

# **Artificial Intelligence in Industry**

Last update: 16 November 2024

Academic Year 2024 – 2025  
Alma Mater Studiorum · University of Bologna

# Contents

<b>1</b>	<b>Preliminaries</b>	<b>1</b>
<b>2</b>	<b>Low dimensional anomaly detection: Taxi calls</b>	<b>2</b>
2.1	Data . . . . .	2
2.2	Approaches . . . . .	2
2.2.1	Gaussian assumption . . . . .	2
2.2.2	Characterize data distribution . . . . .	2
2.2.2.1	Univariate kernel density estimation . . . . .	3
2.2.2.2	Multivariate kernel density estimation . . . . .	4
2.2.2.3	Time-dependent estimator . . . . .	6
2.2.2.4	Time-indexed model . . . . .	7
<b>3</b>	<b>High dimensional anomaly detection: HPC centers</b>	<b>8</b>
3.1	Data . . . . .	8
3.1.1	High-dimensional data visualization . . . . .	8
3.2	Approaches . . . . .	9
3.2.1	Multivariate KDE . . . . .	9
3.2.2	Gaussian mixture model . . . . .	9
3.2.3	Autoencoder . . . . .	11
<b>4</b>	<b>Missing data: Traffic data</b>	<b>14</b>
4.1	Data . . . . .	14
4.2	Preliminaries . . . . .	14
4.2.1	Resampling / Binning . . . . .	14
4.3	Approaches . . . . .	14
4.3.1	Forward/Backward filling . . . . .	15
4.3.2	Geometric interpolation . . . . .	15
4.3.3	Density estimator . . . . .	15
4.3.4	Regressor . . . . .	15
4.3.5	Gaussian processes . . . . .	16
4.3.6	Multiplicative ensemble . . . . .	18
<b>5</b>	<b>Remaining useful life: Turbofan engines</b>	<b>20</b>
5.1	Data . . . . .	20
5.1.1	Data splitting . . . . .	20
5.2	Approaches . . . . .	21
5.2.1	Regressor . . . . .	21
5.2.2	Classifier . . . . .	22
5.2.3	Survival analysis (regression) . . . . .	24
5.2.4	Survival analysis (classification) . . . . .	25
<b>6</b>	<b>Component wear anomalies: Skin wrapper machines</b>	<b>28</b>
6.1	Data . . . . .	28
6.1.1	Binning . . . . .	29

6.2	Approaches . . . . .	29
6.2.1	Autoencoder . . . . .	29
6.2.2	Class rebalancing . . . . .	30
6.2.2.1	Importance sampling applications . . . . .	31
<b>7</b>	<b>Arrivals prediction: Hospital emergency room</b>	<b>32</b>
7.1	Data . . . . .	32
7.2	Approaches . . . . .	32
7.2.1	Neuro-probabilistic model . . . . .	33
<b>8</b>	<b>Feature selection and important: Biomedical analysis</b>	<b>35</b>
8.1	Data . . . . .	35
8.1.1	Preliminary analysis . . . . .	35
8.2	Approaches . . . . .	36
8.2.1	Linear model . . . . .	36
8.2.2	Non-linear model . . . . .	38
8.2.3	Additive feature attribution . . . . .	39
8.2.4	Semantics for feature selection . . . . .	43

# 1 Preliminaries

**Problem formalization** Defines the ideal goal.

**Solution formalization** Defines the actual possible approaches to solve a problem.

**Occam's razor** Principle for which, between two hypotheses, the simpler one is usually correct.

| **Remark.** This approach has less variance and more bias, making it more robust.

Problem  
formalization  
Solution  
formalization  
Occam's razor

## 2 Low dimensional anomaly detection: Taxi calls

**Anomaly** Event that deviates from the usual pattern.

Anomaly

**Time series** Data with an ordering (e.g., chronological).

Time series

### 2.1 Data

The dataset is a time series and it is a `DataFrame` with the following fields:

`timestamp` with a 30 minutes granularity.

`value` number of calls.

The label is a `Series` containing the timestamps of the anomalies.

An additional `DataFrame` contains information about the time window in which the anomalies happen:

`begin` acceptable moment from which an anomaly can be detected.

`end` acceptable moment from which there are no anomalies anymore.

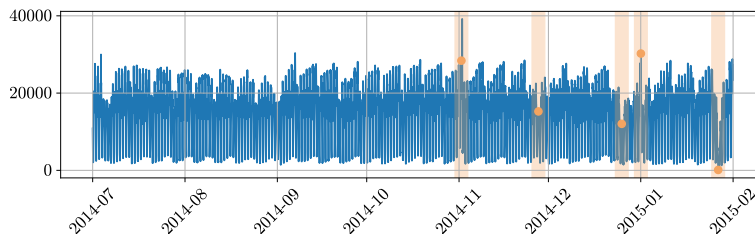


Figure 2.1: Plot of the time series, anomalies, and windows

## 2.2 Approaches

### 2.2.1 Gaussian assumption

Assuming that the data follows a Gaussian distribution, mean and variance can be used to determine anomalies through a threshold.  $z$ -score can also be used.

### 2.2.2 Characterize data distribution

Classify a data point as an anomaly if it is too unlikely.

**Problem formalization** Given a random variable  $X$  with values  $x$  to represent the number of taxi calls, we want to find its probability density function (PDF)  $f(x)$ .

An anomaly is determined whether:

$$f(x) \leq \varepsilon$$

where  $\varepsilon$  is a threshold.

**Remark.** A PDF can be reasonably used even though the dataset is discrete if its data points are sufficiently fine-grained.

**Remark.** It is handy to use negated log probabilities as:

- The logarithm adds numerical stability.
- The negation makes the probability an alarm signal, which is a more common measure.

Therefore, the detection condition becomes:

$$-\log f(x) \geq \varepsilon$$

**Solution formalization** The problem can be tackled using a density estimation technique.

### 2.2.2.1 Univariate kernel density estimation

**Kernel density estimation (KDE)** Based on the assumption that whether there is a data point, there are more around it. Therefore, each data point is the center of a density kernel.

Kernel density estimation (KDE)

**Density kernel** A kernel  $K(x, h)$  is defined by:

- The input variable  $x$ .
- The bandwidth  $h$ .

**Gaussian kernel** Kernel defined as:

$$K(x, h) \propto e^{-\frac{x^2}{2h^2}}$$

where:

- The mean is 0.
- $h$  is the standard deviation.

As the mean is 0, an affine transformation can be used to center the kernel on a data point  $\mu$  as  $K(x - \mu, h)$ .

Given  $m$  training data points  $\bar{x}_i$ , the density of any point  $x$  can be computed as the kernel average:

$$f(x, \bar{x}, h) = \frac{1}{m} \sum_{i=0}^m K(x - \bar{x}_i, h)$$

Therefore, the train data themselves are used as the parameters of the model while the bandwidth  $h$  has to be estimated.

**Data split** Time series are usually split chronologically:

**Train** Should ideally contain only data representing the normal pattern. A small amount of anomalies might be tolerated as they have low probabilities.

**Validation** Used to find the threshold  $\varepsilon$ .

**Test** Used to evaluate the model.

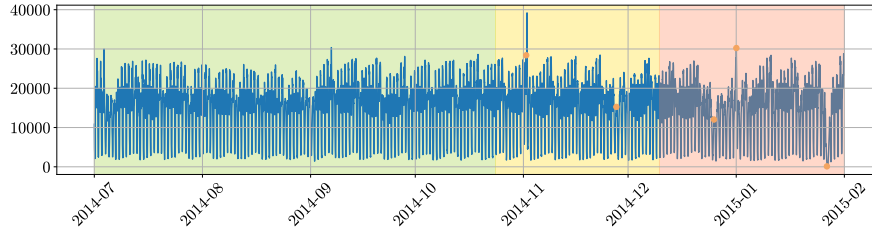


Figure 2.2: Train, validation, and test splits

**Metrics** It is not straightforward to define a metric for anomaly detection. A cost model to measure the benefits of a prediction is more suited. A simple cost model can be based on:

**True positives (TP)** Windows for which at least an anomaly is detected;

**False positives (FP)** Detections that are not actually anomalies;

**False negatives (FN)** Undetected anomalies;

**Advance (adv)** Time between an anomaly and when it is first detected;

and is computed as:

$$(c_{\text{false}} \cdot \text{FP}) + (c_{\text{miss}} \cdot \text{FN}) + (c_{\text{late}} \cdot \text{adv}_{\leq 0})$$

where  $c_{\text{false}}$ ,  $c_{\text{miss}}$ , and  $c_{\text{late}}$  are hyperparameters.

**Bandwidth estimation** According to some statistical arguments, a rule-of-thumb to estimate  $h$  in the univariate case is the following:

$$h = 0.9 \cdot \min \left\{ \hat{\sigma}, \frac{\text{IQR}}{1.34} \right\} \cdot m^{-\frac{1}{5}}$$

where:

- IQR is the inter-quartile range.
- $\hat{\sigma}$  is the standard deviation computed over the whole dataset.

**Threshold optimization** Using the train and validation set, it is possible to find the best threshold  $\varepsilon$  that minimizes the cost model through linear search.

**Remark.** The train set can be used alongside the validation set to estimate  $\varepsilon$  as this operation is not used to prevent overfitting.

**Remark.** The evaluation data should be representative of the real world distribution. Therefore, in this case, to evaluate the model the whole dataset can be used.

**Remark.** KDE assumes that the Markov property holds. Therefore, each data point is considered independent to the others.

### 2.2.2.2 Multivariate kernel density estimation

**Remark.** In this dataset, nearby points tend to have similar values.

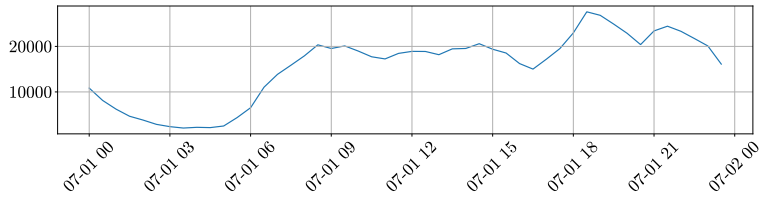


Figure 2.3: Subset of the dataset

**Autocorrelation plot** Plot to visualize the correlation between nearby points of a series. Given the original series, it is duplicated, shifted by a lag  $l$ , and the Pearson correlation coefficient is then computed between the two series. This operation is repeated over different values of  $l$ .

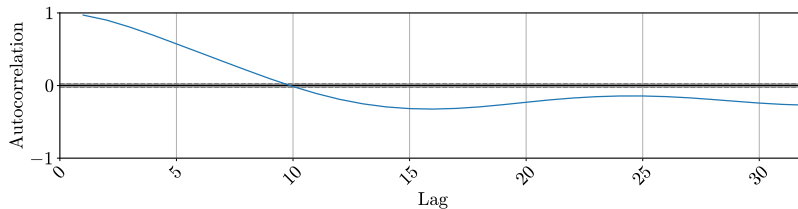


Figure 2.4: Autocorrelation plot of the subset of the dataset.  
There is strong correlation up to 4-5 lags.

**Sliding window** Given a window size  $w$  and a stride  $s$ , the dataset is split into sequences of  $w$  continuous elements.

| **Remark.** Incomplete sequences at the start and end of the dataset are ignored.

| **Remark.** In pandas, the `rolling` method of `Dataframe` allows to create a slicing window iterator. This approach creates the windows row-wise and also considers incomplete windows. However, a usually more efficient approach is to construct the sequences column-wise by hand.

**Multivariate KDE** Extension of KDE to vector variables.

**Window size estimation** By analyzing the autocorrelation plot, an ideal window size can be picked as the lag with a low correlation (e.g., 10 according to Figure 2.4).

**Bandwidth estimation** Differently from the univariate case, the bandwidth has to be estimated by maximizing the log-likelihood on the validation set. Given:

- The validation set  $\tilde{x}$ ,
- The observations  $x$ ,
- The bandwidth  $h$ ,
- The density estimator  $\hat{f}$

the likelihood is computed, by assuming independent observations, as:

$$L(h, x, \tilde{x}) = \prod_{i=1}^m \hat{f}(x_i, \tilde{x}_i, h)$$



Maximum likelihood estimation is defined as:

$$\arg \max \mathbb{E}_{x \sim f(x), \tilde{x} \sim f(x)} [L(h, x, \tilde{x})]$$

where  $f(x)$  is the true distribution.

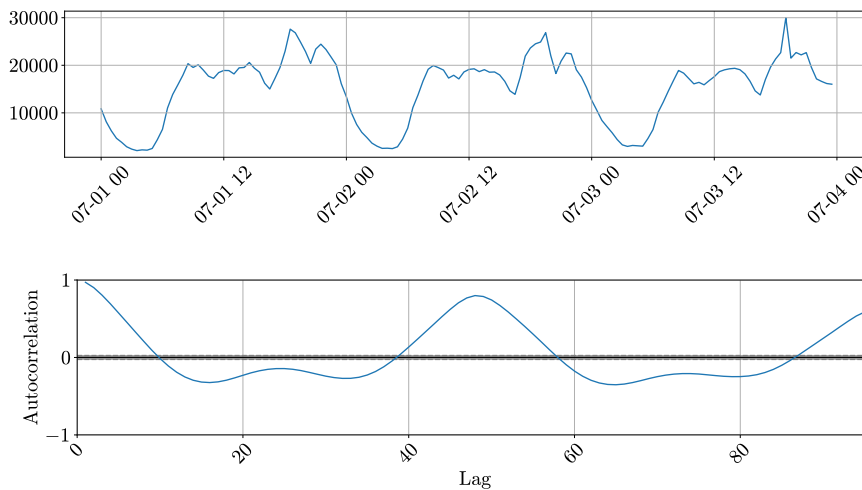
In practice, the expected value is sampled from multiple validation and training sets through cross-validation and grid-search.

