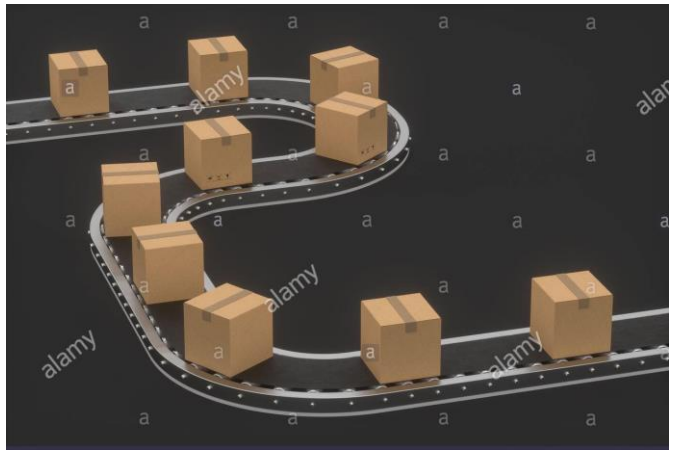
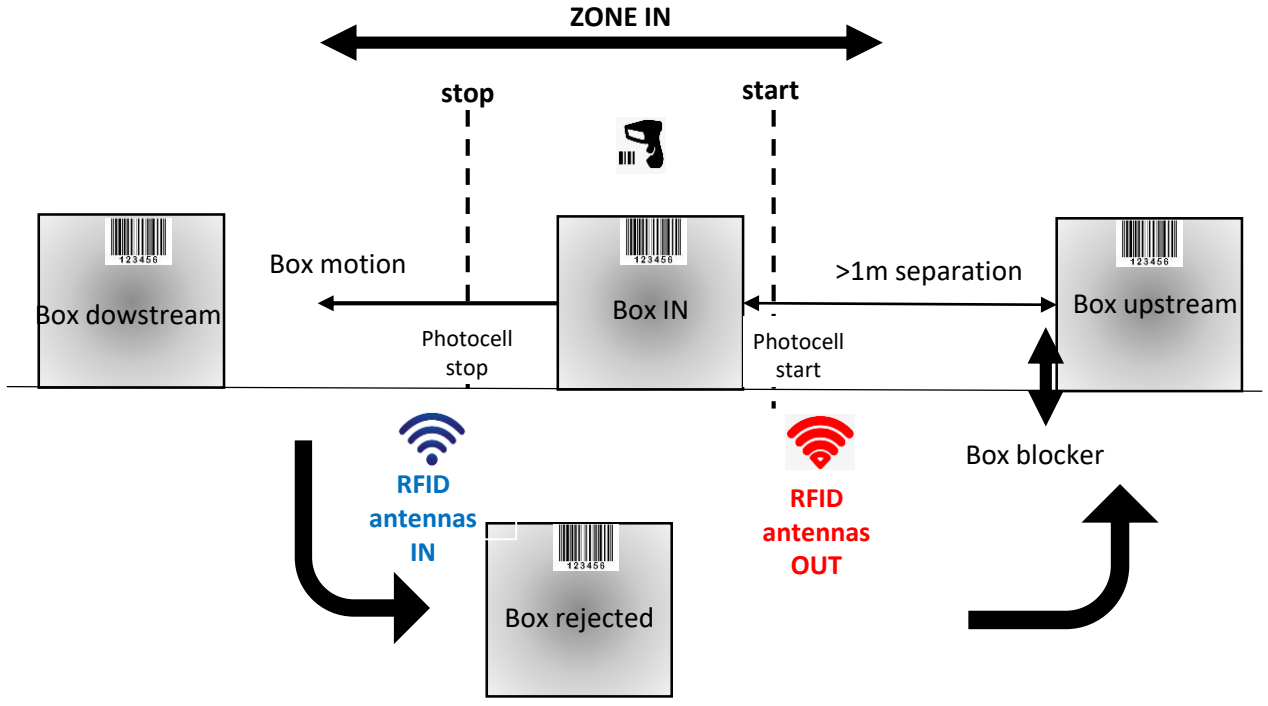
RFID TUNNEL

Tunnel: overall setup and problem to solve



Logistics application: boxes on conveyor
Boxes have RFID tags inside attached on each products (apparel and footwear)
Boxes are identified with a bar code



« zone IN »: one box at a time (separated by box blocker)

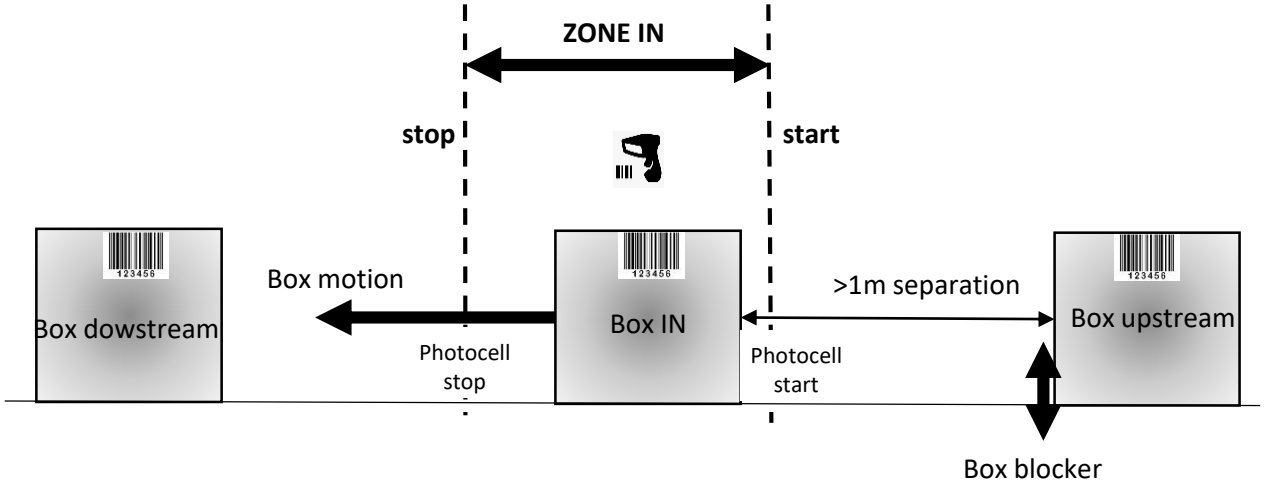
When box hits stop sensor, box content (tags inside) is predicted:
1. If it matches the expected list, box runs through
2. If not, the box is rejected for manually checking and then rerun in zone IN

Problem to solve: box content prediction = tag localization in the right box

Zone IN

Zone IN:

1. Box: one at time (2m separation with box blocker)
2. Photocells: start/stop window
3. Window ID: box bar code



RFID detection: inaccurate localization information

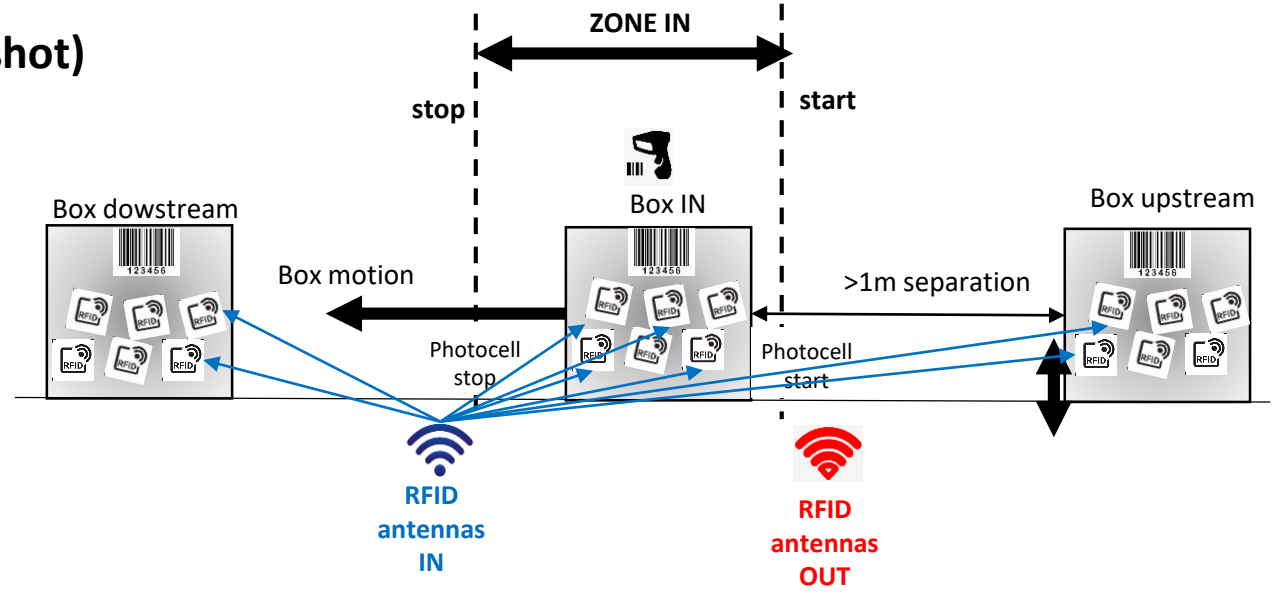
RFID antennas: OUT (upstream zone IN), IN (downstream zone IN)

RFID detection has a large coverage up to +/-4m
boxes separation: 2m
Therefore, IN and OUT antennas detect tags:

1. from box in zone IN
2. also from boxes upstream or downstream zone IN, and even further away

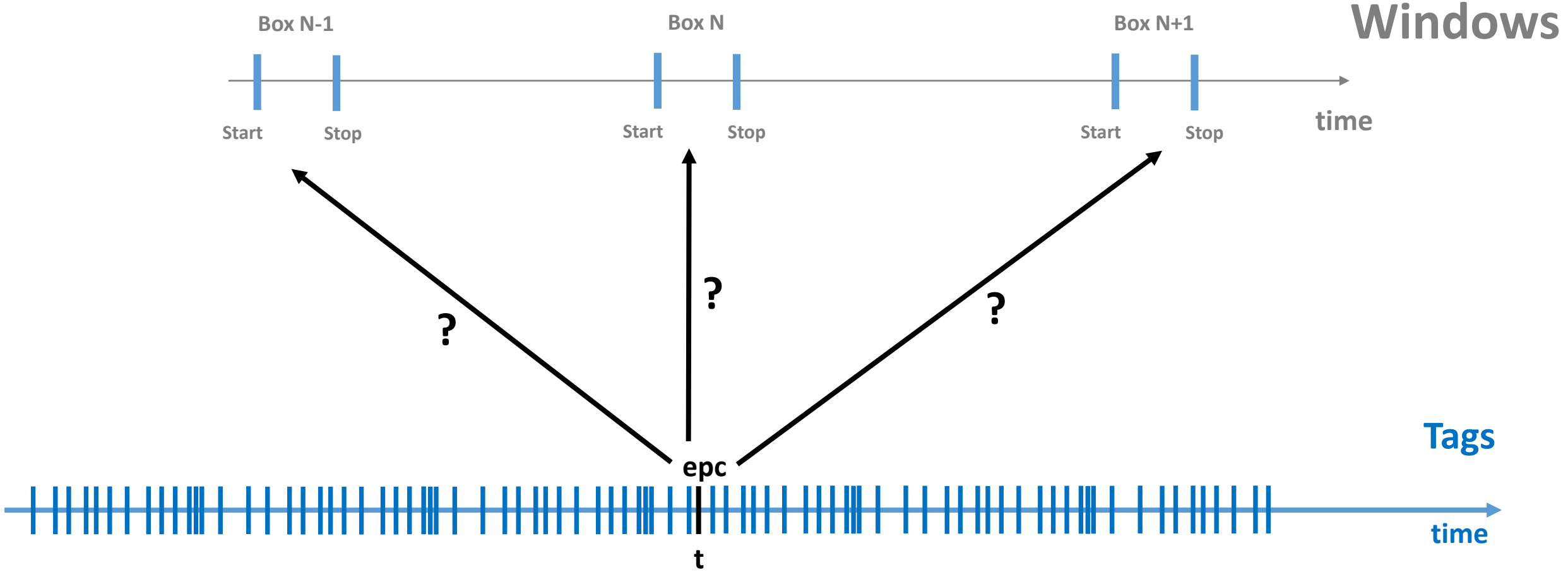
➔ to localize tags in the right box, simple RFID detection is not enough and RFID measurements analysis is needed

Spatial representation (snapshot)



Problem to solve: localize tags in the right box

Time representation



Raw datasets

Operation generates csv files converted into 3 raw dataframes:

1. reflist: epc, box
2. tags: epc, logtime, antenna, rssi
3. windows: window_id, start, stop

Note: window_id is the box reference

Data clean up:

1. tags and windows: restricted to the shared time interval
2. tags restricted to epcs in reflist (but all windows are kept, even without reflist correspondance) for non ambiguous ground truth

```
reflist.head()
```

	Epc	box
0	epc0	box238
1	epc1	box238
2	epc2	box206
3	epc3	box206
4	epc4	box206

```
tags.head()
```

	Epc	LogTime	Rssi	loc
0	epc810	2020-05-04 09:50:40.377	-73.0	out
1	epc809	2020-05-04 09:50:43.277	-72.0	out
2	epc809	2020-05-04 09:50:45.336	-73.0	out
3	epc812	2020-05-04 09:50:46.716	-74.0	out
4	epc809	2020-05-04 09:50:49.356	-71.0	out

```
windows.head()
```

	window_id	Start	Stop
0	box0	2020-05-04 09:50:53.422	2020-05-04 09:50:55.189
1	box1	2020-05-04 09:51:40.317	2020-05-04 09:51:42.091
2	box2	2020-05-04 10:03:20.725	2020-05-04 10:03:22.497
3	box3	2020-05-04 10:06:33.731	2020-05-04 10:06:35.575
4	box4	2020-05-04 10:08:21.239	2020-05-04 10:08:23.002

Data structuring: reflist

Q, runs_box

```
Q = reflist.groupby('box')['Epc'].nunique().rename('Q').reset_index(drop=False)
reflist = pd.merge(reflist, Q, on='box')
#
runs_box = windows.groupby('window_id').size().rename('runs_box').reset_index(drop=False)
reflist = pd.merge(reflist, runs_box, left_on='box', right_on='window_id', how='left')\
            .drop(columns=['window_id'])
reflist.head()
```

	Epc	box	Q	runs_box
0	epc0	box238	2	1
1	epc1	box238	2	1
2	epc2	box206	3	1
3	epc3	box206	3	1
4	epc4	box206	3	1

Data structuring: tags_reflist

tags merges with reflist based on epc ➔ tags_reflist

```
tags.head()
```

	Epc	LogTime	Rssi	loc
0	epc810	2020-05-04 09:50:40.377	-73.0	out
1	epc809	2020-05-04 09:50:43.277	-72.0	out
2	epc809	2020-05-04 09:50:45.336	-73.0	out
3	epc812	2020-05-04 09:50:46.716	-74.0	out
4	epc809	2020-05-04 09:50:49.356	-71.0	out

```
reflist.head()
```

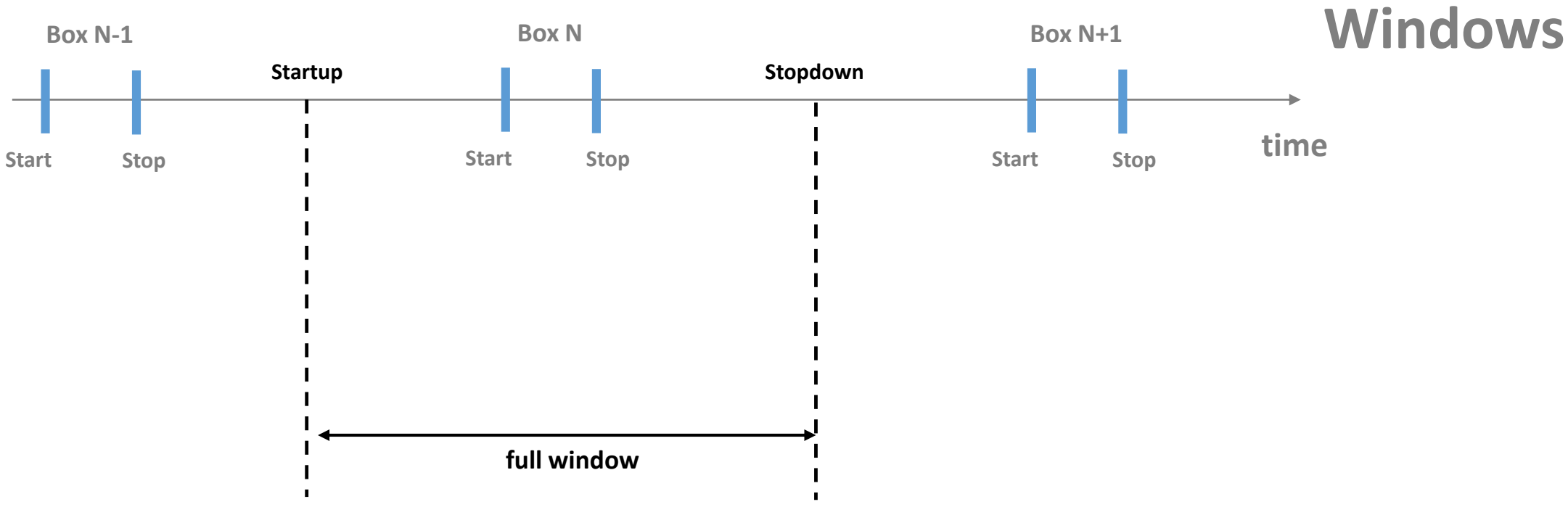
	Epc	box	Q	runs_box
0	epc0	box238	2	1
1	epc1	box238	2	1
2	epc2	box206	3	1
3	epc3	box206	3	1
4	epc4	box206	3	1

```
tags_reflist=pd.merge(tags, reflist , on='Epc', how='left')
tags_reflist.head()
```

	Epc	LogTime	Rssi	loc	box	Q	runs_box
0	epc810	2020-05-04 09:50:40.377	-73.0	out	box0	4	1
1	epc809	2020-05-04 09:50:43.277	-72.0	out	box0	4	1
2	epc809	2020-05-04 09:50:45.336	-73.0	out	box0	4	1
3	epc812	2020-05-04 09:50:46.716	-74.0	out	box0	4	1
4	epc809	2020-05-04 09:50:49.356	-71.0	out	box0	4	1

Data structuring: windows

Startup: in between Stop previous box (N-1) and Start box (N)
Stopdown: in between Stop box N and Start next (box N+1)
Full window



Data structuring: windows

runs (1 if single run, >=2 if rerun) ➔ run_id ➔ window_run_id: unique key
Startup, Stopdown

```
windows.head()
```

	window_id	Start	Stop
0	box0	2020-05-04 09:50:53.422	2020-05-04 09:50:55.189
1	box1	2020-05-04 09:51:40.317	2020-05-04 09:51:42.091
2	box2	2020-05-04 10:03:20.725	2020-05-04 10:03:22.497
3	box3	2020-05-04 10:06:33.731	2020-05-04 10:06:35.575
4	box4	2020-05-04 10:08:21.239	2020-05-04 10:08:23.002

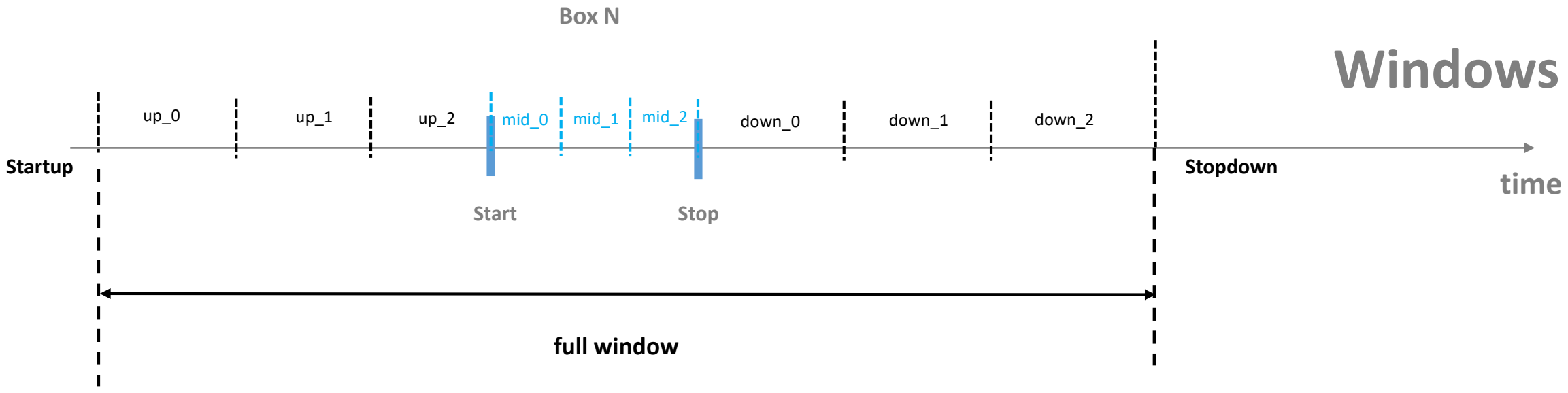
```
windows.head()
```

	window_run_id	window_id	run_id	Startup	Start	Stop	Stopdown	runs
0	box0_0	box0	0	2020-05-04 09:50:40.377	2020-05-04 09:50:53.422	2020-05-04 09:50:55.189	2020-05-04 09:51:17.753	1.0
1	box1_0	box1	0	2020-05-04 09:51:17.753	2020-05-04 09:51:40.317	2020-05-04 09:51:42.091	2020-05-04 09:57:31.408	1.0
2	box2_0	box2	0	2020-05-04 09:57:31.408	2020-05-04 10:03:20.725	2020-05-04 10:03:22.497	2020-05-04 10:04:58.114	1.0
3	box3_0	box3	0	2020-05-04 10:04:58.114	2020-05-04 10:06:33.731	2020-05-04 10:06:35.575	2020-05-04 10:07:28.407	1.0
4	box4_0	box4	0	2020-05-04 10:07:28.407	2020-05-04 10:08:21.239	2020-05-04 10:08:23.002	2020-05-04 10:14:11.755	1.0

Data structuring: subslices

up_0, up_1, up_2: 3 subslices from Startup to Start
mid_0, mid_1, mid_2: 3 subslices from Start to Stop
down_0, down_1, down_2: 3 subslices from Stop to Stopdown

Note: steps=4 (3 intervals) is champion value (annex)



Data structuring: subslices

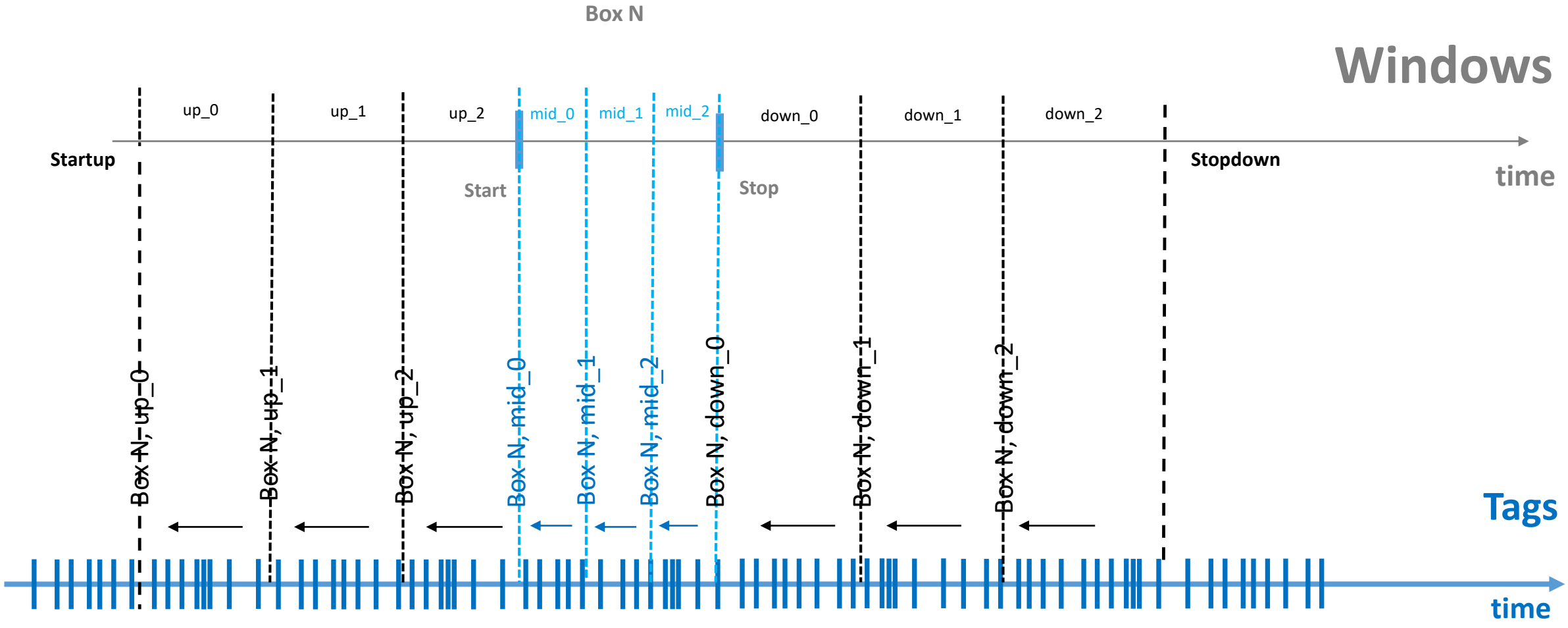
windows ➔ subslices with subsliceStart

```
subslices[:10]
```

	window_run_id	subslice_id	subsliceStart	runs
0	box0_0	up_0	2020-05-04 09:50:40.377000000	1.0
1	box0_0	up_1	2020-05-04 09:50:44.725333333	1.0
2	box0_0	up_2	2020-05-04 09:50:49.073666666	1.0
3	box0_0	mid_0	2020-05-04 09:50:53.422000000	1.0
4	box0_0	mid_1	2020-05-04 09:50:54.011000000	1.0
5	box0_0	mid_2	2020-05-04 09:50:54.600000000	1.0
6	box0_0	down_0	2020-05-04 09:50:55.189000000	1.0
7	box0_0	down_1	2020-05-04 09:51:02.710333333	1.0
8	box0_0	down_2	2020-05-04 09:51:10.231666666	1.0
9	box1_0	up_0	2020-05-04 09:51:17.753000000	1.0

