Anomaly Detection and Kernel Density Estimation

Problem Formalization

How do we formalize this problem?

A possible approach: we know that anomalies are (often) unlikely

- If we can estimate the probability of every occurring observation x
- ...Then we can spot anomalies based on their low probability

We turn a liability into a strenght!

Problem Formalization

We can check our intuition on our data

This is (roughly) the distribution over all the data

In [4]:
```
vmax = data['value'].max()
nab.plot_histogram(data['value'], vmax=vmax, bins=20)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Problem Formalization

We can check our intuition on our data

This is (roughly) the distribution around the first anomaly:

In [8]:
```
w0_start, w0_end = windows.loc[0]['begin'], windows.loc[0]['end']
data_anomaly0 = data[(data.index >= w0_start) & (data.index < w0_end)]
nab.plot_histogram(data_anomaly0['value'], vmax=vmax, bins=30)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

- There seems to be a significant difference

Problem Formalization

How do we formalize this problem?

A possible approach: we know that anomalies are (often) unlikely

- If we can estimate the probability of every occurring observation x
- ...Then we can spot anomalies based on their low probability

We turn a liability into a strength!

Formally, our detection condition can be stated as:

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

- Where $f(x)$ is a Probability Density Function (PDF)
- ...And θ is a (scalar) threshold

What do we need to make this work?

Density Estimation

We need one way to estimate probability densities

For some random process with n-dimensional variable $\boldsymbol{x}$:

- Given the true density function $f^*(x) : \mathbb{R}^n \rightarrow \mathbb{R}^+$
- ...And a second function $f(x, \omega)$ with the same input, and parameters $\boldsymbol{\omega}$

We want to make the two as similar as possible

Density Estimation

We need one way to estimate probability densities

For some random process with n-dimensional variable x:

- Given the true density function $f^*(x) : \mathbb{R}^n \to \mathbb{R}^+$
- ...And a second function $f(x, \omega)$ with the same input, and parameters ω

We want to make the two as similar as possible

What about modeling that as supervised learning?

Given some suitable loss function $L(\mathbf{y}, \mathbf{y}^*)$, that would lead to:

$$\text{argmin}_\omega L(f(\hat{\mathbf{x}}, \omega), f^*(\hat{\mathbf{x}}))$$

- where $\hat{\mathbf{x}}$ represents the training data

Density Estimation

Unfortunately, this approach cannot work

...Because typically we do not have access to the true density f^*

Density estimation is an unsupervised learning problem

It can be solved via a number of techniques:

- Simple histograms
- Kernel Density Estimation
- Gaussian Mixture Models
- Normalizing Flows
- Non Volume Preserving (NVP) transformations

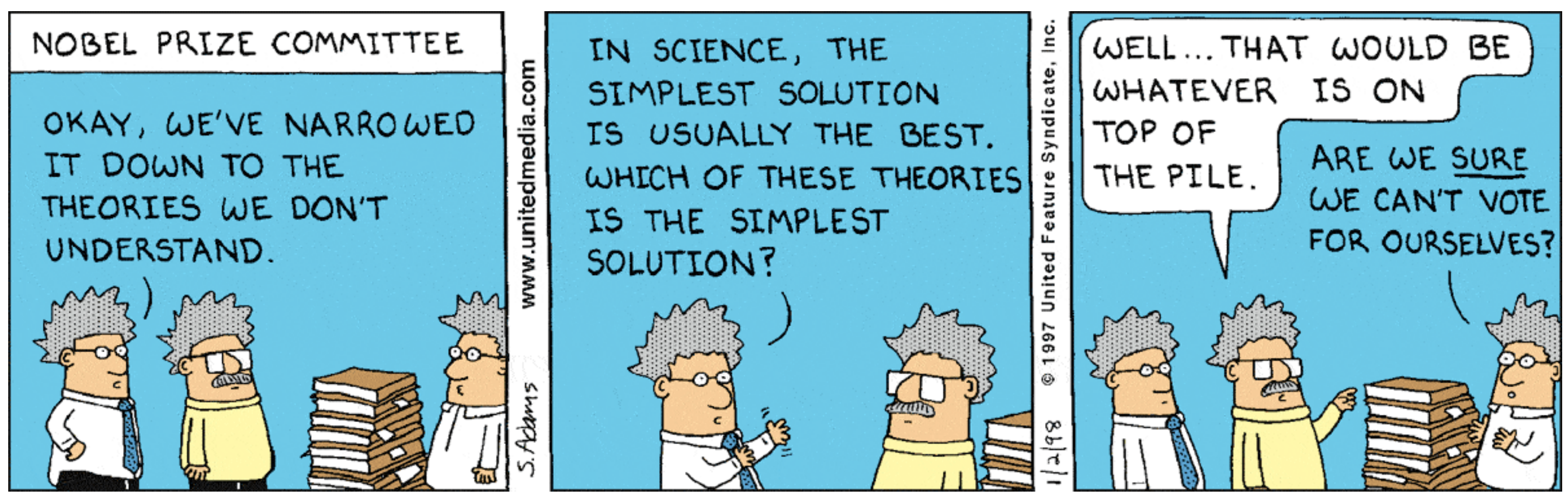
Which one shall we pick?

