Anomaly detection in time series data using autoencoders

Project 25 - Introduction

LiRA Project: Introduction

- http://lira-project.dk/
- Conventional sensors installed in modern vehicles (Green Mobility cars):
  - Acceleration in 3d
  - Car speed

- The goal is to predict road pavements conditions by using such as:
  - road roughness (IRI)
  - noise
  - friction
  - cracks and potholes.

- Targets are recorded using specialised vehicles:
  - laser profilometers (p79) - road roughness (IRI)
  - ARAN (cracks and potholes)
  - Some should be computed using physical models

Data for the project

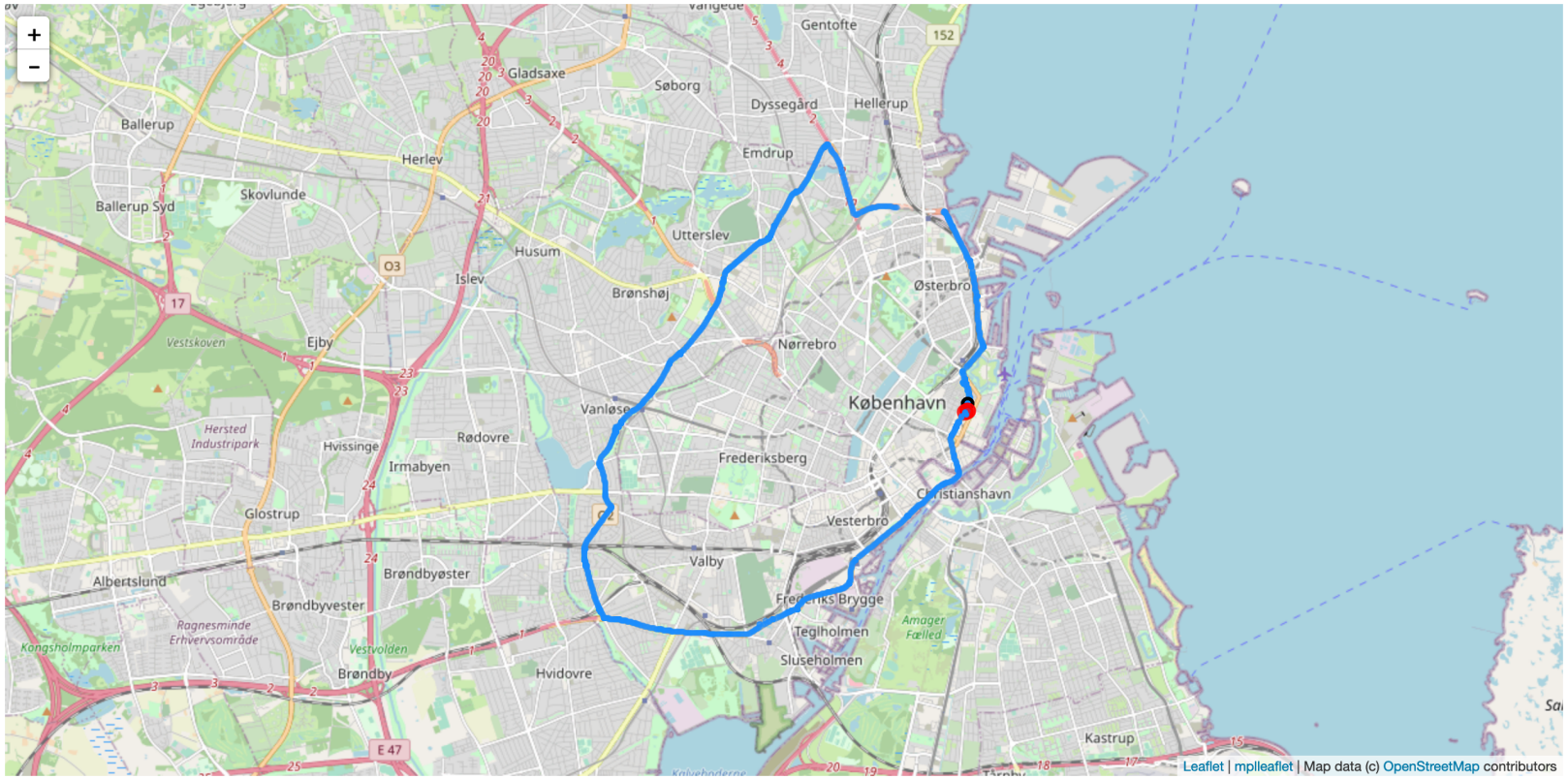
- The used data will be:
  - Green Mobility car data with selected sensors
  - P79 road roughness (IRI) targets