Data structuring: tags_windows

Tags and windows are merged with merge_asof (tags: LogTime and windows: subsliceStart)
All tags in a given subslice slot are associated to the left hand subslice_id



Data structuring: subslices

windows ➔ subslices

```
subslices[:10]
```

	window_run_id	subslice_id	subsliceStart	runs
0	box0_0	up_0	2020-05-04 09:50:40.377000	1.0
1	box0_0	up_1	2020-05-04 09:50:46.899500	1.0
2	box0_0	mid_0	2020-05-04 09:50:53.422000	1.0
3	box0_0	mid_1	2020-05-04 09:50:54.305500	1.0
4	box0_0	down_0	2020-05-04 09:50:55.189000	1.0
5	box0_0	down_1	2020-05-04 09:51:06.471000	1.0
6	box1_0	up_0	2020-05-04 09:51:17.753000	1.0
7	box1_0	up_1	2020-05-04 09:51:29.035000	1.0
8	box1_0	mid_0	2020-05-04 09:51:40.317000	1.0
9	box1_0	mid_1	2020-05-04 09:51:41.204000	1.0

```
tags_reflist.head()
```

	Epc	LogTime	Rssi	loc	box	Q	runs_box
0	epc810	2020-05-04 09:50:40.377	-73.0	out	box0	4	1
1	epc809	2020-05-04 09:50:43.277	-72.0	out	box0	4	1
2	epc809	2020-05-04 09:50:45.336	-73.0	out	box0	4	1
3	epc812	2020-05-04 09:50:46.716	-74.0	out	box0	4	1
4	epc809	2020-05-04 09:50:49.356	-71.0	out	box0	4	1

```
tags_windows[:10]
```

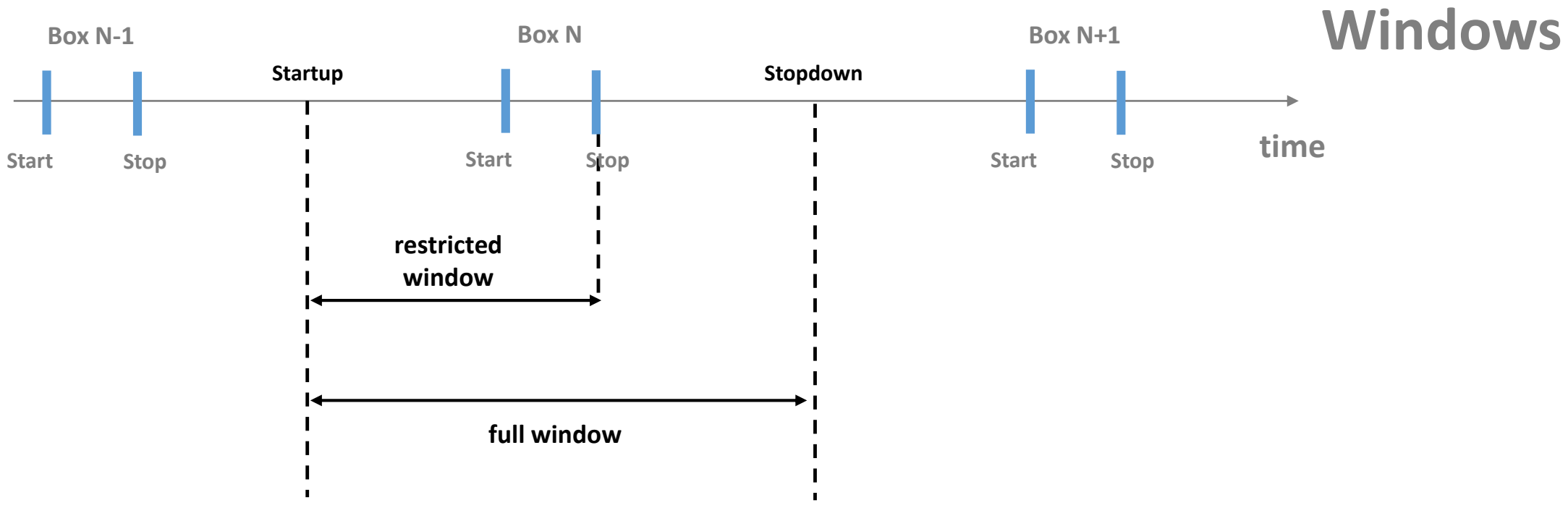
	Epc	LogTime	Rssi	loc	box	Q	runs_box	window_run_id	subslice_id	runs
0	epc810	2020-05-04 09:50:40.377	-73.0	out	box0	4	1	box0_0	up_0	1.0
1	epc809	2020-05-04 09:50:43.277	-72.0	out	box0	4	1	box0_0	up_0	1.0
2	epc809	2020-05-04 09:50:45.336	-73.0	out	box0	4	1	box0_0	up_0	1.0
3	epc812	2020-05-04 09:50:46.716	-74.0	out	box0	4	1	box0_0	up_0	1.0
4	epc809	2020-05-04 09:50:49.356	-71.0	out	box0	4	1	box0_0	up_1	1.0
5	epc812	2020-05-04 09:50:49.916	-73.0	out	box0	4	1	box0_0	up_1	1.0
6	epc809	2020-05-04 09:50:50.017	-64.0	out	box0	4	1	box0_0	up_1	1.0
7	epc812	2020-05-04 09:50:50.157	-66.0	out	box0	4	1	box0_0	up_1	1.0
8	epc809	2020-05-04 09:50:50.357	-61.0	out	box0	4	1	box0_0	up_1	1.0
9	epc812	2020-05-04 09:50:50.497	-65.0	out	box0	4	1	box0_0	up_1	1.0

Data structuring: full/restricted observation

Decision point: box content prediction given @ rejection point (activated if content prediction not matching expected list)
Full/restricted observation:

1. decision point @Stop
2. Decision point @Stopdown (further down, meaning longer zone IN)

The larger the window, the better the observation quality



Data structuring: restricted observation

Restricted observation (limited at Stop) ➔ 0.24% tags are discarded

```
#
# restricted windows
#
```

```
# subslices_restricted=['up', 'mid', 'down']
subslices_restricted=['up', 'mid']
tags_restricted = tags_windows [tags_windows['subslice_id']\
                                                .apply(lambda x:x.split('_')[0] in subslices_restricted)]
```

```
len(tags_windows), len(tags_restricted), tags_windows['Epc'].nunique(), tags_restricted['Epc'].nunique(), \
    tags_restricted['Epc'].nunique() / tags_windows['Epc'].nunique()
```

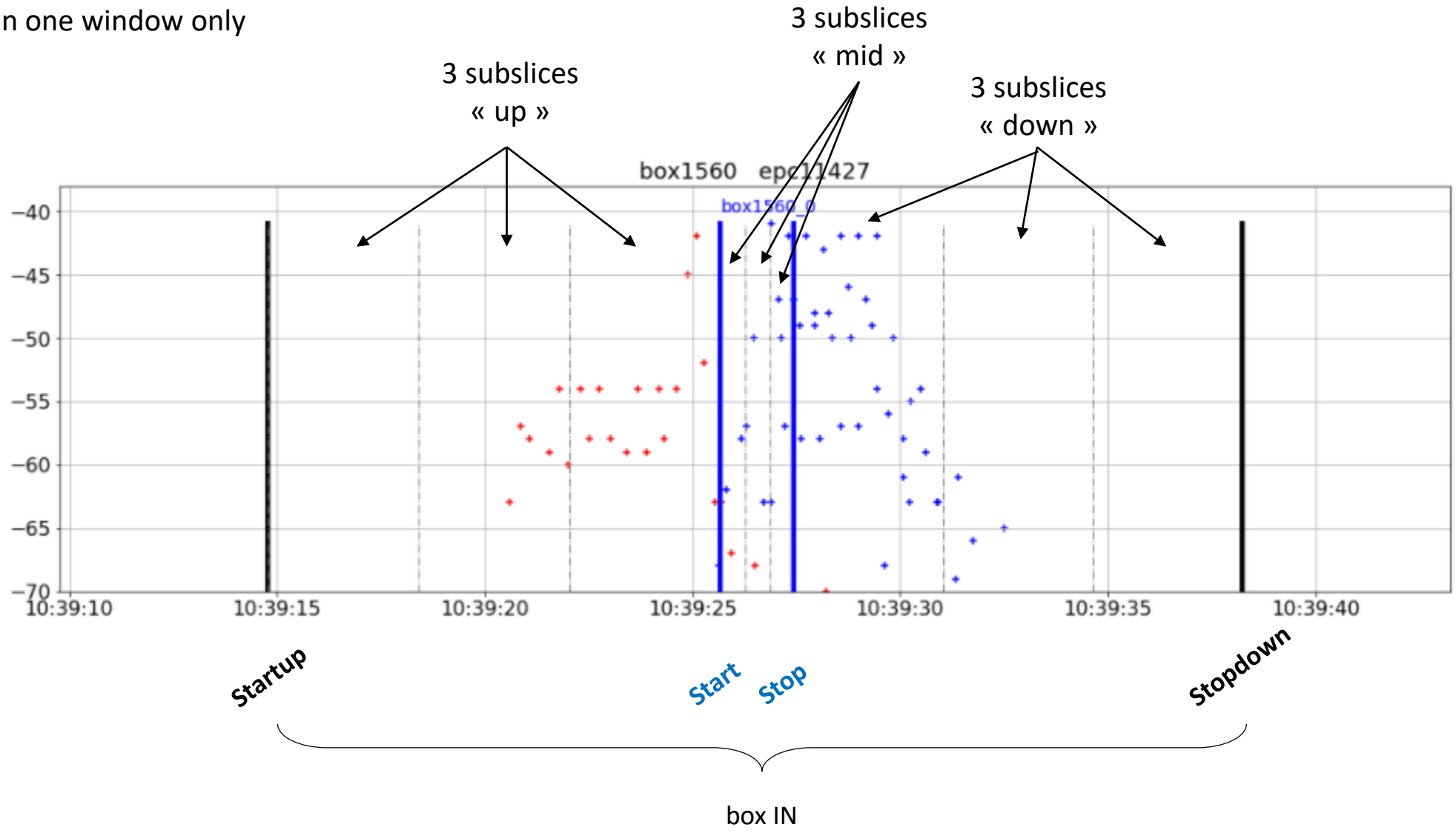
```
(1886241, 996603, 13010, 12979, 0.9976172175249808)
```

Epc visualization

Many reads:

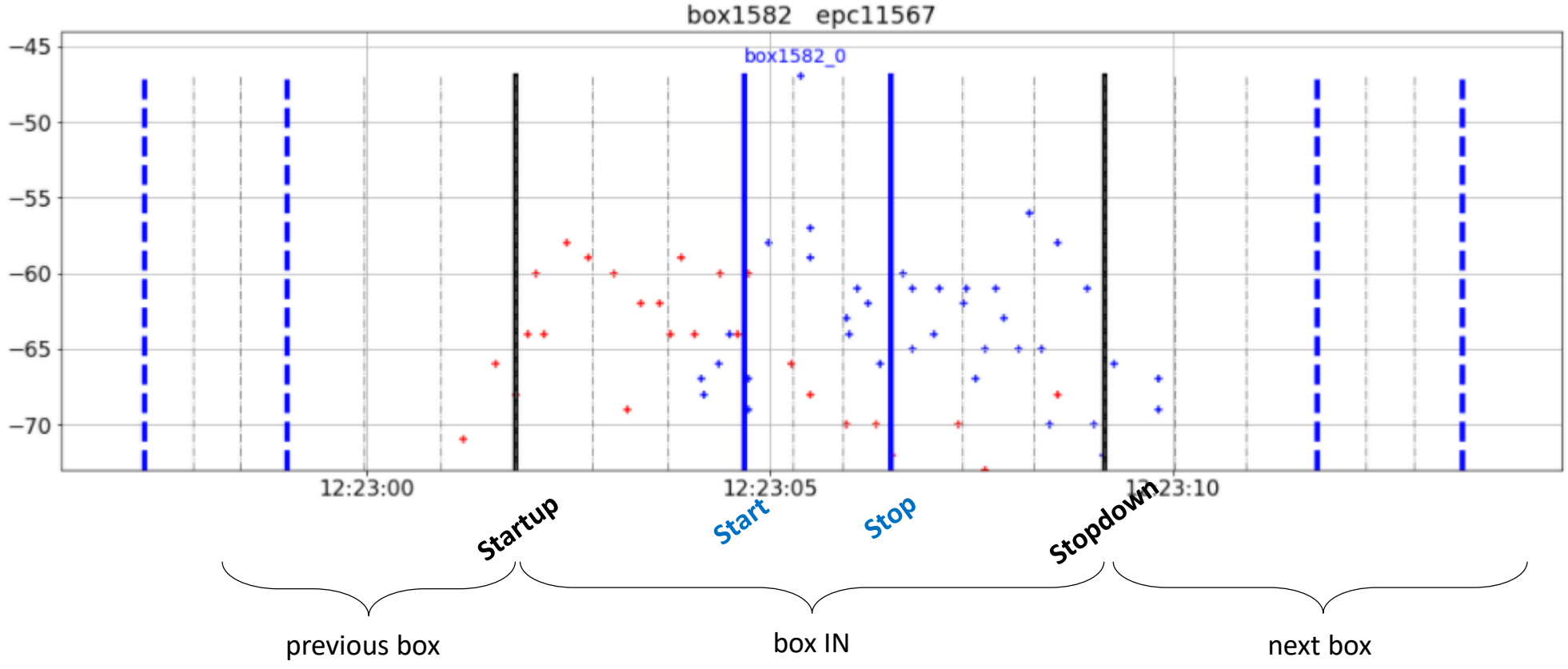
1. Blue: IN antennas
2. Red: OUT antennas

Epc detected in one window only



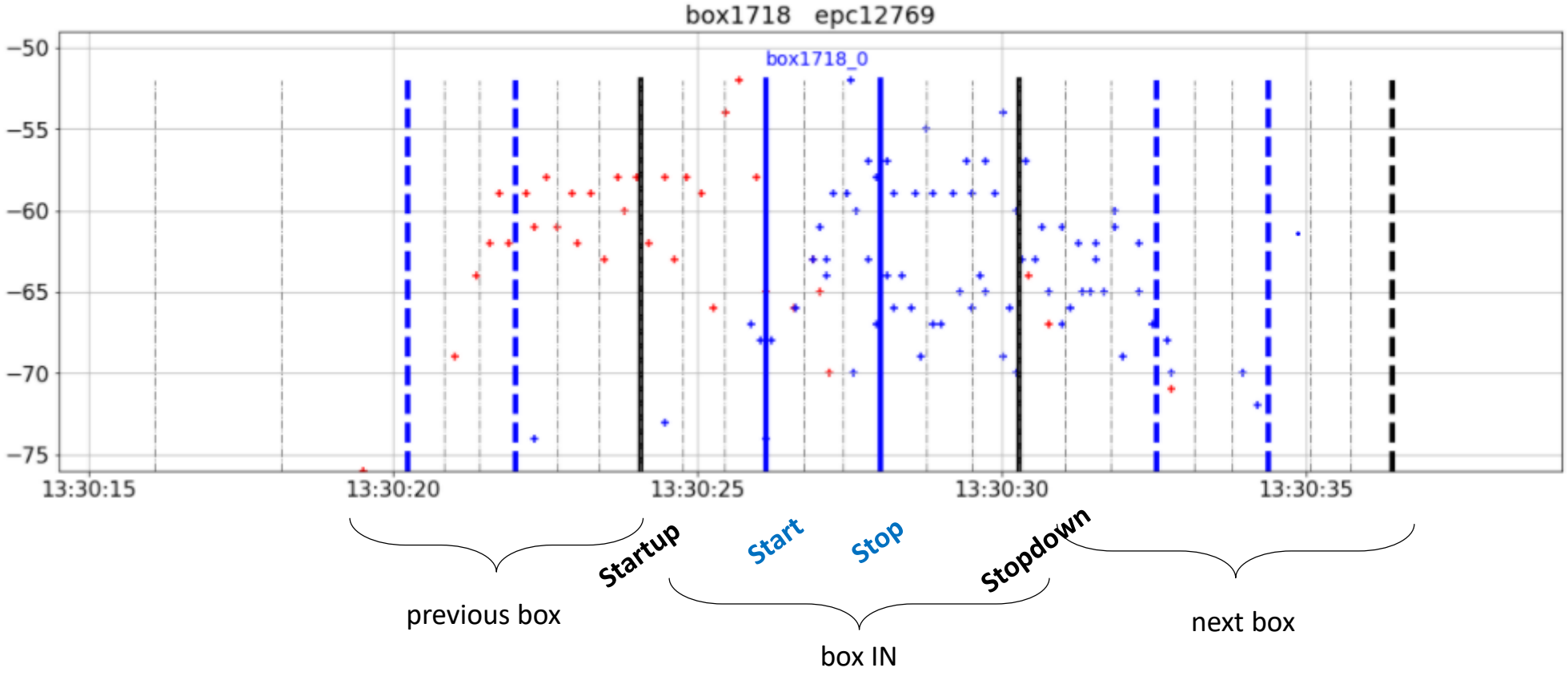
Epc visualization

Epc detected in 3 different windows: box IN, previous box , next box
Very few RFID reads in adjacent windows



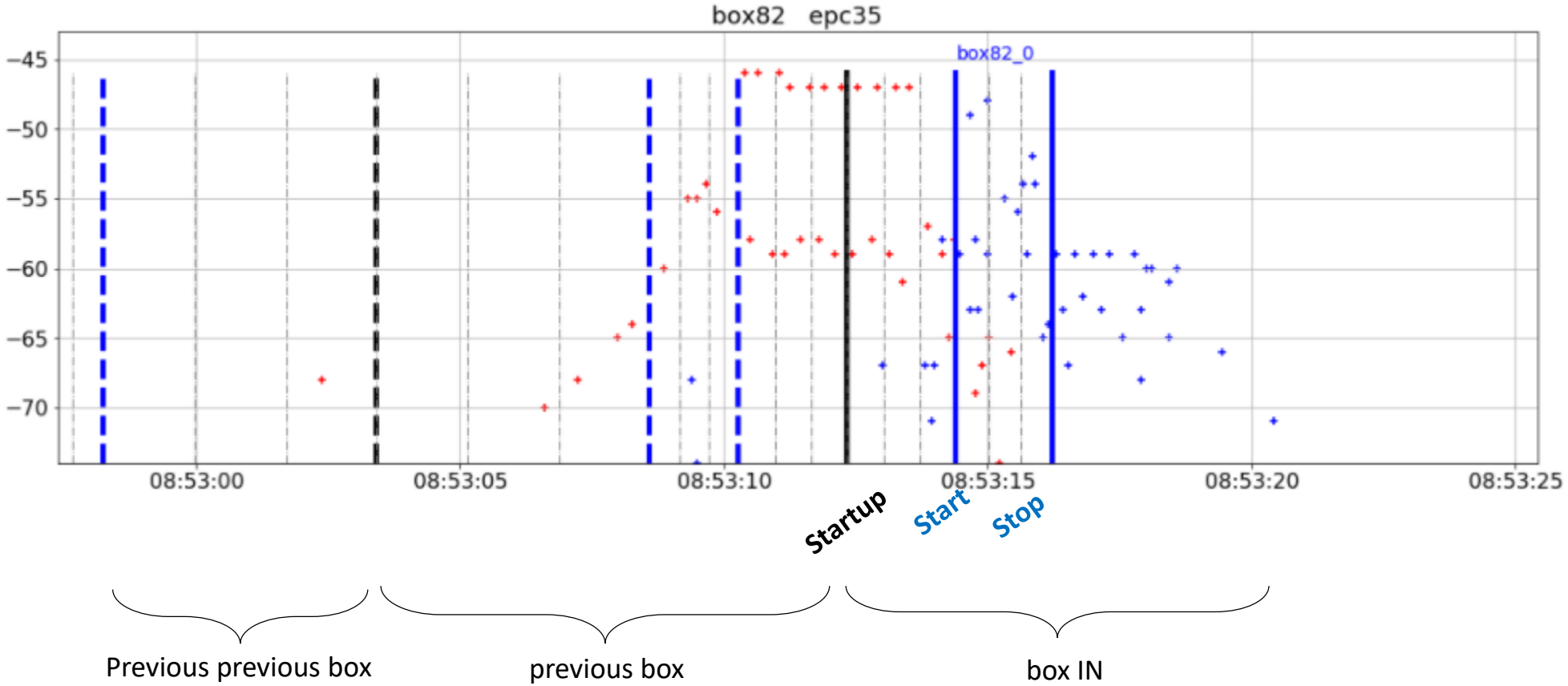
Epc visualization

Epc detected in 3 different windows: box IN, previous box , next box
Lots of RFID reads in adjacent windows



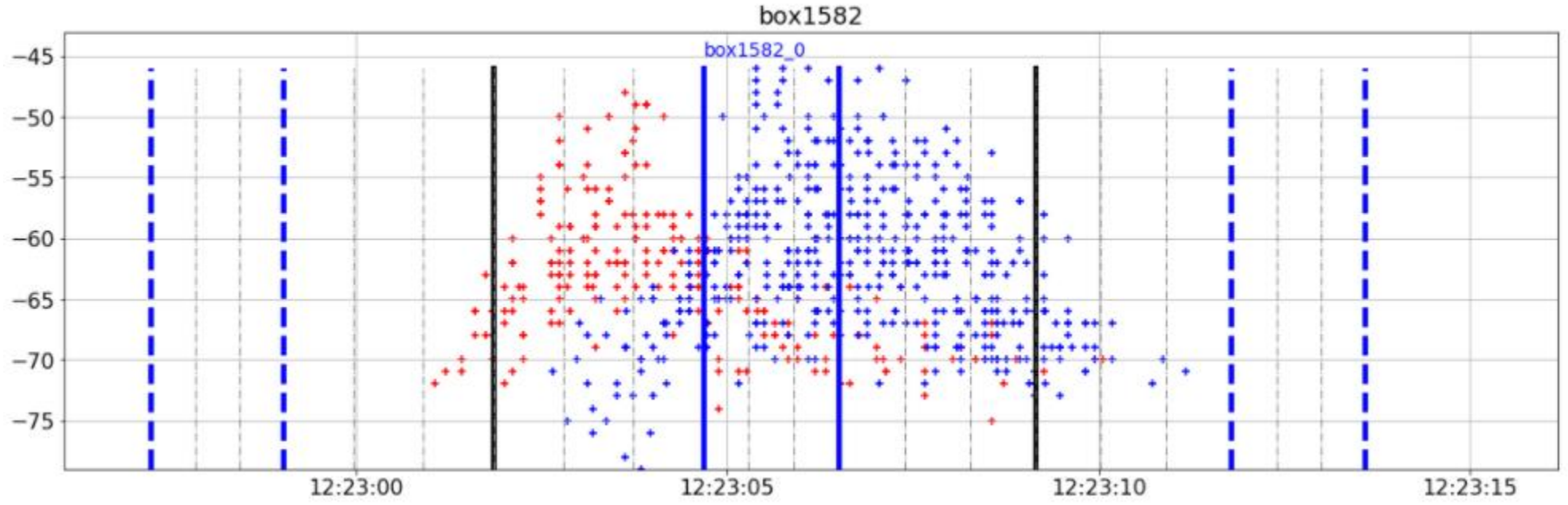
Epc visualization

Epc detected in 3 different windows: box IN, previous box , previous previous box