**Remark.** As different folds for cross-validation must be tried, grid-search is an expensive operation.

**Threshold optimization** The threshold can be determined as in Section 2.2.2.1.

### 2.2.2.3 Time-dependent estimator

The approach taken in Sections 2.2.2.1 and 2.2.2.2 is based on the complete timestamp of the dataset. Therefore, the same times on different days are treated differently. However, it can be seen that this time series is approximately periodic, making the approach used so far more complicated than necessary.



**Time input** It is possible to take time into consideration by adding it as a parameter of the density function:

$$f(t, x)$$

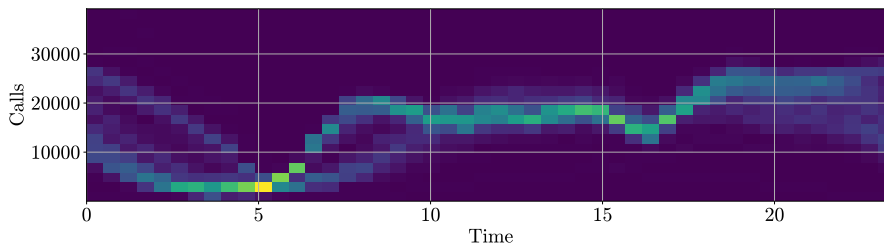


Figure 2.5: 2D histogram of the distribution. Lighter colors indicate a higher frequency of occurrence.

However,  $t$  is a controlled variable and is completely predictable (i.e., if times are samples with different frequencies, they should not affect the overall likelihood). Therefore, a conditional density should be used:

$$f(x | t) = \frac{f(t, x)}{f(t)}$$

where  $f(t)$  can be easily computed using KDE on the time data.

**Remark.** For this dataset, the time distribution is uniform. Therefore,  $f(t)$  is a constant and it is correct to use  $f(t, x)$  as the estimator.

**Bandwidth estimation** The bandwidth can be estimated as in Section 2.2.2.2.

**Remark.** When using the `scikit-learn`, the dataset should be normalized as the implementation of KDE use the same bandwidth for all features.

**Threshold optimization** The threshold can be determined as in Section 2.2.2.1.

#### 2.2.2.4 Time-indexed model

**Ensemble model** Model defined as:

$$f_{g(t)}(x)$$

Ensemble model

where:

- $f_i$  are estimators, each working on subsets of the dataset and solving a smaller problem.
- $g(t)$  determines which particular  $f_i$  should solve the input.

**Time-indexed model** Consider both time and sequence inputs by using an ensemble model. Each estimator is specialized on a single time value (i.e., an estimator for 00:00, one for 00:30, ...).

**Bandwidth estimation** The bandwidth can be estimated as in Section 2.2.2.2.

**Threshold optimization** The threshold can be determined as in Section 2.2.2.1.

# 3 High dimensional anomaly detection: HPC centers

## 3.1 Data

The dataset is a time series with the following fields:

`timestamp` with a 5 minutes granularity.

**HPC data** technical data related to the cluster.

`anomaly` indicates if there is an anomaly.

In practice, the cluster has three operational modes:

**Normal** the frequency is proportional to the workload.

**Power-saving** the frequency is always at the minimum.

**Performance** the frequency is always at the maximum.

For this dataset, both power-saving and performance are considered anomalies.

### 3.1.1 High-dimensional data visualization

**Individual plots** Plot individual columns.

**Statistics** Show overall statistics (e.g., `pandas describe` method)

**Heatmap** Heatmap with standardized data.

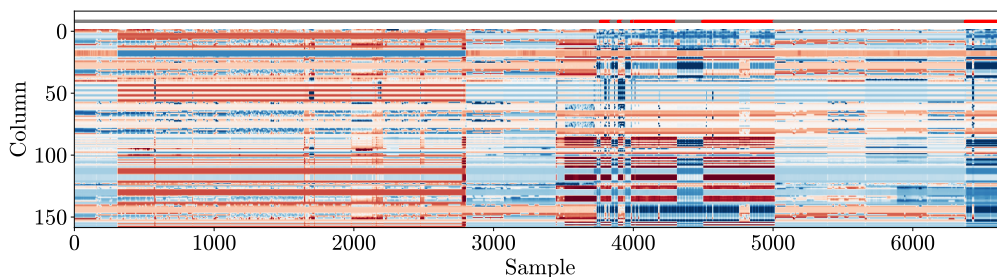


Figure 3.1: Heatmap of the dataset. On the top line, gray points represent normal behavior while red points are anomalies. In the heatmap, white tiles represent the mean, red tiles represent values below average, and blue tiles represent values above average.

## 3.2 Approaches

### 3.2.1 Multivariate KDE

Given the train, validation, and test splits, the dataset is standardized using the training set alone. This avoids leaks from the test set.

The KDE model, bandwidth, and threshold are fitted as in Chapter 2.

**Cost model** A simple cost model can be based on:

- False positives,
- False negatives,
- A time tolerance.

**Problems** KDE on this dataset has the following problems:

- It is highly subject to the curse of dimensionality and requires more data to be reliable.

**Remark.** KDE does not compress the input features and grows with the training set. In other words, it has low bias and high variance.

- As the dataset is used during inference, a large dataset is computationally expensive (with time complexity  $O(mn)$ , where  $m$  is the number of dimensions and  $n$  the number of samples).
- It only provides an alarm signal and not an explanation of the anomaly. In low-dimensionality, this is still acceptable, but with high-dimensional data it is harder to find an explanation.

### 3.2.2 Gaussian mixture model

**Gaussian mixture model (GMM)** Selection-based ensemble that estimates a distribution as a weighted sum of Gaussians. It is assumed that data can be generated by the following probabilistic model:

Gaussian mixture model (GMM)

$$X_Z$$

where:

- $X_k$  are random variables following a multivariate Gaussian distribution. The number of components is a hyperparameter that can be tuned to balance bias-variance.
- $Z$  is a random variable representing the index  $k$  of the variable  $X$  to use to generate data.

More specifically, the PDF of a GMM  $g$  is defined as:

$$g(x, \mu, \Sigma, \tau) = \sum_{k=1}^n \tau_k f(x, \mu_k, \Sigma_k)$$

where:

- $f$  is the PDF of a multivariate normal distribution.
- $\mu_k$  and  $\Sigma_k$  are the mean and covariance matrix for the  $k$ -th component, respectively.
- $\tau_k$  is the weight for the  $k$ -th component and corresponds to  $\mathcal{P}(Z = k)$ .

**Training** Likelihood maximization can be used to train a GMM to approximate another distribution:

$$\arg \max_{\mu, \Sigma, \tau} \mathbb{E}_{x \sim X} [L(x, \mu, \Sigma, \tau)] \quad \text{subject to} \quad \sum_{k=1}^n \tau_k = 1$$

where the expectation can be approximated using the training set (i.e., empirical risk minimization):

$$\mathbb{E}_{x \sim X} [L(x, \mu, \Sigma, \tau)] \approx \prod_{i=1}^m g(x_i, \mu, \Sigma, \tau)$$

**Remark.** Empirical risk minimization can be solved in two ways:

- Use a single large sample (traditional approach).
- Use many smaller samples (as in cross-validation).

By putting the definitions together, we obtain the following problem:

$$\arg \max_{\mu, \Sigma, \tau} \prod_{i=1}^m \sum_{k=1}^n \tau_k f(x, \mu_k, \Sigma_k) \quad \text{subject to} \quad \sum_{k=1}^n \tau_k = 1$$

which:

- Cannot be solved using gradient descent as it is an unconstrained method.
- Cannot be solved using mixed-integer linear programming as the problem is non-linear.
- Cannot be decomposed as the variables  $\mu$ ,  $\Sigma$ , and  $\tau$  appear in every term.

It is possible to simplify the formulation of the problem by introducing new variables:

- A new latent random variable  $Z_i$  is added for each example.  $Z_i = k$  iff the  $i$ -th example is drawn from the  $k$ -th component. The PDF of the GMM can be reformulated to use  $Z_i$  and without the summation as:

$$\tilde{g}_i(x_i, z_i, \mu, \Sigma, \tau) = \tau_{z_i} f(x, \mu_{z_i}, \Sigma_{z_i})$$

The expectation becomes:

$$\mathbb{E}_{x \sim X, \{z_i\} \sim \{Z_i\}} [L(x, z, \mu, \Sigma, \tau)] \approx \mathbb{E}_{\{z_i\} \sim \{Z_i\}} \left[ \prod_{i=1}^m \tilde{g}_i(x_i, z_i, \mu, \Sigma, \tau) \right]$$

Note that, as the distributions to sample  $z_i$  are unknown,  $Z_i$  cannot be approximated in the same way as  $X$ .

- New variables  $\tilde{\tau}_{i,k}$  are introduced to represent the distribution of the  $Z_i$  variables. In other words,  $\tilde{\tau}_{i,k}$  corresponds to  $\mathcal{P}(Z_i = k)$ . This allows to approximate the expectation as:

$$\mathbb{E}_{\hat{x} \sim X, \hat{z} \sim Z} [L(\hat{x}, \hat{z}, \mu, \Sigma, \tau)] \approx \prod_{i=1}^m \prod_{k=1}^n \tilde{g}_i(x_i, z_i, \mu, \Sigma, \tau)^{\tilde{\tau}_{i,k}}$$

The intuitive idea is that, if  $Z_i$  is sampled,  $\tilde{\tau}_{i,k}$  of the samples would be the  $k$ -th component. Therefore, the corresponding density should be multiplied by itself for  $\tilde{\tau}_{i,k}$  times.

Finally, the GMM training can be formulated as follows:

$$\arg \max_{\mu, \Sigma, \tau, \tilde{\tau}} \prod_{i=1}^m \prod_{k=1}^n \tilde{g}_i(x_i, z_i, \mu, \Sigma, \tau)^{\tilde{\tau}_{i,k}} \quad \text{s.t.} \quad \sum_{k=1}^n \tau_k = 1, \forall i = 1 \dots m : \sum_{k=1}^n \tilde{\tau}_{i,k} = 1$$

which can be simplified by applying a logarithm and solved using expectation-maximization.

**Remark.** Differently from KDE, GMM allows making various types of prediction:

- Evaluate the (log) density of a sample.
- Generate a sample.
- Estimate the probability that a sample belongs to a component.
- Assign samples to a component (i.e., clustering).

| **Remark.** GMM can be seen as a generalization of  $k$ -means.

| **Remark.** Differently from KDE, most of the computation is done at training time.

**Number of components estimation** The number of Gaussians to use in GMM can be determined through grid-search and cross-validation.

| **Remark.** Other method such as the elbow method can also be applied. Some variants of GMM are able to infer the number of components.

**Threshold optimization** The threshold can be determined in the same way as in Section 2.2.2.1.

### 3.2.3 Autoencoder

**Autoencoder** Neural network trained to reconstruct its input. It is composed of two components: Autoencoder

**Encoder**  $e(x, \theta_e)$  that maps an input  $x$  into a latent vector  $z$ .

**Decoder**  $d(z, \theta_d)$  that maps  $z$  into a reconstruction of  $x$ .

**Training** Training aims to minimize the reconstruction MSE:

$$\arg \min_{\theta_e, \theta_d} \|d(e(x_i, \theta_e), \theta_d) - x_i\|_2^2$$

To avoid trivial embeddings, the following can be done:

- Use a small-dimensional latent space.
- Use L1 regularization to prefer sparse encodings.

**Autoencoder for anomaly detection** By evaluating the quality of the reconstruction, an autoencoder can be used for anomaly detection:

$$\|x - d(e(x, \theta_e), \theta_d)\|_2^2 \geq \varepsilon$$

The advantages of this approach are the following:

- The size of the neural network does not scale with the training data.
- Neural networks have good performances in high-dimensional spaces.
- Inference is fast.

However, the task of reconstruction can be harder than density estimation.

**Remark.** It is always a good idea to normalize the input of a neural network to have a more stable gradient descent. Moreover, with normalized data, common weight initialization techniques make the output approximately normalized too.

**Remark.** Counterintuitively, neural networks have a higher bias compared to traditional machine learning techniques for mainly two reasons:

- A change in a single parameter affects the whole network.
- Training using SGD inherently prevents overfitting.

**Theorem 3.2.1.** Under the following assumptions:

- Normally distributed noise (i.e., distance between prediction and ground truth),
- Independent noise among each output component (i.e., columns),
- Same variance (i.e., homoscedasticity) in the noise of each output component,

autoencoders are trained as density estimators.

**Remark.** When minimizing MSE, these assumptions hold.

*Proof.* When training an autoencoder  $h$  using MSE on  $m$  examples with  $n$  features, the following problem is solved:

$$\begin{aligned}
 \arg \min_{\theta} \|h(\mathbf{x}, \theta) - \mathbf{x}\|_2^2 &= \arg \min_{\theta} \sum_{i=1}^m \sum_{j=1}^n (h_j(\mathbf{x}_i, \theta) - \mathbf{x}_{i,j})^2 \\
 &= \arg \min_{\theta} \log \exp \left( \sum_{i=1}^m \sum_{j=1}^n (h_j(\mathbf{x}_i, \theta) - \mathbf{x}_{i,j})^2 \right) \\
 &= \arg \min_{\theta} \log \prod_{i=1}^m \exp \left( \sum_{j=1}^n (h_j(\mathbf{x}_i, \theta) - \mathbf{x}_{i,j})^2 \right) \\
 &= \arg \min_{\theta} \log \prod_{i=1}^m \exp \left( (h(\mathbf{x}_i, \theta) - \mathbf{x}_i)^T \mathbf{I} (h(\mathbf{x}_i, \theta) - \mathbf{x}_i) \right)
 \end{aligned}$$

The following adjustments can be done without altering the problem:

- Negate the argument of exp and solve a maximization problem,
- Multiply the argument of exp by  $\frac{1}{2}\sigma$ , for some constant  $\sigma$ ,
- Multiply exp by  $\frac{1}{\sqrt{2\pi\sigma}}$ .

