# epysurv

Rüdiger Busche and Justin Shenk

Feb 03, 2020

# CONTENTS

`epysurv` is a pythonic wrapper around the R surveillance package. It's main goal is to predict disease outbreaks, right now focusing on univariate count time series. `epsurv` operates on pandas `DataFrame`s and strives to implement a scikit-learn like API.

`epysurv` supports two problem formalizations of outbreak detection: time point classification and time series classification.

This documentation mainly explains the usage of `epysurv` and the ideas behind the problem formalizations. For more details about the algorithms have a look at the vignette of the R surveillance package or the literature references in the model docstrings.

This package was originally developed at the Robert Koch Institute in the Signale Project .

CHAPTER

ONE

# QUICKSTART

## 1.1 Installation

`epysurv` should be installed through conda

```
conda create -n epysurv
conda activate epysurv
conda install -c conda-forge epysurv
```

## 1.2 Demo

A quick tour to using `epysurv`.

CHAPTER

# TWO

# OUTBREAK DETECTION

Surveillance algorithms usually work on regular spaced aggregated time series of case counts. Let $\mathbf{x} = (x_1, \ldots x_T)$ be such a time series with entries at regularly spaced, discrete timepoints $t$. An entry $x_t$ of that time series is defined as the number of observed case counts in that time period.

## 2.1 Time Point Classification

Based on this we can view the problem as a *sequential supervised learning problem* [Die02], in which the sequence of counts is paired with a sequence of outbreak labels $(\mathbf{x}, \mathbf{y})$, with $\mathbf{x} = (x_1, \ldots, x_T), x_i \in \mathbb{N}_0$ and $y_i \in \mathbb{B}$. For each timepoint $t$ a boolean label is assigned, corresponding to whether there were outbreak cases present in the aggregation time interval. We call this problem *time point classification*. This is the standard formulation of common surveillance algorithms.

## 2.2 Time Series Classification

The time point formulation can be extended into a time series formulation by dividing the time series $\mathbf{x}$ into smaller time series and assigning the label of the last time point to the whole time series. Thus a data set $\{(\mathbf{x}_j, y_j)\}_{j=1}^{T}$ is obtained. This formulation is especially useful for incorporating reporting delay. That means that the information at time point $t = j$ can be quite different depending on whether $j$ is relatively recent, e.g. $j = T$ or already some time in the past. This is due to the fact that information arrives sometimes slowly in epidemiological surveillance systems. We call this problem formulation *time series classification*.

## 2.3 Models

As of now all models included in `epysurv` work on univariate time series of counts. Extensions to multivariate time series and incorporation of spatial data exist in the `R surveillance package`, but their inclusion is only planned for later releases.

The currently included models can be viewed as semi-supervised techniques from a machine learning or anomaly detection perspective [CBK09]. All models fit historic data, assuming that they represent the normal state of the system. Having fitted the data, an estimate for the case counts of the current week is computed. This estimate is compared to the number of cases reported in the current If the observed case count exceeds the expected number by some threshold, an alarm is raised. Most models in fact compute a predictive distribution for the estimated number of case counts and raise an alarm if the actual number exceeds a certain quantile of this distribution.

### 2.3.1 Window-based Approaches

The simplest form of outbreak detection algorithms are window-based approaches. For them the expectation for the current week is computed from a moving window of fixed size. For example the `EarsC1` algorithm, computes its predictive distribution based the mean and standard deviation of the last seven timepoints, using a normal distribution.

Because of the short time interval considered, these approaches are naturally insensitive against seasonality and trend. However, recent outbreaks can contaminate the data, reducing the sensitivity of the algorithms.

This category includes the Ears-family [HTST03], CDC [SWHK89] and the RKI [SSHohle16] algorithm.

### 2.3.2 GLM-based Approaches

Approaches based on Generalized Linear Models (GLMs) form a popular group of outbreak detection algorithms. They compute a predictive distribution for the current week based on fitting a GLM to previous data. An alarm is raised if the current observation is unlikely under the predictive distribution controlled by some $alpha$ value. Often Poisson or Negative Binomial models are used to do justice to the count nature of the data. Moreover, terms to accommodate seasonality and trend are often incorporated as well. GLM-based approaches included the classical Farrington algorithm [FABC96] and its more recent extension [NEF+13].

### 2.3.3 Cusum-based Approaches

Both window-based and GLM approaches have the downside that they only incorporate evidence from the current week. Larger outbreaks that build up slowly could therefore easily be missed. Cusum-based approaches are inspired by models from statistical process control~cite{Oakland2007} and incorporate evidence from previous timepoints. Instead of computing a predictive distribution, evidence that observed case counts do originate from an epidemic is accumulated until a certain threshold is exceeded and an alarm is raised. Then the sum is reset.

Cusum-based approaches include the Cusum [RLM99], generalized likelihood ratio methods based on Poisson:cite:*Hohle2006* or negative binomial distributions~cite{Hohle2008} and the OutbreakP method [FrisenASchioler09].

# CHAPTER THREE

# USER GUIDE

Using epysurv models should be straightforward if you are familiar with scikit-learn and pandas.

## 3.1 Data Format

Let's first consider the models in the *epysurv.models.timepoint package*. Each model has a fit and a predict method that takes a pandas.DataFrame representing an epidemiological count time series of the following form:

```
            n_cases  n_outbreak_cases
2004-01-05        0                 0
2004-01-12        0                 0
2004-01-19        2                 0
2004-01-26        2                 0
2004-02-02        1                 0
```

The data frame needs to have a regular DatetimeIndex and two columns containing case counts. n_cases represents the total number of cases observed and n_outbreak_cases the number of cases are labeled as belonging to an outbreak. Therefore n_cases should always be bigger or equal to n_outbreak_cases as there can not be more outbreak cases as cases in total. Note also that each row represents the number of cases observed in the **period** between the row's timepoint and the next timepoint. So in the above example the first row denotes that there were zero cases observed from 2004-01-05 up to 2004-01-11 inclusive.

## 3.2 Fitting

When passing the data frame to fit the outbreak cases are subtracted from the total cases to obtain the *in control* time series, i.e. the time series without outbreaks.

If you do not have any labeled outbreak data, but just the raw counts, the n_cases column will be taken as is under the assumption that your data is in fact *in control* data. A warning is still issued in this case.

## 3.3 Prediction

At prediction time only the total case counts are required. The data frame passed to predict needs to consist of observations that are spaced at the same regular time intervals as the training data. All data points should lie strictly in the future of the training data. The data frame returned is the original data augmented by an alarm column that indicated whether the model predicts an outbreak at that time point or not.

```
           n_cases  alarm
2011-01-03        1    0.0
2011-01-10        0    0.0
2011-01-17        3    0.0
2011-01-24        3    0.0
2011-01-31        3    0.0
```

## 3.4 Using Time Series Classification Models

For each each model in the *epysurv.models.timepoint package* there is a corresponding model in the *epysurv.models.timeseries package*. These models basically perform the same task, but make a binary prediction (alarm / no alarm) for an entire time series instead of just a single time point. See *Time Series Classification* for a more detailed discussion. Therefore, bot `fit` and `predict` take iterables of data frames described above and labels: `Iterable[Tuple[DataFrame, bool]]`. The label indicates whether the last time point of the time series is to be considered an outbreak. The `predict` method in this case only returns a time series of alarms.

CHAPTER

# FOUR

# EPYSURV

## 4.1 epysurv package

### 4.1.1 Subpackages

#### epysurv.data package

#### Submodules

#### epysurv.data.disease_loader module

`epysurv.data.disease_loader.`**`load_diseases`**(*path*)

#### epysurv.data.filter_combination module

**`class`** `epysurv.data.filter_combination.`**`FilterCombination`**(*disease: str*, *county: str*, *pathogen: str*, *data: pandas.core.frame.DataFrame*)

Bases: `object`

Representation of case records filtered by combination of county and pathogen.