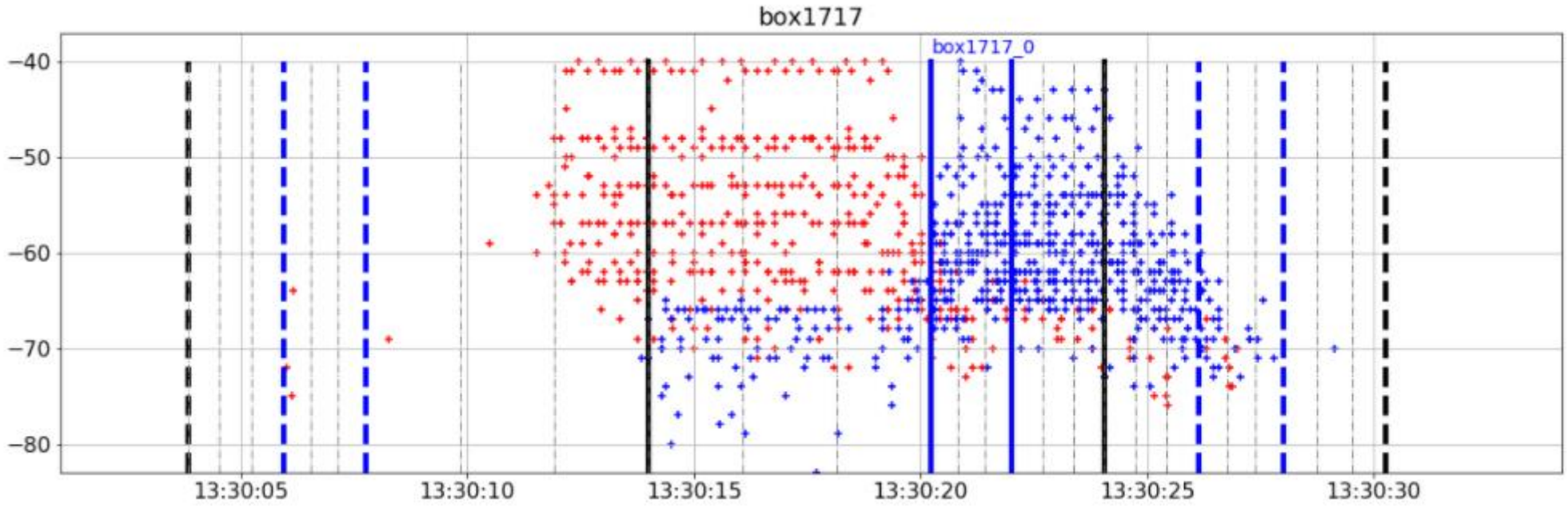
Box visualization

All epcs from the same box
Similar patterns



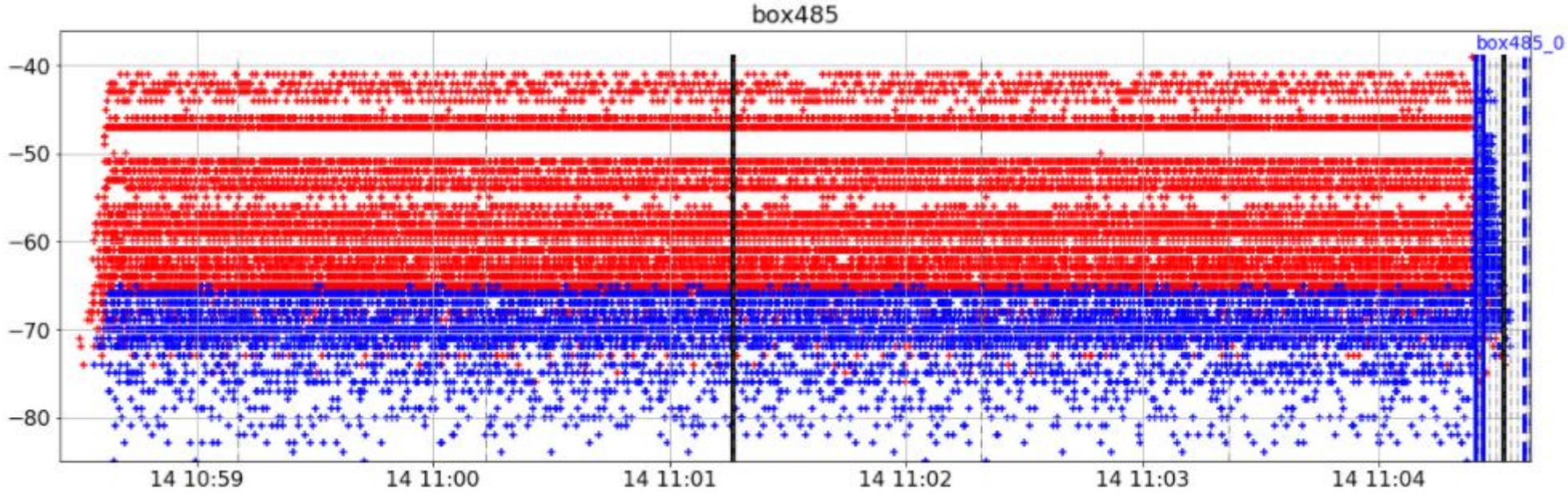
Box visualization

All epcs from the same box
Similar patterns



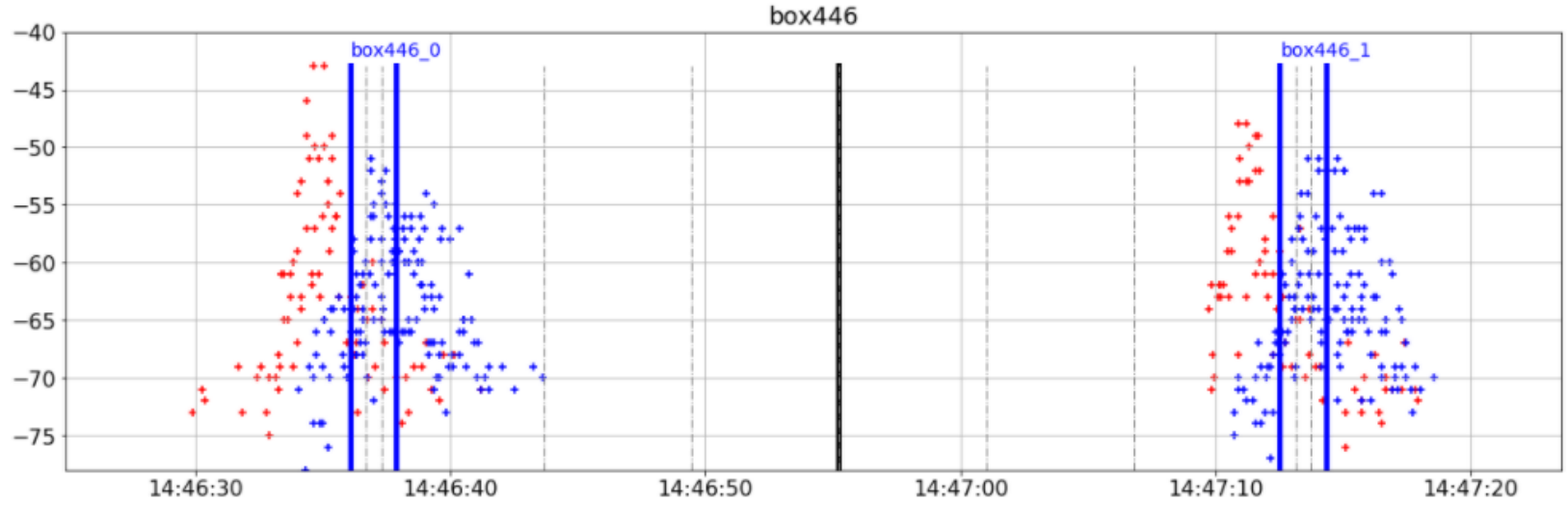
Box visualization

This box stayed static 5min upstream

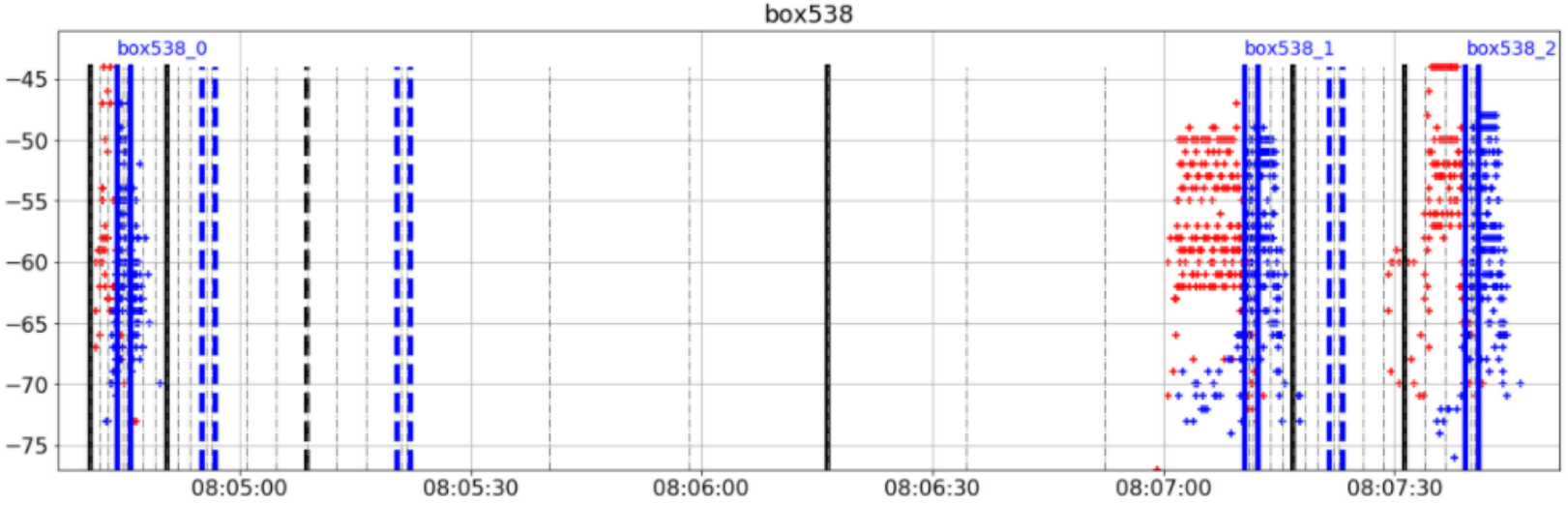


Box visualization: reruns

2 consecutive runs



3 runs, not consecutive



Anomaly

Bar code NOK (no window)
box introduced on the wrong side

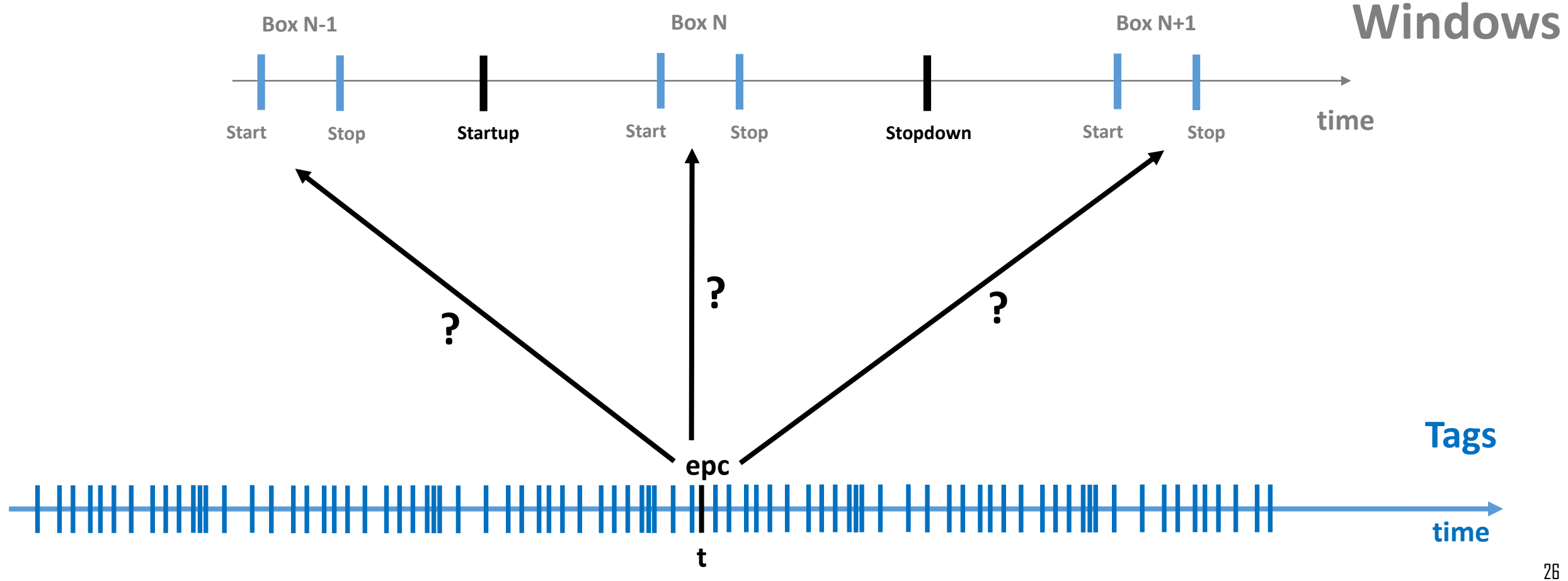
Bar code OK
box introduced on the right side

Problem to solve: localize tags in the right box

associate each tag to the right box with decision after each window (restricted and full):

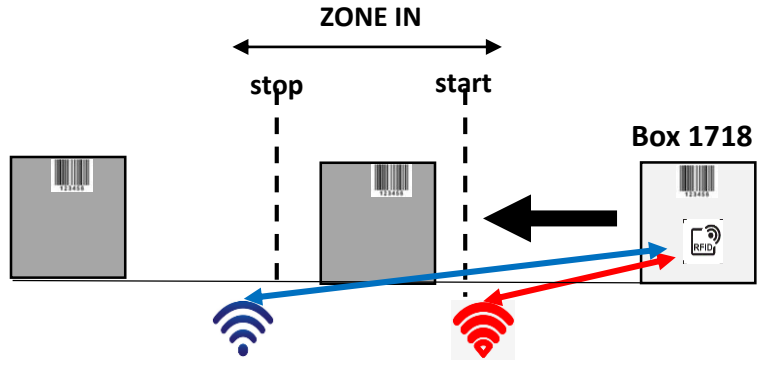
➔ Analytical

➔ Machine Learning

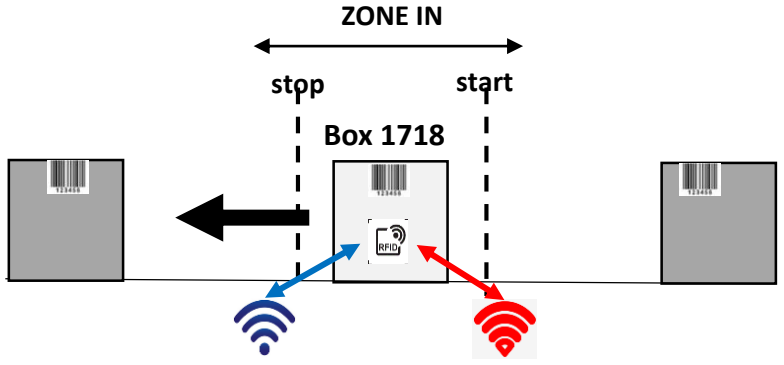


Analytical principle: snapshots

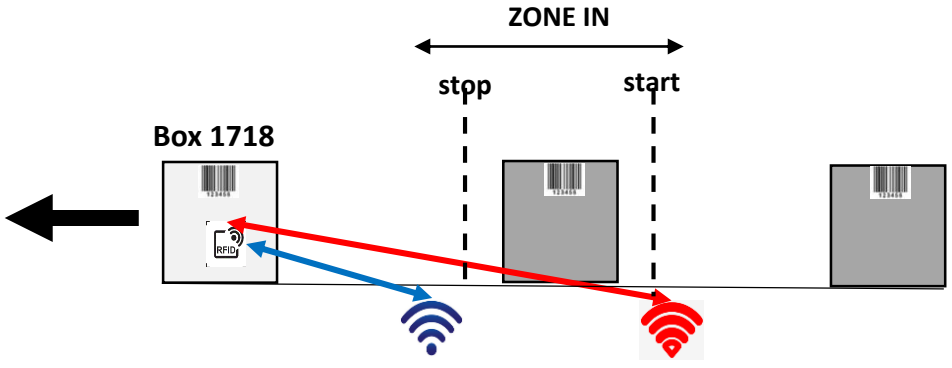
Snapshot: box1718 upstream zone IN
➔ range OUT < range IN
➔ RSSI_OUT > RSSI_IN



Snapshot: box1718 inside zone IN
➔ range OUT = range IN
➔ RSSI_OUT = RSSI_IN



Snapshot: box1718 downstream zone IN
➔ range OUT > range IN
➔ RSSI_OUT < RSSI_IN



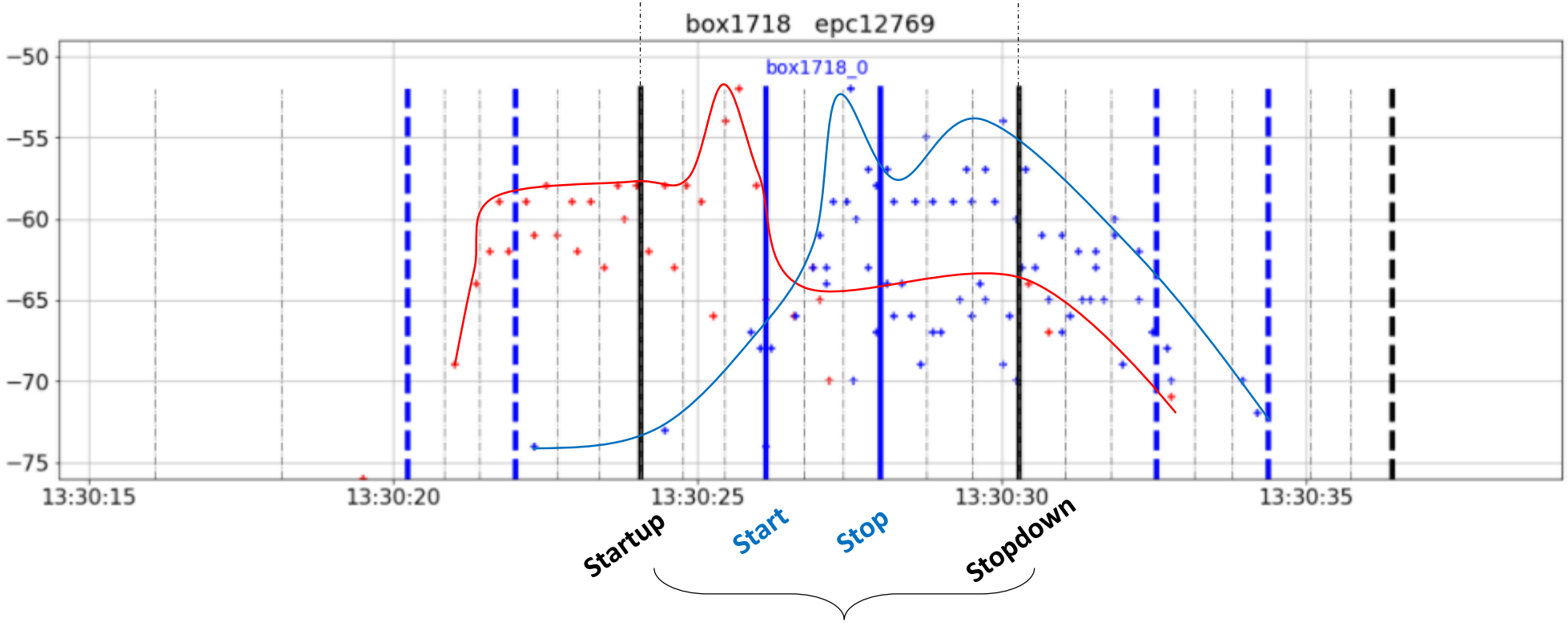
Analytical principle: time based

Epc12769 detected in 3 windows

1st window, OUT>IN
➔ **spatially upstream**

2snd window, OUT=IN
➔ **spatially inside**

3rd window, IN>OUT
➔ **spatially downstream**



Box1718 in zone IN

Analytical principle: rule

Tag localized in window where IN = OUT (IN crossing OUT)

Important note: the analytical rule is based on the following hypothesis:

a. RF free space propagation
b. Ominidirectional radiation pattern for tags
c. these 2 hypothesis are physically incorrect because of multipath and because tag orientation is unknown. It can not give 100% accuracy. However, formal RF predictions based on finite element method (HFSS, CST) are 100% accurate but require long computing time, incompatible with real time system like a tunnel

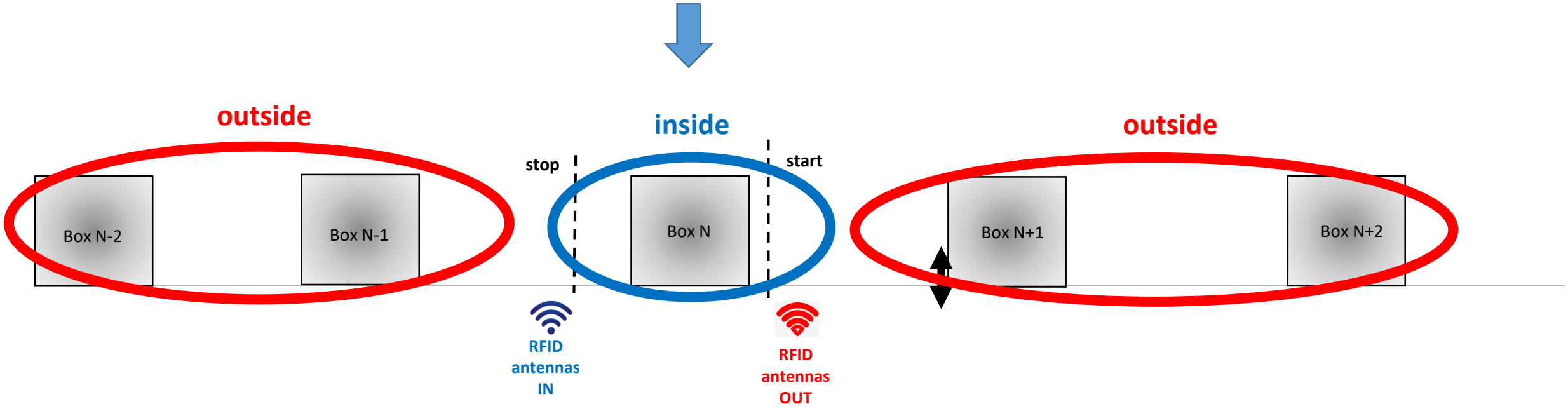
Analytical: script

```python
def analytical(tags, subslices):
    ana = tags.groupby(['Epc', 'window_run_id', 'subslice_id', 'loc'])['Rssi'].max()\
            .unstack('loc', fill_value=-110).reset_index(drop=False)
#
    order=pd.DataFrame(subslices['subslice_id'].unique(), columns=['subslice_id'])
    order['order']=order.index
#
    ana=pd.merge(ana, order, on='subslice_id', how='left')
    ana = ana [['Epc', 'window_run_id', 'subslice_id', 'in', 'out', 'order']]
# last subslice_id with out>in
    ana_out =ana [ ana['out']>ana['in'] ] \
        .sort_values(['Epc', 'window_run_id', 'order'], ascending=False) \
        .drop_duplicates(['Epc', 'window_run_id'])
# first subslice_id with in/out
    ana_in =ana [ ana['in']>ana['out'] ] \
        .sort_values(['Epc', 'window_run_id', 'order'], ascending=True) \
        .drop_duplicates(['Epc', 'window_run_id'])
#
    ana = pd.merge(ana_in, ana_out, on=['Epc', 'window_run_id'], suffixes=['_IN', '_OUT'], how='inner')\
            .sort_values(['Epc', 'window_run_id'])
    ana = pd.merge(ana, reflist, on='Epc', how='left')
#
    ana['pred_ana_bool']= ana['window_run_id'].apply(lambda x:x.split('_')[0]) == ana['box']
#
    return ana
```

```
ana_obs.head()
```

	Epc	window_run_id	subslice_id_IN	in_IN	out_IN	order_IN	subslice_id_OUT	in_OUT	out_OUT	order_OUT	box	Q	runs_box	pred_ana_bool
0	epc0	box238_0	mid_0	-61.0	-69.0	3	up_2	-72.0	-58.0	2	box238	2	1	True
1	epc1	box238_0	mid_0	-61.0	-69.0	3	up_2	-68.0	-54.0	2	box238	2	1	True
2	epc10	box74_0	mid_0	-60.0	-69.0	3	up_1	-110.0	-68.0	1	box74	7	1	True
3	epc100	box55_0	mid_0	-55.0	-58.0	3	up_2	-62.0	-52.0	2	box55	12	1	True
4	epc1000	box179_0	mid_0	-49.0	-60.0	3	up_2	-59.0	-47.0	2	box179	12	1	True