The problem becomes:

$$\arg \max_{\theta} \log \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma}} \exp \left( -\frac{1}{2} (h(\mathbf{x}_i, \theta) - \mathbf{x}_i)^T (\sigma \mathbf{I}) (h(\mathbf{x}_i, \theta) - \mathbf{x}_i) \right)$$

which is the PDF of a multivariate normal distribution  $f(\mathbf{x}_i, h(\mathbf{x}_i), \sigma \mathbf{I})$  with a diagonal covariance matrix. More specifically, this is a distribution:

- Centered on  $h(\mathbf{x}_i)$ ,
- With independent normal components,

- With components with uniform variance.

Therefore, when using MSE as loss, training is a likelihood maximization problem.  $\square$

**Threshold optimization** The threshold can be determined in the same way as in Section 2.2.2.1.

**Multiple signal analysis** With autoencoders, it is possible to compare the reconstruction error of single components.

**Remark.** In most cases, reconstruction errors are often concentrated on a few features.

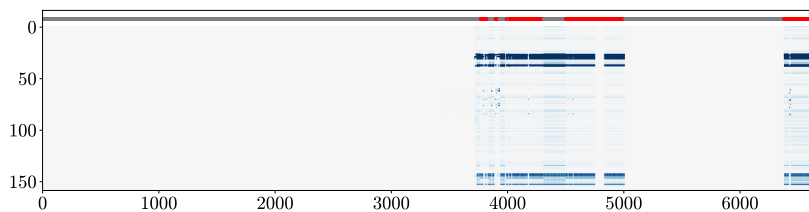


Figure 3.2: Reconstruction error of each feature

**Remark.** It is possible to rank the feature with the highest reconstruction error to provide more insights.

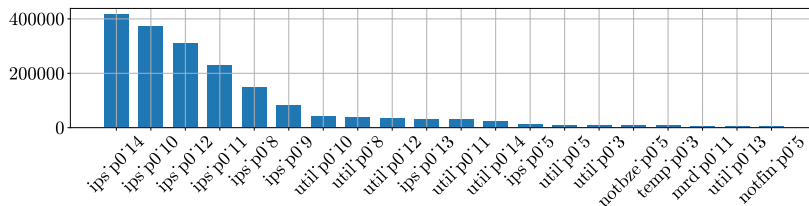


Figure 3.3: Top-20 features with the largest error

**Remark.** In industrial use-cases, a usually well working approach consists of building an estimator for the sensors. More specifically, observed variables can be split into:

- Controlled variables  $x_c$ .
- Measured variables  $x_s$ .

During anomaly detection, controlled variables should not be used to determine anomalous behaviors. In addition, there is a causal relation between the two groups as measured variables are caused by the controlled variables ( $x_c \rightarrow x_s$ ).

A well working method is to use a regressor  $f$  as a causal model, such that:

$$x_s \approx f(x_c; \theta)$$



# 4 Missing data: Traffic data

## 4.1 Data

Time series on traffic anomalies with missing values.

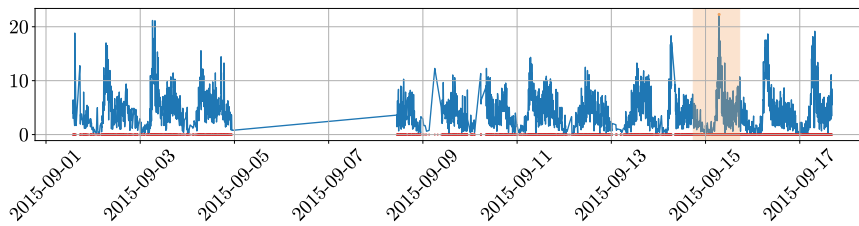


Figure 4.1: Plot of the data. Straight lines are artifacts of missing values. Red dots below represent the actual data points.

## 4.2 Preliminaries

The dataset has sparse indexes (i.e., indexes are non-contiguous) and missing values are represented by gaps. It is necessary to use dense indexes where missing values are explicitly marked as NaN.

### 4.2.1 Resampling / Binning

**Resampling / binning** Resample the indexes of the dataset so that they have a regular step (e.g., 5 minutes).

Resampling / binning

| **Remark.** Values that end up in the same bin need to be aggregated (e.g., mean).

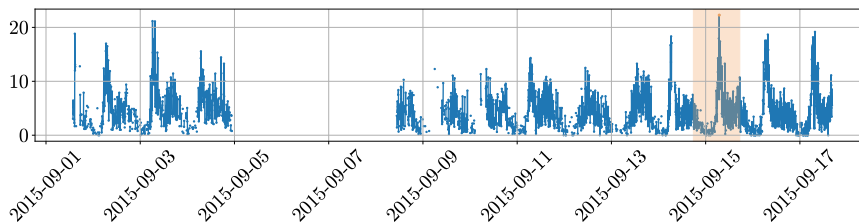


Figure 4.2: Plot of the resampled data without artifacts

## 4.3 Approaches

**Benchmark dataset** A portion of known data where some values are artificially removed can be used to evaluate a filling method. As accuracy metric, RMSE can be used.

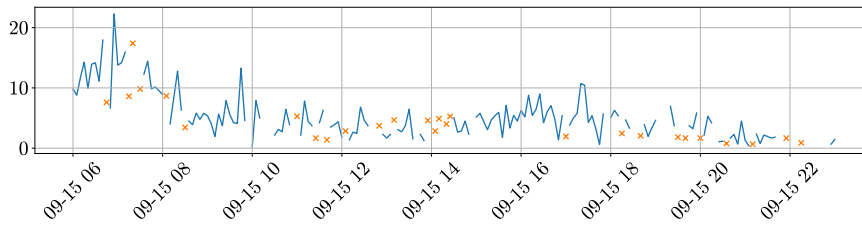


Figure 4.3: Benchmark dataset. Marked points have been artificially removed.

### 4.3.1 Forward/Backward filling

**Forward filling** Set the missing value to the last valid observation.

**Backward filling** Set the missing value to the next valid observation.

**Remark.** The idea of this approach is that time series usually have strong local correlation (i.e., some sort of inertia).

**Remark.** Forward/backward filling tend to work well on low variance portions of the data.

### 4.3.2 Geometric interpolation

Interpolate a function to determine missing points. Possible methods are:

- Linear,
- Nearest value,
- Polynomial,
- Spline.

**Remark.** (R)MSE assumes that the data is normally distributed, independent, and with the same variability at all points. This is not usually true with time series.

### 4.3.3 Density estimator

A density estimator can be used to determine a distribution from all the available data. Given an estimator  $f(x, \theta)$  for  $\mathcal{P}(x)$ , predictions can be obtained through maximum a posteriori (MAP):

$$\arg \max_x f(x, \theta)$$

However, MAP with density estimators is computationally expensive.

**Remark.** Almost all inference approaches in machine learning can be reduced to a maximum a posteriori computation.

### 4.3.4 Regressor

A regressor can be used to fill missing values autoregressively through a rolling forecast by repeatedly making a prediction and including the new point as a training sample. However, regression only relies on the data on one side (past or future) and each autoregressive iteration accumulates compound errors.

### 4.3.5 Gaussian processes

**Remark.** The ideal estimator is the one that is:

- At least as powerful as interpolation (i.e., considers both past and future data).
- Able to detect the expected variability (i.e., a measure of confidence).

**Gaussian process** Stochastic process (i.e., collection of indexed random variables) such that:

Gaussian process

- The index variables  $x$  are continuous and represents an input of arbitrary dimensionality.
- The variable  $y_x$  represents the output for  $x$ .

The random variables  $y_x$  respect the following assumptions:

- They follow a Gaussian distribution.
- The standard deviation depends on the distance between a point and the given observations (i.e., a confidence measure).
- $y_x$ s are correlated. Therefore, every finite subset of  $y_x$  variables follows a multivariate normal distribution.

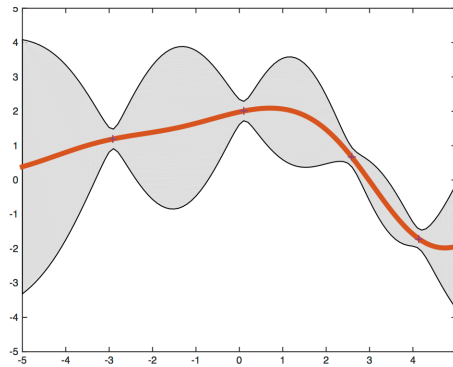


Figure 4.4: Example of Gaussian process. The red line represents the mean and the gray area the confidence interval.

**Remark.** The PDF of a multivariate normal distribution is defined by the mean vector  $\mu$  and the covariance matrix  $\Sigma$ . By recentering ( $\mu = \bar{\mathbf{0}}$ ), knowing  $\Sigma$  is enough to compute the joint or conditional density.

**Remark.** To fill missing values, the conditional density  $f(y_x|\bar{y}_x)$  is used to infer a new observation  $y_x$  given a set of known observations  $\bar{y}_x$ .

#### Naive implementation

**Training** Given the training observations  $\bar{y}_x$ , the covariance matrix can be defined as a parametrized function  $\Sigma(\theta)$  and optimized for maximum likelihood:

$$\arg \max_{\theta} f(\bar{y}_x, \theta)$$

where  $f$  is the joint PDF of a multivariate normal distribution.

**Inference** To infer the variable  $y_x$  associated to an input  $x$ , the joint distribution  $f(y_x|\bar{y}_x)$  has to be computed:

$$f(y_x|\bar{y}_x) = \frac{f(y_x, \bar{y}_x)}{f(\bar{y}_x)}$$

- $f(\bar{y}_x)$  can be computed using the  $\Sigma$  determined during training, which is an  $n \times n$  matrix assuming  $n$  training observations.
- $f(y_x, \bar{y}_x)$  introduces an additional variable and would require an  $(n + 1) \times (n + 1)$  covariance matrix which cannot be determined without further assumptions.

**Kernel implementation** It is assumed that the covariance of two variables can be determined through parametrized kernel functions  $K_\theta(x_i, x_j)$ .

| **Remark.** Typically, the kernel is a distance measure.

**Training** Given the training observations  $\bar{y}_x$  and a parametrized kernel function  $K_\theta(x_i, x_j)$ , training is done by optimizing the kernel for maximum likelihood (e.g., gradient descent).

The trained model is represented by both the kernel parameters  $\theta$  and the training samples  $\bar{y}_x$ .

**Inference** Given a new input  $x$ , the joint distribution  $f(y_x | \bar{y}_x)$  can be computed by obtaining  $\Sigma_{\bar{x}}$  and  $\Sigma_{x, \bar{x}}$  using the kernel.

### Common kernels

**Radial basis function** Based on the Euclidean distance  $d(x_i, x_j)$  between the input points:

$$K(x_i, x_j) = e^{-\frac{d(x_i, x_j)^2}{2l}}$$

where  $l$  is a parameter and represents the scale.

**White kernel** Captures the noise in the data:

$$K(x_i, x_j) = \begin{cases} \sigma^2 & \text{iff } x_i = x_j \\ 0 & \text{otherwise} \end{cases}$$

where  $\sigma$  is a parameter and represents the noise level.

**Constant kernel** Represents a learnable constant factor, useful to tune the magnitude of other kernels.

**Exp-Sine-Squared** Captures a period:

$$K(x_i, x_j) = e^{-2\frac{\sin^2\left(\pi\frac{d(x_i, x_j)}{p}\right)}{l^2}}$$

where  $l$  and  $p$  are parameters representing periodicity and scale, respectively.

**Dot product** Is more or less able to capture a trend:

$$K(x_i, x_j) = \sigma^2 + x_i x_j$$

where  $\sigma$  is a parameter and represents the base level of correlation.

| **Remark.** This kernel is not translation-invariant

| **Remark.** Bounding the domain of the parameters of a kernel can help control training.

| **Remark.** With Gaussian processes, as we have both the prediction and the confidence interval, likelihood can be used as evaluation metric.

**Remark.** With Gaussian processes, by predicting points far away from the training observations (i.e., extrapolation), the mean starts to fall to 0. Only when there is a period, predictions outside the reference observations can be reasonably made.

**Remark.** As the reference observations and the trained kernel are detached, it is possible to change reference observations without retraining.

**Inference** As changing reference observations can be done without retraining the kernel, the whole series can be used when doing inference to obtain more accurate results.

To fill missing values, there are two main strategies:

**Prediction** Use the mean as filling value.

**Sampling** Use mean and variance to sample a point to fill the missing value. Clipping might be needed to make the sampled point valid (e.g., prevent negative values for traffic).

**Remark.** Using the mean results in a smoother filling, while sampling produces more realistic data.

### 4.3.6 Multiplicative ensemble

**Remark.** Gaussian process alone only accounts for covariance and do not consider input-dependent variance (i.e., variance of the traffic at the same week day and time on different days).

**Remark.** Variance scales via multiplication but not summation:

$$\text{Var}(x + \alpha) = \text{Var}(x) \quad \text{Var}(\alpha x) = \alpha^2 \text{Var}(x)$$

for a constant  $\alpha$ .

**Multiplicative ensemble** Product of the outputs of two models  $f$  and  $g$ :

$$g(x, \lambda)f(x, \theta)$$

More specifically, the training process aims to obtain:

$$g(x_i, \lambda)f(x_i, \theta) \approx y_i \Rightarrow f(x_i, \theta) \approx \frac{y_i}{g(x_i, \lambda)}$$

In other words,  $f$  is trained on a series with variance altered by  $g$ .

