glucose_ts

Release 0.0.1

Christoph Lange

Aug 23, 2021

CONTENTS

CHAPTER

ONE

DATA STRUCTURES

As this package predominantly deals with time series we have the following main structure.

1.1 Time-series Container

The idea is to bundle all datapoints that were measures during one experiment.

class glucose_ts.data.**GlucoseTS**(*points_in_time*, *voltages*, *real_concentration*, *comment=''*)

The data structure that represents one glucose sensor experiment.

Parameters

- **points_in_time** (*np.array*) – an array that represents the number of minutes from the start of the experiment
- **voltages** (*np.array*) – an array of voltage numbers that correspond to the points in time when the voltage was measured
- **real_concentration** (*float*) – the "real" concentration of the glucose compound during the experiment
- **comment** (*str*) – anything you want

This data structure can be obtained from excel file that are produced in the lab:

glucose_ts.data.**read_glucose_ts**(*path*, *comment=''*)

Reads an exel file to obtain all relevant values for a Glucose time-series.

Parameters **path** (*str*) – a path to a respective exel file

Returns a glucose time-series

Return type *glucose_ts.data.GlucoseTS*

1.2 Manipulating Time-series

One common operation we need for making predictions is to cut the time-series.

glucose_ts.data.**cut_time_series**(*glucose_ts*, *cutoff_time*)

Creates a subset of a glucose time-series by cutting it off after a certain point in time.

Parameters

- **glucose_ts** (glucose_ts.data.GlucoseTS) – the glucose time-series to create a cut-off from

- **cutoff_time** (*float*) – the point in time until we want to keep the time-series

Returns a glucose time-series

Return type *glucose_ts.data.GlucoseTS*

CHAPTER

TWO

MODELS

The package contains different models to capture the voltage signal of the glucose sensor over time. The types of models that are included in the package right now are the following.

Table of Contents

2.1 Exponential Decay

The model behind exponential decay is

$$V(t) = K + (A - K)\exp(-Bt)$$

The three parameters are represented in the named tuple

class glucose_ts.models.exponential_decay.**ExpDParameter**(*A*, *K*, *B*)

The parameter container has all three parameters needed to specify an exponential decay model.

Parameters

- **A** (*float*) – the upper asymptote
- **K** (*float*) – the lower asymptote, for special cases the carrying capacity
- **B** (*float*) – the growth rate that describles the speed of decay

In order to learn the parameters that are a good fit to your training data we use the following estimator.

class glucose_ts.models.**ExponentialDecay**(*exp_d_params=None*, *gaussian_priors=None*, *variance=None*)

This Estimator learns the paramters of an exponential decay model. It provides the classic maximum likelihood approach as well as the Bayesian approach maximum posterior.

Moreover it implements the interface of a scikit-learn estimator.

fit(*time*, *labels*)

Fits a exponential decay model to training data.

Parameters

- **time** – points in time or the independent variable here
- **labels** – voltage measurements or the dependent variable

Returns the trained exponential decay model

Return type *ExponentialDecay*

fit_least_squares(*features*, *labels*, *loss=None*)
Fits a exponential decay model to training data.

Parameters

- **time** – points in time or the independent variable here
- **labels** – voltage measurements or the dependent variable

Returns the trained exponential decay model

Return type *ExponentialDecay*

inverse(*inputs*)
computes the inverse function using the model internal parameters

Parameters **inputs** (*numpy.array*) – points we want to get the inverse values for

Returns the inverse function values

Return type numpy.array

predict(*time*)
makes predictions by using the model internal parameters

Parameters **time** (*numpy.array*) – points in time we want to make predictions for

Returns the predictions

Return type numpy.array

time_derivative(*time*)
computes the derivative with respect to time using the model internal parameters

Parameters **time** (*numpy.array*) – points in time we want to get the derivative for

Returns the derivatives

Return type numpy.array

2.2 Logistic Growth

The family of function we refer to as logistic growth models is described by

$$V(t) = A + \frac{K - A}{1 + \exp(-B(t - M))}$$

The four parameters are represented in the named tuple

class glucose_ts.models.logistic_decrease.**LDParameter**(*A*, *K*, *B*, *M*)
This data structure contains the four parameter that are needed to identify a logistic function. The notations are identical with the Wikipedia artile with $\nu = 1$, $Q = 1$ and $C = 1$ being set to fixed values.