Machine Learning: principle



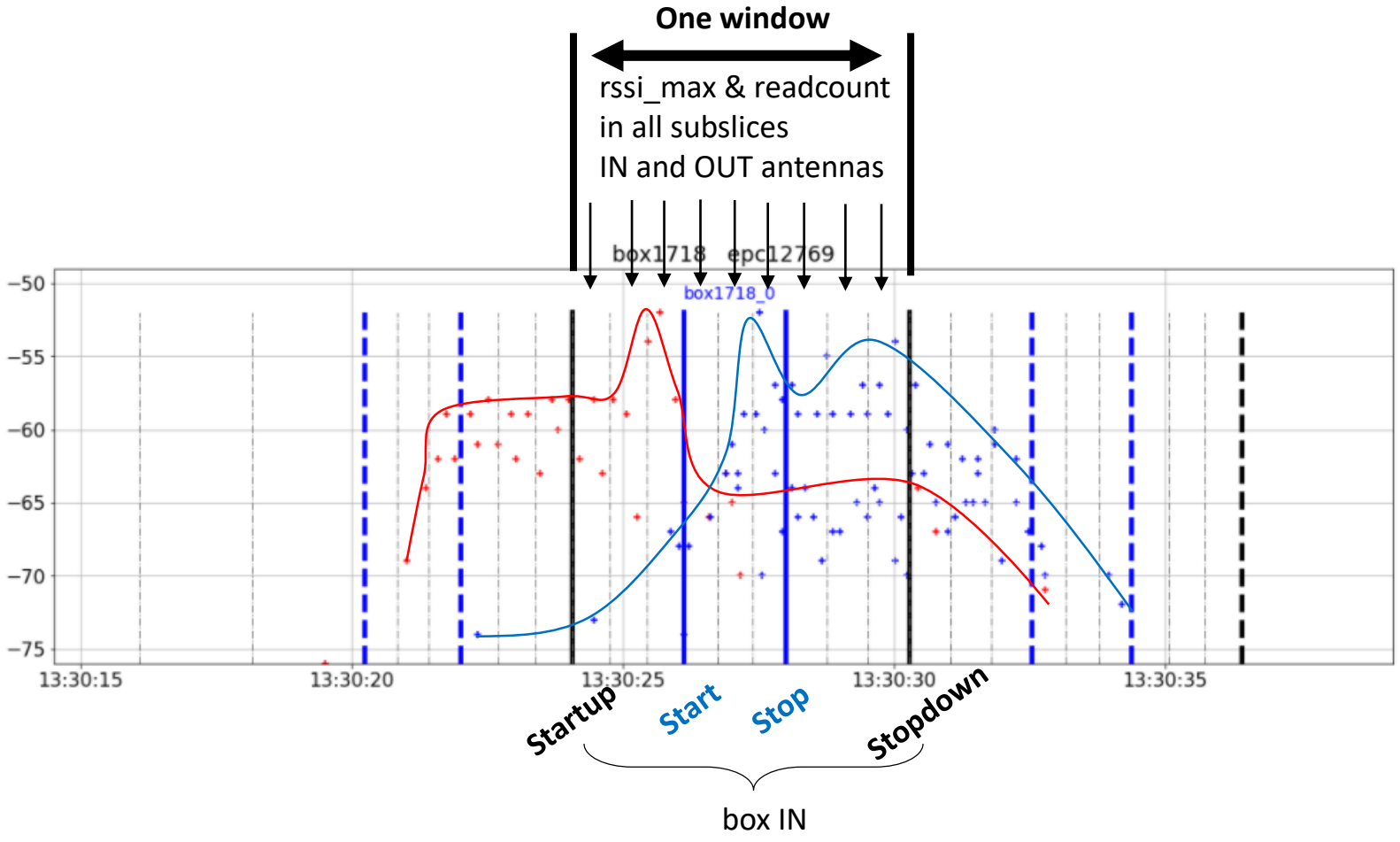
From observation of a single window, can we predict if a tag is:

1. inside box N
2. outside box N

Machine Learning: sample and features

1. **Sample**: one epc in one window
2. **Features**: rssi_max and readcount in all subslices, with IN and OUT antennas

➔ **classify** whether this epc is « inside » or « outside »



Machine Learning: sample and features

```
#
# ds:
# sample: one tag in one window
#
def dataset(tags, windows, rssi_quantile):
    ds_rssi = tags.groupby(['Epc', 'window_run_id', 'subslice_id', 'loc'])['Rssi'].quantile(rssi_quantile)\
            .unstack(['subslice_id', 'loc'], fill_value=-110)
    ds_rssi.columns = [x[0]+'_'+x[1] for x in ds_rssi.columns]
    ds_rssi = ds_rssi.reset_index(drop=False)
#
    ds_rc = tags.groupby(['Epc', 'window_run_id', 'subslice_id', 'loc']).size()\
            .unstack(['subslice_id', 'loc'], fill_value=0)
    ds_rc.columns = [x[0]+'_'+x[1] for x in ds_rc.columns]
    ds_rc = ds_rc.reset_index(drop=False)
#
    ds = pd.merge(ds_rssi, ds_rc, on=['Epc', 'window_run_id'], suffixes=['_rssi', '_rc'])
# window_width
    ds = pd.merge(ds, windows[['window_run_id', 'window_width']], on='window_run_id', how='left')
# Epcs_window
    Q_Epcs_window = tags.groupby(['window_run_id'])['Epc'].nunique().rename('Epcs_window').reset_index(drop=False)
    ds = pd.merge(ds, Q_Epcs_window, on='window_run_id', how='left')
# reads_window
    Q_reads_window = tags.groupby(['window_run_id']).size().rename('reads_window').reset_index(drop=False)
    ds = pd.merge(ds, Q_reads_window, on='window_run_id', how='left')
    return ds
```

Machine Learning: sample and features

```
#
# retries in ds, objective: identify error Epcs
#
# classifier parameters extracted from gs
clf = RandomForestClassifier(\
                n_estimators=gs.best_params_['n_estimators'],\
                max_depth=gs.best_params_['max_depth'],\
               )
# pred_ml stores prediction results: [Epc, Window_run_id, actual, pred_ml]
pred_ml=pd.DataFrame()
retries=50
for retry in range(retries):
#
    Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size=0.8, stratify=y)
#
    scaler = MinMaxScaler()
    scaler.fit(Xtrain)
    Xtrain_std = scaler.transform(Xtrain)
    Xtest_std = scaler.transform(Xtest)
#
    clf.fit(Xtrain_std, ytrain)
#
    ypred = clf.predict(Xtest_std)
#
    print (retry, (ytest==ypred).mean())
    ypred_series = pd.Series(ypred, index=ytest.index, name='pred_ml')
    temp = ds.loc[ytest.index, ['Epc', 'window_run_id', 'actual']]
    temp = temp.join(ypred_series)
    temp.loc[:, 'retry'] = retry
    pred_ml = pred_ml.append(temp)
#
pred_ml.loc[:, 'pred_ml_bool'] = (pred_ml.loc[:, 'actual']==pred_ml.loc[:, 'pred_ml'])
pred_ml = pred_ml [['Epc', 'window_run_id', 'actual', 'pred_ml', 'pred_ml_bool', 'retry']]
```

```
0 1.0
1 0.998282377189969
2 0.9993129508759876
3 0.9986259017519753
4 0.9993129508759876
5 0.9989694263139814
```

Machine Learning: sample and features

```
ds.head()
```

	Epc	window_run_id	down_0_in_rssi	down_0_out_rssi	mid_0_in_rssi	mid_0_out_rssi	mid_1_in_rssi	mid_1_out_rssi	mid_2_in_rssi	up_2_in_rssi	...
0	epc0	box238_0	-55.0	-69.0	-61.0	-69.0	-45.0	-67.0	-57.0	-72.0	...
1	epc1	box238_0	-53.0	-70.0	-61.0	-69.0	-51.0	-66.0	-55.0	-68.0	...
2	epc10	box74_0	-66.0	-110.0	-60.0	-69.0	-58.0	-66.0	-56.0	-64.0	...
3	epc100	box55_0	-57.0	-65.0	-55.0	-58.0	-55.0	-62.0	-58.0	-62.0	...
4	epc1000	box179_0	-47.0	-63.0	-49.0	-60.0	-52.0	-58.0	-42.0	-59.0	...

5 rows × 45 columns

```
ds.head()
```

rssi	mid_2_in_rssi	up_2_in_rssi	...	down_1_out_rc	down_2_out_rc	down_2_in_rc	window_width	Epcs_window	reads_window
57.0	-57.0	-72.0	...	0	0	0	45.9765	2	110
66.0	-55.0	-68.0	...	0	0	0	45.9765	2	110
66.0	-56.0	-64.0	...	0	0	0	11.5210	7	399
62.0	-58.0	-62.0	...	0	0	0	14.6555	12	683
58.0	-42.0	-59.0	...	0	0	0	90.0715	12	519

Machine Learning: features selection

```
#
# reduction of Xcols to the selected features
#
def Xcols_func(features, Xcols_all):
#
    Features_temp = Features [Features['features']==features]
#
    X=[]
    rssi = Features_temp ['rssi'].values[0]
    rc = Features_temp['rc'].values[0]
    rc_mid_only = Features_temp['rc_mid_only'].values[0]
    Epcs_window = Features_temp['Epcs_window'].values[0]
    reads_window = Features_temp['reads_window'].values[0]
    window_width = Features_temp['window_width'].values[0]
#
    X_rssi = [x for x in Xcols_all if rssi*'rssi' in x.split('_') ]
#
    X_rc = [x for x in Xcols_all if rc*'rc' in x.split('_') ]
#
    X = X_rssi + X_rc
#
    if Epcs_window:
        X.append('Epcs_window')
#
    if reads_window:
        X.append('reads_window')
#
    if window_width:
        X.append('window_width')
#
    return X
```

```
Features=pd.DataFrame(\
        [\
         ['all', True, True, False, True, True, True ],\
         ['rssi & rc only', True, True, False, False, False, False ],\
         ['rssi & rc_mid', True, True, True, False, False, False ],\
         ['rssi only', True, False, True, False, False, False ],\
         ['rc only', False, True, False, False, False, False ],\
        ], columns=['features', 'rssi', 'rc', 'rc_mid_only', 'Epcs_window', 'reads_window','window_width'])
Features
```

	features	rssi	rc	rc_mid_only	Epcs_window	reads_window	window_width
0	all	True	True	False	True	True	True
1	rssi & rc only	True	True	False	False	False	False
2	rssi & rc_mid	True	True	True	False	False	False
3	rssi only	True	False	True	False	False	False
4	rc only	False	True	False	False	False	False

Machine Learning: classes

2 actual classes: IN or OUT

```
ds[['Epc', 'window_run_id', 'box', 'actual']] [50:60]
```

	Epc	window_run_id	box	actual
50	epc10041	box1497_0	box1497	in
51	epc10042	box1497_0	box1497	in
52	epc10043	box1497_0	box1497	in
53	epc10044	box1430_0	box1430	in
54	epc10045	box1429_0	box1430	out
55	epc10045	box1430_0	box1430	in
56	epc10046	box1429_0	box1430	out
57	epc10046	box1430_0	box1430	in
58	epc10047	box1429_0	box1430	out
59	epc10047	box1430_0	box1430	in

```
ds.groupby('actual')['Epc'].nunique()
```

```
actual
in     12979
out      803
```

Results

Champion parameters

See parametric analysis in annex

Parameters	Champion value	Perf impact
observation	1	from 0 to 1: 3x less errors (0.1%) from 1 to « full »: 2x less errors (0.04%) **1sec after stop means that rejection point must be moved downstream by 30cm**
Steps	4	Low
features	all	Key: rssi: key Low: Rc, Epcs in window, reads in window
Rssi_quantile	1	No influence between 1, 0.9, 0.5
Classifier	RandomForest	Significant
Classifiers_parameters	Max_depth=10, n_estimators=100	

Machine Learning: GridSearch best score

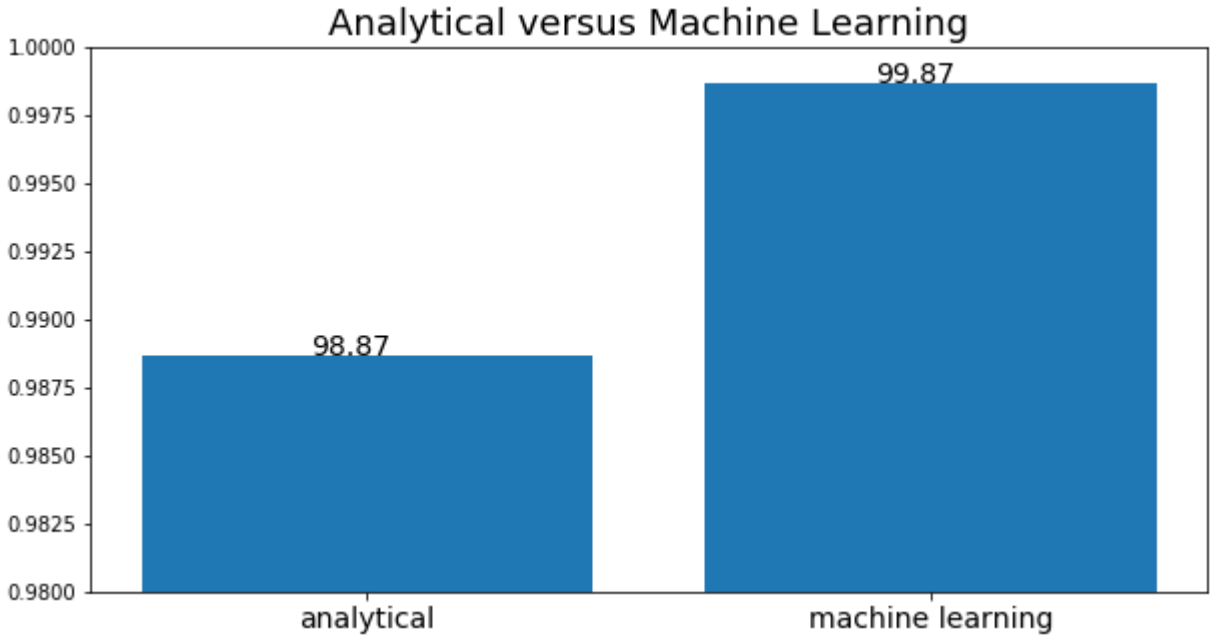
CV=50
Best_score = 99.87%

```
gs.best_score_
```

```
0.9986800716221448
```

Analytical versus Machine Learning

ML outperforms Analytical: 10x times less errors



Machine Learning: retries on ds

50 retries in ds with test_size=20%
➔ Each Epc is tested 10 times in average (min=1, max=87)

```
pred_ml.groupby (['Epc']).size().describe()
```

```
count    13004.000000
mean        11.069671
std          5.457046
min          1.000000
25%          8.000000
50%         10.000000
75%         12.000000
max         87.000000
```

Retries ➔ 99.91%
slight difference with GridSearch (99.87%)

```
pred_ml['pred_ml_bool'].mean()
```

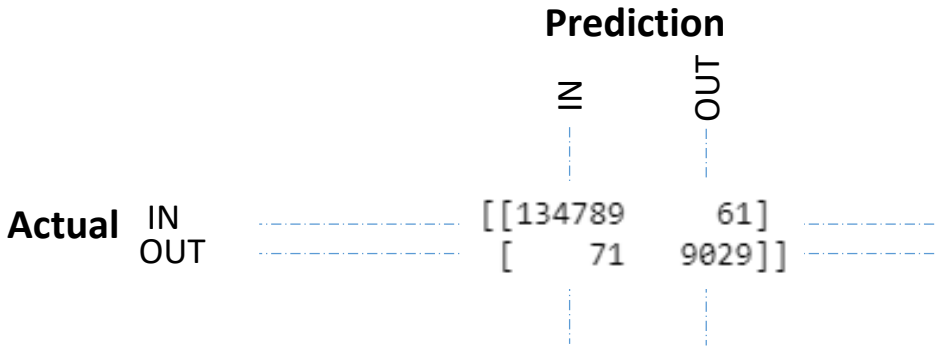
```
0.9990830149357416
```

Machine Learning: confusion_matrix, classification_report

Confusion matrix

71 FP ➔ overpacking

61 FN ➔ underpacking



Classification report: 99.91% accuracy

```
print(classification_report(pred_ml['actual'], pred_ml['pred_ml'], digits=4))
```

	precision	recall	f1-score	support
in	0.9995	0.9995	0.9995	134850
out	0.9933	0.9922	0.9927	9100
accuracy			0.9991	143950
macro avg	0.9964	0.9959	0.9961	143950
weighted avg	0.9991	0.9991	0.9991	143950

Machine Learning: error Epcs

71+61=132 errors … but only 23 error Epcs

```
: tags_obs['Epc'].nunique(), pred_ml ['Epc'].nunique()
(13004, 13004)

: ds['Epc'].nunique(), pred_ml [~pred_ml['pred_ml_bool']] ['Epc'].nunique()
(13004, 23)
```

Machine Learning: 3 categories of error Epcs

24 (Epcs, window_run_id) errors:
True and False prediction counts

1. Epcs_error_high: no good prediction (True=0)
2. Epcs_error_mid: good prediction (True!=0) but False>True
3. Epcs_error_low: True>False

```
: Epc_status [Epc_status[False]!=0].sort_values (False, ascending=False).reset_index(drop=True)
```

pred_ml_bool	Epc	window_run_id	False	True
0	epc8822	box1242_0	12	0
1	epc460	box94_0	10	0
2	epc6835	box1025_0	9	1
3	epc10871	box1458_0	9	0
4	epc4054	box605_1	8	0
5	epc1296	box183_0	7	5
6	epc210	box59_0	7	2
7	epc3709	box515_0	7	2
8	epc4458	box639_0	6	0
9	epc4059	box605_3	5	1
10	epc3746	box535_0	5	0
11	epc5805	box945_0	3	4
12	epc7114	box1062_0	3	13
13	epc12284	box1635_0	2	10
14	epc12437	box1644_0	2	10
15	epc7020	box1030_0	2	9
16	epc4059	box605_2	2	8
17	epc4688	box685_2	2	4
18	epc11547	box1614_0	1	13
19	epc12165	box1629_0	1	13
20	epc497	box79_0	1	11
21	epc4057	box605_2	1	10
22	epc6143	box1039_0	1	9
23	epc12198	box1716_0	1	7

Machine Learning: Epcs_error_high

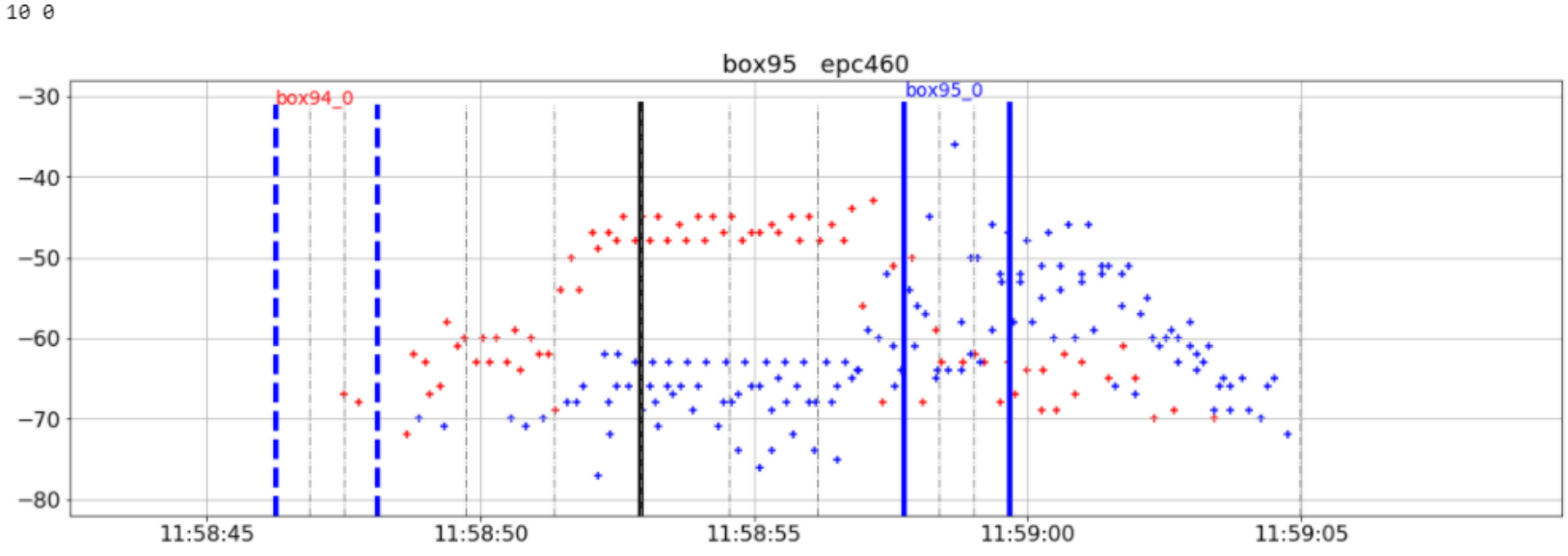
```
: Epc_error_high = Epc_status [ (Epc_status[False]!=0) & (Epc_status[True]==0) ].reset_index(drop=True)
  Epc_error_high
```

pred_ml_bool	Epc	window_run_id	False	True
0	epc8822	box1242_0	12	0
1	epc460	box94_0	10	0
2	epc10871	box1458_0	9	0
3	epc4054	box605_1	8	0
4	epc4458	box639_0	6	0
5	epc3746	box535_0	5	0