**`disease`**
The disease from which the cases suffer.

**`county`**
The county in which the cases where reported.

**`pathogen`**
The pathogen subtype.

**`data`**
The case records.

**`expanding_windows`**(*min_len_in_weeks: int*, *split_years: epysurv.data.filter_combination.SplitYears*) → epysurv.data.filter_combination.TimeseriesClassificationData
Transform case records into expanding time series.

**Parameters**

- **`min_len_in_weeks`** – The minimum length of each time series.
- **`split_years`** – The years at which to split the data into train and test data.

**Returns** *Compound object of train and test data as generators and dataframes.*

**class** `epysurv.data.filter_combination.`**`SplitYears`**(*start: pandas._libs.tslibs.timestamps.Timestamp*, *middle: pandas._libs.tslibs.timestamps.Timestamp*, *end: pandas._libs.tslibs.timestamps.Timestamp*)

Bases: `object`

Data structure that holds the years data should be split into training and test set.

start to middle is the training data. middle to end is the test data.

**classmethod from_ts_input**(*start*, *middle*, *end*)
Create instance from inputs that are passed through `pd.Timestamp`.

**class** `epysurv.data.filter_combination.`**`TimeseriesClassificationData`**(*train_final*, *test_final*, *train_gen*, *test_gen*)

Bases: `tuple`

**property test_final**
Alias for field number 1

**property test_gen**
Alias for field number 3

**property train_final**
Alias for field number 0

**property train_gen**
Alias for field number 2

## epysurv.data.salmonella_data module

**class** `epysurv.data.salmonella_data.`**`TimeseriesClassificationData`**(*train*, *test*, *train_gen*, *test_gen*)

Bases: `tuple`

**property test**
Alias for field number 1

**property test_gen**
Alias for field number 3

**property train**
Alias for field number 0

**property train_gen**
Alias for field number 2

`epysurv.data.salmonella_data.`**`salmonella`**()
Count data from Salmonella newport in Germany.

epysurv.data.salmonella_data.**timeseries_classifaction_generator**(*train: pandas.core.frame.DataFrame*, *test: pandas.core.frame.DataFrame*, *offset_in_weeks: int*) → Tuple[Generator, Generator]

Turn a time point classification problem into a time series classification problem.

epysurv.data.salmonella_data.**timeseries_classifcation**(*train: pandas.core.frame.DataFrame*, *test: pandas.core.frame.DataFrame*, *offset_in_weeks: int*) → epysurv.data.salmonella_data.TimeseriesClassificationDat

Convert standard timeseries for usage in time series classification.

## epysurv.data.utils module

epysurv.data.utils.**timedelta_weeks**(*weeks: int*)

## Module contents

Module for handling data transformation and example data.

epysurv.data.**load_diseases**(*path*)

**class** epysurv.data.**TimeseriesClassificationData**(*train*, *test*, *train_gen*, *test_gen*)

Bases: `tuple`

**property test**

Alias for field number 1

**property test_gen**

Alias for field number 3

**property train**

Alias for field number 0

**property train_gen**

Alias for field number 2

epysurv.data.**salmonella**()

Count data from Salmonella newport in Germany.

epysurv.data.**timeseries_classifaction_generator**(*train: pandas.core.frame.DataFrame*, *test: pandas.core.frame.DataFrame*, *offset_in_weeks: int*) → Tuple[Generator, Generator]

Turn a time point classification problem into a time series classification problem.

epysurv.data.**timeseries_classifcation**(*train: pandas.core.frame.DataFrame*, *test: pandas.core.frame.DataFrame*, *offset_in_weeks: int*) → epysurv.data.salmonella_data.TimeseriesClassificationData

Convert standard timeseries for usage in time series classification.

## epysurv.metrics package

### Submodules

### epysurv.metrics.outbreak_detection module

`epysurv.metrics.outbreak_detection.`**`ghozzi_case_score`**(*prediction_result: pandas.core.frame.DataFrame*) → float

Evalutes the performance of an outbreak detection.

Using the following formula: sum(p[t] * c[t] - (1 - p[t]) * c[t] - (p[t] != o[t]) * e[t] for t in timeseries) / sum(c) p: alarm c: count of outbreak cases o: outbreak e: endemic cases

**Parameters** **`prediction_result`** – Dataframe containing the columns "alarm", "outbreak" and "outbreak_cases"

**Returns** *A maximum score of 1.*

`epysurv.metrics.outbreak_detection.`**`ghozzi_score`**(*prediction_result: pandas.core.frame.DataFrame*) → float

Evalutes the performance of an outbreak detection.

Using the following formula: sum(p[t] * c[t] - (1 - p[t]) * c[t] - (p[t] != o[t]) * mean(c) for t in timeseries) / sum(c) p: alarm c: count of outbreak cases o: outbreak

**Parameters** **`prediction_result`** – Dataframe containing the columns "alarm", "outbreak" and "outbreak_cases"

**Returns** *A maximum score of 1.*

### Module contents

`epysurv.metrics.`**`ghozzi_score`**(*prediction_result: pandas.core.frame.DataFrame*) → float

Evalutes the performance of an outbreak detection.

Using the following formula: sum(p[t] * c[t] - (1 - p[t]) * c[t] - (p[t] != o[t]) * mean(c) for t in timeseries) / sum(c) p: alarm c: count of outbreak cases o: outbreak

**Parameters** **`prediction_result`** – Dataframe containing the columns "alarm", "outbreak" and "outbreak_cases"

**Returns** *A maximum score of 1.*

## epysurv.models package

### Subpackages

### epysurv.models.timepoint package

### Submodules

### epysurv.models.timepoint.bayes module

**class** `epysurv.models.timepoint.bayes.`**`Bayes`**(*years_back: int = 0*, *window_half_width: int = 6*, *include_recent_year: bool = True*, *alpha: float = 0.05*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

Evaluation of timepoints with the Bayes subsystem.

**`years_back`**
How many years back in time to include when forming the base counts.

**`window_half_width`**
Number of weeks to include before and after the current week in each year.

**`include_recent_year`**
is a boolean to decide if the year of timePoint also contributes w reference values.

**`alpha`**
The parameter alpha is the (1 )-quantile to use in order to calculate the upper threshold. As default b, w, actY are set for the Bayes 1 system with alpha=0.05.

#### References

### epysurv.models.timepoint.boda module

**class** `epysurv.models.timepoint.boda.`**`Boda`**(*trend: bool = False*, *season: bool = False*, *prior: str = 'iid'*, *alpha: float = 0.05*, *mc_munu: int = 100*, *mc_y: int = 10*, *quantile_method: str = 'MM'*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

The Boda model.

**`trend`**
Boolean indicating whether a linear trend term should be included in the model for the expectation the log-scale

**`season`**
Boolean to indicate whether a cyclic spline should be included.

**`prior`**
Either of “iid”, “rw1” or “rw2”.

**`alpha`**
The threshold for declaring an observed count as an aberration is the (1 ) · 100% quantile of the predictive posterior.

**`mc_munu`**

**`mc_y`**
Number of samples of y to generate for each pair of the mean and size parameter. A total of mc.munu × mc.y samples are generated.

**`sampling_method`**
Should one sample from the parameters joint distribution (joint) or from their respective marginal posterior distribution (marginals)

**`quantile_method`**
Either of “MC” or “MM”. Indicates how to compute the quantile based on the posterior distribution (no

matter the inference method): either by sampling mc.munu values from the posterior distribution of the parameters and then for each sampled parameters vector sampling mc.y response values so that one gets a vector of response values based on which one computes an empirical quantile (MC method, as explained in Manitz and Höhle 2013); or by sampling mc_munu from the posterior distribution of the parameters and then compute the quantile of the mixture distribution using bisectioning, which is faster.

## epysurv.models.timepoint.cdc module

**class** `epysurv.models.timepoint.cdc.`**CDC**(*years_back: int = 5*, *window_half_width: int = 1*, *alpha: float = 0.001*)

Bases: `epysurv.models.timepoint._base.DisProgBasedAlgorithm`

The CDC model.