- Full data is available in a PostgreSQL database
  - The car data GPS coordinates are very noisy, hence done is map matching (Hidden Markov Models) and interpolation
  - The car data is aligned with p79 data

- For this project, extracted and prepared (map matched, interpolated, aligned and cleaned) is a part of recorded data
  - Available here:
    https://drive.google.com/drive/folders/1SS_IpHoa0S4WfVYzxabk8zFGLHGqmc6e

Data for the project

- 2 pickle files with Pandas objects
- Each file contains data covering ~25km of Copenhagen 1 route (~50km combined)



Data for the project

- **The data is aligned per road segments:**
  - Each 100m window of Green Mobility car data is aligned with 100m of p79 IRI data
  - Sliding step of 10m is applied to get more windows
- **The road roughness is characterised using IRI (1 average value per 100m)**
  - Bad road (=larger IRI) -> expect more oscillation in the acceleration-z direction (perpendicular to the road)
  - Due to different car speeds, 100m segment will have different lengths
    - Slow car -> many points (easier to predict)
    - Fast car -> few points (harder)
- **Acceleration-z and speed** are already intuitively very important (predictive) sensors and will be considered for this analysis, but acceleration-x and y can be tested in addition
- **The sensors sample at different time points**, example: acc-z, acc-z, speed, acc-z …(readout sorted in time)

*files are large, loading them will take some time

Data Structure

- Each row is a 100m segment
- Columns:
  - **P79 recordings:**
    - ‘IRI_mean’ - IRI averaged over 100m
    - ‘IRI_sequence' - all IRI values recorded in 100m using p79
  - **Car (Green Mobility) recordings**
    - ‘GM.lat_int’,'GM.lon_int' - interpolated latitude and longitudes of all measurements
    - 'GM.acc.xyz.x', 'GM.acc.xyz.y', ‘GM.acc.xyz.z’ - accelerometer readings in x (along),y (side) and z(upwards wtr to the road surface) directions
    - ‘GM.obd.spd_veh.value' - car speed
    - ‘GM.TS_or_Distance' - timestamps of all recordings (all sensors sorted and combined)
    - ‘GM.T’ - sensor type