Machine Learning: Epcs_error_high

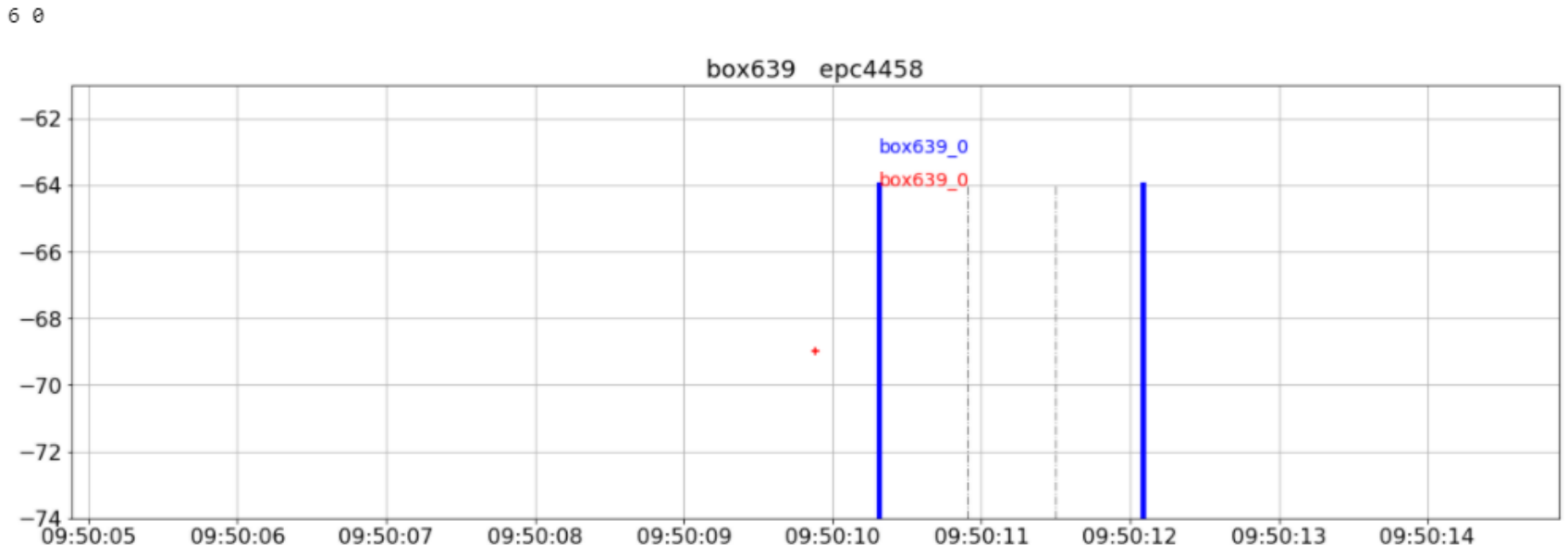
FN
12 False, 0 True

FP
10 False, 0 True

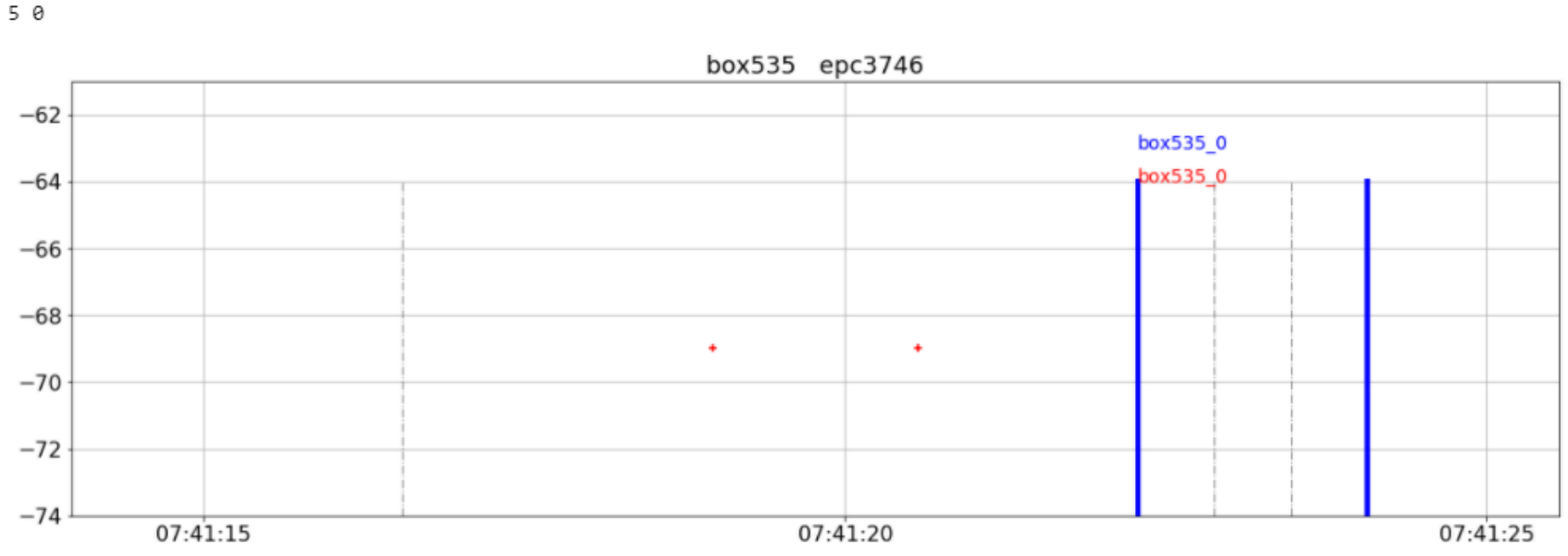


Machine Learning: Epcs_error_high

FN
6 False, 0 True

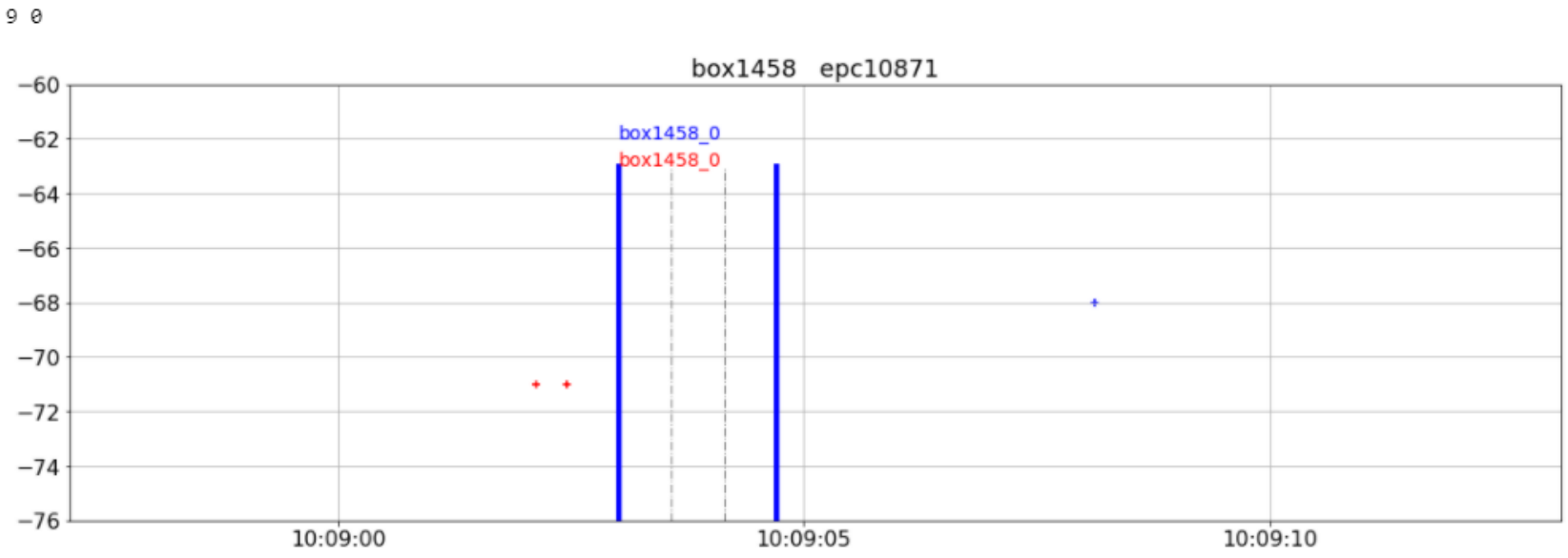


FN
5 False, 0 True



Machine Learning: Epcs_error_high

FN
9 False, 0 True



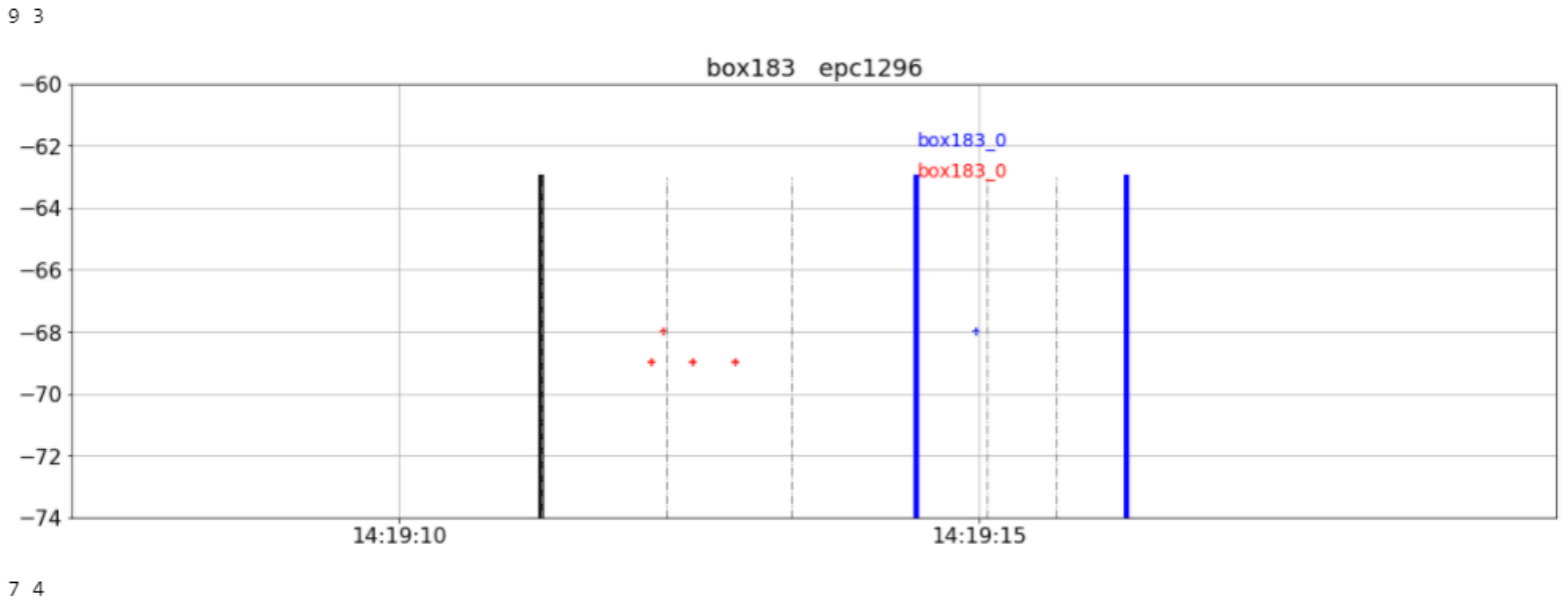
Machine Learning: Epcs_error_mid

```
Epc_error_mid = Epc_status [ (Epc_status[False]!=0) & (Epc_status[True]!=0) & (Epc_status[False]>Epc_status[True]) ]\
                    .reset_index(drop=True)
Epc_error_mid
```

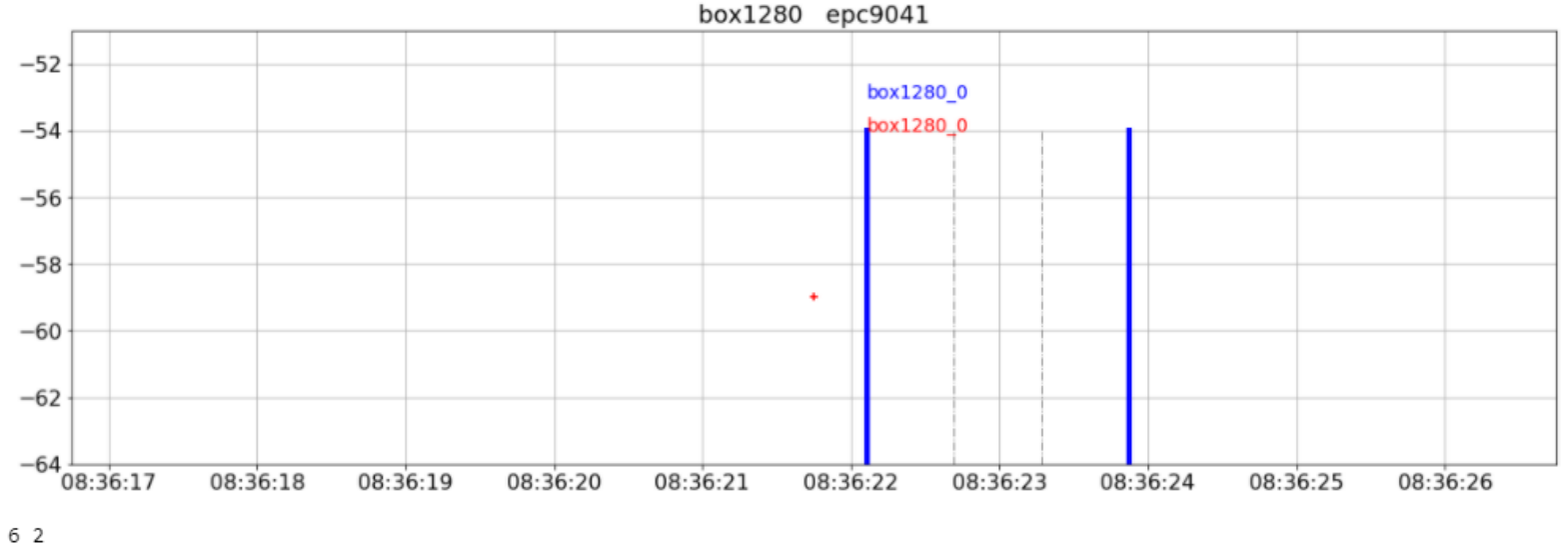
pred_ml_bool	Epc	window_run_id	False	True
0	epc1296	box183_0	9	3
1	epc9041	box1280_0	7	4
2	epc4688	box685_2	6	2

Machine Learning: Epcs_error_mid

FN
9 False, 3 True

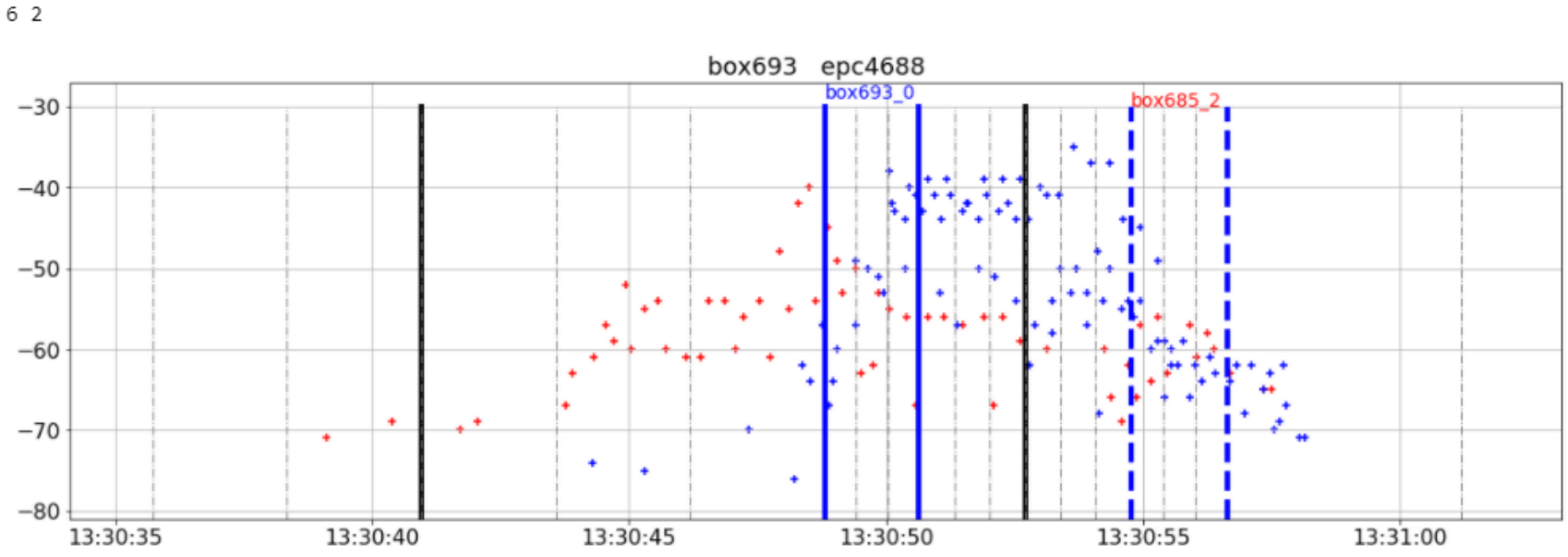


FN
7 False, 4 True



Machine Learning: Epcs_error_mid

FP
6 False, 2 True



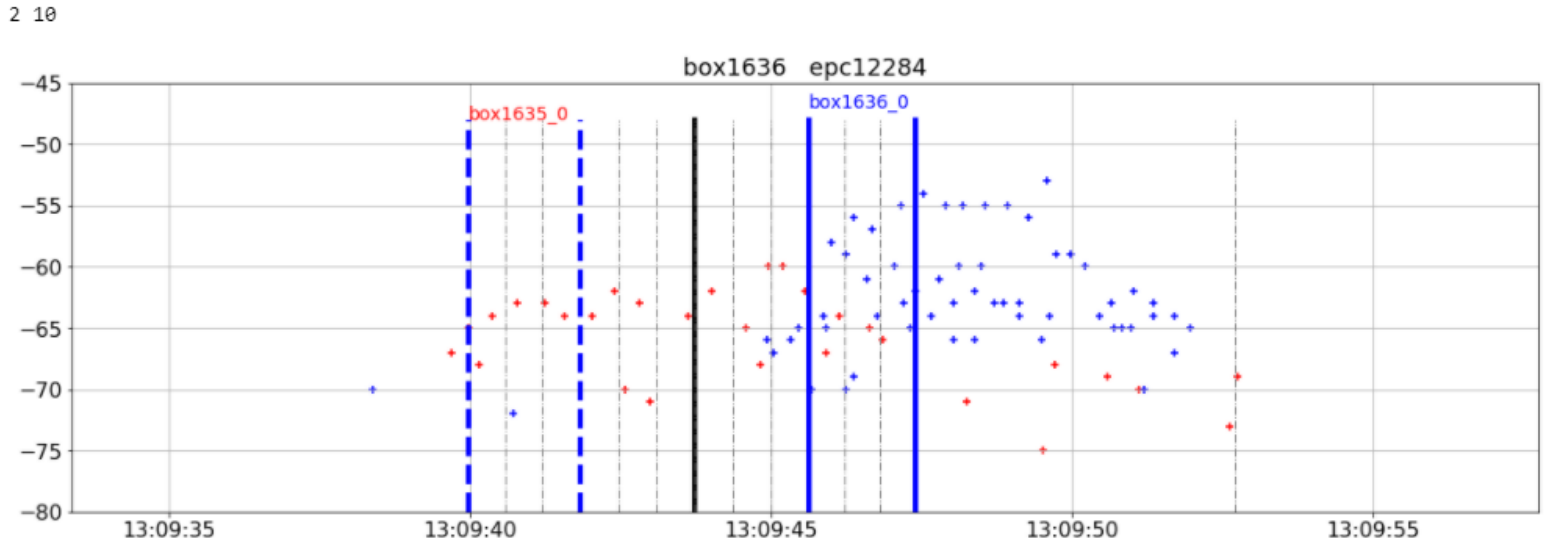
Machine Learning: Epcs_error_low

```
Epc_error_low = Epc_status [ (Epc_status[False]!=0) & (Epc_status[True]>=Epc_status[False]) ].reset_index(drop=True)
Epc_error_low
```

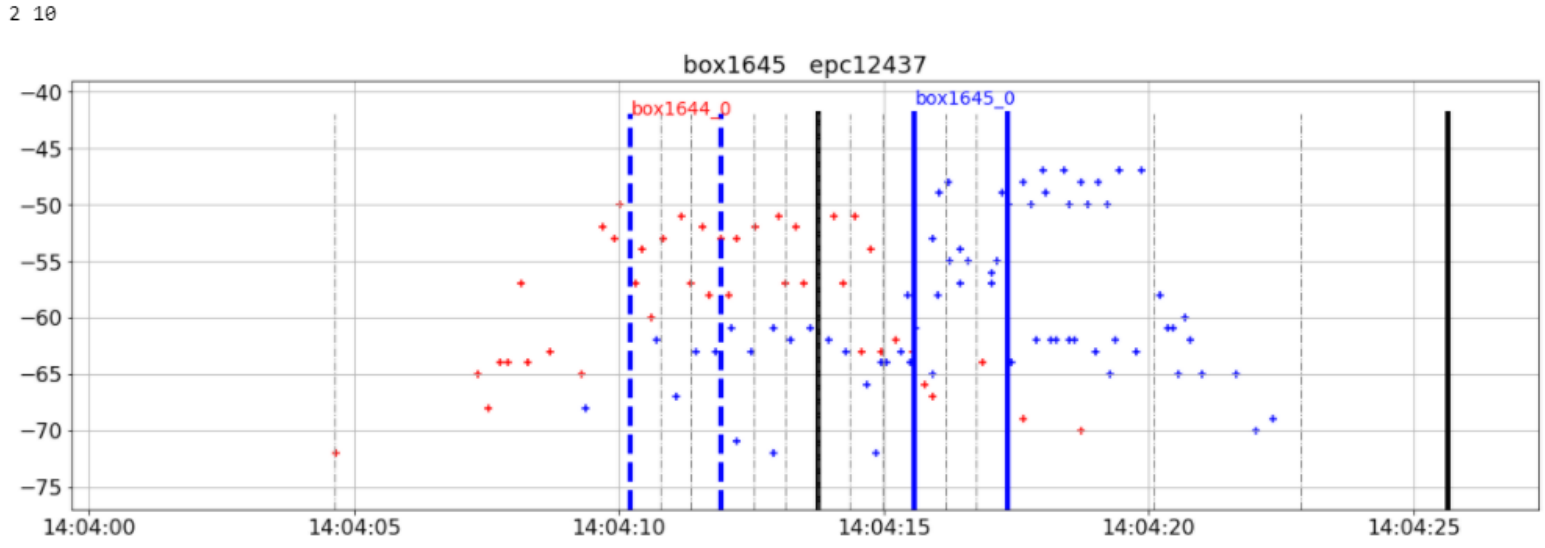
pred_ml_bool	Epc	window_run_id	False	True
0	epc3726	box515_0	5	9
1	epc4059	box605_3	4	4
2	epc6835	box1025_0	3	11
3	epc9038	box1274_0	3	3
4	epc7114	box1062_0	2	8
5	epc12284	box1635_0	1	13
6	epc6143	box1039_0	1	13
7	epc7020	box1030_0	1	10
8	epc12447	box1644_0	1	7