For this specific problem,  $f$  is a Gaussian process and  $g$  a standard deviation model.

**Standard deviation model** A simple standard deviation model consists of mapping time intervals to their standard deviations. This approach is sensitive to the choice of the granularity of the interval (time unit in this problem):

- If it has too many missing values or too little samples, it is not enough to compute a reliable standard deviation.
- If it is too coarse, the computed standard deviation is not useful.

**Remark.** From empirical considerations, the central limit theorem is observable starting from 30 samples. Therefore, 30 data points are enough to make a reasonably stable prediction of the standard deviation.

As the final model might be too coarse, the following can be done:

**Upsampling** Use a finer grain unit (i.e., x-axis, time unit in this problem) and fill missing values through linear interpolation.

**Smoothing** Smooth the upsampled data through a low-pass filter.

For this problem, an exponentially weighted moving average works best as recent data are more relevant.

# 5 Remaining useful life: Turbofan engines

Maintenance can be of three types:

**Reactive maintenance** Repair when something is broken.

**Preventive maintenance** Periodically change something, in a conservative way, before it breaks.

**Predictive maintenance** Change when something is close to break.

Remaining useful life (RUL) is a metric useful for predictive maintenance.

## 5.1 Data

The dataset contains run-to-failure experiments on NASA turbofan engines. Excluding domain specific features, the main columns are:

`machine` Index of the experiment.

`cycle` Time step of the experiment.

`rul` Remaining useful life.

From the dataset heatmap, the following can be observed:

- Rows with a uniform blue or red color represent features that contain frequent short-lived variations (i.e., peeks) that skew the standard deviation.
- Some features show a trend synced with the experiments (see rows around 15 at the y-axis with blue peeks at the end of each experiment).

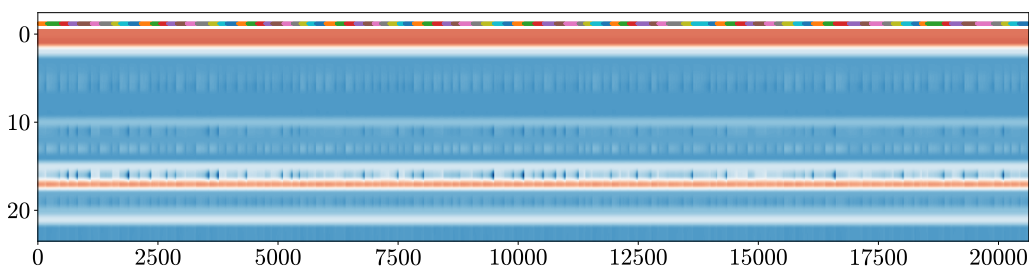


Figure 5.1: Heatmap of the dataset. On the top line, each section represents an experiment.

### 5.1.1 Data splitting

As the dataset is composed of experiments, standard random sampling will mix experiments and leak information. Therefore, sampling is done on the experiments in chronological order (train first).

**Remark.** When splitting, the train data should be representative of the test data. Moreover, the test set should be representative of the real world.

## 5.2 Approaches

### 5.2.1 Regressor

Predict RUL with a regressor  $f$  and set a threshold to trigger maintenance:

$$f(x, \theta) \leq \varepsilon$$

**Linear regression** A linear regressor can be used as a simple baseline for further experiments.

**Remark.** For convenience, a neural network without hidden layers can be used as the regressor.

**Remark.** Training linear models with gradient descent tend to be slower to converge.

**Multi-layer perceptron** Use a neural network to solve the regression problem.

**Remark.** A more complex model causes more training instability as it is more prone to overfitting (i.e., more variance).

**Remark.** Being more expressive, deeper models tend to converge faster.

**Remark (Self-similarity curves).** Exponential curves (e.g., the loss decrease plot) are self-similar. By considering a portion of the full plot, the subplot will look like the whole original plot.

**Remark (Common batch size reason).** A batch size of 32 follows the empirical statistical rule of 30 samples. This allows to obtain more stable gradients as irrelevant noise is reduced.

**Convolutional neural network** Instead of feeding the network a single state, a sequence can be packed as the input using a sliding window as in Section 2.2.2.2. 1D convolutions can then be used to process the input so that time proximity can be exploited.

**Remark.** Using sequence inputs might expose the model to more noise.

**Remark.** Even if the dataset is a time series, it is not always the case (as in this problem) that a sequence input is useful.

**Remark.** The accuracy of the trained regressors are particularly poor at the beginning of the experiments due to the fact that faulty effects are noticeable only after a while. However, we are interested in predicting when a component is reaching its end-of-life. Therefore, wrong predictions when the RUL is high are acceptable.

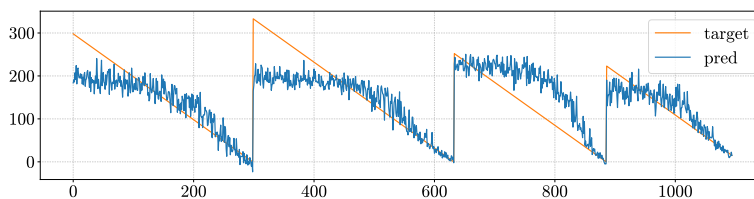


Figure 5.2: Predictions on a portion of the test set



**Cost model** Intuitively, a cost model can be defined as follows:

- The steps of the experiments are considered as the atomic value units.
- The cost for a failure (i.e., untriggered maintenance) is high.
- Triggering a maintenance too early also has a cost.

**Threshold estimation** To determine the threshold  $\varepsilon$  that determines when to trigger a maintenance, a grid-search to minimize the cost model can be performed.

Formally, given a set of training experiments  $K$ , the overall problem to solve is the following:

$$\begin{aligned} \arg \min_{\varepsilon} \sum_{k=1}^{|K|} \text{cost}(f(\mathbf{x}_k, \boldsymbol{\theta}^*), \varepsilon) \\ \text{subject to } \boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \mathcal{L}(f(\mathbf{x}_k, \boldsymbol{\theta}), y_k) \end{aligned}$$

Note that  $\varepsilon$  do not appear in the regression problem. Therefore, this problem can be solved as two sequential subproblems (i.e., regression and then threshold optimization).

## 5.2.2 Classifier

Predict RUL with a classifier  $f_{\varepsilon}$  (for a chosen  $\varepsilon$ ) that determines whether a failure will occur in  $\varepsilon$  steps:

$$f_{\varepsilon}(x, \theta) = \begin{cases} 1 & \text{failure in } \varepsilon \text{ steps} \\ 0 & \text{otherwise} \end{cases}$$

**Remark.** Training a classifier might be easier than a regressor.

**Logistic regression** A logistic regressor can be used as baseline.

**Remark.** For convenience, a neural network without hidden layers can be used as the classifier.

**Multi-layer perceptron** Use a neural network to solve the classification problem.

**Remark.** As classifiers output a probability, they can be interpreted as the probability of not failing.

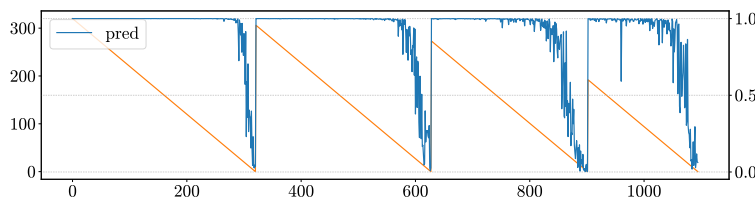
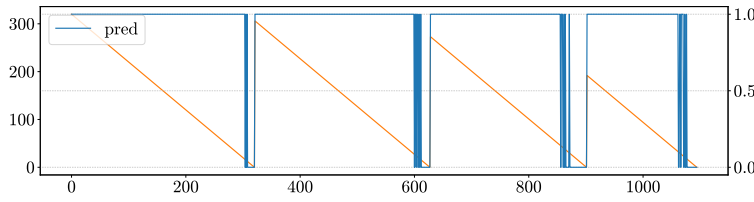


Figure 5.3: Output probabilities of a classifier

By rounding, it is easier to visualize a threshold:



**Cost model** As defined in Section 5.2.1.

### Threshold estimation

**Remark.** The possibility of using a user-defined threshold  $\varepsilon$  allows choosing how close to a failure maintenance has to be done. However, early signs of a failure might not be evident at some thresholds so automatically optimizing it might be better.

Formally, given a set of training experiments  $K$ , the overall problem to solve is the following:

$$\arg \min_{\varepsilon} \sum_{k=1}^{|K|} \text{cost} \left( f(\mathbf{x}_k, \boldsymbol{\theta}^*), \frac{1}{2} \right)$$

$$\text{subject to } \boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \mathcal{L}(f(\mathbf{x}_k, \boldsymbol{\theta}), \mathbb{1}_{y_k \geq \varepsilon})$$

Differently from regression, the threshold  $\varepsilon$  appears in both the classification and threshold optimization problems, so they cannot be decomposed.

**Black-box optimization** Optimization approach based on brute-force.

Black-box optimization

For this problem, black-box optimization can be done as follows:

1. Over the possible values of  $\varepsilon$ :
  - a) Determine the ground truth based on  $\varepsilon$ .
  - b) Train the classifier.
  - c) Compute the cost.

**Remark.** Black-box optimization with grid-search is very costly.

**Bayesian surrogate-based optimization** Class of approaches to optimize a black-box function  $f$  under trivial constraints. It is assumed that  $f$  is expensive to evaluate and a surrogate model (i.e., a proxy function) is instead used to optimize it.

Bayesian surrogate-based optimization

Formally, Gaussian surrogate-based optimization solves problems in the form:

$$\min_{x \in B} f(x)$$

where  $B$  is a box (i.e., hypercube).  $f$  is optimized through a surrogate model  $\hat{f}$  that is trained on some observations from  $f$  (prior). At each iteration, an acquisition function that uses  $\hat{f}$  is used to determine a new point to explore and evaluate with  $f$ . The newly discovered sample (posterior) is used to improve  $\hat{f}$  and the process is repeated.

**Remark.** Under the correct assumptions, the result is optimal.

**Gaussian process surrogate** A good surrogate model should be able to accurately approximate all available training samples and provide a prediction confidence.

A Gaussian process has these properties and can be used as surrogate model.

**Acquisition function** Function to determine which point to explore next. It should account for both predictions and confidence.

**Remark.** Acquisition functions should balance exploration (i.e., area with low predictions) and exploitation (i.e., area with high confidence).

**Lower confidence bound** Acquisition function defined as:

$$\text{LCB}(x) = \mu(x) - Z_\alpha \sigma(x)$$

where  $\mu(x)$  and  $\sigma(x)$  are the predicted mean and standard deviation, respectively.  $Z_\alpha$  is a multiplier for the confidence interval.

Given a set of training samples  $\{x_i, y_i\}$  and a black-box function  $f$ , surrogate-based optimization does the following:

1. Until a termination condition is not met:
  - a) Train a surrogate model  $\hat{f}$  on  $\{x_i, y_i\}$  to approximate  $f$ .
  - b) Optimize an acquisition function  $a_{\hat{f}}(x)$  to find the next data point  $x'$  to explore.
  - c) Evaluate  $x'$  with the black-box function  $y' = f(x')$ .
  - d) Store  $y'$  if it is the current best optimum for  $f$ .
  - e) Add  $(x', y')$  to the training set.

### 5.2.3 Survival analysis (regression)

**Remark.** Chronologically, this approach has been presented after Chapter 7.

**Survival analysis model** Probabilistic model to estimate the survival time of an entity.

Survival analysis model

**Survival analysis formalization** Consider a random variable  $T$  to model the survival time. The simplest model can be defined as:

$$t \sim \mathcal{P}(T)$$

Remaining survival time depends on the time  $\bar{t}$ , therefore we have that:

$$t \sim \mathcal{P}(T | \bar{t})$$

Remaining survival time also depends on the past sensor readings  $X_{\leq \bar{t}}$  and future readings  $X_{> \bar{t}}$  (at this stage, we want to capture what affects the survival time, even if we do not have access to some variables), therefore we have that:

$$t \sim \mathcal{P}(T | \bar{t}, X_{\leq \bar{t}}, X_{> \bar{t}})$$

**Marginalization** Average over all the possible outcomes of a random variable to cancel it out.

For this problem, we do not have access to the future sensor readings, so we can marginalize them:

$$t = \mathbb{E}_{X_{> \bar{t}} \sim \mathcal{P}(X_{> \bar{t}})} [\mathcal{P}(T | \bar{t}, X_{\leq \bar{t}})]$$

**Remark.** In probabilistic terms, the regression approach previously taken can be modelled as:

$$t \sim \mathcal{N}(\mu(X_{\bar{t}}), \sigma)$$

where  $\mu(\cdot)$  is the regressor.

Compared to the survival analysis model, there are the following differences:

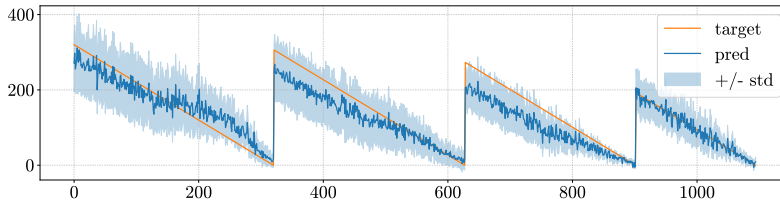
- Regressor only reasons on the sensor readings  $X_{\bar{t}}$  at a single time step.
- Regressor does not consider the current time.
- Regressor assumes normal distribution with constant variance.