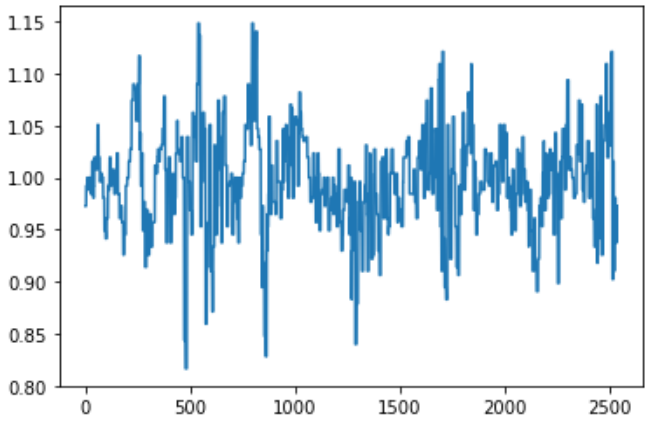
df - DataFrame

Index	IRI_mean	IRI_sequence	GM.TS_or_Distance	GM.T	GM.lat_int	GM.lon_int	GM.acc.xyz.z
0	4.93354	[6.6448828 5.5573473… 5.8527534 2.0332293…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65471331 55… 55.65554094]	[12.4866869 12.48668653 1… 12.48611034]	[0.9727 0.9727 0.9727 ... 0.9375 0.9375 0.9727]
1	4.69622	[5.55734731 7.6663199… 2.03322937 2.8570602…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65478146 55… 55.65562473]	[12.48662899 12.48662888 1… 12.48606118]	[0.9727 0.9727 0.9727 ... 1.0703 1.0703 1.0703]
2	4.69915	[7.66631995 4.1605364… 2.85706026 4.6520824…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65486231 55… 55.65571197]	[12.48656058 12.48656033 1… 12.48601007]	[1.0391 1.0391 1.0391 ... 1.0352 1.0352 1.0352]
3	4.39757	[4.16053647 5.3599070… 4.65208249 4.2717243…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65494254 55… 55.65579754]	[12.48649296 12.48649288 1… 12.48596003]	[1.0391 1.0391 0.9805 ... 0.9883 0.9883 0.9883]
4	4.40055	[5.35990705 4.5512844… 4.2717243 5.5865571…	Timestamp('202… Timestamp('202…	['obd.spd_ve…	[55.65502238 55… 55.65588877]	[12.48642492 12.48642466 1… 12.48590919]	[0.9492 0.9961 0.9961 ... 0.9766 0.9766 0.9766]
5	4.44037	[4.55128446 5.8527534… 5.58655713 4.6505287…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65511527 55… 55.65597295]	[12.48635153 12.48635153 1… 12.4858633]	[0.9063 0.9063 0.9063 ... 1.0117 1.0117 1.0156]
6	4.3715	[5.8527534 2.0332293… 4.65052872 4.1903628…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65520049 55… 55.65605681]	[12.48630597 12.48630556 1… 12.48581817]	[1.0195 1.0195 1.0195 ... 0.9883 0.9766 0.9766]
7	4.12271	[2.03322937 2.8570602… 4.19036283 5.7580724…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65528926 55… 55.65614293]	[12.4862577 12.48625748 1… 12.48577203]	[0.9883 0.9883 0.9883 ... 0.9883 0.9883 0.9883]
8	4.24496	[2.85706026 4.6520824… 5.75807246 3.8626593…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65537202 55… 55.65623125]	[12.48620959 12.48620948 1… 12.48572517]	[1.0391 1.0391 1.0391 ... 0.9727 0.9727 0.9766]
9	4.29173	[4.65208249 4.2717243… 3.86265931 3.3648383…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.6554573 55… 55.65631644]	[12.4861598 12.48615955 1… 12.48567908]	[0.8984 0.9805 0.9805 ... 0.9766 1.0234 1.0234]
10	4.03366	[4.2717243 5.5865571… 3.36483837 3.2557203…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65554185 55… 55.65640019]	[12.48610983 12.48610971 1… 12.48563412]	[0.9727 0.9063 0.9063 ... 1.0078 1.0078 1.0078]
11	3.8437	[5.58655713 4.6505287… 3.25572039 3.3247534…	Timestamp('202… Timestamp('202…	['obd.spd_ve…	[55.65562585 55… 55.65648843]	[12.48606052 12.4860604 1… 12.48558705]	[1.1016 1.1016 0.9766 ... 0.9375 0.9375 0.9375]
12	3.70043	[4.65052872 4.1903628… 3.32475342 2.0714105…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65571218 55… 55.65657055]	[12.48600993 12.4860098 1… 12.48554356]	[1.0273 1.0273 1.0273 ... 0.9531 0.9531 0.9531]
13	3.46654	[4.19036283 5.7580724… 2.07141053 2.3721431…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65579805 55… 55.65666097]	[12.48595975 12.48595962 1… 12.4854954]	[0.9883 0.9648 0.9648 ... 0.9648 0.9648 0.9648]
14	3.2047	[5.75807246 3.8626593… 2.37214313 4.1538046…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65588916 55… 55.65674484]	[12.48590898 12.48590883 1… 12.48545085]	[0.9531 0.9531 0.9531 ... 1.0234 1.0234 1.0078]
15	2.78836	[3.86265931 3.3648383… 4.1538046 2.3116809…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65597352 55… 55.65683089]	[12.48586299 12.48586283 1… 12.48540424]	[1.0156 1.0156 1.0156 ... 1. 1. 1.]
16	2.57506	[3.36483837 3.2557203… 2.31168094 1.5719552…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65605856 55… 55.65692001]	[12.48581723 12.48581698 1… 12.48535608]	[1.0039 1.0039 1.0039 ... 1.0469 1.0078 1.0078]
17	2.41431	[3.25572039 3.3247534… 1.5719552 1.5946838…	Timestamp('202… Timestamp('202…	['acc.xyz' '…	[55.65614321 55… 55.65700288]	[12.4857719 12.48577176 1… 12.48531206]	[0.9922 0.9922 0.9922 ... 1.0156 1.0156 1.0156]
18	2.29731	[3.32475342 2.0714105… 1.59468382 1.7296317	Timestamp('202… Timestamp('202	['acc.xyz' '…	[55.65623401 55… 55.65708701]	[12.48572368 12.48572323 1… 12.48526709]	[0.9766 0.9727 0.9727 ... 1.0195 1.0195 1.0234]