Please note that we are always dealing with decays in case of the glucose sensor. Therefore we role a lower and upper asymptote is flipped.

Parameters

- **A** (*float*) – the upper asymptote
- **K** (*float*) – the lower asymptote, for the growth case it is the carrying capacity
- **B** (*float*) – the growth rate
- **M** (*float*) – the location parameter of the logistic curve

In order to learn the parameters that are a good fit to your training data we use the following estimator.

class glucose_ts.models.**LogisticDecrease**(*parameter=None*, *gaussian_priors=None*, *std=None*, *time_horizon=None*)

The Estimator that learns the paramters of a logistic growth model. You can use it for classic maximum likelihood and a Baysian approach.

fit(*time*, *labels*)

Finds the logistic growth model parameters that fit the training data

Parameters

- **time** (*np.array*) – points in time or the independent variable in this case
- **labels** (*np.array*) – voltage measurements or the dependent variable

Returns the trained generalized logistic model

Return type *GeneralizedLogisticGrowth*

predict(*time*)

makes predictions for all points in time by using the model internal parameters

Parameters **time** (*numpy.array*) – points in time we want to make predictions for

Returns the predictions

Return type numpy.array

time_derivative(*time*)

computes the derivative with respect to time using the model internal parameters

Parameters **time** (*numpy.array*) – all the points in time we want to get the derivative for

Returns the derivative values

Return type numpy.array

2.3 Generalized Logistic Growth

The formula behind the generalized exponential growth is very similar to the last one.

$$V(t) = A + \frac{K - A}{(1 + \exp(-B(t - M))^{\frac{1}{\nu}}}$$

The *Logistic Growth* is a special case of this model for $\nu = 1$ which breaks the symmetry of the curve. The five parameter that are needed to characterize one specific growth curve are stored in the following namedtuple:

class glucose_ts.models.generalized_logistics.**GLParameter**(*A*, *K*, *B*, *nu*, *M*)

The data structure represents all parameters that are needed for a generalized logistic function. The notations are identical with the Wikipedia artile with $Q = 1$ and $C = 1$ being set to fixed values. Please note that we are always dealing with decays in case of the glucose sensor. Therefore we role a lower and upper asymptote is flipped.

Parameters

- **A** (*float*) – the upper asymptote
- **K** (*float*) – the lower asymptote, for special cases the carrying capacity
- **B** (*float*) – the growth rate
- **nu** (*float*) – exponent for approximating the growth change
- **M** (*float*) – the location parameter of the logistic curve

So learn a specific parameter set from training data we use the following estimator.

class glucose_ts.models.**GeneralizedLogisticGrowth**(*parameter=None*, *gaussian_priors=None*, *std_model=None*, *time_horizon=None*)

The Estimator that learns the paramters of a generalized logistic growth model. It provides the classic maximum likelihood approach as well as the Bayesian approach maximum posterior.

fit(*time*, *labels*)

Fits a generalized logistic model to data.

Parameters

- **time** (*np.array*) – points in time or the independent variable here
- **labels** (*np.array*) – voltage measurements or the dependent variable

Returns the trained generalized logistic model

Return type *GeneralizedLogisticGrowth*

predict(*time*)

makes predictions by using the model internal parameters

Parameters **time** (*numpy.array*) – points in time we want to make predictions for

Returns the predictions

Return type numpy.array

time_derivative(*time*)

computes the derivative with respect to time using the model internal parameters

Parameters **time** (*numpy.array*) – points in time we want to get the derivative for