Machine Learning: Epcs_error_low

FP
10 True, 2 False

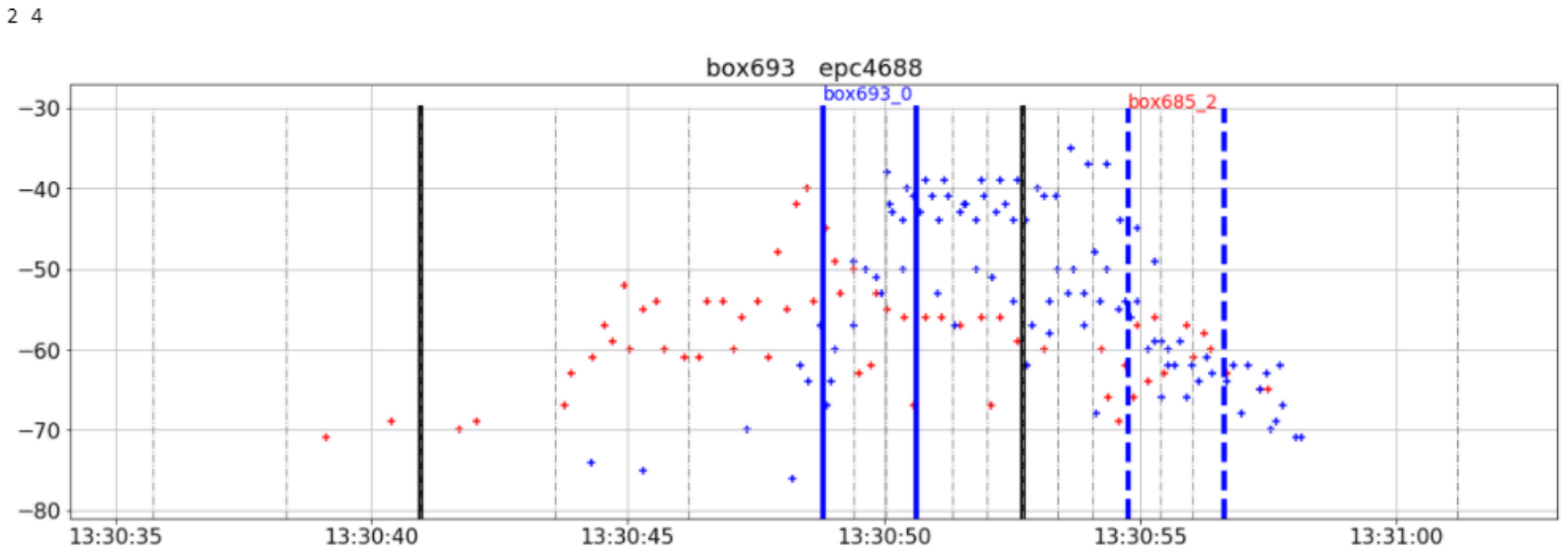


FP
10 True, 2 False

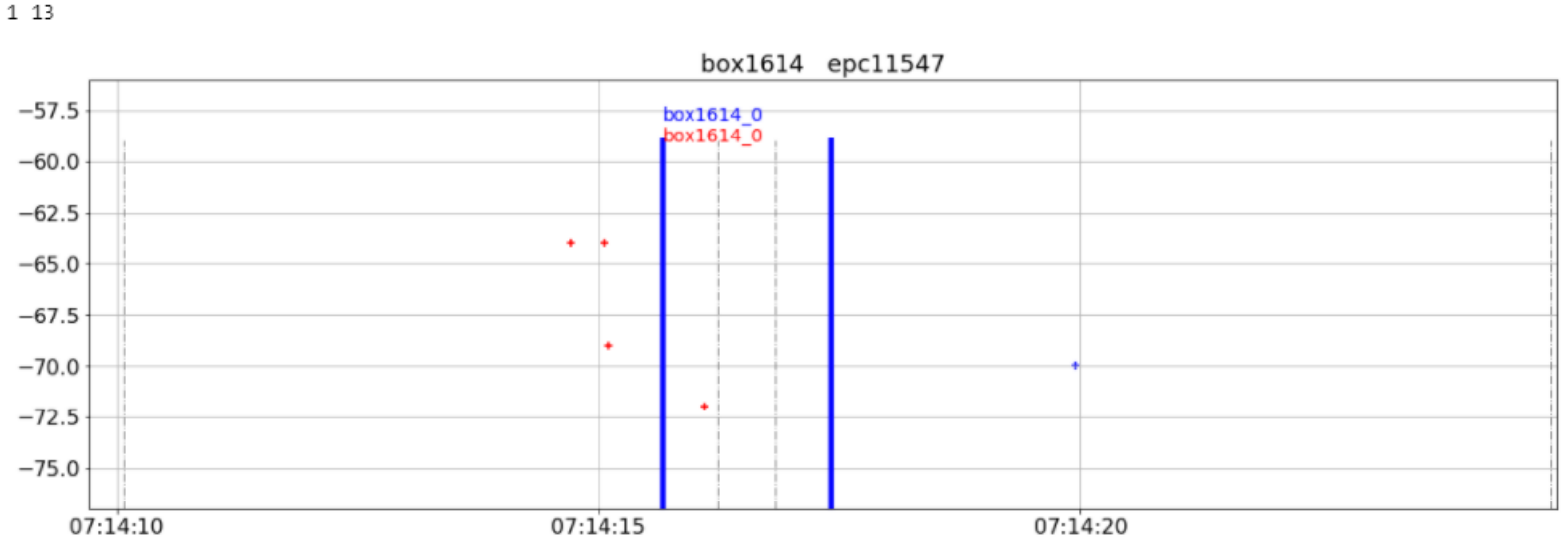


Machine Learning: Epcs_error_low

FP
4 True, 2 False

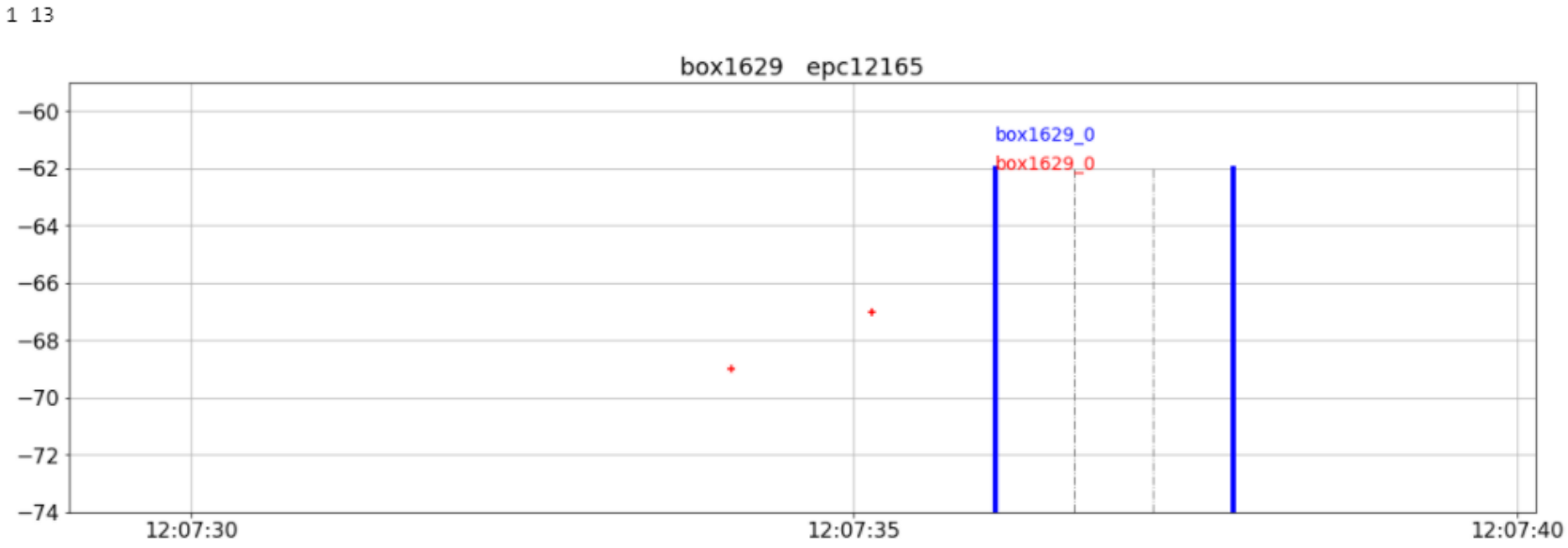


FN
13 True, 1 False

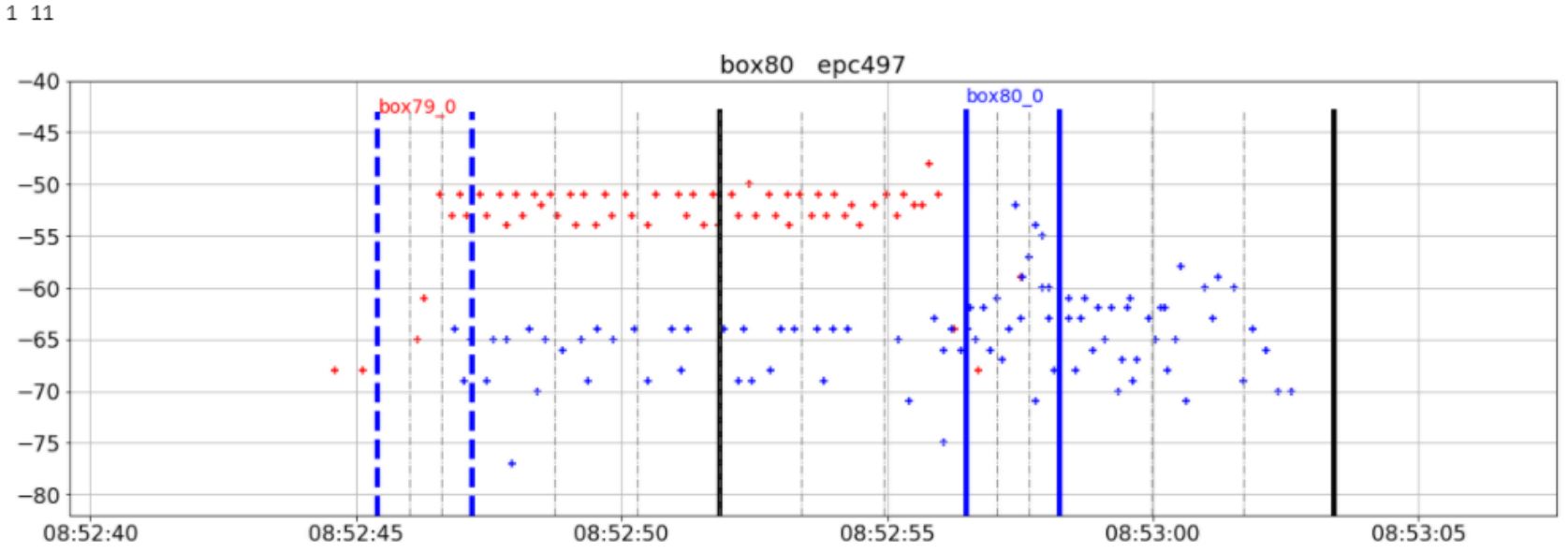


Machine Learning: Epcs_error_low

FP
13 True, 1 False

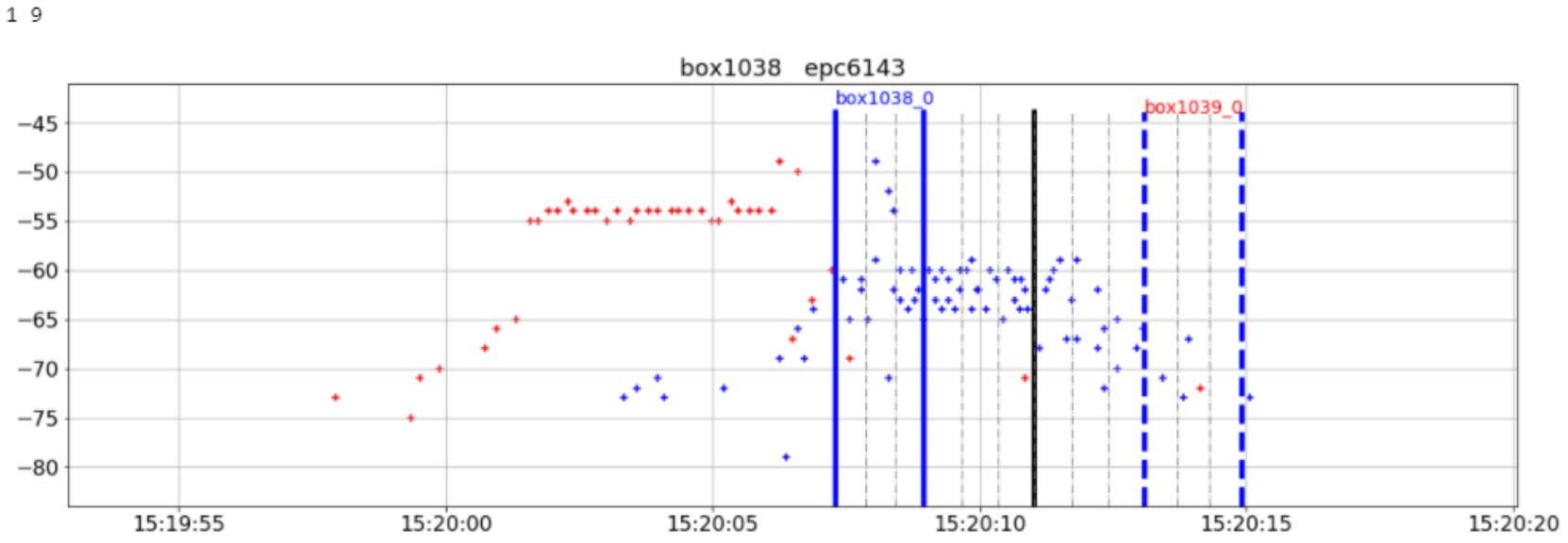


FP
11 True, 1 False

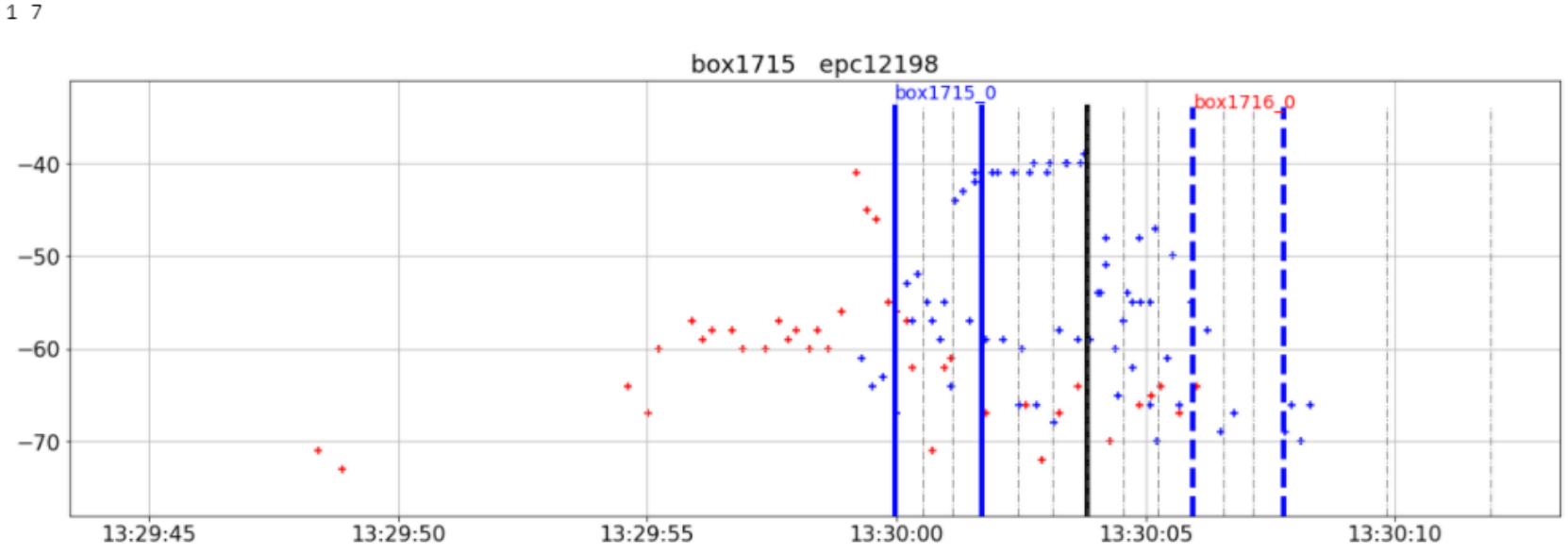


Machine Learning: Epcs_error_low

FP
9 True, 1 False

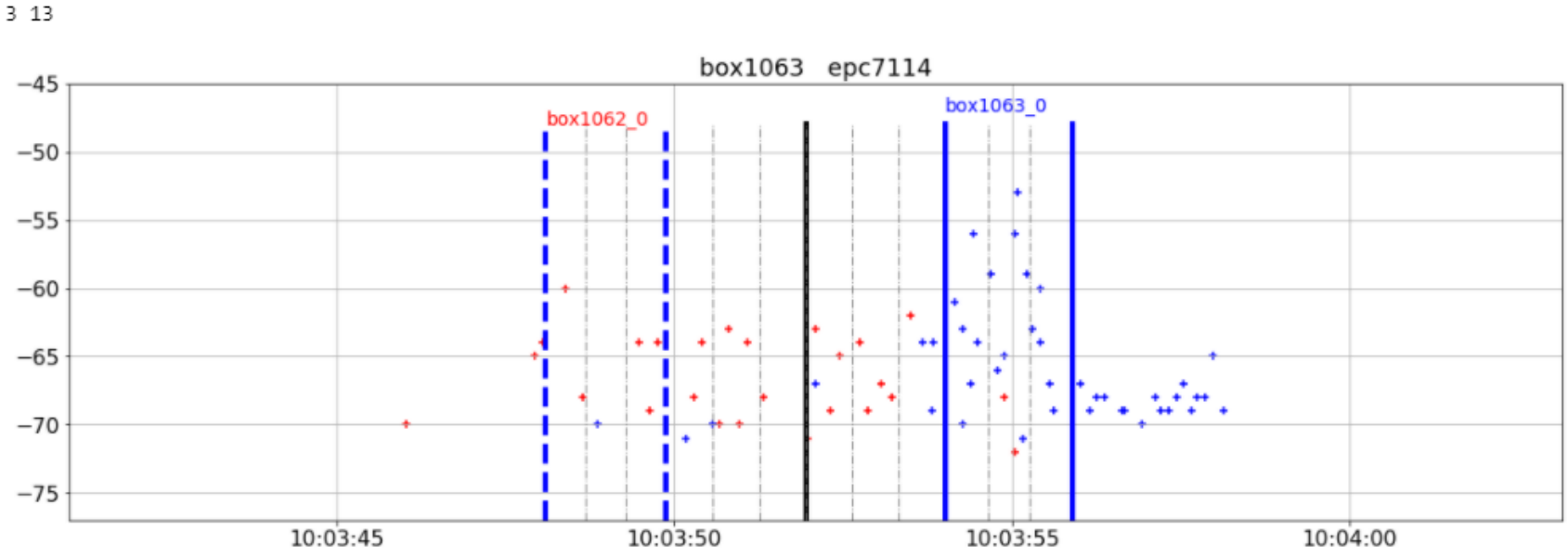


FP
7 True, 1 False

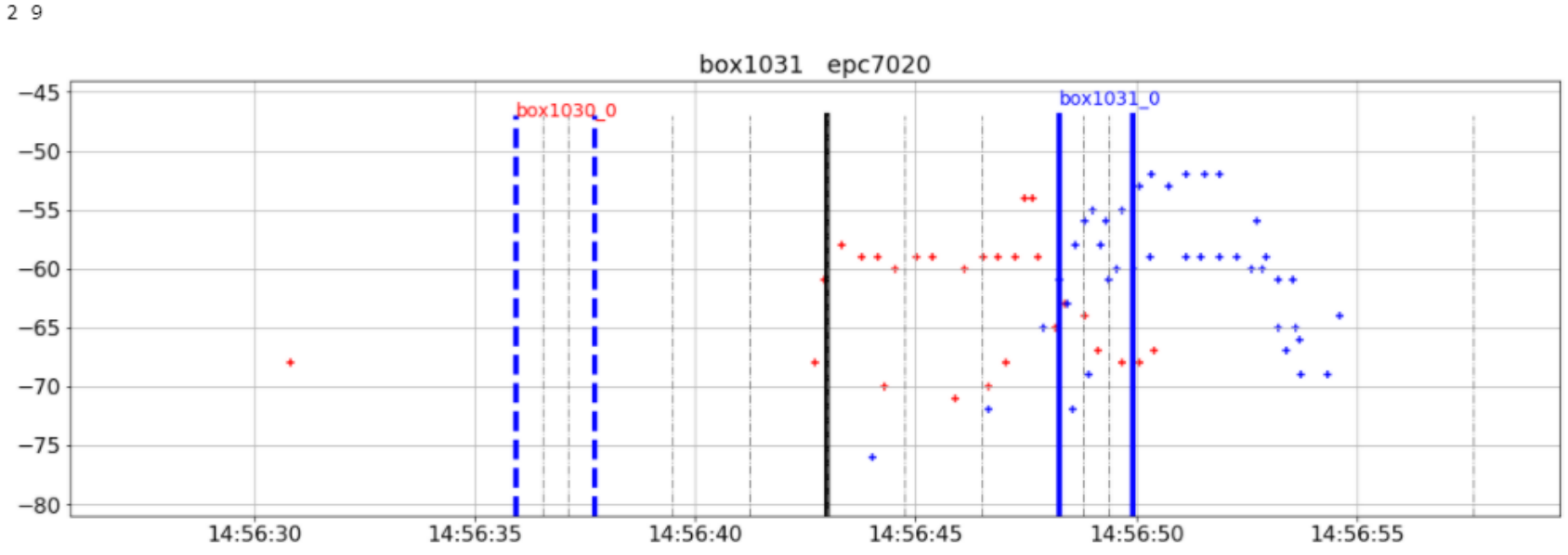


Machine Learning: Epcs_error_low

FP
13 True, 3 False



FP
9 True, 2 False



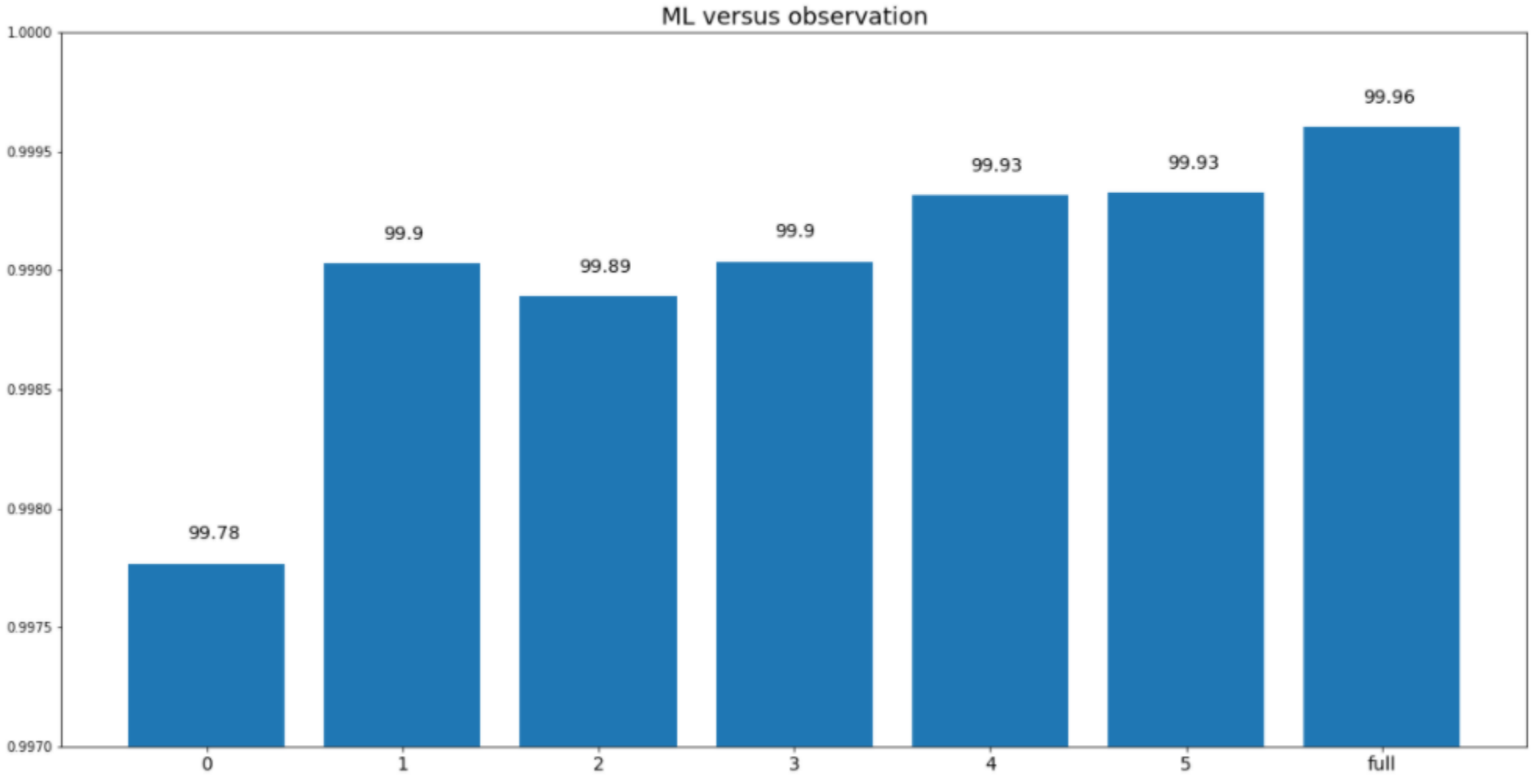
PARAMETRIC ANALYSIS

Parametric analysis

parameter			
Observations	7	Stop+0sec, Stop+1sec, Stop+2sec, Stop+3sec, Stop+4sec, Stop+5sec, full	
Steps	6	2/3/4/5/6/7	
Features	5	all	rssi, rc, Epcs_window, reads_window, window_width
		rssi & rc only	rssi, rc
		rssi & rc_mid	rssi, rc_mid
		rssi only	rssi
		rc_only	rc
Rssi_quantile	3	1 (max), 0.9, 0.5	
Classifiers	4	KNN SVC logisticRegression RandomForest	
Classifiers_gs_parameters	34	34 parameters in average CV=5	

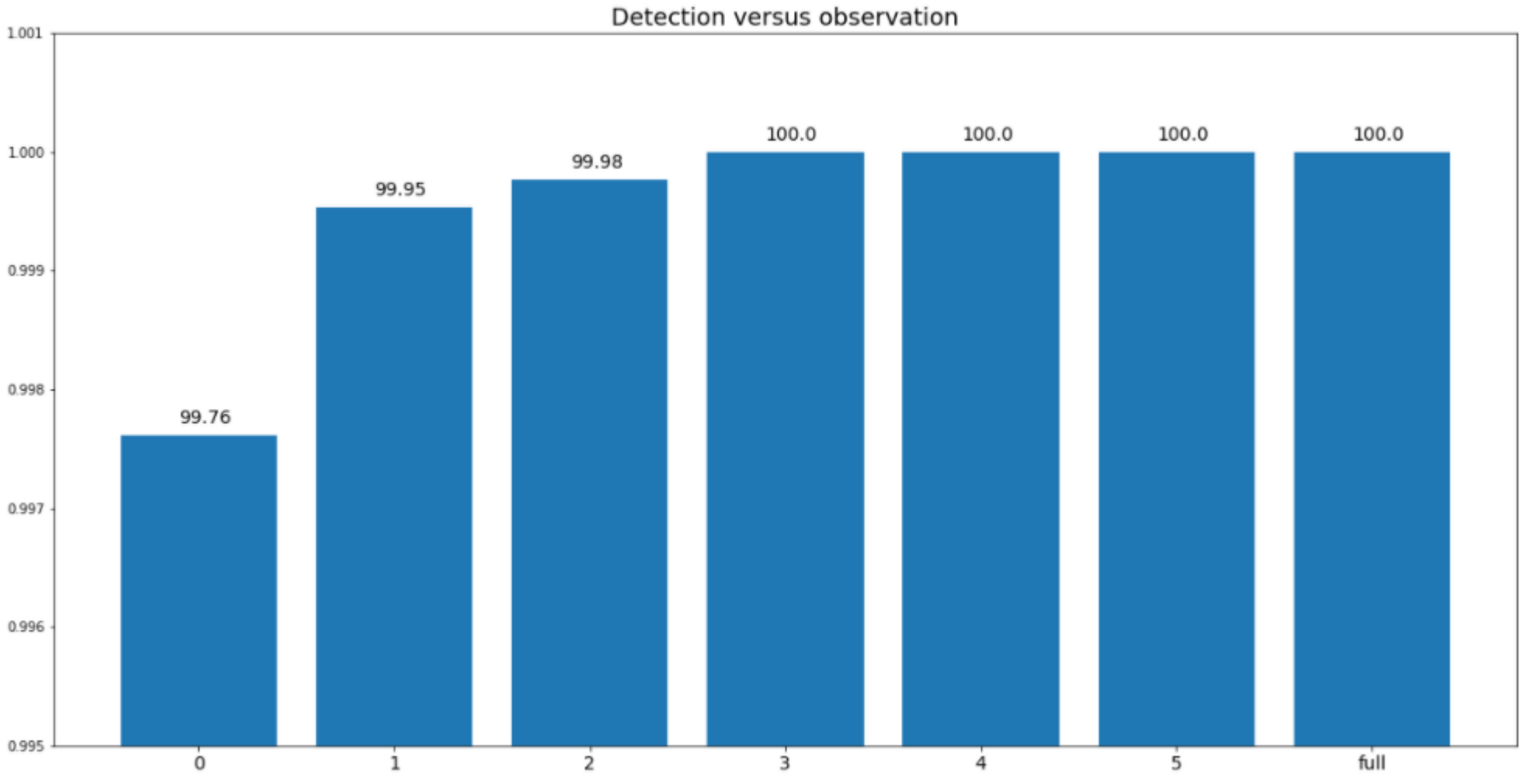
ML prediction versus observation

The larger the observation, the higher the prediction
From 0 to 1, errors x3 less