**Neuro-probabilistic model** We can assume that the model follows a normal distribution parametrized on both the mean and standard deviation:

$$t \sim \mathcal{N}(\mu(X_{\bar{t}}, \bar{t}), \sigma(X_{\bar{t}}, \bar{t}))$$

**Remark.** The readings of a single time step are used as it can be shown that using multiple time steps does not yield significant improvements for this dataset.

**Architecture** Use a neural network that output both  $\mu$  and  $\sigma$  that are passed to a distribution head.



## 5.2.4 Survival analysis (classification)

**Censoring** Hide key events from the dataset.

Censoring

**Remark.** For this dataset, it is more realistic to have more partial runs than run-to-failure experiments as they are expensive to obtain.

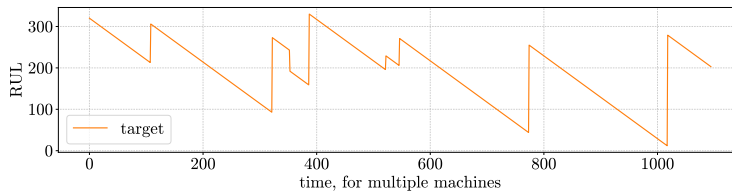


Figure 5.4: Example of partial runs

**Survival function** Given the random variable  $T$  to model survival time, the survival function  $S$  is defined as:

Survival function

$$S(\bar{t}) = \mathcal{P}(T > \bar{t})$$

In other words, it is the probability of surviving at least until time  $\bar{t}$ .

For this problem, the survival function can account for past sensor readings  $X_{\leq \bar{t}}$ :

$$S(\bar{t}, X_{\leq \bar{t}}) = \mathcal{P}(T > \bar{t} | X_{\leq \bar{t}})$$

**Hazard function** Given the random variable  $T$  to model survival time, the hazard function  $\lambda$  is defined as:

$$\lambda(\bar{t}) = \mathcal{P}(T < \bar{t} \mid T > \bar{t} - 1)$$

In other words, it is the conditional probability of not surviving at time  $\bar{t}$  knowing that the entity survived until time  $\bar{t} - 1$ .

With discrete time, the survival function can be factorized using the hazard function:

$$S(\bar{t}) \approx (1 - \lambda(\bar{t})) \cdot (1 - \lambda(\bar{t} - 1)) \cdot \dots$$

**Remark.** The hazard function only depends on one observation.

**Remark.** The fact that the probability of not surviving is used is for historical reasons.

For this problem, the hazard function is the following:

$$\begin{aligned} \lambda(\bar{t}, X_{\bar{t}}) &= \mathcal{P}(T < \bar{t} \mid T > \bar{t} - 1, X_{\bar{t}}) \\ S(\bar{t}, X_{\bar{t}}) &\approx (1 - \lambda(\bar{t}, X_{\bar{t}})) \cdot (1 - \lambda(\bar{t} - 1, X_{\bar{t}-1})) \cdot \dots \end{aligned}$$

**Hazard estimator** We want to train an estimator  $\hat{\lambda}_{\theta}(\bar{t}, x_{\bar{t}})$  for the hazard function.

Consider the  $k$ -th experiment that ended at time  $e^{(k)}$ . The probability of the survival event can be modelled as follows:

$$\underbrace{\hat{\lambda}_{\theta}(e^{(k)}, x_{e^{(k)}})}_{\text{Not surviving at time } e^{(k)}} \underbrace{\prod_{t=1}^{e^{(k)}-1} (1 - \hat{\lambda}_{\theta}(t, x_t^{(k)}))}_{\text{Surviving until time } e^{(k)} - 1}$$

**Training** In likelihood maximization terms, the problem for  $m$  experiments is formulated as:

$$\arg \max_{\theta} \prod_{k=1}^m \left( \hat{\lambda}_{\theta}(e^{(k)}, x_{e^{(k)}}) \prod_{t=1}^{e^{(k)}-1} (1 - \hat{\lambda}_{\theta}(t, x_t^{(k)})) \right)$$

Let  $d_t^{(k)} = 1 \iff t = e^{(k)}$  (i.e., 1 when not surviving at time  $t$ ), the problem can be rewritten as:

$$\begin{aligned} &\arg \max_{\theta} \prod_{k=1}^m \prod_{t=1}^{e^{(k)}} d_t^{(k)} \hat{\lambda}_{\theta}(t, x_t^{(k)}) + (1 - d_t^{(k)}) (1 - \hat{\lambda}_{\theta}(t, x_t^{(k)})) \\ &= \arg \max_{\theta} \sum_{k=1}^m \sum_{t=1}^{e^{(k)}} \log \left( d_t^{(k)} \hat{\lambda}_{\theta}(t, x_t^{(k)}) + (1 - d_t^{(k)}) (1 - \hat{\lambda}_{\theta}(t, x_t^{(k)})) \right) \\ &= \arg \max_{\theta} \sum_{k=1}^m \sum_{t=1}^{e^{(k)}} d_t^{(k)} \log \left( \hat{\lambda}_{\theta}(t, x_t^{(k)}) \right) + (1 - d_t^{(k)}) \log \left( 1 - \hat{\lambda}_{\theta}(t, x_t^{(k)}) \right) \\ &= \arg \min_{\theta} - \sum_{k=1}^m \sum_{t=1}^{e^{(k)}} d_t^{(k)} \log \left( \hat{\lambda}_{\theta}(t, x_t^{(k)}) \right) + (1 - d_t^{(k)}) \log \left( 1 - \hat{\lambda}_{\theta}(t, x_t^{(k)}) \right) \end{aligned}$$

Which corresponds to a binary cross-entropy minimization problem where  $d_t^{(k)}$  can be seen as the class. Therefore,  $\hat{\lambda}_{\theta}(t, x_t^{(k)})$  can be seen as a classifier.

**Remark.** This shows that the classification approach used in Section 5.2.2 is not strictly wrong. Instead of predicting whether the machine fails in  $\varepsilon$  time steps, in probabilistic terms, the correct approach is to consider whether the machine fails at a time step  $t$  knowing it survived at time  $t - 1$ .

**Remark.** Censored data skews the distribution of the data. Sample weights can be used to deal with the issue.

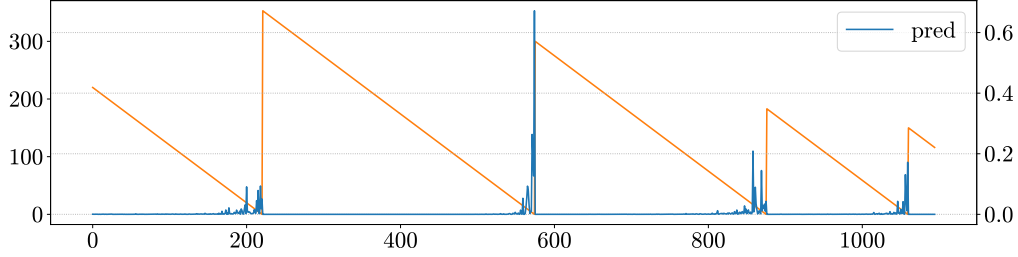


Figure 5.5: Failure probability of the classifier

**Inference** Given a threshold  $\varepsilon$ , maintenance is triggered when:

$$\hat{\lambda}(t, x_t) \geq \varepsilon$$

**Forecasting** Determine the probability of surviving  $n$  more steps as:

$$\begin{aligned} \frac{S(t+n)}{S(t)} &= \prod_{h=0}^n (1 - \lambda(t+h, X_{t+h})) \\ &\approx \prod_{h=0}^n (1 - \hat{\lambda}(t+h, x_{t+h}^{(k)})) \end{aligned}$$

However, this would require accessing future values of  $X_t$ . Possible workarounds are:

- Ignore time-varying inputs (i.e., sort of marginalization) so that  $\hat{\lambda}(t+h, x_{t+h}) \approx \hat{\lambda}(t+h, x^{(k)})$  where  $x^{(k)}$  only contains stable information.
- Predict future values of  $x_t^{(k)}$  by training another estimator.
- Assume that all the variables  $x_t^{(k)}$  are stable for some time so that:

$$\frac{S(t+n)}{S(t)} \approx \prod_{h=0}^n (1 - \hat{\lambda}(t+h, x_t^{(k)}))$$

# 6 Component wear anomalies: Skin wrapper machines

## 6.1 Data

The dataset represents a single run-to-failure experiment of a skin wrapper machine with a 4 ms sampling period and different segments (i.e., operating mode).

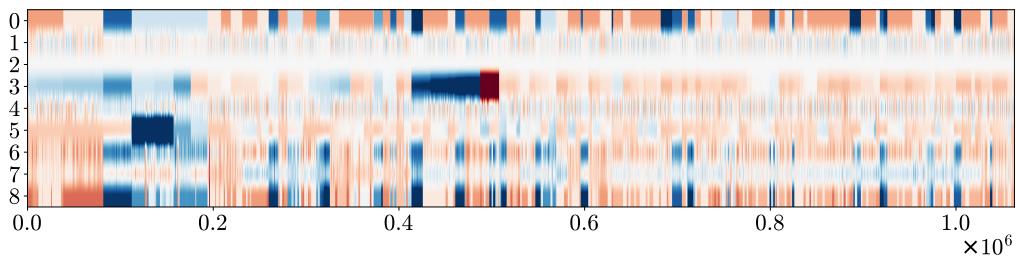
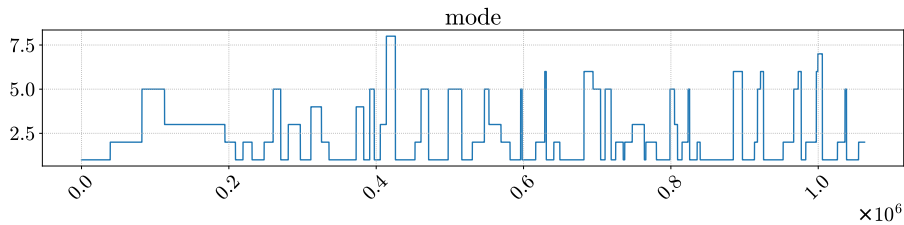


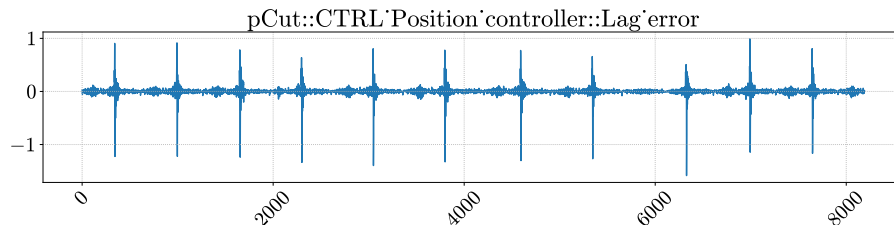
Figure 6.1: Dataset heatmap

The following observations can be made:

- Some features (i.e., rows 0 and 8 of the heatmap) are fixed for long periods of time (i.e., piece-wise constant). They are most likely controlled parameters.

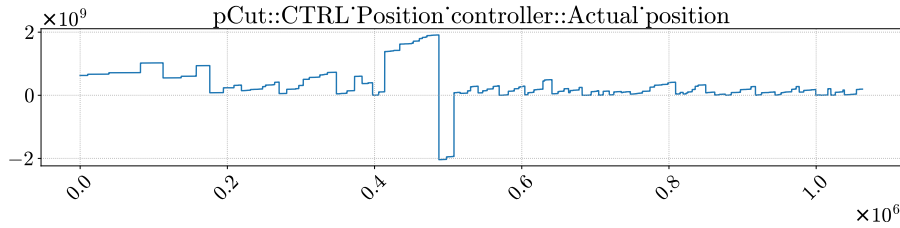


- A feature (i.e., row 2 of the heatmap) seems constant but in reality it has many short-lived peaks.



**|Remark.** This type of signal might be useful to determine a period in the series.

- Some features (i.e., rows 3 and 5 of the heatmap) contain sudden localized deviations. This is most likely due to external interventions.



### 6.1.1 Binning

**Binning** Reduce the granularity of the data using a non-overlapping sliding window and aggregation functions. Binning

**Remark.** With high-frequency data, some approaches (e.g., a neural network) might not be able to keep up if real-time predictions are required.

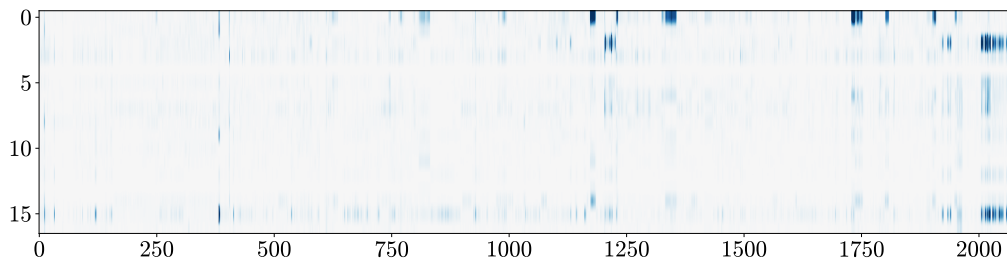
**Remark.** After binning, the number of samples is reduced, but the number of features might increase.