Returns the derivatives

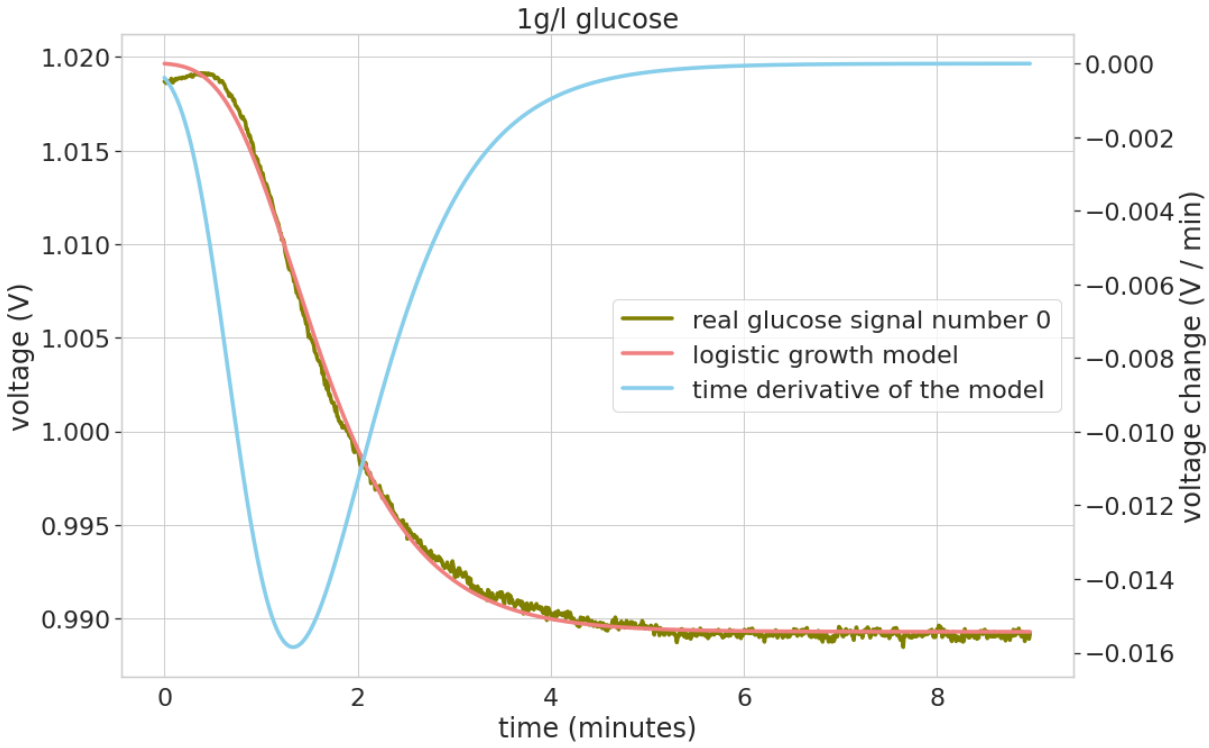
Return type numpy.array

CHAPTER

THREE

EXTRAPOLATION

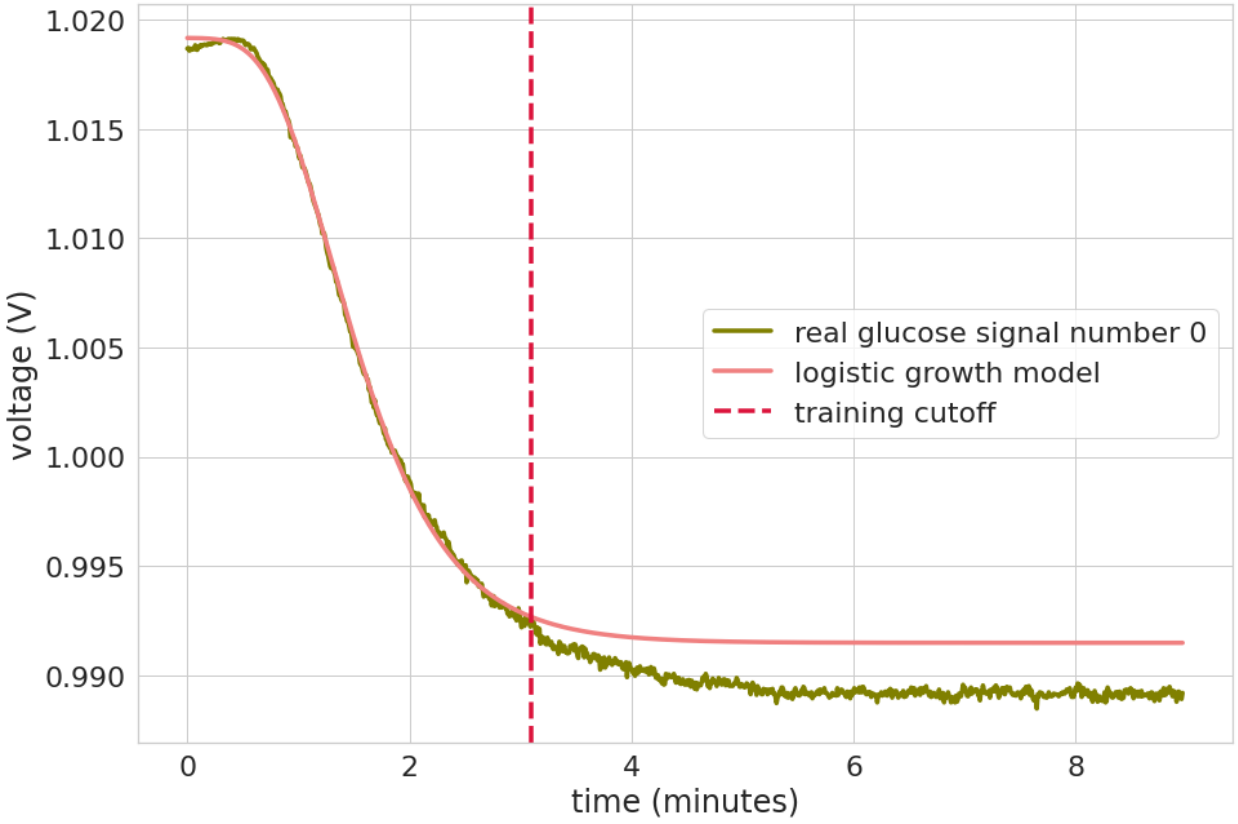
When we look at the whole time-series its rather easy to predict the final voltage of the time series, as you can observe here.