**years_back**
How many years back in time to include when forming the base counts.

**window_half_width**
Number of weeks to include before and after the current week in each year.

**alpha**
An approximate (two-sided)(1  ) prediction interval is calculated.

### References

## epysurv.models.timepoint.cusum module

**class** `epysurv.models.timepoint.cusum.`**Cusum**(*reference_value: float = 1.04*, *decision_boundary: float = 2.26*, *expected_numbers_method: str = 'mean'*, *transform: str = 'standard'*, *negbin_alpha: float = 0.1*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

The Cusum model.

**reference_value**

**decision_boundary**

**expected_numbers_method**
How to determine the expected number of cases – the following arguments are possible: {"glm", "mean"}.

**mean** Use the mean of all data points passed to `fit`.

**glm** Fit a glm to the data ponts passed to `fit`.

**transform**
One of the following transformations (warning: Anscombe and NegBin transformations are experimental) - standard standardized variables z1 (based on asymptotic normality) - This is the default. - rossi standardized variables z3 as proposed by Rossi - anscombe anscombe residuals – experimental - anscombe2nd anscombe residuals as in Pierce and Schafer (1986) based on 2nd order approximation of E(X) – experimental - pearsonNegBin compute Pearson residuals for NegBin – experimental - anscombeNegBin anscombe residuals for NegBin – experimental - `"none"` no transformation

**negbin_alpha**
Parameter of the negative binomial distribution, such that the variance is $m + \cdot m2$.

### References

## epysurv.models.timepoint.ears module

**class** `epysurv.models.timepoint.ears.`**`EarsC1`**(*alpha: float = 0.001*, *baseline: int = 7*, *min_sigma: float = 0*)

Bases: `epysurv.models.timepoint.ears._EarsBase`

Computes a threshold for the number of counts based on values from the recent past.

This is then compared to the observed number of counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised. This method is especially useful for data without many historic values, since it only needs counts from the recent past.

**`alpha`**
An approximate (two-sided)(1  ) prediction interval is calculated.

**`baseline`**
How many time points to use for calculating the baseline.

**`min_sigma`**
If minSigma is higher than 0, the quantity zAlpha * minSigma is then the alerting threshold if the baseline is zero.

### References

**class** `epysurv.models.timepoint.ears.`**`EarsC2`**(*alpha: float = 0.001*, *baseline: int = 7*, *min_sigma: float = 0*)

Bases: `epysurv.models.timepoint.ears._EarsBase`

Computes a threshold for the number of counts based on values from the recent past.

This is then compared to the observed number of counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised. This method is especially useful for data without many historic values, since it only needs counts from the recent past.

**`alpha`**
An approximate (two-sided)(1  ) prediction interval is calculated.

**`baseline`**
How many time points to use for calculating the baseline.

**`min_sigma`**
If minSigma is higher than 0, zAlpha * minSigma is then the alerting threshold if the baseline is zero.

### References

**class** `epysurv.models.timepoint.ears.`**`EarsC3`**(*alpha: float = 0.001*, *baseline: int = 7*, *min_sigma: float = 0*)

Bases: `epysurv.models.timepoint.ears._EarsBase`

The EarsC3 model.

Computes a threshold for the number of counts based on values from the recent past. This is then compared to the observed number of counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised. This method is especially useful for data without many historic values, since it only needs counts from the recent past.

**alpha**
An approximate (two-sided)(1  ) prediction interval is calculated.

**baseline**
How many time points to use for calculating the baseline.

### References

## epysurv.models.timepoint.farrington module

**class** `epysurv.models.timepoint.farrington.`**`Farrington`**(*years_back: int = 3*, *window_half_width: int = 3*, *reweight: bool = True*, *alpha: float = 0.01*, *trend: bool = True*, *past_period_cutoff: int = 4*, *min_cases_in_past_periods: int = 5*, *power_transform: str = '2/3'*)

Bases: `epysurv.models.timepoint._base.DisProgBasedAlgorithm`

The Farrington algorithm.

For each time point uses a GLM to predict the number of counts according to the procedure by Farrington et al. (1996). This is then compared to the observed number of counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised.

**years_back**
How many years back in time to include when forming the base counts.

**window_half_width**
Number of weeks to include before and after the current week in each year.

**reweight**
Boolean specifying whether to perform reweighting step.

**alpha**
An approximate (two-sided) (1  ) prediction interval is calculated.

**trend**
Boolean indicating whether a trend should be included and kept in case the conditions in the Farrington et. al. paper are met (see the results). If false then no trend is fit.

**past_period_cutoff**
Periods considered for suppression of low case numbers.

**min_cases_in_past_periods**
The minimal number of cases in past periods such that an outbreak is considered.

**power_transform**
Power transformation to apply to the data if the threshold is to be computed with the method described in Farrington et al. (1996). Use either - “2/3” for skewness correction (Default) - “1/2” for variance stabilizing transformation - “none” for no transformation.

### References

**class** `epysurv.models.timepoint.farrington.`**`FarringtonFlexible`**(*years_back: int = 3*, *window_half_width: int = 3*, *reweight: bool = True*, *weights_threshold: float = 2.58*, *alpha: float = 0.01*, *trend: bool = True*, *trend_threshold: float = 0.05*, *past_period_cutoff: int = 4*, *min_cases_in_past_periods: int = 5*, *power_transform: str = '2/3'*, *past_weeks_not_included: int = 26*, *threshold_method: str = 'delta'*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

The extended Farrington algorithm.

For each time point uses a Poisson GLM with overdispersion to predict an upper bound on the number of counts according to the procedure by Farrington et al. (1996) and by Noufaily et al. (2012). This bound is then compared to the observed number of counts. If the observation is above the bound, then an alarm is raised.

**`years_back`**
How many years back in time to include when forming the base counts.

**`window_half_width`**
Number of weeks to include before and after the current week in each year.

**`reweight`**
Boolean specifying whether to perform reweighting step.

**`weights_threshold`**
Defines the threshold for reweighting past outbreaks using the Anscombe residuals (1 in the original method, 2.58 advised in the improved method).

**`alpha`**
An approximate (one-sided) (1  ) · 100% prediction interval is calculated unlike the original method where it was a two-sided interval. The upper limit of this interval i.e. the (1  ) · 100% quantile serves as an upperbound.

**`trend`**
Boolean indicating whether a trend should be included and kept in case the conditions in the Farrington et. al. paper are met (see the results). If false then NO trend is fit.

**`trend_threshold`**
Threshold for deciding whether to keep trend in the model (0.05 in the original method, 1 advised in the improved method).

**`past_period_cutoff`**
Periods considered for suppression of low case numbers.

**`min_cases_in_past_periods`**
The minimal number of cases in past periods such that an outbreak is considered. power_transform Power transformation to apply to the data if the threshold is to be computed with the method described in Farrington et al. (1996). Use either - "2/3" for skewness correction (Default) - "1/2" for variance stabilizing transformation - "none" for no transformation.

**`past_weeks_not_included`**
Number of past weeks to ignore in the calculation.

**`threshold_method`**
Method to be used to derive the upperbound. Options are - "delta" for the method described in Farrington et al. (1996) - "Noufaily" for the method described in Noufaily et al. (2012) - "muan" for the method extended from Noufaily et al. (2012)

#### References

### epysurv.models.timepoint.glr module

Count data regression charts for the monitoring of surveillance time series.

Method as proposed by Höhle and Paul (2008). The implementation is described in Salmon et al. (2016).

**class** `epysurv.models.timepoint.glr.`**`GLRNegativeBinomial`**(*alpha: float = 0, glr_test_threshold: int = 5, m: int = -1, change: str = 'intercept', direction: Union[Tuple[str, str], Tuple[str]] = ('inc', 'dec'), upperbound_statistic: str = 'cases', x_max: float = 10000.0*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

Generalized likelihood ratio algorithm using negative binomial distribution.