## 6.2 Approaches

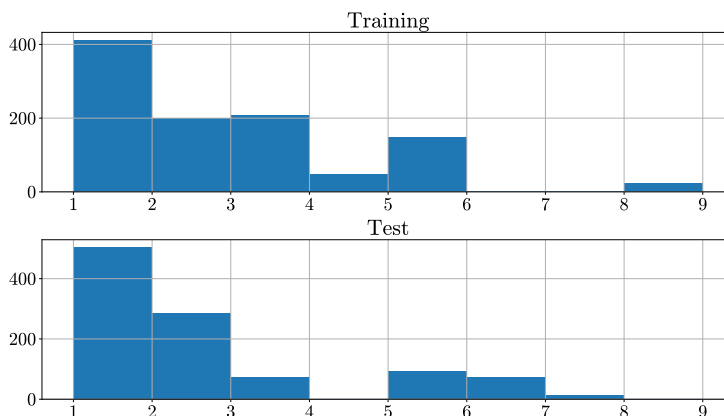
### 6.2.1 Autoencoder

**Autoencoder** Train an autoencoder on earlier non-anomalous data and use the reconstruction error to detect component wear.

Results show that there is a high reconstruction error for the operating mode feature (i.e., first row), which is a controlled parameter. This hints at the fact that some operating modes are too infrequent, so the model is unable to reconstruct them.



**Remark.** There is a distribution drift (i.e., future does not behave like the past) in the dataset.





**Remark.** To account for distribution drift, chronological splitting for the training and test sets is a better approach instead of random sampling (which ignores the drift). The validation set can be created through random sampling as, in case it is created chronologically, it might create a gap between training and test data. However, this approach ignores drift when validating.

## 6.2.2 Class rebalancing

**Remark.** Consider two training samples  $\{(x_1, y_1), (x_2, y_2)\}$ , the optimization problem over an objective function  $f_\theta$  would be:

$$\arg \max_{\theta} f_{\theta}(y_1|x_1) f_{\theta}(y_2|x_2)$$

If the second sample  $(x_2, y_2)$  appears twice, the problem is:

$$\begin{aligned} & \arg \max_{\theta} f_{\theta}(y_1|x_1) f_{\theta}(y_2|x_2)^2 \\ & = \arg \max_{\theta} f_{\theta}(y_1|x_1)^{\frac{1}{3}} f_{\theta}(y_2|x_2)^{\frac{2}{3}} \end{aligned}$$

**Importance sampling** Given an empirical risk minimization problem:

Importance sampling

$$\arg \min_{\theta} - \prod_{i=1}^m f_{\theta}(y_i|x_i)$$

Each training sample can have associated a different weight  $w_i$ :

$$\begin{aligned} & \arg \min_{\theta} - \prod_{i=1}^m f_{\theta}(y_i|x_i)^{w_i} \\ & = \arg \min_{\theta} - \sum_{i=1}^m w_i \log(f_{\theta}(y_i|x_i)) \end{aligned}$$

The weights can be seen as a ratio:

$$w_i = \frac{p_i^*}{p_i}$$

where:

- $p_i$  is the sampling bias that has to be canceled out.
- $p_i^*$  is the target distribution to emulate.

In this problem, we can define the weights as follows:

$$\begin{aligned} p_i &= \frac{1}{n} |\text{samples with the same operating mode as } x_i| \\ p_i^* &= \frac{1}{n} \end{aligned}$$

**Remark.** As uniform distribution is assumed, the detector will not be sensitive to the mode.

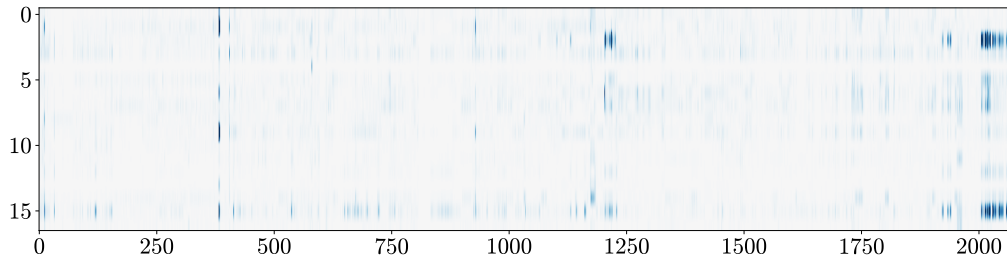


Figure 6.2: Autoencoder feature reconstruction error with importance sampling. Note that the operation mode (row 0) now has fewer errors.

### 6.2.2.1 Importance sampling applications

**Class rebalancing** Oversample or undersample training samples based on the class unbalance.

**Remark.** When evaluating a model trained on a rebalanced training set, accuracy might not be the ideal metric. A cost model or confusion matrix can be used.

**Remark.** Rebalancing should not be used when the data is unbalanced but representative of the real distribution.

**Remove bias from continuous attributes**  $p_i$  and  $p_i^*$  can be probability densities. Therefore, with a continuous feature, it is sufficient to estimate  $p_i$  using a density estimator.

**Remark.** It can be useful to clip  $p_i$  for numerical stability:

$$p_i = \max(l, \min(u, f(x_i, y_i)))$$

**Remove bias from external attributes** If the dataset is generated from biased external sources, it is possible to attempt to debias it by determining the probability that a sample belong to the dataset and use it as  $p_i$ .

**Example.** A dataset for organ transplants already contains patients for which doctors think the surgery is more likely to be successful.

$p_i$  can be the probability that the patient has been selected to be an organ receiver.

**Sample-specific variance** Importance sampling applied with MSE allows to remove its homoscedasticity (i.e., allow non-constant variance).

Consider MSE formulated as follows:

$$\arg \min_{\theta} - \sum_{i=1}^m \log \left( \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{1}{2} (y_i - h_{\theta}(x_i))^2 \right) \right)$$

By adding as weights  $\frac{1}{\hat{\sigma}_i^2}$ , the problem becomes:

$$\begin{aligned} \arg \min_{\theta} - \sum_{i=1}^m \frac{1}{\hat{\sigma}_i^2} \log \left( \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{1}{2} (y_i - h_{\theta}(x_i))^2 \right) \right) \\ = \arg \min_{\theta} - \sum_{i=1}^m \log \left( \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{(y_i - h_{\theta}(x_i))^2}{2\hat{\sigma}_i^2} \right) \right) \end{aligned}$$

In other words, the weight in the form  $\frac{1}{\hat{\sigma}_i^2}$  represents the inverse sample variance. Therefore, it is possible to specify a different variance for each sample.

# 7 Arrivals prediction: Hospital emergency room

## 7.1 Data

The dataset contains accesses to the emergency room of the Maggiore Hospital in Bologna. Each row of the dataset represents a patient and the features are:

**Triage** Time of arrival.

**TKCharge** Time of first visit.

**Code** Priority (white, green, yellow, and red).

**Outcome** Indicates whether the patient got admitted or left.

**Binning** As the problem is to predict the total number of arrivals at fixed intervals, binning can be used to obtain a dataset with an hour granularity.

## 7.2 Approaches

**Remark.** MSE assumes that the conditional distribution  $\mathcal{P}(y|x;\theta)$  of the predictions follows a normal distribution (i.e.,  $y \sim \mathcal{N}(\mu(x;\theta), \sigma)$ ).

**Remark.** Arrivals usually follow a Poisson distribution as:

- There is a skew in the tail.
- All values are positive.
- Events are independent.

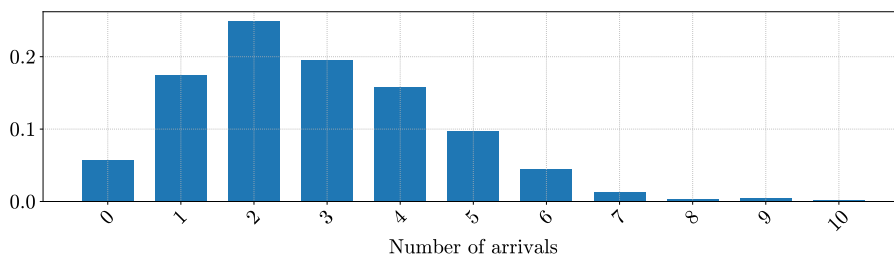


Figure 7.1: Arrivals distribution at 6 a.m.

**Theorem 7.2.1.** If the inter-arrival time is exponential with respect to a fixed rate, the counts of the arrivals will always follow a Poisson distribution.

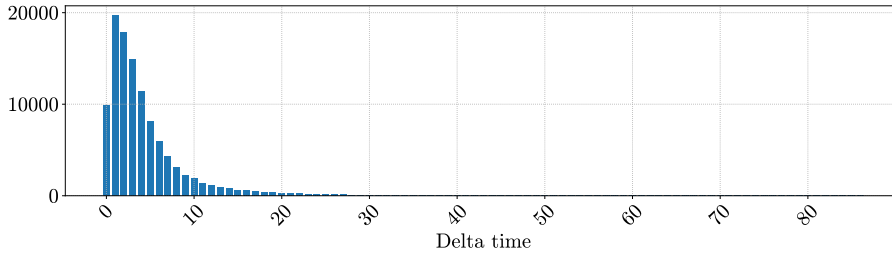


Figure 7.2: Dataset inter-arrival counts (the first bar is due to binning artifacts)

### 7.2.1 Neuro-probabilistic model

**Poisson distribution** Distribution with discrete support whose probability mass function is defined as:

Poisson distribution

$$p(k, \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

where  $\lambda$  is the occurrence rate of the events.

**Remark.** Both mean and standard deviation of a Poisson distribution are  $\lambda$ .

**Remark.**  $\lambda^{-\frac{1}{2}}$  represent the distribution skewness. For smaller values of  $\lambda$ , there is a positive skew to the left. For larger values of  $\lambda$ , the skew becomes less observable.

**Remark.** For this problem,  $\lambda$  is the average number of arrivals in each bin.

**Neuro-probabilistic model** Model that combines statistics and machine learning.

Neuro-probabilistic model

**Remark.** This class of models is known in the statistics literature as generalized linear model. With neural networks (i.e., non-linearity), they do not have an official name. “Neuro-probabilistic model” is an unofficial name.

This problem can be formulated through the following probabilistic model:

$$y \sim \text{Poisson}(\lambda(x))$$

where  $y$  is the number of arrivals and  $\lambda(x)$  is the rate parametrized on the temporal information  $x$ .

The rate can be approximated using an estimator as:

$$y \sim \text{Poisson}(\lambda(x, \theta))$$

where  $\lambda(x, \theta)$  is a regression model.

**Loss** Training is done for maximum likelihood estimation using the Poisson distribution:

$$\begin{aligned} & \arg \min_{\theta} - \sum_{i=1}^m \log(f(y_i, \lambda(x_i, \theta))) \\ &= \arg \min_{\theta} - \sum_{i=1}^m \log \left( \frac{\lambda(x_i, \theta)^{y_i} e^{-\lambda(x_i, \theta)}}{y_i!} \right) \end{aligned}$$

**Architecture** Use an MLP to predict  $\hat{\lambda}$  and then, instead of outputting the prediction directly, output a Poisson distribution object with rate  $\hat{\lambda}$ :

$$\begin{aligned} \hat{\lambda} &= \text{MLP}(x) \\ \text{out} &= \text{Poisson}(\cdot, \hat{\lambda}) \end{aligned}$$

Some considerations must be made:

**Only positive rates** As  $\hat{\lambda}$  must be positive, it is possible to combine a logarithm (i.e., assume that the MLP outputs a log-rate) and an exponentiation to achieve this:

$$\begin{aligned} \log(\hat{\lambda}) &= \text{MLP}(x) \\ \text{out} &= \text{Poisson}\left(\cdot, \exp\left(\log(\hat{\lambda})\right)\right) = \text{Poisson}(\cdot, \hat{\lambda}) \end{aligned}$$

| **Remark.** A strictly positive activation function can also be used.

**Standardization** The input of the network can be standardized. On the other hand, standardizing the output is wrong as the Poisson distribution is discrete.

However, if the input is standardized (for training stability), the output of the network with normal weights initialization will have 0 mean. Therefore, at the beginning, the network will predict  $\hat{\lambda} \approx 1$  (i.e., the MLP predicts  $\log(\hat{\lambda}) \approx 0$  and then  $\hat{\lambda} = \exp\left(\log(\hat{\lambda})\right) \approx 1$ ) which might not be a reasonable starting point.

It is possible to provide an initial guess  $\bar{\lambda}$  for  $\hat{\lambda}$ . This value can be used as a multiplicative factor, so that the first prediction will be close to  $\bar{\lambda} \cdot 1$ :

$$\begin{aligned} \log(\hat{\lambda}) &= \text{MLP}(x) \\ \text{out} &= \text{Poisson}\left(\cdot, \bar{\lambda} \exp\left(\log(\hat{\lambda})\right)\right) = \text{Poisson}(\cdot, \bar{\lambda}\hat{\lambda}) \end{aligned}$$

| **Remark.** For this problem  $\bar{\lambda}$  can be the average number of arrivals in each bin.

**Remark.** With one-hot encoding, a linear model can be non-linearized into a lookup table.

**Example.**

Consider the dataset:      A linear model learns a straight line:

$x$	$y$
0	1
1	4
2	2

$$f(x) = \alpha x$$

With the dataset:      A linear model learns a linear combination:

$x_0$	$x_1$	$x_2$	$y$
1	0	0	1
0	1	0	4
0	0	1	2

$$f(x) = \alpha x_0 + \beta x_1 + \gamma x_2$$

Which allows to easily learn the dataset with  $\alpha = 1$ ,  $\beta = 4$ , and  $\gamma = 2$

| **Remark.** Having access to the distribution allows to compute all possible statistics. For instance, it is possible to plot mean and standard deviation.

# 8 Feature selection and important: Biomedical analysis

## 8.1 Data

The dataset contains anonymized biomedical data and is composed of a binary target variable and unknown variables.

**Remark.** In reality, this dataset contains real-world examples. Features have been anonymized for the purpose of feature analysis.