The green curve is the actual measurements of the glucose sensor. The orange curve is a generalized logistics growth model that is fitted to the sensor data.

3.1 Maximum Likelihood

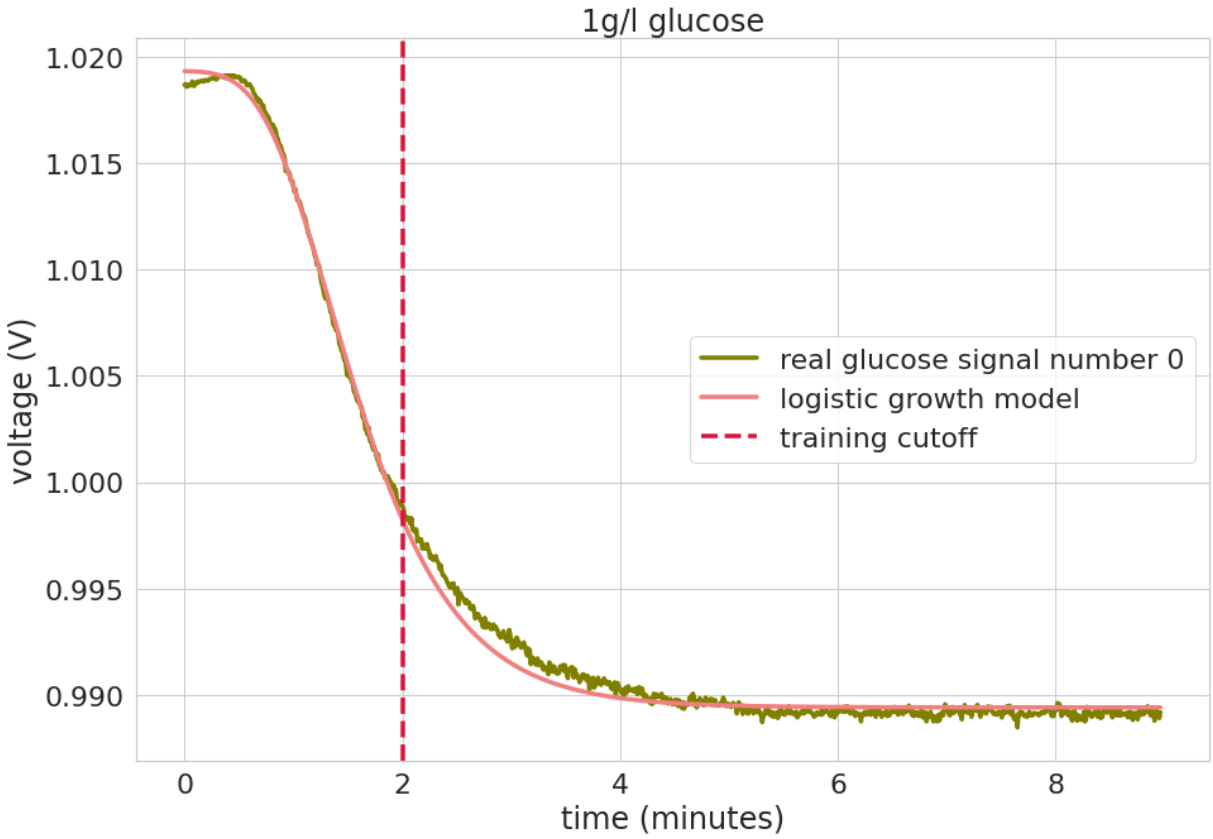
When we make a cut-off after 3 minutes the fit of the model to the data becomes worse