**`alpha`**
The (known) dispersion parameter of the negative binomial distribution, i.e. the parametrization of the negative binomial is such that the variance is mean + alpha mean2. Note: This parametrization is the inverse of the shape parametrization used in R – for example in dnbinom and glr.nb. Hence, if alpha=0 then the negative binomial distribution boils down to the Poisson distribution and a call of algo.glrnb is equivalent to a call to algo.glrpois. If alpha=NULL the parameter is calculated as part of the in-control estimation. However, the parameter is estimated only once from the first fit. Subsequent fittings are only for the parameters of the linear predictor with alpha fixed.

**`glr_test_threshold`**
Threshold in the GLR test, i.e. c.

**`m`**
Number of time instances back in time in the window-limited approach, i.e. the last value considered is max(1, n m). To always look back until the first observation use -1.

**`change`**
A string specifying the type of the alternative. The two choices are "intercept" and "epi".

**`direction`**
Specifying the direction of testing in GLR scheme. - ("inc",) only increases in x are considered in the GLR-statistic - ("dec",) only decreases are regarded - ("inc", "dec") both increases and decreases are regarded.

**upperbound_statistic**
A string specifying the type of upperbound-statistic that is returned. - "cases" for the number of cases that would have been necessary to produce an alarm - "value" for the GLR-statistic

**x_max**
Maximum value to try for x to see if this is the upperbound number of cases before sounding an alarm (Default: 1e4). This only applies only when `upperbound_statistic == "cases"`.

### References

**class** `epysurv.models.timepoint.glr.`**GLRPoisson**(*glr_test_threshold: int = 5, m: int = -1, change: str = 'intercept', direction: Union[Tuple[str, str], Tuple[str]] = ('inc', 'dec'), upperbound_statistic: str = 'cases'*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

Generalized likelihood ratio algorithm using Poisson distribution.

**glr_test_threshold**
Threshold in the GLR test, i.e. c.

**m**
Number of time instances back in time in the window-limited approach, i.e. the last value considered is max(1, n m). To always look back until the first observation use -1.

**change**
A string specifying the type of the alternative. The two choices are "intercept" and "epi".

**direction**
Specifying the direction of testing in GLR scheme. - ("inc",) only increases in x are considered in the GLR-statistic - ("dec",) only decreases are regarded - ("inc", "dec") both increases and decreases are regarded.

**upperbound_statistic**
a string specifying the type of upperbound-statistic that is returned. With "cases" the number of cases that would have been necessary to produce an alarm or with "value" the GLR-statistic is computed.

### References

**change = 'intercept'**
a string specifying the type of the alternative. Currently the two choices are intercept and epi. See the SFB Discussion Paper 500 for details

**direction = ('inc', 'dec')**
Specifying the direction of testing in GLR scheme. With "inc" only increases in x are considered in the GLR-statistic, with "dec" decreases are regarded.

**glr_test_threshold = 5**
threshold in the GLR test, i.e. c.

**m = -1**
number of time instances back in time in the window-limited approach, i.e. the last value considered is max 1, n M. To always look back until the first observation use M=-1.

**upperbound_statistic = 'cases'**
a string specifying the type of upperbound-statistic that is returned. With "cases" the number of cases that would have been necessary to produce an alarm or with "value" the GLR-statistic is computed (see below)

### epysurv.models.timepoint.hmm module

**class** `epysurv.models.timepoint.hmm.`**`HMM`**(*n_observations: int = -1*, *n_hidden_states: int = 2*, *trend: bool = True*, *n_harmonics: int = 1*, *equal_covariate_effects: bool = False*)

Bases: `epysurv.models.timepoint._base.DisProgBasedAlgorithm`

Hidden Markov model for outbreak detection.

**`n_observations`**
number of observations back in time to use for fitting the HMM (including the current observation). Reasonable values are a multiple of observations per year, the default is -1, which means to use all possible values - for long series this might take very long time!

**`n_hidden_states`**
number of hidden states in the HMM – the typical choice is 2. The initial rates are set such that the noStates'th state is the one having the highest rate. In other words: this state is considered the outbreak state.

**`trend`**
The two choices are "intercept" and "epi".

**`n_harmonics`**
Number of harmonic waves to include in the linear predictor.

**`equal_covariate_effects`**
If set then all covariate effects parameters are equal for the states.

#### References

### epysurv.models.timepoint.outbreak_p module

**class** `epysurv.models.timepoint.outbreak_p.`**`OutbreakP`**(*threshold: int = 100*, *upperbound_statistic: str = 'cases'*, *max_upperbound_cases: int = 100000*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

The OutbreakP model.

**`threshold`**
The threshold value. Once the outbreak statistic is above this threshold an alarm is sounded.

**`upperbound_statistic`**
A string specifying the type of upperbound-statistic that is returned. With "cases" the number of cases that would have been necessary to produce an alarm (NNBA) or with "value" the outbreakP-statistic is computed.

**`max_upperbound_cases`**
Upperbound when numerically searching for NNBA. Default is 1e5.

#### References

### epysurv.models.timepoint.rki module

**class** epysurv.models.timepoint.rki.**RKI**(*years_back: int = 0*, *window_half_width: int = 6*, *include_recent_year: bool = True*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

The old algorithm from the Robert Koch Institute.

**years_back**
: How many years back in time to include when forming the base counts.

**window_half_width**
: Number of weeks to include before and after the current week in each year.

**include_recent_year**
: Is a boolean to decide if the year of timePoint also contributes w reference values.

### Module contents

**class** epysurv.models.timepoint.**Bayes**(*years_back: int = 0*, *window_half_width: int = 6*, *include_recent_year: bool = True*, *alpha: float = 0.05*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

Evaluation of timepoints with the Bayes subsystem.

**years_back**
: How many years back in time to include when forming the base counts.

**window_half_width**
: Number of weeks to include before and after the current week in each year.

**include_recent_year**
: is a boolean to decide if the year of timePoint also contributes w reference values.

**alpha**
: The parameter alpha is the (1 )-quantile to use in order to calculate the upper threshold. As default b, w, actY are set for the Bayes 1 system with alpha=0.05.

#### References

**class** epysurv.models.timepoint.**Boda**(*trend: bool = False*, *season: bool = False*, *prior: str = 'iid'*, *alpha: float = 0.05*, *mc_munu: int = 100*, *mc_y: int = 10*, *quantile_method: str = 'MM'*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

The Boda model.

**trend**
: Boolean indicating whether a linear trend term should be included in the model for the expectation the log-scale

**season**
: Boolean to indicate whether a cyclic spline should be included.

**prior**
: Either of "iid", "rw1" or "rw2".

**alpha**
: The threshold for declaring an observed count as an aberration is the (1 ) · 100% quantile of the predictive posterior.

**mc_munu**

**mc_y**
Number of samples of y to generate for each pair of the mean and size parameter. A total of mc.munu × mc.y samples are generated.

**sampling_method**
Should one sample from the parameters joint distribution (joint) or from their respective marginal posterior distribution (marginals)

**quantile_method**
Either of "MC" or "MM". Indicates how to compute the quantile based on the posterior distribution (no matter the inference method): either by sampling mc.munu values from the posterior distribution of the parameters and then for each sampled parameters vector sampling mc.y response values so that one gets a vector of response values based on which one computes an empirical quantile (MC method, as explained in Manitz and Höhle 2013); or by sampling mc_munu from the posterior distribution of the parameters and then compute the quantile of the mixture distribution using bisectioning, which is faster.

**class** `epysurv.models.timepoint.`**`CDC`**(*years_back: int = 5*, *window_half_width: int = 1*, *alpha: float = 0.001*)
Bases: `epysurv.models.timepoint._base.DisProgBasedAlgorithm`

The CDC model.

