else -> stored in a label vector
  - 2nd way Maxima Segmentation:
    - In this approach all maxima of the speed signal are considered as segmentation points
    - Then, a window of one second before the segmentation point is used as window
    - Features and labels are extracted as in the sliding window approach
- Window preprocessing
  - Input timeseries are: speed, xpos, ypos, xsmooth (smoothed x), ysmooth, absxpos (absolute x pos), absypos
  - Each signal in the window is preprocessed as follows
    - xpos: This signal was centred to start at point zero as I assume that similar shots should have similar movements – which can only be compared if they start at same times. Also it does not matter if someone shoots on the left or the right goal when it comes to classification. Thus, a shot on the right goal will be inverted such that it looks as going left. With this I can treat left and right shots the same.
    - ypos: same centering as in xpos. Same it does not matter if someon shoots from top or bottom. Invert bottom shots to be looking like top shots.
    - Xsmooth and ysmooth: identical preprocessing as xpos and ypos
    - Speed: nothing particular
    - ALSO all signals were normalized to be between 0 and 1

- Feature Extraction

    - Per window extract a set of features from EACH signal to capture characteristics of the window
    - This results in a feature vector per window: Mean, std, Min, Max, Skew, kurtosis, gradient mean, gradient std
    - A label is set if a shot was within the window giving a label vector
- This will all be done when running main.py with above parameters

- Train Test validation split
    - As described above 5 data sets result. I used 2 for training 1 for validation and 2 for testing (see main.py for which ones I used) – not the usual 60:20:20 split. In my experiments the test set was 40% of the training set and validation set is still around 20% but within this evaluation it should not matter in practice more training samples are always better

- Preprocessing the feature vector
    - Normalization:
        - First I normalized all features to be between 0 and 1
    - Balancing:
        - The data set is highly imbalanced so I used SMOTETOMEK which is an approach that uses under and oversampling to balance the data set (without this the classifier would prefer the dominant label)

- Next, run feature selection to find the best features. For this in main.py
    - Set resegment=False
    - Set approach = APPROACH.FEATURE_BASED
    - Set the segmentation to one of the above
    - Set
        - fb_feature_selection = True
        - fb_apply_selection = True
        - fb_best_features = []
        - Note if fb_best_features is empty feature selection (FW BW search) is run to find the best features – if it is a set of numbers the features corresponding to the numbers are used as features
    - To reproduce my result use random.seed(123) # throughout all results shown here (I tried it with different seeds -> gives similar results)
- Feature selection
    - So before running it should look like this

```
approach = APPROACH.FEATURE_BASED # Different
segmentation = SEGMENTATION.MAX_SPEED_1_SEC #
resegment = False # if true perform segmentat
random.seed(123)

# Params for FB Approach
fb_normalize = True # use normalization
fb_balance = True # use balancing
fb_feature_selection = True # run feature sel
fb_apply_selection = True # apply fw bw featu
fb_best_features = []#24, 35, 4, 3, 19, 34, 2
fb_clf = RandomForestClassifier()
fb_pca = False # run pca -> seems to not make
fb_n_pca = 20 # number of pca components
fb_decision_pca = False # show plot of releva
fb_train_plot_bad = False # plot examples tha
```

- For the
  - Maxima segmentation this gave as best features (on the validation set)
    - [33, 25, 17, 35, 6, 5, 24, 8, 9, 23, 0, 30, 14, 41, 42, 43, 16, 18, 37, 1, 29, 27, 15, 7, 12]
  - Sliding window this gave as best features (on the validation set)
    - [38, 33, 25, 36, 35, 12, 22, 6, 29, 43, 9, 13, 5, 27, 24, 0, 23, 30, 15, 42, 37, 1, 40, 2, 11]

- Feature transformation
  - Initially I tried to do a PCA to reduce dimensions. But after selection dimensionality is already low so removing further ones did not give increased performance
  - Thus, PCA is omitted here

- Training:
  - With the optimal feature selection the classifier can be trained. For this the selection set needs to be provided e.g.
    - fb_best_features = [33, 25, 17, 35, 6, 5, 24, 8, 9, 23, 0, 30, 14, 41, 42, 43, 16, 18, 37, 1, 29, 27, 15, 7, 12]
  - I tried LogisticRegression, Naïve Bayes, SVMs and Random Forest. Random Forest was the only classifier that gave acceptable results
- Results
  - To get the testing results run: fb_test=True
  - For both segmentation windows I was able to get a moderate score
  - Running it on maxima segmentation gave

Performance Test set

	precision	recall	f1-score	support
No Shot	0.98	0.99	0.99	2448
Shot	0.58	0.47	0.52	80
accuracy			0.97	2528
macro avg	0.78	0.73	0.76	2528
weighted avg	0.97	0.97	0.97	2528

    - So basically every second to third shot was detected and a little less than half of the classified shots are no shots
    - A F1 Score of 52 per cent is ok but can be improved.