3.2 Maximum A Posteriori

When we take gaussian priors into account, we improve the curve fit. In particular we want the lower asymptote to be close to the final signal, as that is our prediction for the final voltage that correlate with the glucose concentration.

When we make a cut-off after 2 minutes the fit of the model to the data improves.


1g/l glucose
1.020
1.015
1.010
1.005
1.000
0.995
0.990
voltage (V)
0
2
4
6
8
time (minutes)
real glucose signal number 0
logistic growth model
training cutoff

CHAPTER

FOUR

EXPERIMENTS

4.1 Procedures to setup the Rasberry Pi

Just to mention it, the current setup is not well documented, but let us start here. When switching on the Raspberry Pi we need to ssh to the address `130.149.148.195`. Then we need to execute the following commands.

```
cd glucose_system
nohup python glucosesystem_server.py &
nohup python X_MeasureAllTheTime.py &
python glucosesystem_client_Setup.py
```

For the question about the port, you may choose `"/dev/ttyACM0"` If you try it in this order, it might work.

4.2 Glucose Measurement in Batch Cultivation

- 4 reactors
- 4 samples per hour

4.3 Callibration Standards

- putting everything in a thermal block and measure on 37 degrees
- initialization finished at 16:20
- basline on 37 degrees 1010 mV
- heat up the test solution 2 minutes on the thermal block
- start with 2.5 g/L
- **afterwards we go down and measure 1.25, 1, 0.75, 0.5, 0.4, 0.3, 0.2, 0.1,** 0 g / L
- **then we did a second round of measurements to check if the sensors behavior** is stable over time