### 8.1.1 Preliminary analysis

**Data distribution** There are both categorical and numerical features.

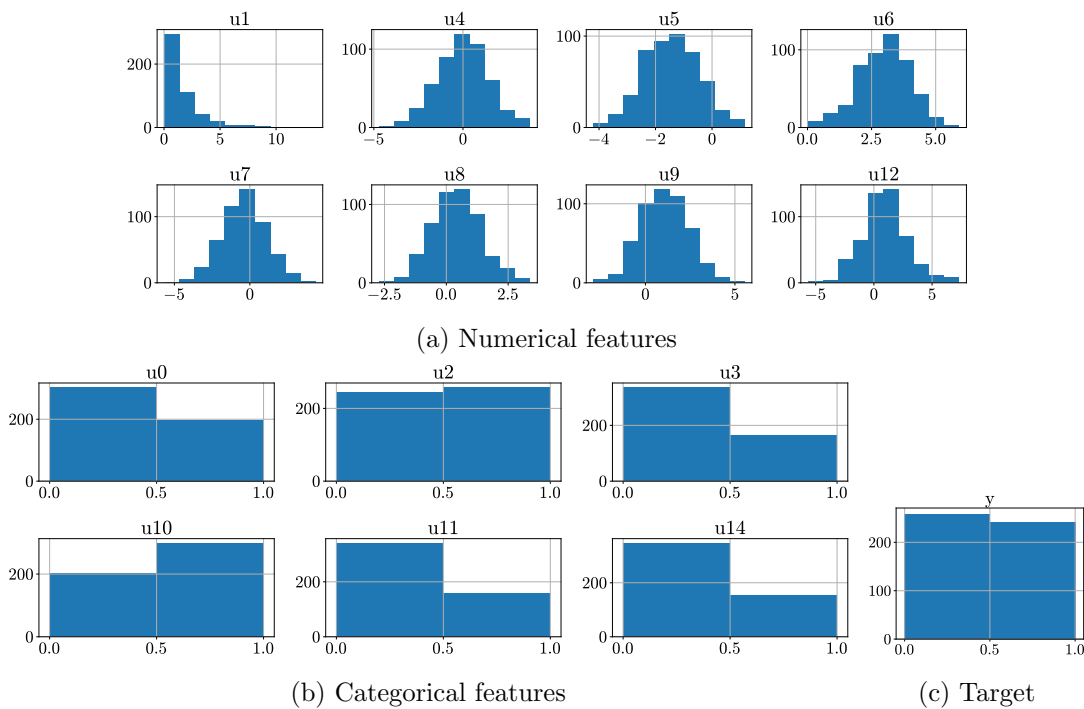
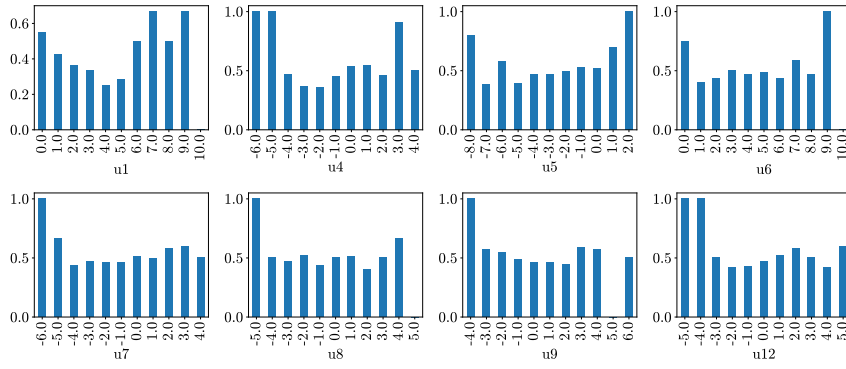
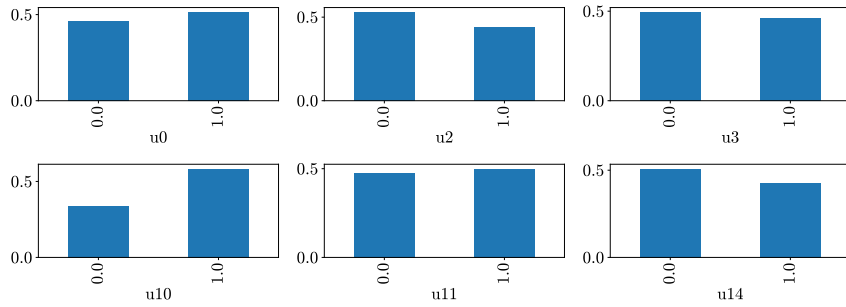


Figure 8.1: Distribution of the dataset

**Univariate dependencies** Determine the fraction of examples with target  $Y = 1$  (i.e., likelihood that a feature has a specific value while the target is 1).



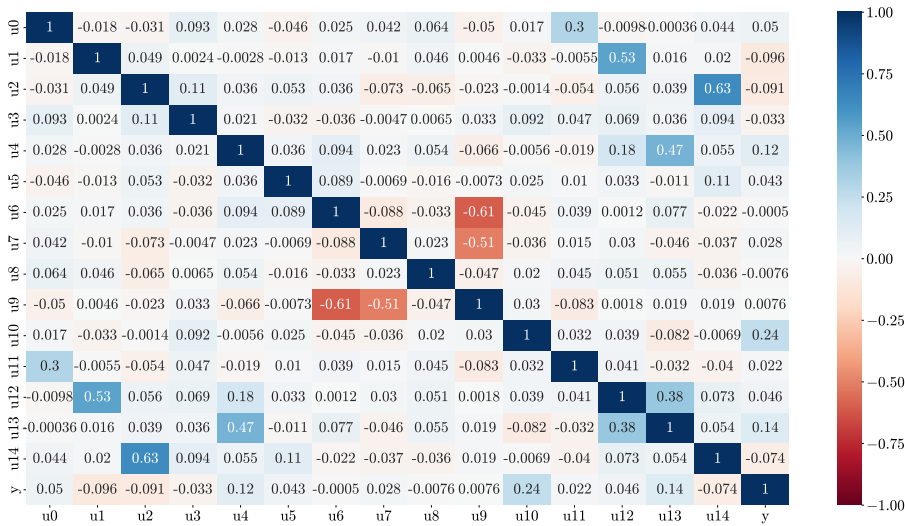
(a) Numerical features



(b) Categorical features

Figure 8.2: Univariate dependencies with  $Y = 1$

**Linear correlation** Determine the Pearson's correlation between variables.



## 8.2 Approaches

### 8.2.1 Linear model

**Linear model for explanation** Use a linear model (logistic regression in this case) as a proxy to explain the features.

Linear model for explanation

**Remark.** Preventing overfitting is important as the model has to generalize well. L1 regularization ( $\alpha\|\theta\|_1$ ) can help by:

- Preventing overfitting.
- Sparsifying the coefficients (i.e., make them mostly close to 0). The hyperparameter  $\alpha$  controls this behavior.

**Example.** Consider the case of MSE with L1 regularization. The gradient of L1 is never 0 and is upper-bounded whereas the gradient of MSE can be 0 and can go to  $\infty$ . This produces the following phenomenon:

- When close to 0, the slope of  $\nabla\text{MSE}$  is larger and outweighs  $\nabla\text{L1}$ , making updates follow MSE.
- When moving away from 0,  $\nabla\text{MSE}$  becomes smaller up to a point where it balances with  $\nabla\text{L1}$ . At this point, updating that parameter does not improve the loss anymore.

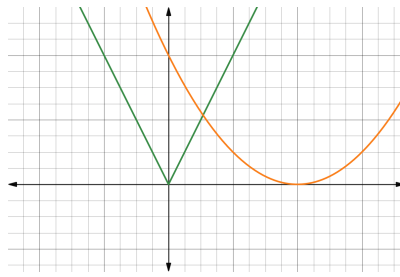


Figure 8.3: MSE (orange) and L1 (green)

**Remark.** ROC-AUC is a good metric for non-deterministic classification problems. While accuracy is more suited for deterministic approaches.

**Coefficient analysis** By using a linear model, its coefficients can be analyzed to determine the weights of the features.

**Remark.** For the weights to make sense, it is important to normalize the features.

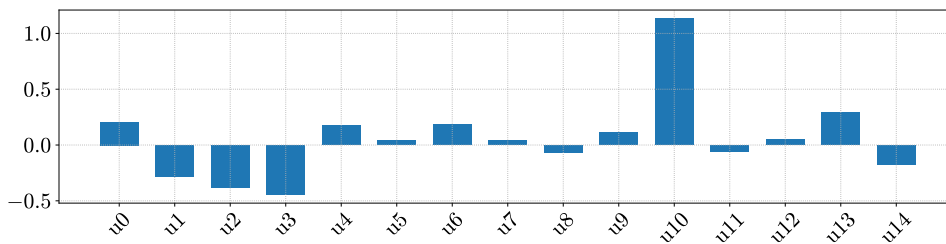


Figure 8.4: Coefficients for this problem

**Issues** A linear model has several limitations for explainability:

- Being linear, it might not have good performance on the training data. Therefore, feature analysis might not be accurate.
- Only linear correlations can be captured.



- Coefficients are not necessarily sparse as L1 has to balance between sparsification and overfitting.

**Remark.** A linear model approach is similar to using the correlation matrix. The only difference is that:

- Pearson's coefficient is computed between a pair of variables.
- A linear model considers all the variables at once.

## 8.2.2 Non-linear model

**Non-linear model for explanation** Use a non-linear model (gradient boosted trees in this case) as a proxy to explain the features.

Non-linear model for explanation

**Remark.** Compared to linear models, non-linear models might have better performance but are less explainable and risks overfitting.

**Features importance analysis** By using trees, feature importances can be computed.

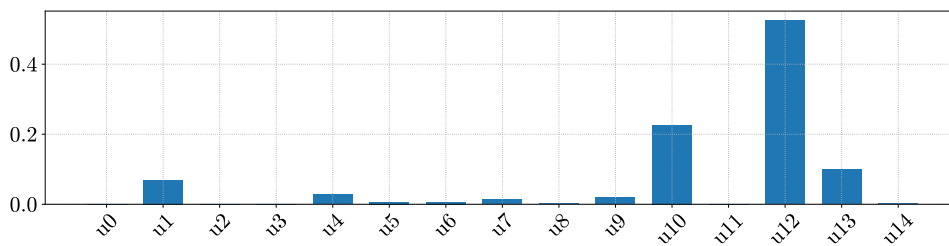


Figure 8.5: Feature importances (gain) for this problem

**Remark.** Feature importance can be computed using different metrics which will not necessarily produce the same results. Some possible metrics are:

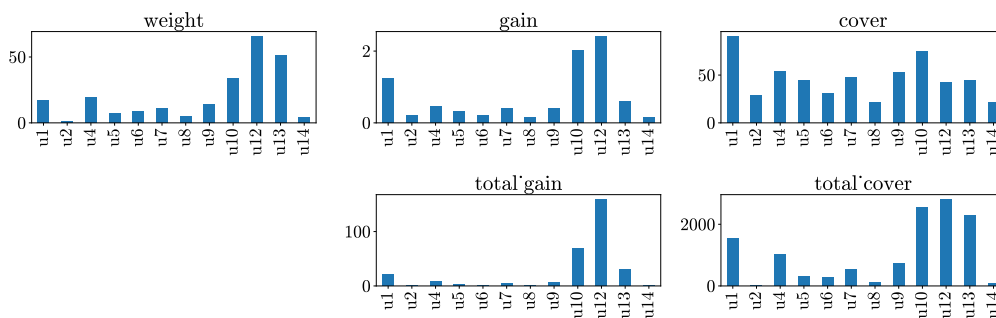
**Weight** The number of times a feature has been used to split.

**Gain** Average gain from the splits of a feature.

**Cover** Average number of examples for which a feature is used as decision criteria.

**Total gain** Sum of the gains from the splits of a feature.

**Total cover** Sum of the number of examples for which a feature is used as decision criteria.



Moreover, feature importances are associated with the training data. Therefore, it risks overfitting.

**Permutation importance analysis** Given:

- A trained model,
- An evaluation metric,
- A reference dataset  $\{x_i, y_i\}_{i=1}^m$ .

Permutation importance for the  $j$ -th feature is computed by randomly permuting the  $j$ -th column in the dataset and computing the difference between the performance on the original and permuted dataset.

For more statistical relevance (i.e., remove spurious correlations), the process can be repeated multiple times and presented using mean and standard deviation.

**Remark.** Permuting the values of a feature aims at preserving the distribution of the column while breaking all correlations.

**Remark.** This approach works on black box models. However, it tends to be expensive.

**Remark.** If the reference dataset is the training set, permutation importance can be used to determine how the model is using the data. On the other hand, by using the test set, the scores will reflect the correlation between the attributes and the target.

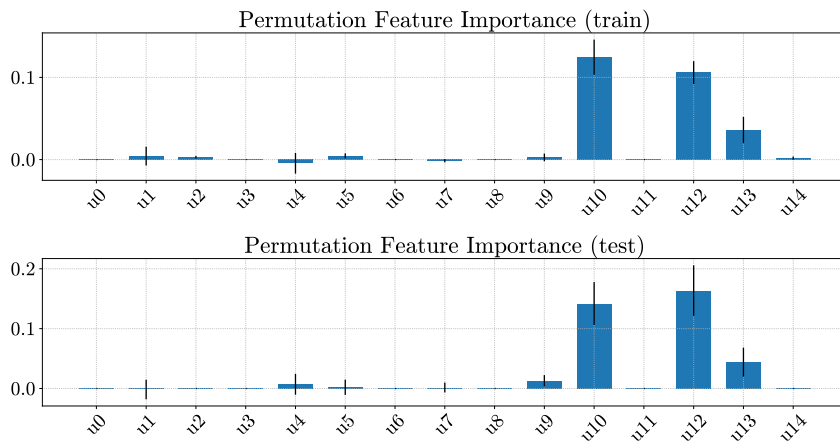


Figure 8.6: Permutation importance on train and test set.  $u_{10}$  is slightly higher on the train set, which indicates that it might be causing a bit of overfitting.