**years_back**
How many years back in time to include when forming the base counts.

**window_half_width**
Number of weeks to include before and after the current week in each year.

**alpha**
An approximate (two-sided)(1  ) prediction interval is calculated.

## References

**class** `epysurv.models.timepoint.`**`Cusum`**(*reference_value: float = 1.04*, *decision_boundary: float = 2.26*, *expected_numbers_method: str = 'mean'*, *transform: str = 'standard'*, *negbin_alpha: float = 0.1*)
Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

The Cusum model.

**reference_value**

**decision_boundary**

**expected_numbers_method**
How to determine the expected number of cases – the following arguments are possible: {"glm", "mean"}.

**mean** Use the mean of all data points passed to `fit`.

**glm** Fit a glm to the data ponts passed to `fit`.

**transform**
One of the following transformations (warning: Anscombe and NegBin transformations are experimental) - standard standardized variables z1 (based on asymptotic normality) - This is the default. - rossi standardized variables z3 as proposed by Rossi - anscombe anscombe residuals – experimental - anscombe2nd anscombe residuals as in Pierce and Schafer (1986) based on 2nd order approximation of E(X) – experimental - pearsonNegBin compute Pearson residuals for NegBin – experimental - anscombeNegBin anscombe residuals for NegBin – experimental - `"none"` no transformation

**negbin_alpha**
Parameter of the negative binomial distribution, such that the variance is $m + \cdot m2$.

### References

**class** epysurv.models.timepoint.**EarsC1**(*alpha: float = 0.001*, *baseline: int = 7*, *min_sigma: float = 0*)
Bases: `epysurv.models.timepoint.ears._EarsBase`

Computes a threshold for the number of counts based on values from the recent past.

This is then compared to the observed number of counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised. This method is especially useful for data without many historic values, since it only needs counts from the recent past.

**alpha**
An approximate (two-sided)(1  ) prediction interval is calculated.

**baseline**
How many time points to use for calculating the baseline.

**min_sigma**
If minSigma is higher than 0, the quantity zAlpha * minSigma is then the alerting threshold if the baseline is zero.

### References

**class** epysurv.models.timepoint.**EarsC2**(*alpha: float = 0.001*, *baseline: int = 7*, *min_sigma: float = 0*)
Bases: `epysurv.models.timepoint.ears._EarsBase`

Computes a threshold for the number of counts based on values from the recent past.

This is then compared to the observed number of counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised. This method is especially useful for data without many historic values, since it only needs counts from the recent past.

**alpha**
An approximate (two-sided)(1  ) prediction interval is calculated.

**baseline**
How many time points to use for calculating the baseline.

**min_sigma**
If minSigma is higher than 0, zAlpha * minSigma is then the alerting threshold if the baseline is zero.

### References

**class** epysurv.models.timepoint.**EarsC3**(*alpha: float = 0.001*, *baseline: int = 7*, *min_sigma: float = 0*)
Bases: `epysurv.models.timepoint.ears._EarsBase`

The EarsC3 model.

Computes a threshold for the number of counts based on values from the recent past. This is then compared to the observed number of counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised. This method is especially useful for data without many historic values, since it only needs counts from the recent past.

**alpha**
: An approximate (two-sided)(1 ) prediction interval is calculated.

**baseline**
: How many time points to use for calculating the baseline.

## References

**class** `epysurv.models.timepoint.`**`FarringtonFlexible`**(*years_back: int = 3, window_half_width: int = 3, reweight: bool = True, weights_threshold: float = 2.58, alpha: float = 0.01, trend: bool = True, trend_threshold: float = 0.05, past_period_cutoff: int = 4, min_cases_in_past_periods: int = 5, power_transform: str = '2/3', past_weeks_not_included: int = 26, threshold_method: str = 'delta'*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

The extended Farrington algorithm.

For each time point uses a Poisson GLM with overdispersion to predict an upper bound on the number of counts according to the procedure by Farrington et al. (1996) and by Noufaily et al. (2012). This bound is then compared to the observed number of counts. If the observation is above the bound, then an alarm is raised.

**years_back**
: How many years back in time to include when forming the base counts.

**window_half_width**
: Number of weeks to include before and after the current week in each year.

**reweight**
: Boolean specifying whether to perform reweighting step.

**weights_threshold**
: Defines the threshold for reweighting past outbreaks using the Anscombe residuals (1 in the original method, 2.58 advised in the improved method).

**alpha**
: An approximate (one-sided) (1 ) · 100% prediction interval is calculated unlike the original method where it was a two-sided interval. The upper limit of this interval i.e. the (1 ) · 100% quantile serves as an upperbound.

**trend**
: Boolean indicating whether a trend should be included and kept in case the conditions in the Farrington et. al. paper are met (see the results). If false then NO trend is fit.

**trend_threshold**
: Threshold for deciding whether to keep trend in the model (0.05 in the original method, 1 advised in the improved method).

**past_period_cutoff**
: Periods considered for suppression of low case numbers.

**min_cases_in_past_periods**
: The minimal number of cases in past periods such that an outbreak is considered. power_transform Power

transformation to apply to the data if the threshold is to be computed with the method described in Farrington et al. (1996). Use either - "2/3" for skewness correction (Default) - "1/2" for variance stabilizing transformation - "none" for no transformation.

**past_weeks_not_included**
Number of past weeks to ignore in the calculation.

**threshold_method**
Method to be used to derive the upperbound. Options are - "delta" for the method described in Farrington et al. (1996) - "Noufaily" for the method described in Noufaily et al. (2012) - "muan" for the method extended from Noufaily et al. (2012)

### References

**class** `epysurv.models.timepoint.`**`Farrington`**(*years_back: int = 3*, *window_half_width: int = 3*, *reweight: bool = True*, *alpha: float = 0.01*, *trend: bool = True*, *past_period_cutoff: int = 4*, *min_cases_in_past_periods: int = 5*, *power_transform: str = '2/3'*)

Bases: `epysurv.models.timepoint._base.DisProgBasedAlgorithm`

The Farrington algorithm.

For each time point uses a GLM to predict the number of counts according to the procedure by Farrington et al. (1996). This is then compared to the observed number of counts. If the observation is above a specific quantile of the prediction interval, then an alarm is raised.

**years_back**
How many years back in time to include when forming the base counts.

**window_half_width**
Number of weeks to include before and after the current week in each year.

**reweight**
Boolean specifying whether to perform reweighting step.

**alpha**
An approximate (two-sided) (1  ) prediction interval is calculated.

**trend**
Boolean indicating whether a trend should be included and kept in case the conditions in the Farrington et. al. paper are met (see the results). If false then no trend is fit.

**past_period_cutoff**
Periods considered for suppression of low case numbers.

**min_cases_in_past_periods**
The minimal number of cases in past periods such that an outbreak is considered.

**power_transform**
Power transformation to apply to the data if the threshold is to be computed with the method described in Farrington et al. (1996). Use either - "2/3" for skewness correction (Default) - "1/2" for variance stabilizing transformation - "none" for no transformation.

### References

**class** epysurv.models.timepoint.**GLRNegativeBinomial**(*alpha: float = 0, glr_test_threshold: int = 5, m: int = -1, change: str = 'intercept', direction: Union[Tuple[str, str], Tuple[str]] = ('inc', 'dec'), upperbound_statistic: str = 'cases', x_max: float = 10000.0*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

Generalized likelihood ratio algorithm using negative binomial distribution.

**alpha**
The (known) dispersion parameter of the negative binomial distribution, i.e. the parametrization of the negative binomial is such that the variance is mean + alpha mean2. Note: This parametrization is the inverse of the shape parametrization used in R – for example in dnbinom and glr.nb. Hence, if alpha=0 then the negative binomial distribution boils down to the Poisson distribution and a call of algo.glrnb is equivalent to a call to algo.glrpois. If alpha=NULL the parameter is calculated as part of the in-control estimation. However, the parameter is estimated only once from the first fit. Subsequent fittings are only for the parameters of the linear predictor with alpha fixed.

**glr_test_threshold**
Threshold in the GLR test, i.e. c.

**m**
Number of time instances back in time in the window-limited approach, i.e. the last value considered is max(1, n m). To always look back until the first observation use -1.