Our Friend, Occam's Razor

We will go with Occam's razor

It's a philosophical principle stating that:

Between two hypotheses, the simpler one is usually correct



Our Friend, Occam's Razor

We will go with Occam's razor

It's a philosophical principle stating that:

Between two hypotheses, the simpler one is usually correct

For its simplicity we will pick Kernel Density Estimation

- This will be returning principle in the course
- ...We will typically try simpler approaches first
- Especially at the beginning! Brace up for a slow start

Kernel Density Estimation

Kernal Density Estimation

In Kernel Density Estimation (KDE), the main idea is that:

- Wherever (in input space) there is a sample
- ...It's likely that there are more

So, we assume that each training sample is the center for a density "kernel"

Formally, the kernel $K(x, h)$ is just a valid PDF:

- x is the input variable (scalar or vector)
- h is a parameter (resp. scalar or matrix) called bandwidth

Typical kernels: Gaussian, exponential, cosine, linear...

Kernels

An example with one sample and a Guassian kernel:

In [9]:
```
x = np.array(0.5).reshape(1,1) # single sample
kde = KernelDensity(kernel='gaussian', bandwidth=0.1) # build the estimator
kde.fit(x) # fit the estimator on the data
# We use a plotting function from our module
nab.plot_density_estimator_1D(kde, xr=np.linspace(0, 1, 200))
ymin, ymax = plt.ylim()
plt.vlines(x, ymin, ymax, color='tab:red')
plt.ylim((ymin, ymax)); # ; = suppress output
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Kernel

An example with one sample and a Tophat kernel:

In [11]:
```
x = np.array(0.5).reshape(1,1) # single sample
kde = KernelDensity(kernel='tophat', bandwidth=0.1) # build the estimator
kde.fit(x) # fit the estimator on the data
# We use a plotting function from our module
nab.plot_density_estimator_1D(kde, xr=np.linspace(0, 1, 200))
ymin, ymax = plt.ylim()
plt.vlines(x, ymin, ymax, color='tab:red')
plt.ylim((ymin, ymax)); # ; = suppress output
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Kernel

An example with one sample and a linear kernel:

In [12]:
```
x = np.array(0.5).reshape(1,1) # single sample
kde = KernelDensity(kernel='linear', bandwidth=0.1) # build the estimator
kde.fit(x) # fit the estimator on the data
# We use a plotting function from our module
nab.plot_density_estimator_1D(kde, xr=np.linspace(0, 1, 200))
ymin, ymax = plt.ylim()
plt.vlines(x, ymin, ymax, color='tab:red')
plt.ylim((ymin, ymax)); # ; = suppress output
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Kernels

As an example, a Gaussian kernel in sklearn is given by:

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

- The $\propto$ ("proportional to") means that there is an implicit normalization constant
- The constant is handled by scikit-learn (in an efficient way)

All kernels in KDE:

- Are by default zero-centered (their mode is at 0)
- Can be relocated via an affine transformation (i.e. summing a constant to x)

In practice:

$$K(x - \mu, h)$$

Kernel Density Estimation

The estimated density of any point is obtained as a kernel average:

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

- x is the input for which we want an estimate
- $\hat{\mathbf{x}}$ is the matrix with the training samples
  - The training samples are part of the model parameters
- $x - \hat{x}_i$ is the difference between x and the i-th training sample
  - I.e. the value at x of the kernel centered on $\hat{x}_i$

Changing the kernel function:

- Allows to adjust the properties of the distribution (e.g. smoothness)
- ...By exploiting our prior knowledge

Kernel Density Estimation

An example with two samples and a Guassian kernel:

In [13]:
```
x = np.array([0.25, 0.75]).reshape(-1,1) # two sample, univariate
kde = KernelDensity(kernel='gaussian', bandwidth=0.1) # build the estimator
kde.fit(x) # fit the estimator on the data
nab.plot_density_estimator_1D(kde, xr=np.linspace(0, 1, 200))
ymin, ymax = plt.ylim()
plt.vlines(x, ymin, ymax, color='tab:red')
plt.ylim((ymin, ymax)); # ; = suppress output
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Kernel Density Estimation

How do we tune the bandwidth?

A nice approach (in principle): minimize the Mean Integrated Squared Error:

$$MISE(h) = \int (f(x, \hat{x}, h) - f^*(x))^2 dx$$

- In practice, we can't use it (we don't have f^*)
- ...But we can use an approximation, i.e. the validation set (more about this later)

A rule of thumb for the univariate case:

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

- IQR is the inter-quartile range