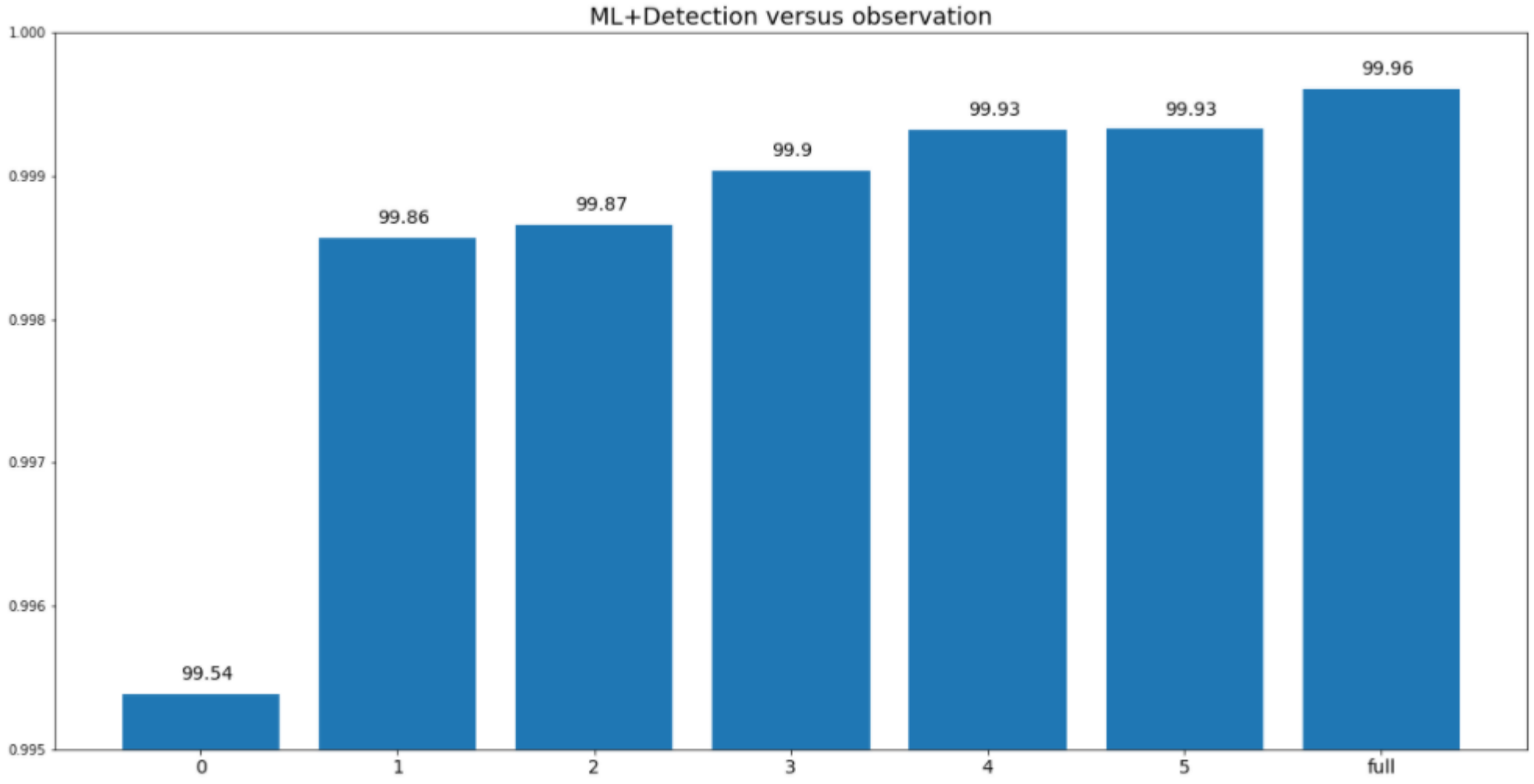
Detection versus observation

13,010 different tags
100% detection from Stop+3sec



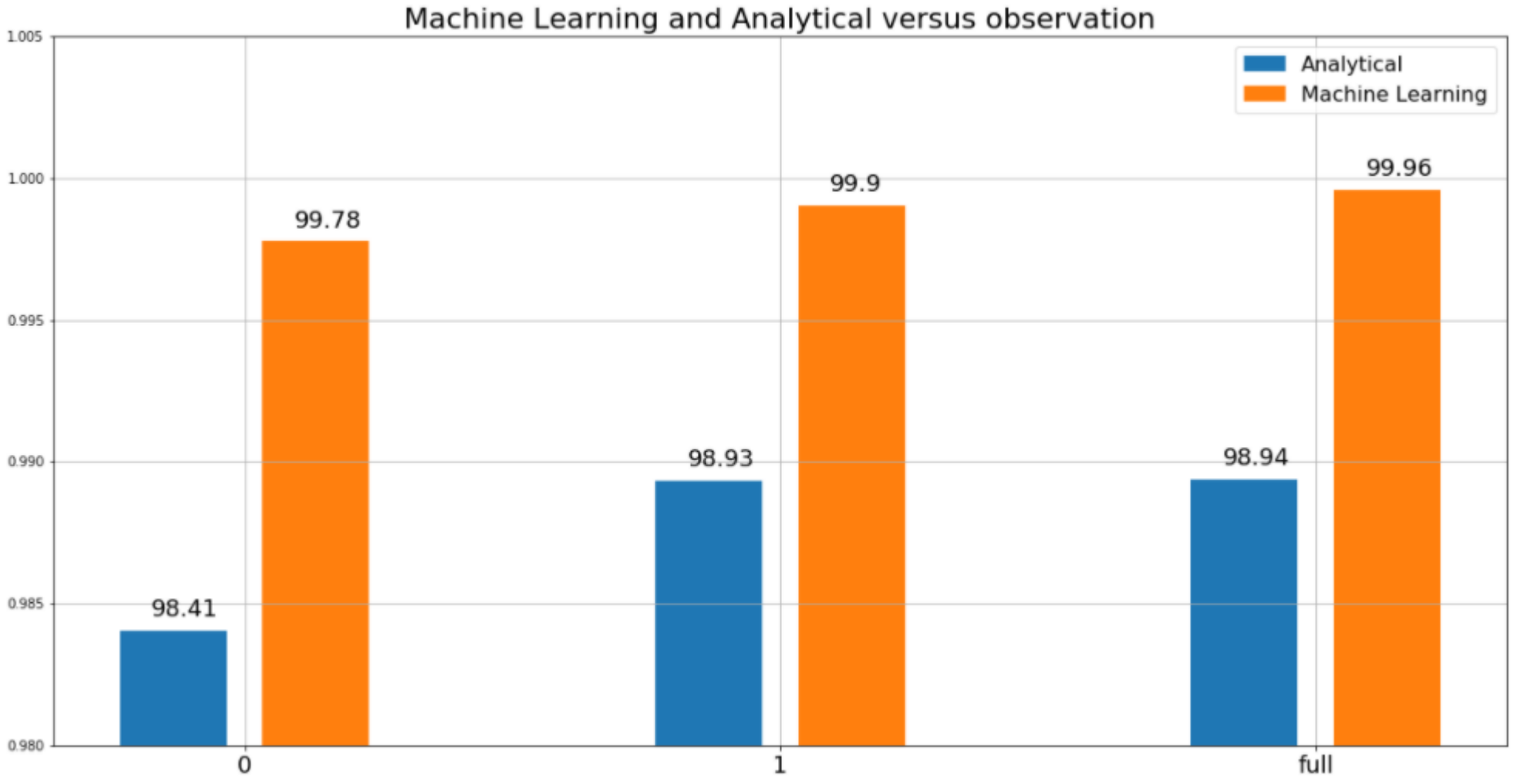
Detection + ML prediction versus observation

Significant improvement step with Stop+1sec



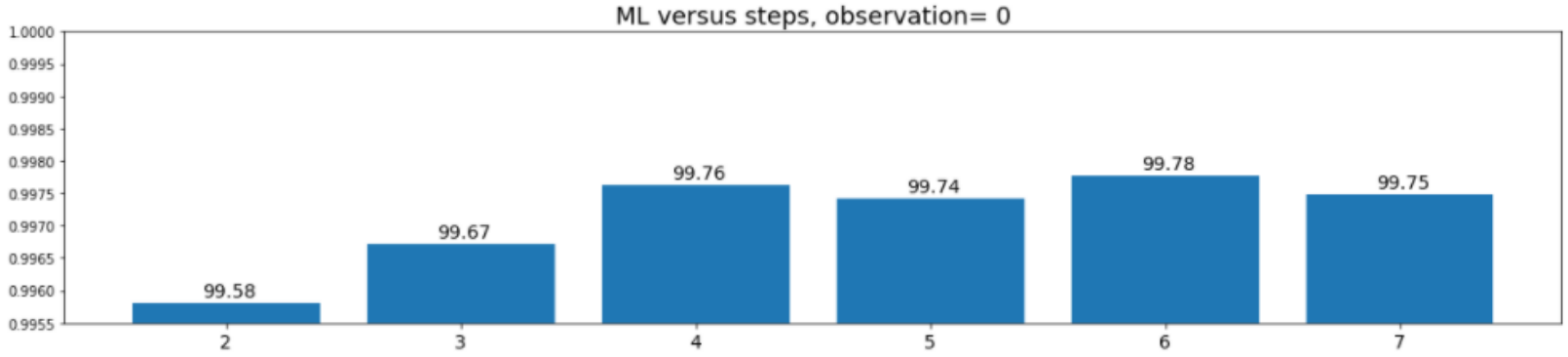
ML prediction & Analytical versus observation

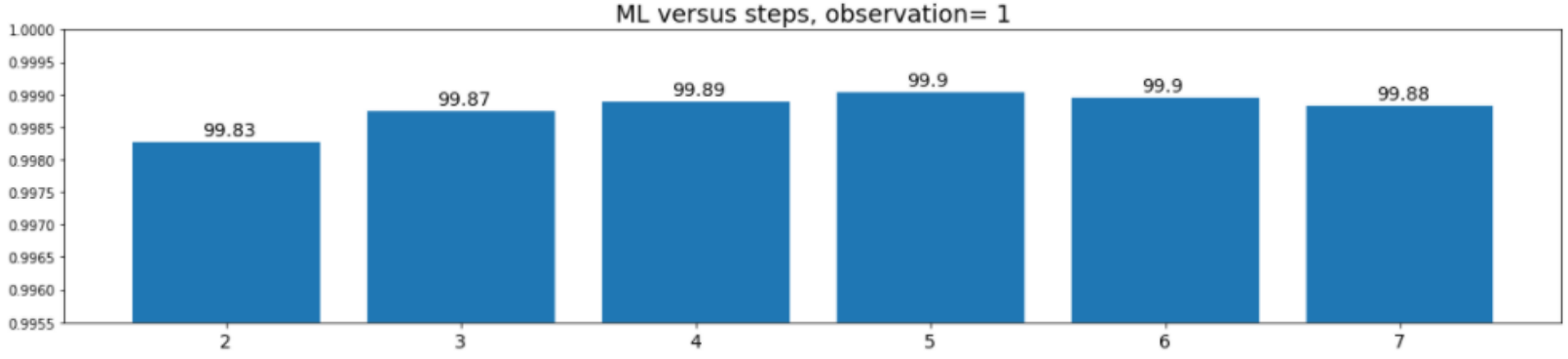
ML outperforms Analytical in a ratio >10

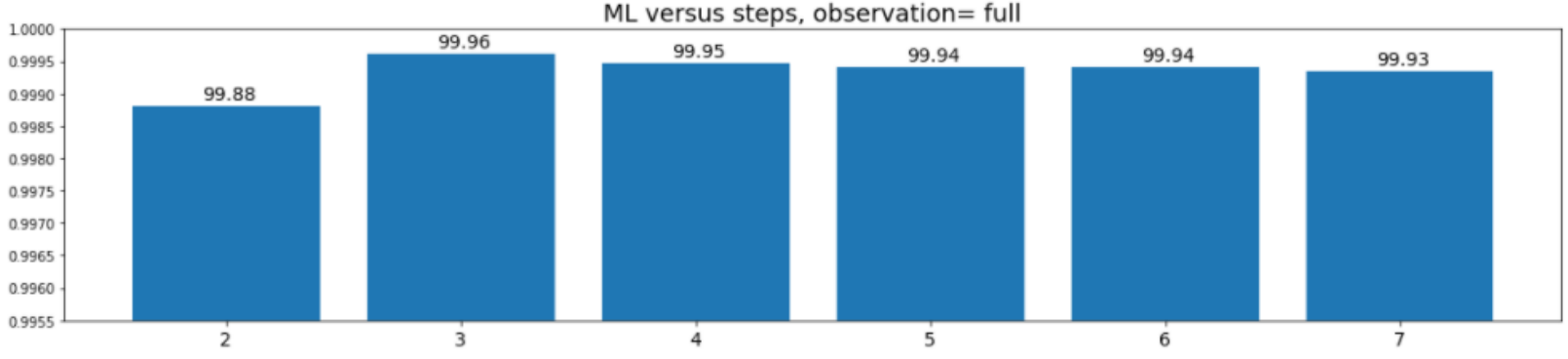


ML prediction versus (observation, steps)

Low impact
Steps=4

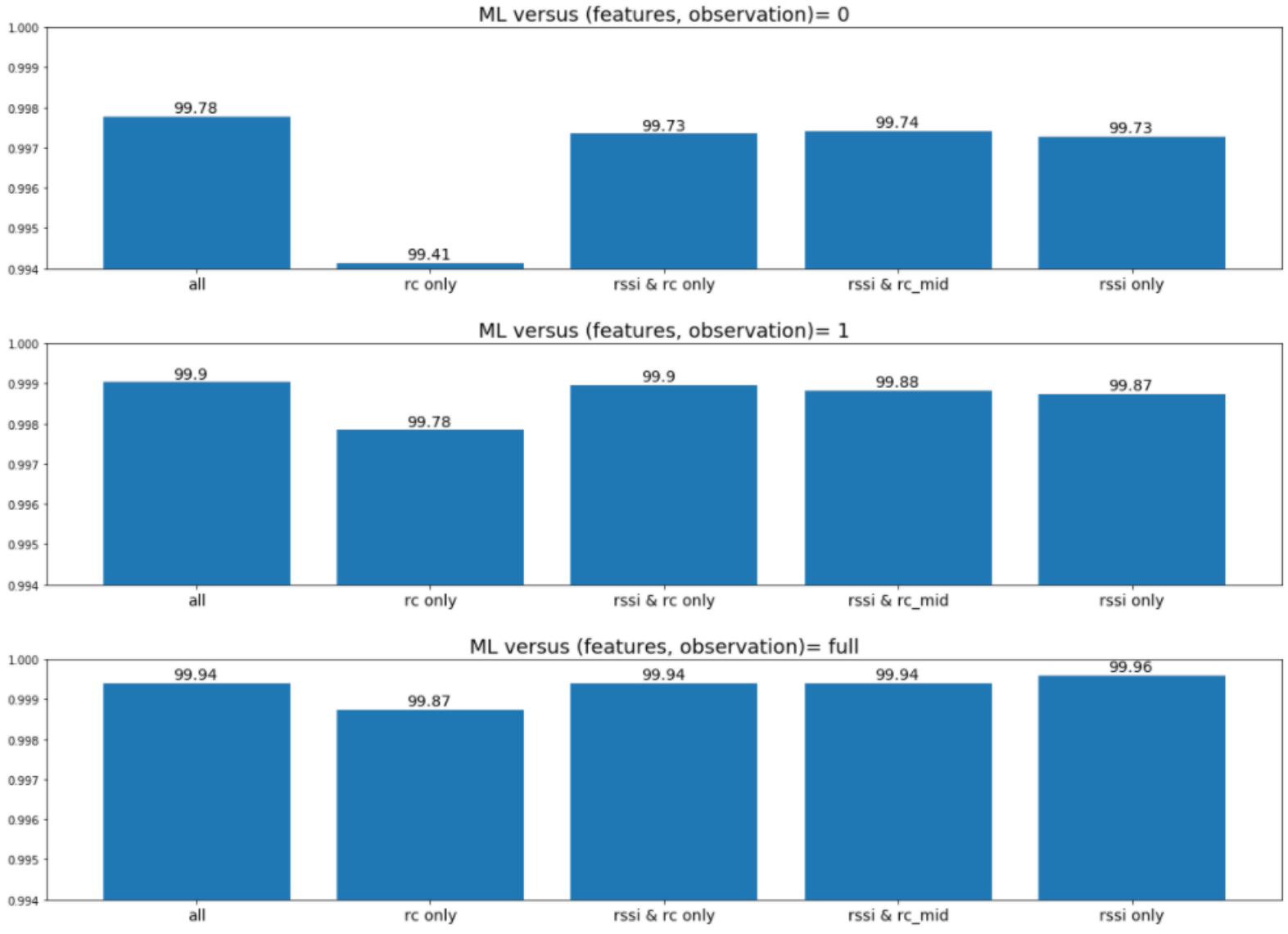






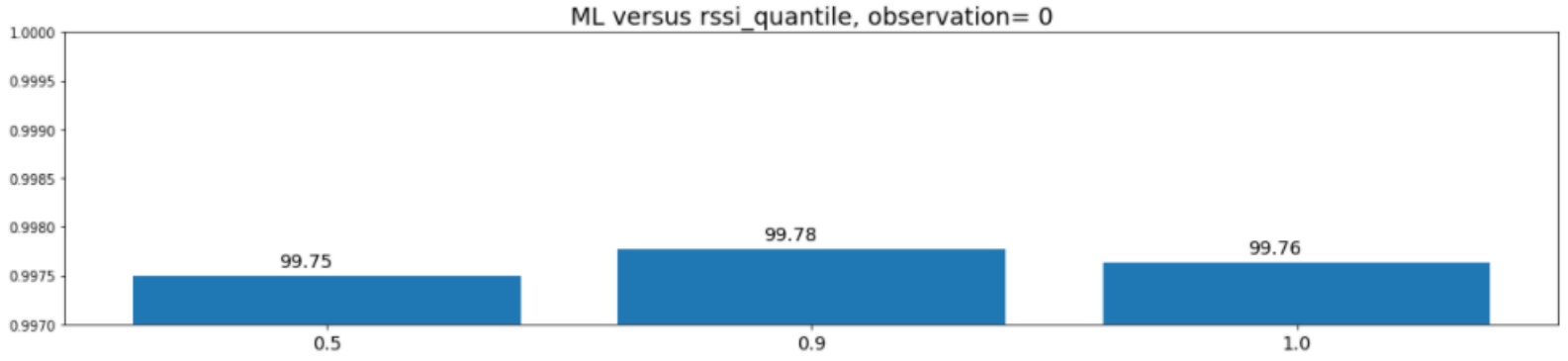
ML prediction versus (observation, features)

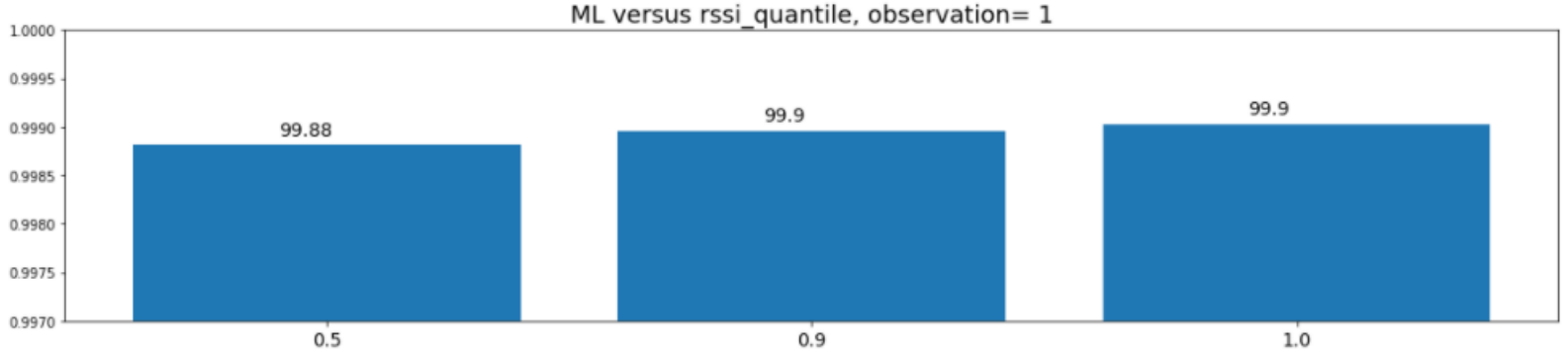
rssi is key
Other features: very low impact

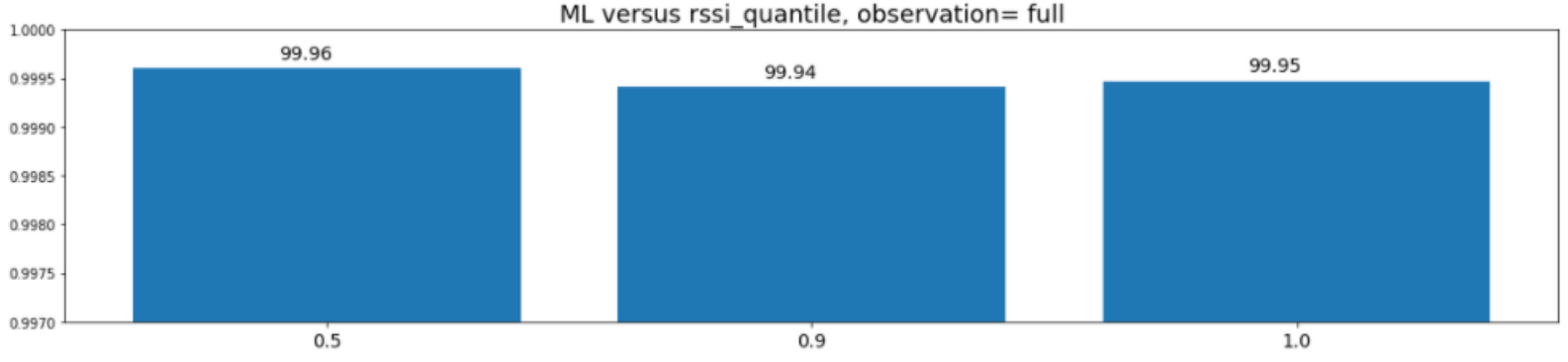


ML prediction versus (observation, rssi_quantile)