**change**
A string specifying the type of the alternative. The two choices are "intercept" and "epi".

**direction**
Specifying the direction of testing in GLR scheme. - ("inc",) only increases in x are considered in the GLR-statistic - ("dec",) only decreases are regarded - ("inc", "dec") both increases and decreases are regarded.

**upperbound_statistic**
A string specifying the type of upperbound-statistic that is returned. - "cases" for the number of cases that would have been necessary to produce an alarm - "value" for the GLR-statistic

**x_max**
Maximum value to try for x to see if this is the upperbound number of cases before sounding an alarm (Default: 1e4). This only applies only when `upperbound_statistic == "cases"`.

### References

**class** epysurv.models.timepoint.**GLRPoisson**(*glr_test_threshold: int = 5, m: int = -1, change: str = 'intercept', direction: Union[Tuple[str, str], Tuple[str]] = ('inc', 'dec'), upperbound_statistic: str = 'cases'*)

Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

Generalized likelihood ratio algorithm using Poisson distribution.

**glr_test_threshold**
Threshold in the GLR test, i.e. c.

**m**
: Number of time instances back in time in the window-limited approach, i.e. the last value considered is max(1, n m). To always look back until the first observation use -1.

**change**
: A string specifying the type of the alternative. The two choices are "intercept" and "epi".

**direction**
: Specifying the direction of testing in GLR scheme. - ("inc",) only increases in x are considered in the GLR-statistic - ("dec",) only decreases are regarded - ("inc", "dec") both increases and decreases are regarded.

**upperbound_statistic**
: a string specifying the type of upperbound-statistic that is returned. With "cases" the number of cases that would have been necessary to produce an alarm or with "value" the GLR-statistic is computed.

### References

**change = 'intercept'**
: a string specifying the type of the alternative. Currently the two choices are intercept and epi. See the SFB Discussion Paper 500 for details

**direction = ('inc', 'dec')**
: Specifying the direction of testing in GLR scheme. With "inc" only increases in x are considered in the GLR-statistic, with "dec" decreases are regarded.

**glr_test_threshold = 5**
: threshold in the GLR test, i.e. c.

**m = -1**
: number of time instances back in time in the window-limited approach, i.e. the last value considered is max 1, n M. To always look back until the first observation use M=-1.

**upperbound_statistic = 'cases'**
: a string specifying the type of upperbound-statistic that is returned. With "cases" the number of cases that would have been necessary to produce an alarm or with "value" the GLR-statistic is computed (see below)

**class** `epysurv.models.timepoint.`**HMM**(*n_observations: int = -1, n_hidden_states: int = 2, trend: bool = True, n_harmonics: int = 1, equal_covariate_effects: bool = False*)

Bases: `epysurv.models.timepoint._base.DisProgBasedAlgorithm`

Hidden Markov model for outbreak detection.

**n_observations**
: number of observations back in time to use for fitting the HMM (including the current observation). Reasonable values are a multiple of observations per year, the default is -1, which means to use all possible values - for long series this might take very long time!

**n_hidden_states**
: number of hidden states in the HMM – the typical choice is 2. The initial rates are set such that the noStates'th state is the one having the highest rate. In other words: this state is considered the outbreak state.

**trend**
: The two choices are "intercept" and "epi".

**n_harmonics**
: Number of harmonic waves to include in the linear predictor.

**equal_covariate_effects**
If set then all covariate effects parameters are equal for the states.

#### References

**class** `epysurv.models.timepoint.`**OutbreakP**(*threshold: int = 100*, *upperbound_statistic: str = 'cases'*, *max_upperbound_cases: int = 100000*)
Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

The OutbreakP model.

**threshold**
The threshold value. Once the outbreak statistic is above this threshold an alarm is sounded.

**upperbound_statistic**
A string specifying the type of upperbound-statistic that is returned. With "cases" the number of cases that would have been necessary to produce an alarm (NNBA) or with "value" the outbreakP-statistic is computed.

**max_upperbound_cases**
Upperbound when numerically searching for NNBA. Default is 1e5.

#### References

**class** `epysurv.models.timepoint.`**RKI**(*years_back: int = 0*, *window_half_width: int = 6*, *include_recent_year: bool = True*)
Bases: `epysurv.models.timepoint._base.STSBasedAlgorithm`

The old algorithm from the Robert Koch Institute.

**years_back**
How many years back in time to include when forming the base counts.

**window_half_width**
Number of weeks to include before and after the current week in each year.

**include_recent_year**
Is a boolean to decide if the year of timePoint also contributes w reference values.

### epysurv.models.timeseries package

#### Submodules

#### epysurv.models.timeseries.convert_interface module

Put a timeseries interface in front of all timepoint algorithms.

**class** `epysurv.models.timeseries.convert_interface.`**Bayes**(*years_back: int = 0*, *window_half_width: int = 6*, *include_recent_year: bool = True*, *alpha: float = 0.05*)
Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.bayes.Bayes`*

**class** `epysurv.models.timeseries.convert_interface.`**`Boda`**(*trend: bool = False, season: bool = False, prior: str = 'iid', alpha: float = 0.05, mc_munu: int = 100, mc_y: int = 10, quantile_method: str = 'MM'*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.boda.Boda`*

**class** `epysurv.models.timeseries.convert_interface.`**`CDC`**(*years_back: int = 5, window_half_width: int = 1, alpha: float = 0.001*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.cdc.CDC`*

**class** `epysurv.models.timeseries.convert_interface.`**`Cusum`**(*reference_value: float = 1.04, decision_boundary: float = 2.26, expected_numbers_method: str = 'mean', transform: str = 'standard', negbin_alpha: float = 0.1*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.cusum.Cusum`*

**class** `epysurv.models.timeseries.convert_interface.`**`EarsC1`**(*alpha: float = 0.001, baseline: int = 7, min_sigma: float = 0*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.ears.EarsC1`*

**class** `epysurv.models.timeseries.convert_interface.`**`EarsC2`**(*alpha: float = 0.001, baseline: int = 7, min_sigma: float = 0*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.ears.EarsC2`*

**class** `epysurv.models.timeseries.convert_interface.`**`Farrington`**(*years_back: int = 3, window_half_width: int = 3, reweight: bool = True, alpha: float = 0.01, trend: bool = True, past_period_cutoff: int = 4, min_cases_in_past_periods: int = 5, power_transform: str = '2/3'*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.farrington.Farrington`*

**class** `epysurv.models.timeseries.convert_interface.`**`FarringtonFlexible`**(*years_back: int = 3, window_half_width: int = 3, reweight: bool = True, weights_threshold: float = 2.58, alpha: float = 0.01, trend: bool = True, trend_threshold: float = 0.05, past_period_cutoff: int = 4, min_cases_in_past_periods: int = 5, power_transform: str = '2/3', past_weeks_not_included: int = 26, threshold_method: str = 'delta'*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.farrington.FarringtonFlexible`*

**class** `epysurv.models.timeseries.convert_interface.`**`GLRNegativeBinomial`**(*alpha: float = 0, glr_test_threshold: int = 5, m: int = -1, change: str = 'intercept', direction: Union[Tuple[str, str], Tuple[str]] = ('inc', 'dec'), upperbound_statistic: str = 'cases', x_max: float = 10000.0*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.glr.GLRNegativeBinomial`*

**class** `epysurv.models.timeseries.convert_interface.`**`GLRPoisson`**(*glr_test_threshold: int = 5, m: int = -1, change: str = 'intercept', direction: Union[Tuple[str, str], Tuple[str]] = ('inc', 'dec'), upperbound_statistic: str = 'cases'*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.glr.GLRPoisson`*

**class** `epysurv.models.timeseries.convert_interface.`**`HMM`**(*n_observations: int = -1, n_hidden_states: int = 2, trend: bool = True, n_harmonics: int = 1, equal_covariate_effects: bool = False*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.hmm.HMM`*