4.4 Callibration Standards 9th of June

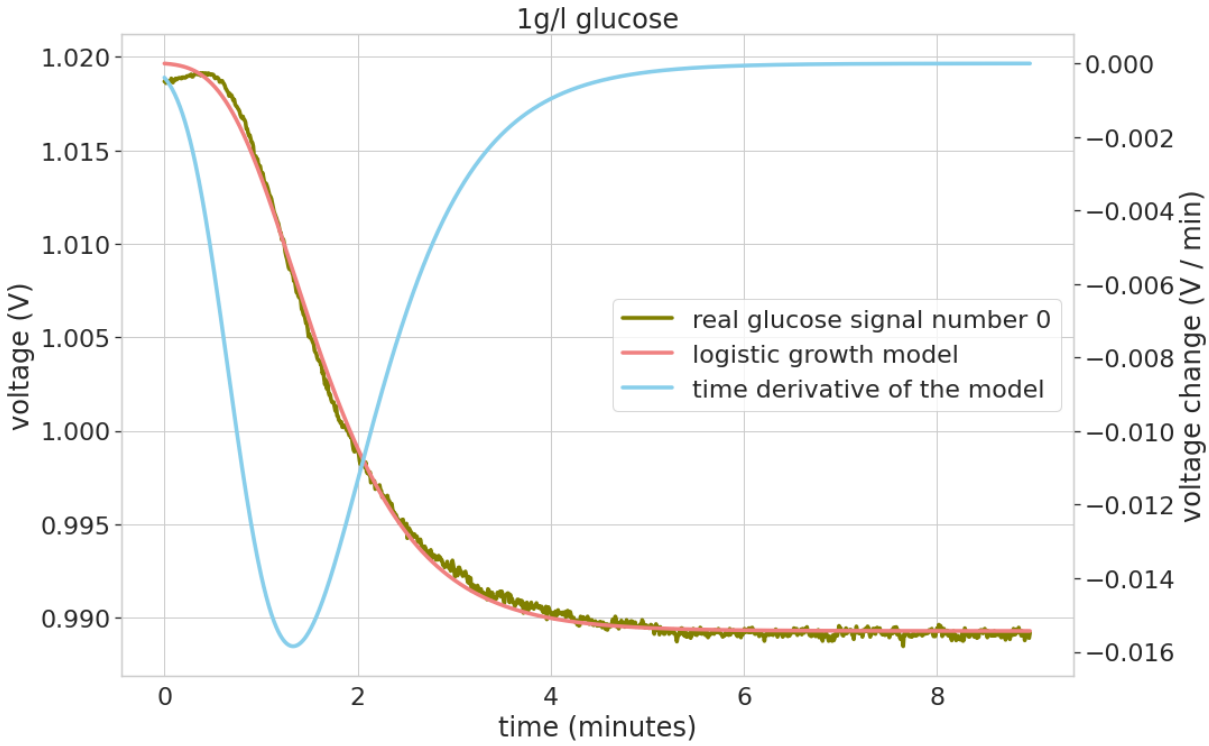
- started initializing at 11 a.m.
- put it in 2.5 g / L for one hour
- at 12 p.m we changed it to 5 g / L
- 37.2 degree celsius
- Start all time data `nohup python X_MeasureAllTheTime.py &`
- At 2:32 we are already down to 658 mV
- put sensor in buffer at 2:33 p.m.
- last standard is 1 g / L at 0:55

CHAPTER

FIVE

PURPOSE

The python package *glucose_ts* will help you to measure glucose concentrations closer to real time. When you measure a glucose concentration with a enzyme based sensor you normally have to wait for 5 - 10 minutes to get the glucose concentration. This projects aims for telling you this value earlier.



The green curve is the actual measurements of the glucose sensor. The orange curve is a generalized logistics growth model that is fitted to the sensor data. This model gets all the sensor measurements to fit a model. The idea would be to get the final voltage much earlier.

To ease the usage this package tries to follow the guidelines of scikit-learn estimators https://scikit-learn.org/stable/developers/develop.html. In practise the usage looks like this:

```
import glucose_ts

trained_model = glucose_ts.models.ExponentialDecay().fit(points_in_time, labels)
trained_model.predict(points_in_time)
```

5.1 Features

The package implements the following methods to explain and predict the glucose sensor voltage signal

- exponential decay
- logistic growth
- generalized logistic growth

5.2 Installation

Install the glucose package using *pip* by

```
cd glucose-prediction
pip install -e .
```

Here we assume that you want to install the package in editable mode, because you would like to contribute to it. This package is not available on PyPI, it might be in the future, though.

5.3 Contribute

- Issue Tracker: https://git.tu-berlin.de/ch.lange/glucose-prediction/-/issues
- Source Code: https://git.tu-berlin.de/ch.lange/glucose-prediction

5.4 Support

If you encounter issues, please let us know.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

INDEX