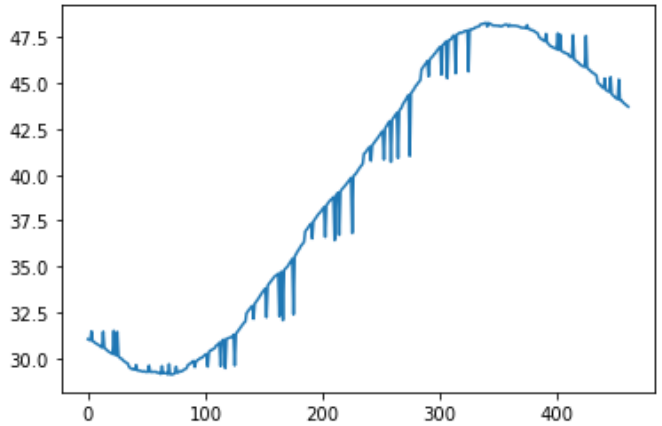
*files are large, loading them will take some time

Data Structure

- Car (Green Mobility) recordings are given as arrays (sequences)
- Each array consists of all readouts of a specific sensor per 100m segment
- Most important: ‘GM.acc.xyz.z' and ‘GM.obd.spd_veh.value’
- **Example:** the 1st 100m segment



plt.plot(df[‘GM.acc.xyz.z'].iloc[0])



plt.plot(df['GM.obd.spd_veh.value'].iloc[0])

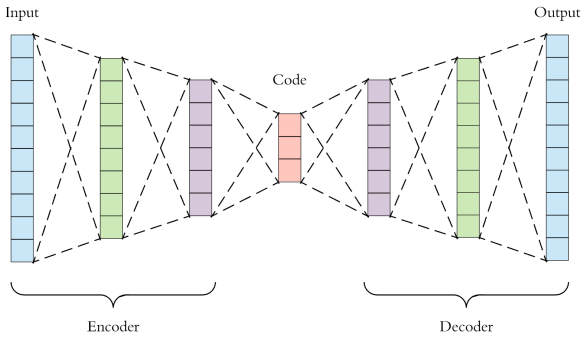
Related to IRI: df[‘IRI_mean’].iloc[0] = 4.93 (not so good road)

Autoencoders

- Represent the time series data using a compressed representation (find a set of features which describes that time series well)

- Encode (compress) a multidimensional sequence (acceleration-z, speed…) to a vector which encompasses the information in those sequences

- **Autoencoder:**
  - Can try a simple autoencoder and/or a variational autoencoder
  - Autoencoder architecture can be built using: Dense, RNN, CNN or combined units
  - **RNN and CNN (CNN-1D for time series) are typically used for sequence data since they take into account the correlations between points in a sample**
    - A road defect with generate a longer oscillation within the signal… hence the points in the signal are often correlated within each other



Anomaly detection

1) Supervised

- Example: train a classification algorithm to distinguish between the normal and anomalous class
- Learns to detect known anomalies

2) Unsupervised

- Learn to detect anything that deviates from normal samples (in principle, detects both known and unknown anomalies)
- Autoencoder - a model that learns to reconstruct the data it was trained on
- Is it reconstructing well? Define an anomaly score (reconstruction error between input and output sequence)
- **Training**
  - Use **only normal** samples for training
  - The autoencoder will learn the latent representation of normal samples and will reconstruct normal samples well
- **How to evaluate the anomaly detection?** Use both normal and anomalous samples -> good model should compute small reconstruction errors for normal and large for anomalous samples
  - A threshold on reconstruction error can be chosen to detect a % of *known* anomalies

Plan

- Find a paper for inspiration (or combine few and add your own ideas) and try to implement the main part of it or try to compile the mentioned ideas
- Submit a realistic project synopses
- Explore and understand the structure of the data
- Build a model
- Make a poster and write the report