**class** `epysurv.models.timeseries.convert_interface.`**`OutbreakP`**(*threshold: int = 100, upperbound_statistic: str = 'cases', max_upperbound_cases: int = 100000*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.outbreak_p.OutbreakP`*

**class** `epysurv.models.timeseries.convert_interface.`**`RKI`**(*years_back: int = 0, window_half_width: int = 6, include_recent_year: bool = True*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.rki.RKI`*

## Module contents

**class** `epysurv.models.timeseries.`**`Bayes`**(*years_back: int = 0, window_half_width: int = 6, include_recent_year: bool = True, alpha: float = 0.05*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.bayes.Bayes`*

**class** `epysurv.models.timeseries.`**`Boda`**(*trend: bool = False, season: bool = False, prior: str = 'iid', alpha: float = 0.05, mc_munu: int = 100, mc_y: int = 10, quantile_method: str = 'MM'*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.boda.Boda`*

**class** `epysurv.models.timeseries.`**`CDC`**(*years_back: int = 5, window_half_width: int = 1, alpha: float = 0.001*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.cdc.CDC`*

**class** `epysurv.models.timeseries.`**`Cusum`**(*reference_value: float = 1.04, decision_boundary: float = 2.26, expected_numbers_method: str = 'mean', transform: str = 'standard', negbin_alpha: float = 0.1*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.cusum.Cusum`*

**class** `epysurv.models.timeseries.`**`EarsC1`**(*alpha: float = 0.001, baseline: int = 7, min_sigma: float = 0*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.ears.EarsC1`*

**class** `epysurv.models.timeseries.`**`EarsC2`**(*alpha: float = 0.001, baseline: int = 7, min_sigma: float = 0*)

Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.ears.EarsC2`*

**class** epysurv.models.timeseries.**FarringtonFlexible**(*years_back: int = 3, window_half_width: int = 3, reweight: bool = True, weights_threshold: float = 2.58, alpha: float = 0.01, trend: bool = True, trend_threshold: float = 0.05, past_period_cutoff: int = 4, min_cases_in_past_periods: int = 5, power_transform: str = '2/3', past_weeks_not_included: int = 26, threshold_method: str = 'delta'*)
  Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.farrington.FarringtonFlexible`*

**class** epysurv.models.timeseries.**Farrington**(*years_back: int = 3, window_half_width: int = 3, reweight: bool = True, alpha: float = 0.01, trend: bool = True, past_period_cutoff: int = 4, min_cases_in_past_periods: int = 5, power_transform: str = '2/3'*)
  Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.farrington.Farrington`*

**class** epysurv.models.timeseries.**GLRNegativeBinomial**(*alpha: float = 0, glr_test_threshold: int = 5, m: int = -1, change: str = 'intercept', direction: Union[Tuple[str, str], Tuple[str]] = ('inc', 'dec'), upperbound_statistic: str = 'cases', x_max: float = 10000.0*)
  Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.glr.GLRNegativeBinomial`*

**class** epysurv.models.timeseries.**GLRPoisson**(*glr_test_threshold: int = 5, m: int = -1, change: str = 'intercept', direction: Union[Tuple[str, str], Tuple[str]] = ('inc', 'dec'), upperbound_statistic: str = 'cases'*)
  Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.glr.GLRPoisson`*

**class** epysurv.models.timeseries.**HMM**(*n_observations: int = -1, n_hidden_states: int = 2, trend: bool = True, n_harmonics: int = 1, equal_covariate_effects: bool = False*)
  Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.hmm.HMM`*

**class** epysurv.models.timeseries.**OutbreakP**(*threshold: int = 100, upperbound_statistic: str = 'cases', max_upperbound_cases: int = 100000*)
  Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.outbreak_p.OutbreakP`*

**class** epysurv.models.timeseries.**RKI**(*years_back: int = 0, window_half_width: int = 6, include_recent_year: bool = True*)
  Bases: `epysurv.models.timeseries._base.NonLearningTimeseriesClassificationMixin`, *`epysurv.models.timepoint.rki.RKI`*

### Module contents

## epysurv.simulation package

### Submodules

### epysurv.simulation.naive_poisson module

`epysurv.simulation.naive_poisson.`**`get_outbreak_begins`**(*n: int*, *outbreak_length: int*, *n_outbreaks: int*) → Set[int]

`epysurv.simulation.naive_poisson.`**`simulate_outbreaks`**(*n: int = 104*, *outbreak_length: int = 5*, *n_outbreaks: int = 3*, *mu: float = 1*, *outbreak_mu: float = 10*) → pandas.core.frame.DataFrame

Simulate outbreaks based on Poisson distribution.

**Parameters**

- **`n`** – Number of weeks.
- **`outbreak_length`** – Number of weeks each outbreak is long.
- **`n_outbreaks`** – Number of outbreaks.
- **`mu`** – Mean for the baseline.
- **`outbreak_mu`** – Mean for the outbreaks.

**Returns** *Simulated case counts per week, separated into baseline and outbreak cases.*

### Module contents

Module for simulating epidemiological data.

**`class`** `epysurv.simulation.`**`PointSource`**(*alpha: float = 1.0*, *amplitude: float = 1.0*, *frequency: int = 1*, *p: float = 0.99*, *r: float = 0.01*, *seasonal_move: int = 0*, *seed: Optional[int] = None*, *trend: float = 0.0*)

Bases: `epysurv.simulation.base.BaseSimulation`

Simulation of epidemics which were introduced by point sources.

The basis of this programme is a combination of a Hidden Markov Model (to get random time points for outbreaks) and a simple model (compare `epysurv.simulation.SeasonalNoise`) to simulate the baseline.

**Parameters**

- **`amplitude`** – Amplitude of the sine. Determines the possible range of simulated seasonal cases.
- **`alpha`** – Parameter to move along the y-axis (negative values are not allowed) with *alpha >= amplitude*.
- **`frequency`** – Factor in oscillation term. Is multiplied with the annual term $\omega$ and the current time point.
- **`p`** – Probability to get a new outbreak at time $t$ if there was one at time $t - 1$.
- **`r`** – Probability to get no new outbreak at time $t$ if there was none at time $t - 1$.

- **`seasonal_move`** – A term added to time point $t$ to move the curve along the x-axis.
- **`seed`** – Seed for the random number generation.
- **`trend`** – Controls the influence of the current week on $\mu$.

## References

http://surveillance.r-forge.r-project.org/

**`simulate`**(*length: int, state_weight: float = 0, state: Optional[Sequence[int]] = None*) → pandas.core.frame.DataFrame

Simulate outbreaks.

**Parameters**

- **`length`** – Number of weeks to model. `length` is ignored if `state` is given. In this case, the length of `state` is used.
- **`state`** – Use a state chain to define the status at this time point (outbreak or not). If not given, a Markov chain is generated automatically.
- **`state_weight`** – Additional weight for an outbreak which influences the distribution parameter mu.

**Returns** A `DataFrame` of simulated case counts per week, separated into baseline and outbreak cases.

**class** `epysurv.simulation.`**`SeasonalNoiseNegativeBinomial`**(*baseline_frequency: float = 1.5, dispersion: float = 1.0, seasonality_cos: float = 0.2, seasonality_sin: float = -0.4, seasonality_length: int = 1, seed: Optional[int] = None, trend: float = 0.003*)

Bases: `epysurv.simulation.base.BaseSimulation`

A time series simulation that generates case counts based on a negative binomial model.

The model is described by a mean $\mu$, variance $\phi \cdot \mu$, and a linear predictor including trend and seasonality determined by Fourier terms. $\mu$ of the model depends on the current week and is defined as follows:

$$\mu(t) = \exp\left\{\theta + \beta t + \sum_{j=1}^{m}\left\{\gamma_1 \cos(\tfrac{2\pi jt}{52}) + \gamma_2 \sin(\tfrac{2\pi jt}{52})\right\}\right\}$$

where $t$ is the current week, $m$ the seasonality length, $\beta$ equals to the trend parameter, $\gamma$ is a seasonality parameter, and $\theta$ is the baseline frequency of the cases.