- So the maxima segmentation is not so promising after all. Probably it would require some further optimization and more training samples as here the window number is reduce by choosing this segmentation – I decided to implement other approaches instead (as time is limited)
- For evaluation the estimated labels are assumed to be correct if they detected a shot within a epsilon of 6 == 300ms as sometimes the window before or after the true label was classified as shot

- Running it on sliding window segmentation gave

```
Performance Test set
              precision    recall  f1-score   support

      No Shot       0.99      0.99      0.99     14317
         Shot       0.75      0.85      0.80       628

     accuracy                           0.98     14945
    macro avg       0.87      0.92      0.89     14945
 weighted avg       0.98      0.98      0.98     14945
```

- Here also if the detection window (i.e. the estimate classified the window as shot) overlaps with the true window here the labels are adjusted such that all labels of the detection window for the overlap are assumed correct -> is more fair then saying that the non overlapping part of the detection window is counting as wrong

- So the feature based approach works pretty well with a F1 score of 80 per cent. Probably with some optimization it might be improved. I.e. more hyperparameter tuning on the classifier better segmentation etc.
- However, for my limited time I decided to not continue optimizing, but rather implemented a more state of the art approach which is deep learning i.e. LSTMs

b. LSTM Approach

- All parameters for this approach can be set in main.py (I will describe the procedure in detail in the following)

- First, do the segmentation (together with above preprocessing)
  - In main.py:
    - Set resegment=True
    - Main.py should look like this

```
segmentation = SEGMENTATION.LSTM_RAW # Different segmentation techniques
resegment = True # if true perform segmentation - else run ML
```

    - Choose segmentation to use. Here only a sliding window approach was implemented. For this an area of +- 250 ms before and after the shot are labelled as shot (as I do not know if your algorithm always produces the exact same location). Then, with a step size of 3 a window of 20 steps == 1 second is used.

        - Per window do the window processing described below to prepare the timeseries. Then, store the whole sequence and all its features per window. Further a label per window is stored, which is true if a shot label as described above is in this window.

    - Window preprocessing
        - Input timeseries are: speed, xpos, ypos, xsmooth (smoothed x), ysmooth, absxpos (absolute x pos), absypos
        - Each signal in the window is preprocessed as follows
            - xpos: This signal was centred to start at point zero as I assume that similar shots should have similar movements – which can only be compared if they start at same times. Also it does not matter if someone shoots on the left or the right goal when it comes to classification. Thus, a shot on the right goal will be inverted such that it looks as going left. With this I can treat left and right shots the same.
            - ypos: same centering as in xpos. Same it does not matter if someon shoots from top or bottom. Invert bottom shots to be looking like top shots.
            - Xsmooth and ysmooth: identical preprocessing as xpos and ypos
            - Speed: nothing particular
            - ALSO all signals were normalized to be between 0 and 1
        - The idea: the normalized xpos, ypos etc. capture the shape of the shot, while the absolute x and y values prefer to predict shots at close proximity to the goal.
        - With this, basically, all signals are normalized and can be used as an input for DL
- Training the model
    - Architecture:
        - I chose an architecture of two consecutive LSTMs. Those are state of the art models to handle the prediction of time series data as they are able to capture the characteristics of a sequence, which is exactly what we need here

    - Parameterization
        - As parameters I found the following to work quite well
            - The input dimensions of the first LSTM
                - lstm_lstm1_dim=16
            - And of the second LSTM
                - lstm_lstm2_dim=8
            - Here the dimension of the model captures its capacity. Choosing a big value yields a big loss during training and bad validation performance. This problem is low dimensional (7 dims) so a small dimension here is sufficient
            - For regularization (i.e. to avoid overfitting on the data that we used for training) for both LSTMS a dropout percentage of stm_dropout_perc=0.1 and a L1-penalty of lstm_l1_penalty=.01
            - The learning rate was lstm_learning_rate=.01

- The loss metric during optimization was lstm_loss='mean_squared_error' and the optimizer is the state of the art optimizer called ADAM
- As a batch size for training I used lstm_batch_size=1000 and I trained the model over lstm_epochs=800 epochs
- The validation split during training was lstm_validation_split=0.1, where the training data and the validation data is first merged and then fed into the KERAS model. Keras then automatically takes 10 percent of that data for computation of the validation loss

  - Balancing
    - The data set we have is very unbalanced. So when the LSTM is trained, during optimization mini-batches nearly always contain no shot labels. This results in a predictor that is highly pessimistic and never predicts a shot. Therefore, I copied the training data with shot labels multiple times until it had a ratio that is similar to the number of no shot labelled data points. With this, the mini-batches always contain both types in a more equal ratio.
    - The amount of balancing can be adjusted with the parameter balance_ratio = 0.8