**Remark.** Compared to linear models, non-linear models do not have the capability of identifying the direction of correlation.

### 8.2.3 Additive feature attribution

**Linear models** Given a linear model, an individual example  $\mathbf{x}$  can be explained using an additive attribution model as the difference between the prediction and average prediction over the distribution:

$$\boldsymbol{\theta}^T \mathbf{x} - \mathbb{E}_{\mathbf{x}' \in P(\mathbf{X})} [\boldsymbol{\theta}^T \mathbf{x}']$$

$\mathbb{E}_{\mathbf{x}' \in P(\mathbf{X})} [\boldsymbol{\theta}^T \mathbf{x}']$  can be approximated as the average prediction on the training data and it represents the prediction that the model would make if no information is provided.

Due to linearity, the formula can be rewritten as follows:

$$\begin{aligned}\boldsymbol{\theta}^T \mathbf{x} - \mathbb{E}_{\mathbf{x}' \in P(\mathbf{X})} [\boldsymbol{\theta}^T \mathbf{x}'] &= \boldsymbol{\theta}^T (\mathbf{x} - \mathbb{E}_{\mathbf{x}' \in P(\mathbf{X})} [\mathbf{x}']) \\ &= \sum_{j=1}^n \boldsymbol{\theta}_j (\mathbf{x}_j - \mathbb{E}_{\mathbf{x}' \in P(\mathbf{X})} [\mathbf{x}'_j]) \\ &= \sum_{j=1}^n \phi_j(\mathbf{x})\end{aligned}$$

where  $\phi_j(\mathbf{x}) = \boldsymbol{\theta}_j (\mathbf{x}_j - \mathbb{E}_{\mathbf{x}' \in P(\mathbf{X})} [\mathbf{x}'_j])$  is the **effect** of the  $j$ -th attribute on the output for the example  $\mathbf{x}$  (i.e., how much it moves from the baseline prediction). Effect

**Non-linear models** Given a non-linear model, an example  $\mathbf{x}$  can be explained using an additive attribution model defined as follows:

$$g(\mathbf{z}, \mathbf{x}) = \phi_0 + \sum_{j=1}^n \phi_j(\mathbf{x}) \mathbf{z}_j$$

where:

- $\phi_0$  is the baseline prediction (i.e., average prediction over the dataset).
- $\phi_j$  is the effect of the  $j$ -th feature.
- $\mathbf{z}_j \in \{0, 1\}$  indicates whether the  $j$ -th feature is known.

In practice, given an example  $\mathbf{x}$  and the flags  $\mathbf{z}$ , a linear explanation model is trained locally to tune the effect coefficients  $\phi_j$  of  $g(\mathbf{z}, \mathbf{x})$ .

**Shapely values** Closed-form formula to determine the effect coefficients. Given the set of all input features  $\mathcal{J}$ , the Shapely value of the  $j$ -th feature is computed as: Shapely values

$$\phi_j(\mathbf{x}) = \sum_{\mathcal{S} \subset \mathcal{J} \setminus j} \frac{|\mathcal{S}|!(n - |\mathcal{S}| - 1)!}{n!} (\hat{f}(\mathbf{x}_{\mathcal{S} \cup j}) - \hat{f}(\mathbf{x}_{\mathcal{S}}))$$

Intuitively, by fixing the  $j$ -th attribute,  $\phi_j$  is computed by considering all the possible combinations of the  $\mathbf{z}_i$  values of the other attributes (i.e., different subsets  $\mathcal{S}$  of the features without  $j$ ) and averaging the difference between the prediction that uses as features  $\mathcal{S} \cup j$  and the baseline that only uses  $\mathcal{S}$ .

**Remark.** Shapely values are expensive to compute due to the exponential number of terms. Moreover, many models do not support missing values.

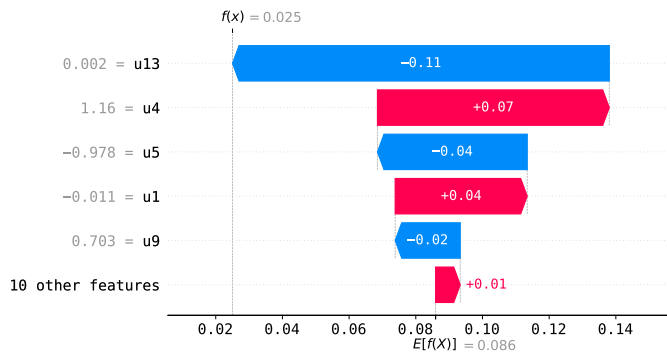
**SHAP** Framework to approximate Shapely values for black-box models. SHAP

**Kernel SHAP** Given an example  $\mathbf{x}$ , kernel SHAP does the following:

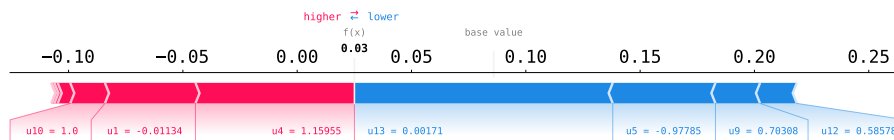
1. Sample multiple  $\mathbf{z}'$  vectors from  $\{0, 1\}^n$ .
2. For every sampled vector, construct an example  $\mathbf{x}'$  such that:
  - $\mathbf{x}'_j = \mathbf{x}_j$  if  $\mathbf{z}'_j = 1$ .
  - $\mathbf{x}'_j$  is sampled from a background set (e.g., training set) if  $\mathbf{z}'_j = 0$ .
3. Train a linear model on the created examples and determine Shapely values using it.

**Plots** Shapely values can be visualized through several plots:

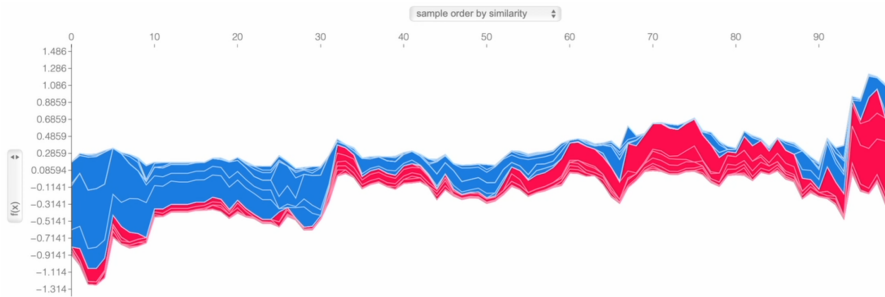
**Waterfall plot** Shows the most relevant features that allowed to reach the prediction  $f(x)$  starting from the baseline  $E[f(x)]$ .



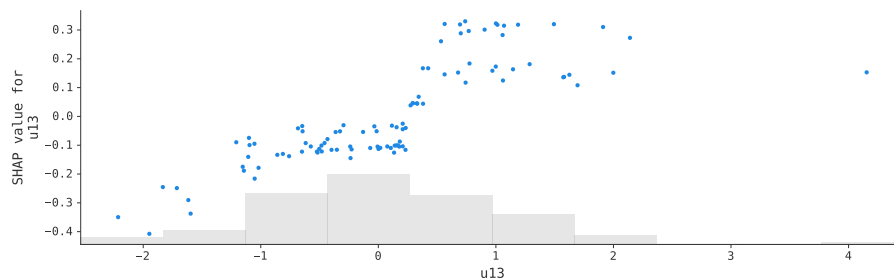
**Force plot** Compact version of waterfall plots.



**Global force plot** Stack of force plots where the  $x$ -axis represents the examples.

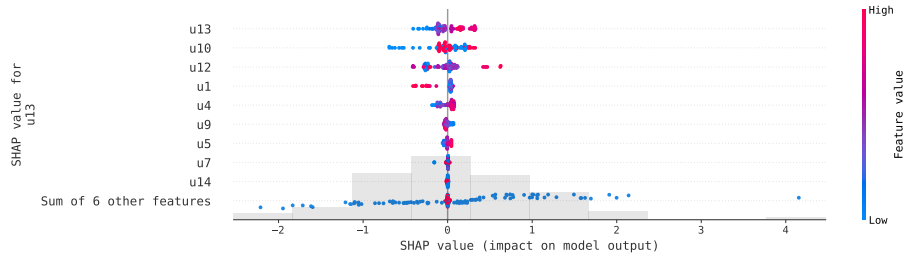


**Scatter plot** Links the values of an attribute to its Shapely values.



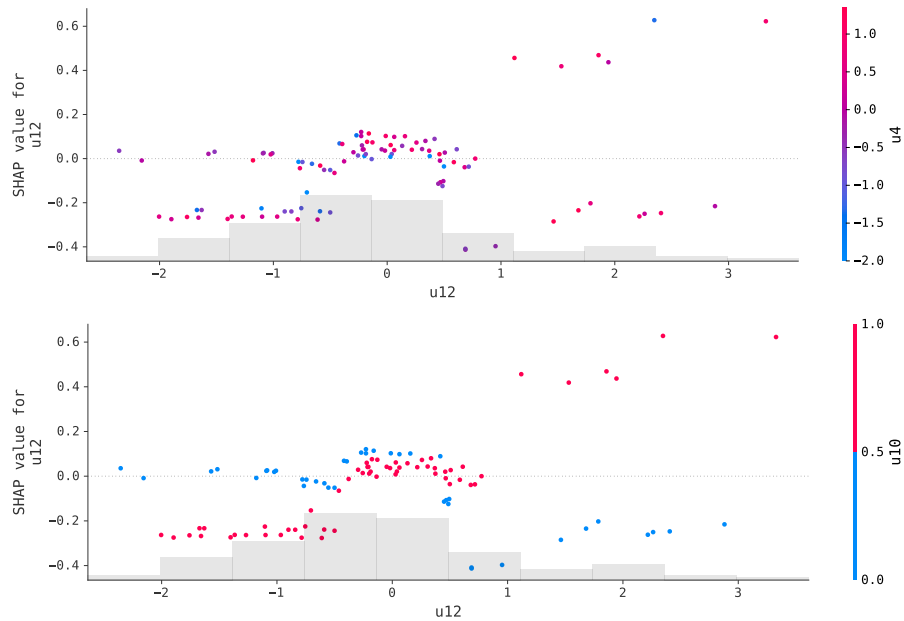
**[Remark.** A growing trend indicates a positive correlation.

**Beeswarm summary plot** Stack of multiple scatter plots. The values of the attributes are color coded (instead of being at the  $x$ -axis).



**Remark.** A color gradient indicates correlation.

**Scatter dependency plot** Links the combined effect of two attributes. Starting from the scatter plot of the first attribute, points are colored based on the values of the second attribute.

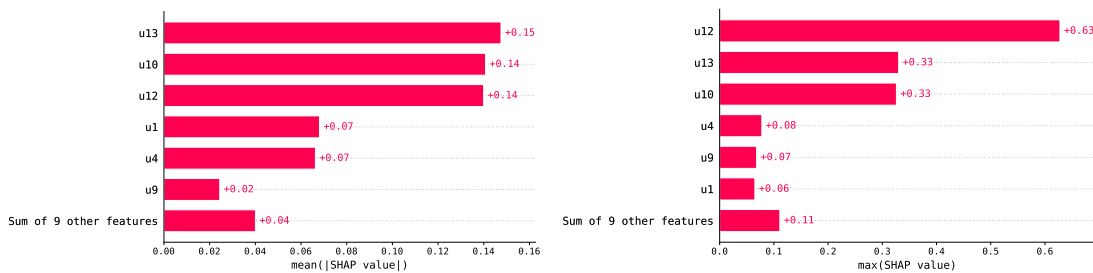


**Remark.** As is, SHAP does not provide single values (e.g., feature importance) but only values for each example.

**Global feature analysis** Aggregate local Shapely values to obtain global explanations. For instance, a possible approach is averaging over the examples:

$$\bar{\phi}_j(x) = \frac{1}{n} \sum_{i=1}^m |\phi_j(x_i)|$$

**Remark.** This approach is heuristic. Shapely values have well-defined semantics and the result of aggregation is not clear.



## 8.2.4 Semantics for feature selection

**Semantics for feature selection** Given the set of features  $\mathcal{J}$ , solve the problem:

Semantics for feature selection

$$\arg \min_{\mathcal{S} \subseteq \mathcal{J}} \left\{ |\mathcal{S}| : \forall (x, y). \hat{y} = \hat{f}_{\mathcal{S}}(x_{\mathcal{S}}), \mathcal{L}(y, \hat{y}) \leq \theta \right\}$$

In other words, this is the problem of determining the smallest subset of features  $\mathcal{S}$  such that a model  $\hat{f}_{\mathcal{S}}$  trained only considering them has an acceptable performance.

**Remark** (Pareto fronts). This problem can be solved as a multi-objective optimization problem using Pareto fronts to discard unnecessary points and only consider those that graphically do not have other points in the top-left area above them.

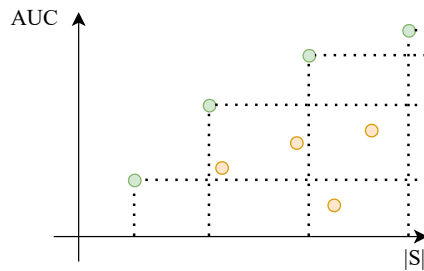


Figure 8.9: Green points are non-dominated and orange points are dominated

**Remark.** This approach aims at obtaining the best possible performance. Therefore, it is reasonable for a normal machine learning problem. However, for determining features importance, this is not ideal.