The simulation is then run using $\mu$ and the dispersion parameter $\phi$ to specify the negative binomial model we draw case counts from.

**Parameters**

- **`baseline_frequency`** – Baseline frequency of cases.
- **`dispersion`** – Regulates the overdispersion compared to the Poisson distribution ($\phi \cdot \mu$).
- **`seasonality_cos`** – Seasonality parameter to model $\cos$ of the Fourier term.
- **`seasonality_sin`** – seasonality parameter to model $\sin$ of the Fourier term.
- **`seasonality_length`** – Models the annual-wise seasonality. 0 equals to no seasonality, 1 to annual seasonality, 2 to biannual seasonality and so forth.
- **`seed`** – A seed for the random number generation.

- **`trend`** – Controls the influence of the current week on $\mu$.

### References

**`simulate`**(*length: int*) → pandas.core.frame.DataFrame
Simulate outbreaks.

**length** Number of weeks to model.

**Returns** A `DataFrame` of an endemic time series where each row contains the case counts ot this week.

**class** `epysurv.simulation.`**`SeasonalNoisePoisson`**(*alpha: float = 1.0*, *amplitude: float = 1.0*, *frequency: int = 1*, *seasonal_move: int = 0*, *seed: Optional[int] = None*, *trend: float = 0.0*)

Bases: `epysurv.simulation.base.BaseSimulation`

Simulation of an endemic time series based on a Poisson distribution.

The mean of the Poisson distribution is modelled as:

$$\mu(t) = \exp\left(A \sin\left(frequency \cdot \omega \cdot (t + \phi)\right) + \alpha + \beta \cdot t + K \cdot state\right)$$

with $\omega = \pi/52$, $A$ being the amplitude, $\beta$ the trend parameter, $t$ the current week, and $\theta$ the seasonal move.

**Parameters**

- **`amplitude`** – Amplitude of the sine. Determines the range of simulated cases.
- **`alpha`** – Parameter to move simulation along the y-axis (negative values are not allowed) with *alpha >= amplitude*.
- **`frequency`** – Factor in oscillation term. Is multiplied with the annual term $\omega$ and the current time point.
- **`seasonal_move`** – A term added to each time point $t$ to move the curve along the x-axis.
- **`seed`** – Seed for the random number generation.
- **`trend`** – Controls the influence of the current week on $\mu$.

### References

http://surveillance.r-forge.r-project.org/

**`simulate`**(*length: int*, *state_weight: Optional[float] = None*, *state: Optional[Sequence[int]] = None*) → pandas.core.frame.DataFrame
Simulate outbreaks.

**Parameters**

- **`length`** – Number of weeks to model. `length` is ignored if `state` is given. In this case the length of `state` is used.
- **`state`** – Use a state chain to define the status at this time point (outbreak or not). If not given, a Markov chain is generated automatically.
- **`state_weight`** – Additional weight for an outbreak which influences the distribution parameter $\mu$.

**Returns**

- A `DataFrame` of an endemic time series where each row contains the case counts of this week.
- *It also contains the mean case count value based on the underlying sinus model.*

## epysurv.visualization package

### Submodules

### epysurv.visualization.model_diagnostics module

`epysurv.visualization.model_diagnostics.`**`ghozzi_score_plot`**(*prediction_result: pandas.core.frame.DataFrame*, *filename: str*)

Plots case counts and detector predictions with ghozzi weighting.

**Parameters**

- **`prediction_result`** – DataFrame containing 'alarm', 'county', 'pathogen', 'n_cases', 'n_outbreak_cases', 'outbreak'.
- **`filename`** – File name to write the plot to.

`epysurv.visualization.model_diagnostics.`**`plot_confusion_matrix`**(*confusion_matrix: numpy.ndarray*, *class_names: list*, *ax: matplotlib.axes._axes.Axes = None*) → matplotlib.axes._axes.Axes

Plots a confusion matrix, as returned by sklearn.metrics.confusion_matrix, as a heatmap.

Based on https://gist.github.com/shaypal5/94c53d765083101efc0240d776a23823

**Parameters**

- **`confusion_matrix`** – The numpy.ndarray object returned from a call to sklearn.metrics.confusion_matrix. Similarly constructed ndarrays can also be used.
- **`class_names`** – An ordered list of class names, in the order they index the given confusion matrix.
- **`figsize`** – A 2-long tuple, the first value determining the horizontal size of the ouputted figure, the second determining the vertical size. Defaults to (10,7).

**Returns** *The resulting confusion matrix figure*

`epysurv.visualization.model_diagnostics.`**`plot_prediction`**(*train_data*, *test_data*, *prediction*, *ax: matplotlib.axes._axes.Axes = None*) → matplotlib.axes._axes.Axes

Plots case counts as step line, with outbreaks and alarms indicated by triangles.

### Module contents

Module for visualizing epidemiological data and performance of outbreak detection models.

### 4.1.2 Module contents

Surveillance algorithms in Python.

**Currently not implemented:**

- rogerson as it requires in control values to be specified

CHAPTER

# FIVE

# INDICES AND TABLES

- genindex
- modindex
- search

# BIBLIOGRAPHY

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection. *ACM Computing Surveys*, 41(3):1–58, jul 2009. URL: http://www.cs.umn.edu/sites/cs.umn.edu/files/tech_reports/07-017.pdf, doi:10.1145/1541880.1541882.

[Die02] Thomas G Dietterich. Machine Learning for Sequential Data: A Review. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, 15–30. Springer, Berlin, Heidelberg, 2002. URL: http://link.springer.com/10.1007/3-540-70659-3\T1\textbackslash{ }2, arXiv:0-387-31073-8, doi:10.1007/3-540-70659-3_2.

[FABC96] C. P. Farrington, N. J. Andrews, A. D. Beale, and M. A. Catchpole. A Statistical Algorithm for the Early Detection of Outbreaks of Infectious Disease. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 159(3):547, 1996. URL: https://www.jstor.org/stable/10.2307/2983331?origin=crossrefhttp://www.jstor.org/stable/10.2307/2983331?origin=crossref, doi:10.2307/2983331.

[FrisenASchioler09] Marianne Frisén, E Andersson, and L Schiöler. Robust outbreak surveillance of epidemics in Sweden. *Statistics in Medicine*, 28(3):476–493, 2009. URL: www.interscience.wiley.com, arXiv:NIHMS150003, doi:10.1002/sim.3483.

[HTST03] Lori Hutwagner, William Thompson, G Matthew Seeman, and Tracee Treadwell. The bioterrorism preparedness and response Early Aberration Reporting System (EARS). *Journal of urban health : bulletin of the New York Academy of Medicine*, 80(2 Suppl 1):i89–i96, 2003. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3456557/pdf/11524_2006_Article_200.pdf, doi:10.1007/PL00022319.

[NEF+13] Angela Noufaily, Doyo G. Enki, Paddy Farrington, Paul Garthwaite, Nick Andrews, and André Charlett. An improved algorithm for outbreak detection in multiple surveillance systems. *Statistics in Medicine*, 32(7):1206–1222, 2013. URL: http://ojphi.org, doi:10.1002/sim.5595.

[RLM99] G Rossi, L Lampugnani, and M Marchi. an Approximate Cusum Procedure for. *Statistics in Medicine*, 2122(November 1997):2111–2122, 1999.

[SSHohle16] Maëlle Salmon, Dirk Schumacher, and Michael Höhle. Monitoring Count Time Series in R : Aberration Detection in Public Health Surveillance. *Journal of Statistical Software*, 2016. URL: http://www.jstatsoft.org/v70/i10/, arXiv:1411.1292, doi:10.18637/jss.v070.i10.

[SWHK89] Donna F Stroup, G David Williamson, Joy L Herndon, and John M Karon. Detection of aberrations in the occurrence of notifiable diseases surveillance data. *Statistics in medicine*, 8(3):323–329, 1989.

# PYTHON MODULE INDEX

# INDEX