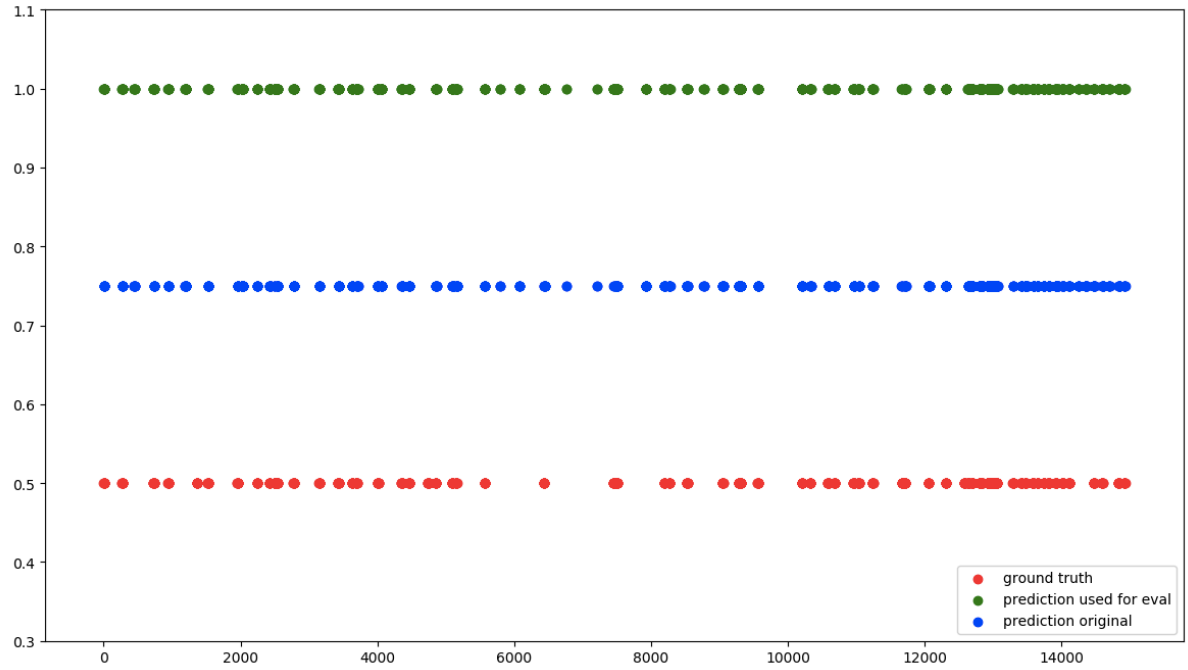
  - With the balanced data keras is used to train the model, which can be set using
    - lstm_training = True
    - This model will be stored and used for any further prediction
    - NOTE: I pretrained the model with above parameters (on my BMW workstation of high power). If you want run this training yourself you can do so. However, depending on your HW it might take time. So I recommend to leave lstm_training = False, as it will load the model which you can use to do the prediction

- Testing
  - The resulting classfier can be used for testing if you set lstm_testing=True
  - It will output a prediction per window
    - Here also if the detection window (i.e. the estimate classified the window as shot) overlaps with the true window here the labels are adjusted such that all labels of the detection window for the overlap are assumed correct -> is more fair then saying that the non overlapping part of the detection window is counting as wrong

  - Decision Threshold
    - Also, the LSTM will output a continuous estimate between 0 and 1. This is pretty useful, as the number indicates somewhat the confidence of the prediction, i.e. if it predicts a value closer to 1 for a window it is more sure that this window has a shot.
    - Therefore, I added a decision threshold dec_threshold = 0.8 which estimates a shot to be a shot if the prediction is higher than this value. Else it is considered no shot.
  - With this running the trained model on the test set gave me

	precision	recall	f1-score	support
No Shot	1.00	0.98	0.99	14213
Shot	0.76	0.94	0.84	720
accuracy			0.98	14933
macro avg	0.88	0.96	0.92	14933
weighted avg	0.99	0.98	0.98	14933

- For each testing I also output the comparison of prediction and truth, which gave here for testing



  - Note in green is the adjustment made (i.e. overlapping is considered right) on the truth labels, while the blue dots is exactly what keras gives us
  - We see that the classifier is pretty good in estimating shots already. At some periods (middle part) it seems to predict shots too often. However, it fits pretty well.
  - This can also be read from the classification table. We see that it is able to predict 76 per cent of shots with a recall of 94 per cent, which is pretty good already.
  - This can be further optimized with more time. But, as this is a challenge and as a F1 score of 84 per cent is already pretty good I will stop at this point

- Summary:
  - So we see that the LSTM is pretty good in capturing shots, and perform slightly better than the feature based approaches