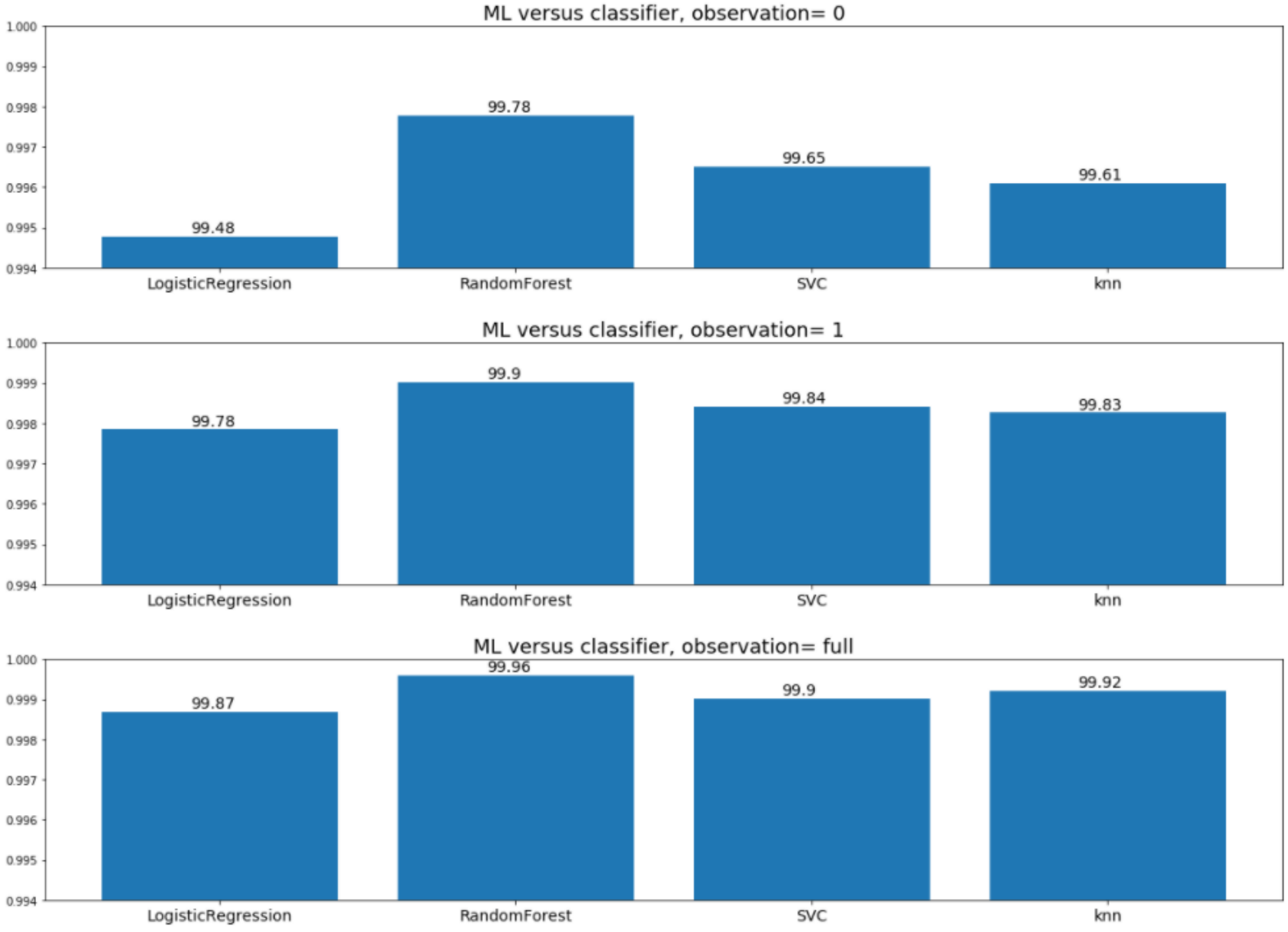
No impact
➔1







ML prediction versus classifier



➔ RandomForest
2x less errors than Logistic Regression

Classifier parameters

```
param_svc ={\
            'C':[10],\
            'kernel':['rbf'],\
             'gamma':['scale'],\
            }
param_lr ={\
            'penalty':['l1'],\
            'C':[10],\
             'solver':['liblinear'],\
            }
param_rf ={\
            'n_estimators':[100],\
            'max_depth':[20],\
            }
param_knn ={\
              'algorithm': ['auto'],
              'metric': ['manhattan'],
              'n_neighbors': [3],
              'weights': ['distance']\
             }

Classifiers_gs=pd.DataFrame([\
                                ['SVC', SVC(), param_svc],\
                                ['LogisticRegression', LogisticRegression(), param_lr],\
                                ['RandomForest', RandomForestClassifier(), param_rf],\
                                 ['knn', KNeighborsClassifier(), param_knn],\
                                ], columns=['classifier', 'clf', 'params_gs'])
Classifiers_gs
```

TUNNEL UPGRADE

RFID tunnel hardware upgrade

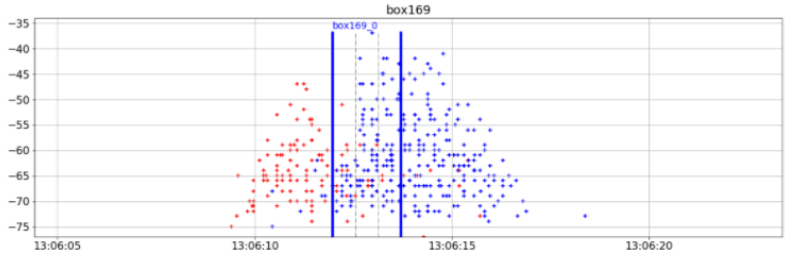
RFID tunnel hardware upgrade	Measured limitations
Extra antenna on top	No reads in (Start-Stop) window ➔ Lack of y polar
Higher power	12 mis-detected tags
	1% lowest RSSI: extremely low (-70dBm)
Faster reader (automatic Q) like Impinj	1% lowest ReadCount tags Down to 1 with high RSSI (-54dBm)

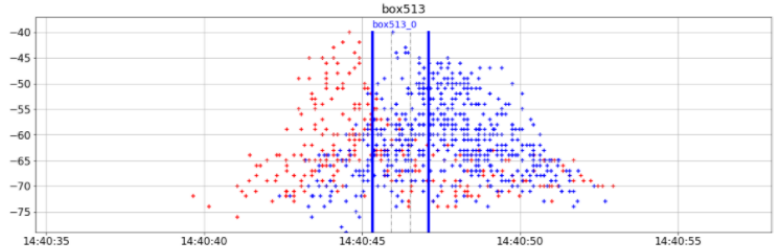
New RFID tunnel hardware:

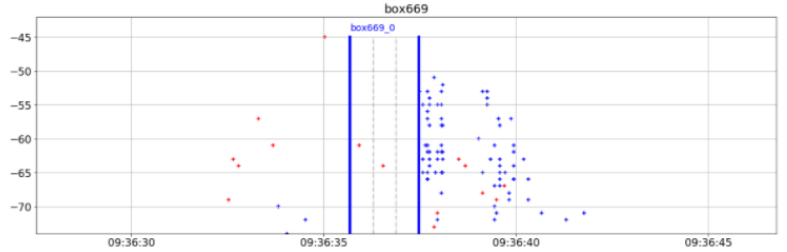
1. RFID reader: latest R700 (Impinj) because faster (and more sensitive) with antenna hub to connect up 6 antennas
2. Antennas:
   a. Same positioning as actual (OUT antennas upstream and IN antennas downstream)
   b. OUT antennas: 3 antennas in arch left/right + top)
   c. IN antennas: 3 antennas in arch left/right + top)
3. All other elements unchanged: shielding on rejected line, box separator, bar code reader, …

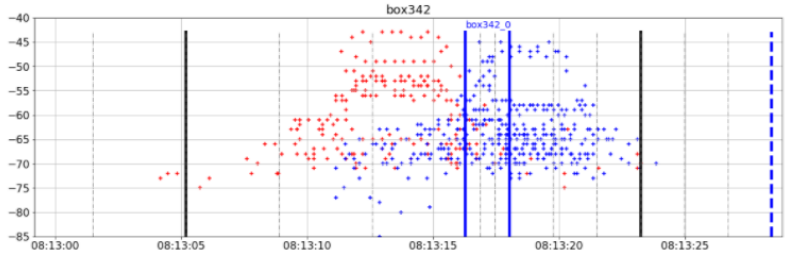
Higher power

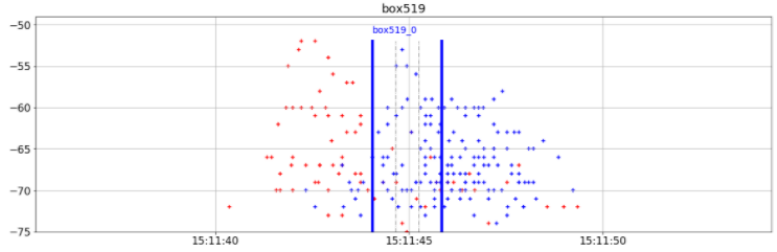
12 tags missing out of 13010 ➔ 0.09% misdetected tags
10 corresponding error boxes but looking OK with other tags
Possible causes: reflist human error? Lack of Power with « difficult » tags?
➔ **higher power**

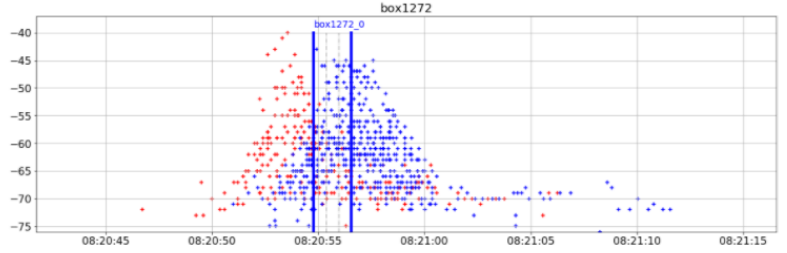


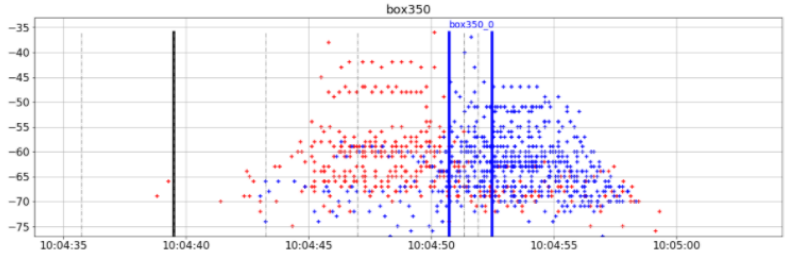


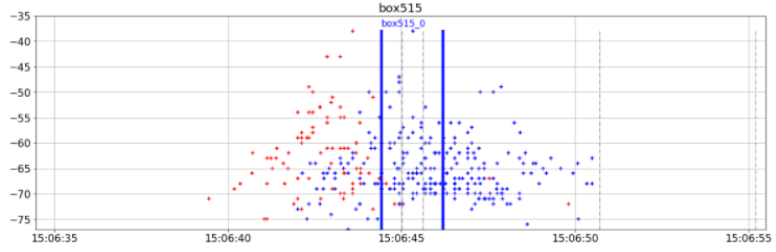


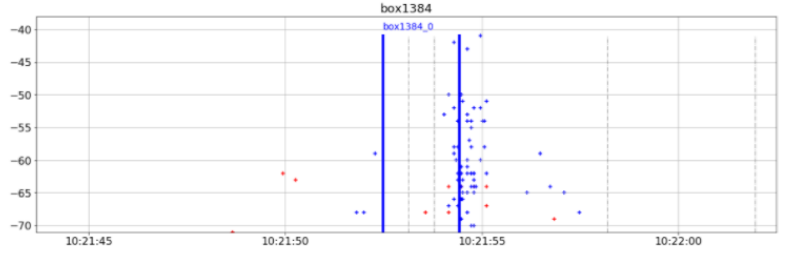


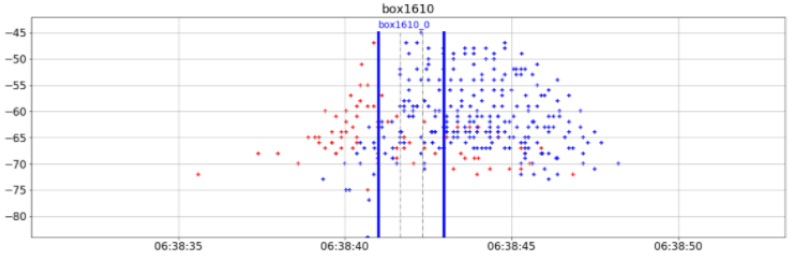












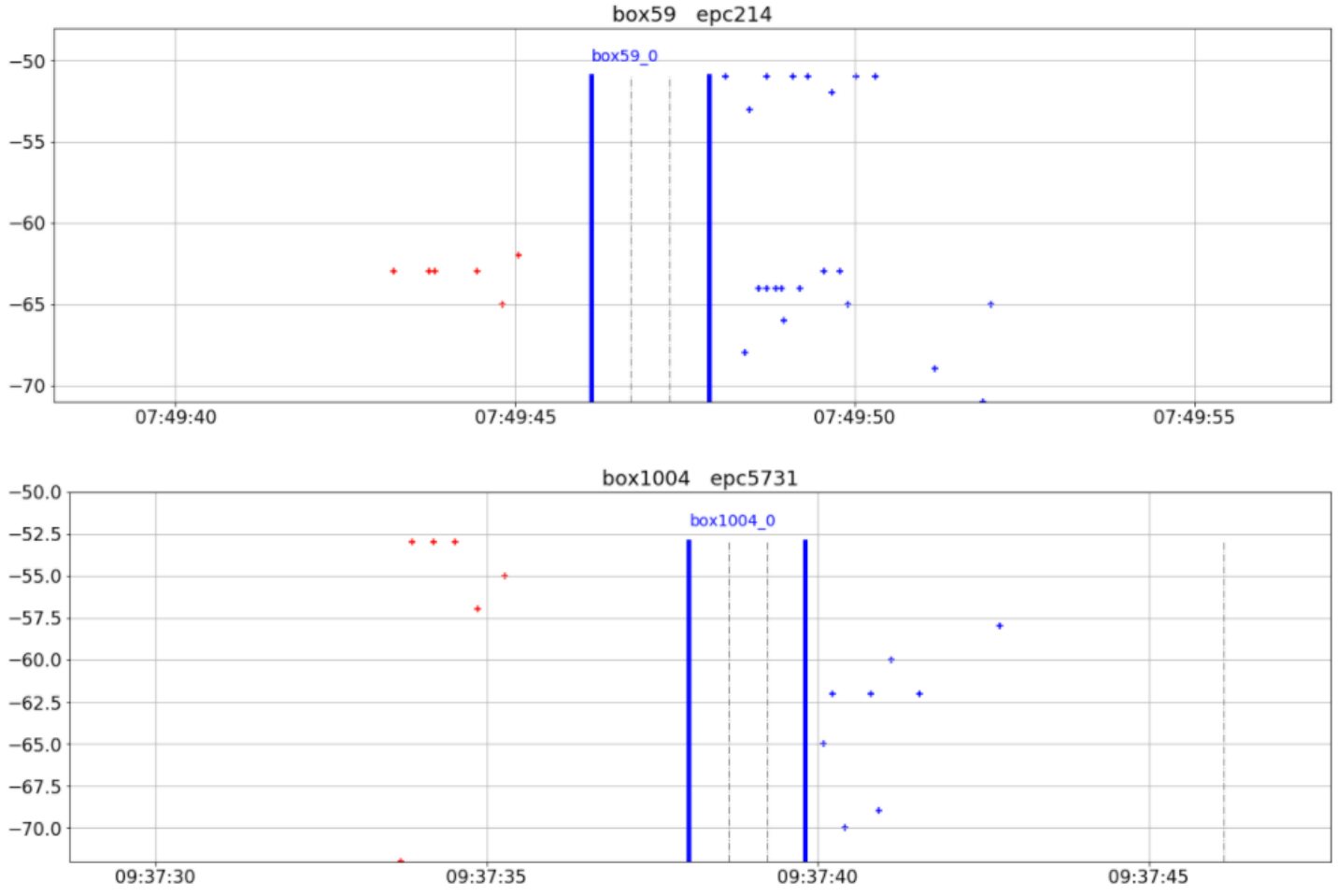
Extra antenna over or below the conveyor

138 tags out of 13010:

1. not detected in mid slice when facing antennas
2. But detected in up and down when away and angled with the antennas

➔ **Lack of polar y**

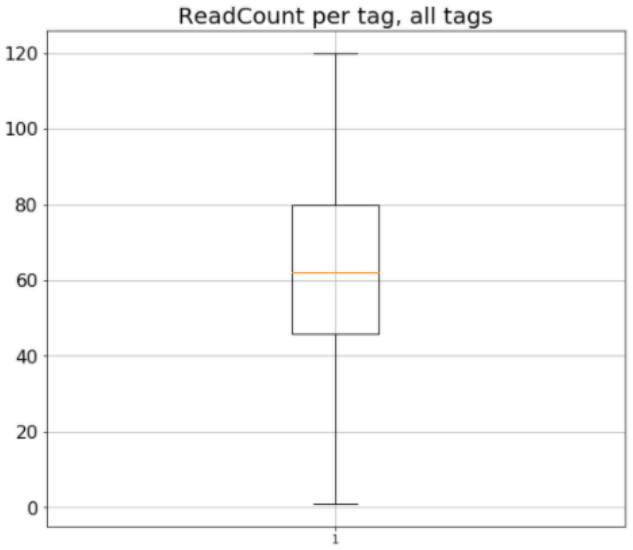
➔ **extra antenna over or below the conveyor**

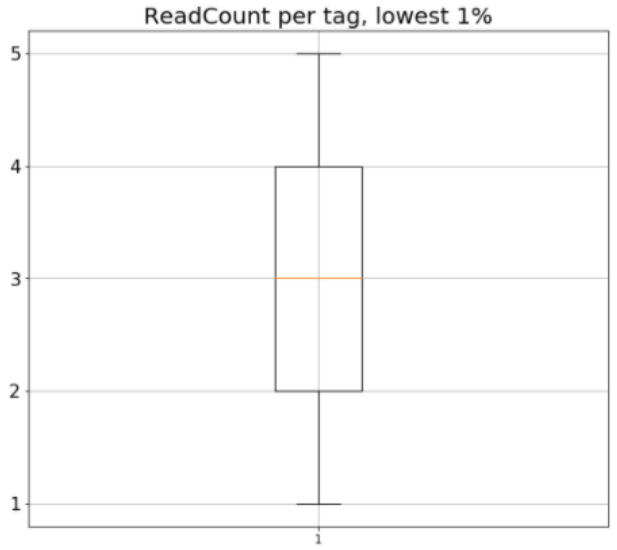


Faster reading

ReadCount per tag:

1. All tags: >60
2. Lowest 1%: 3



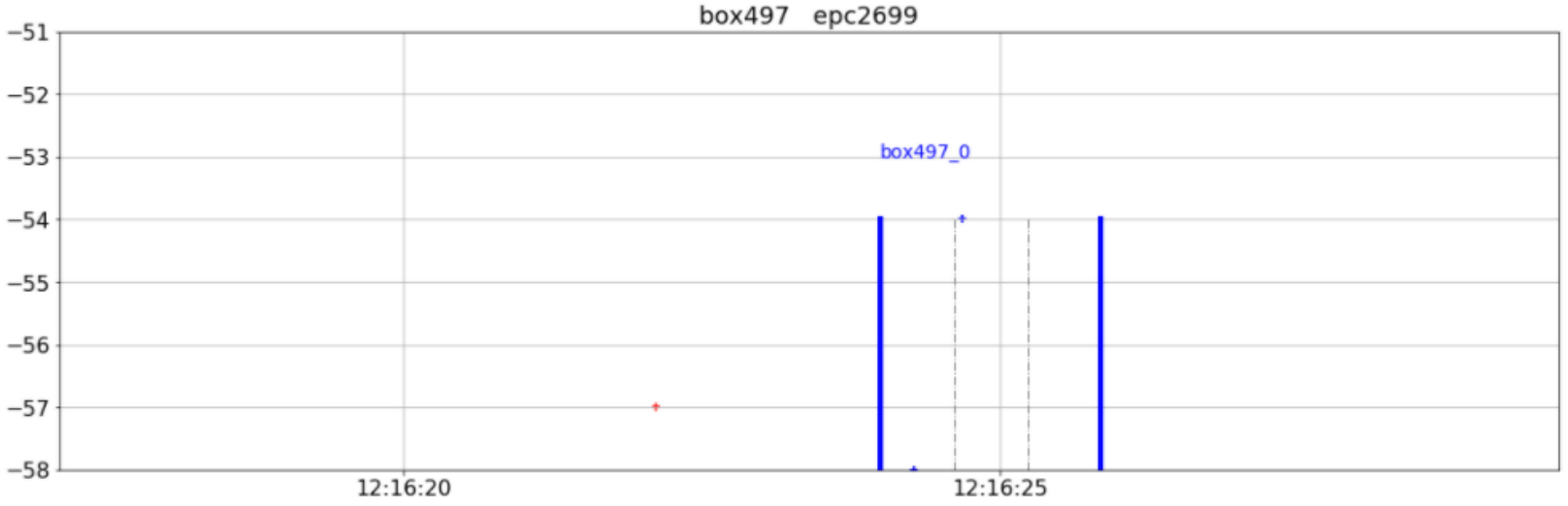


Lowest 1% ReadCount:

➔ RSSI can be pretty high: -54dBm

➔ Faster reading

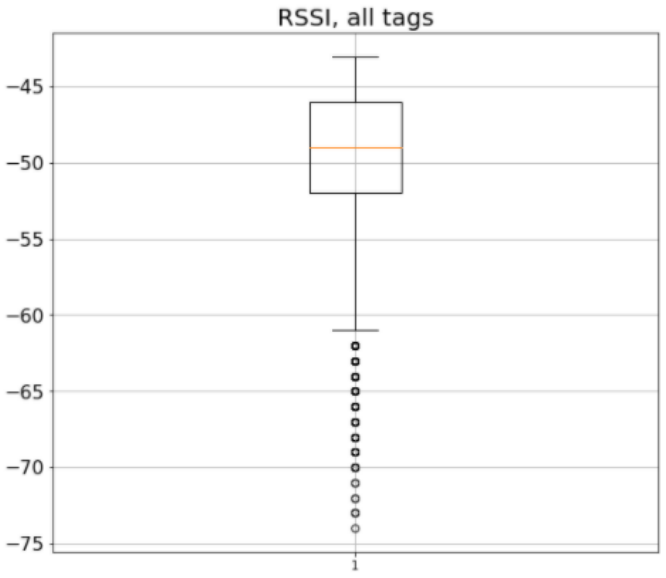
➔ **automatic Q reader (impinj reader)**

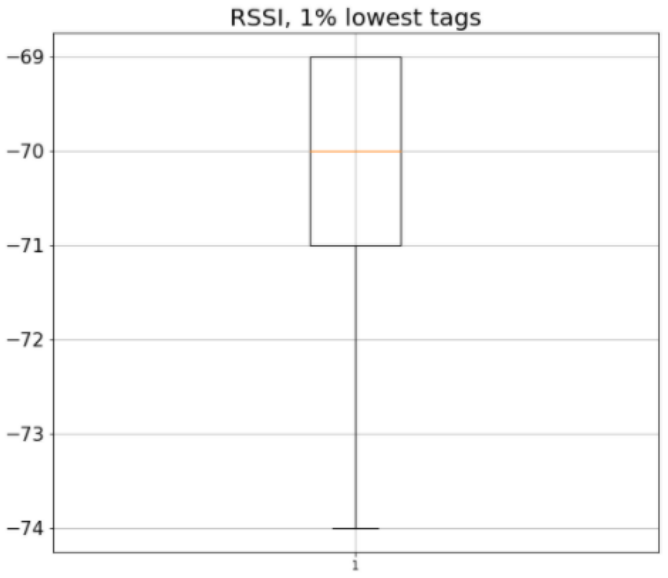


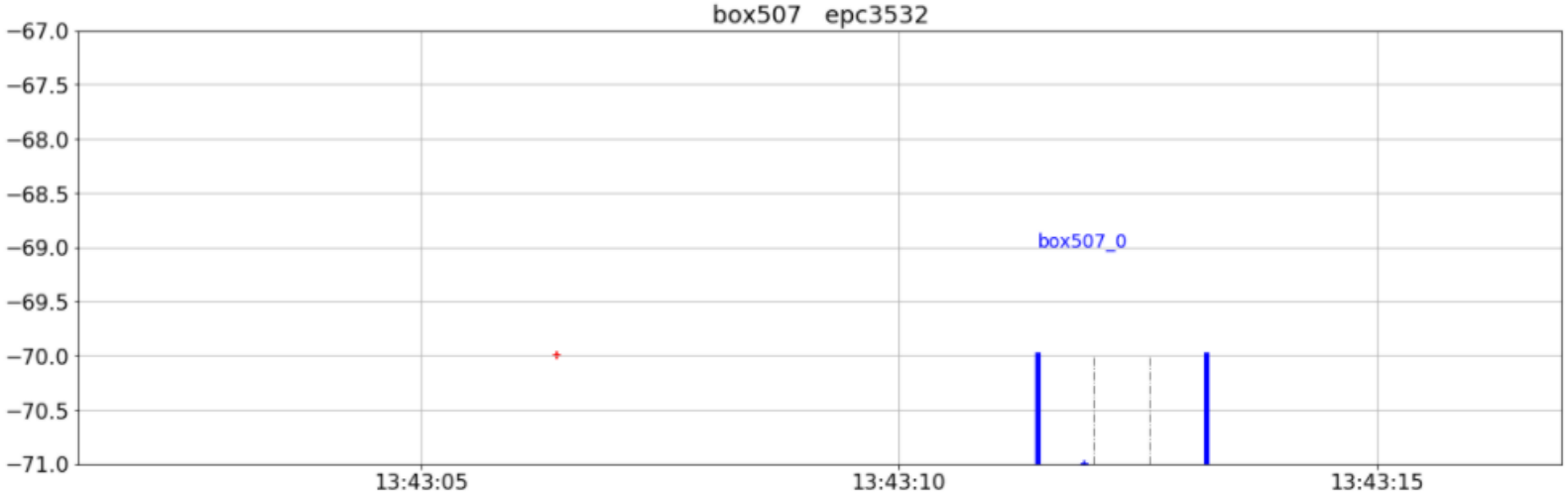
Higher Power

RSSI per tag:

1. All tags: >-50dBm
2. Lowest 1%: -70dBm







Lowest 1% RSSI:

➔ RSSI can be very low -70dBm

➔ **Higher Power**

Success rate versus train size

The larger the train set, the higher the performance: 99.94% with 1400 train boxes

However, 400 boxes: >99.9%

400 boxes: 1h30