Artificial Intelligence in Industry

Matteo Donati

October 20, 2022

Contents

1 Anomaly Detection via Simple Methods

1.1 Problem and Data

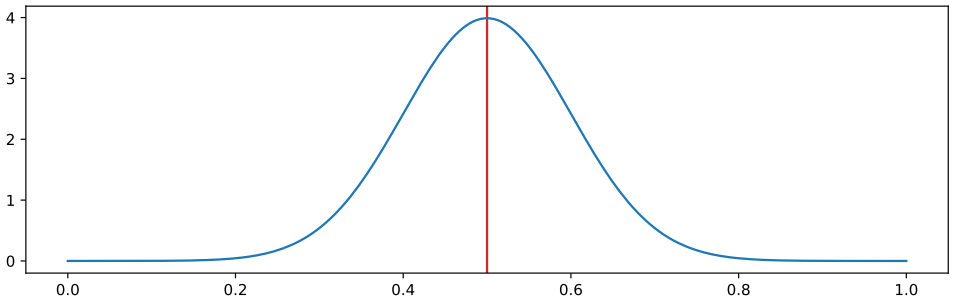
The goal of anomaly detection is to detect, analyze and anticipate abnormal situations (i.e. **anomalies**). Usually, anomaly detection is based on time-series analysis, where a time-series is a sequence whose index represents time. Time-series have one difference with respect to classical table datasets: their row index is meaningful, since it represents the position of the example in the sequence. The labels associated with such a dataset usually indicate an instant of time at which an anomaly occurs. Considering a specific anomaly, one could also compute the window of time inside which the specific anomaly occurs.

1.2 Anomaly Detection and Kernel Density Estimation

A possible approach to detect anomalies is based on the fact that anomalies are often unlikely. If one can estimate the probability of every occurring observation x, then one can spot anomalies based on their low probability. Formally, a detection condition can be states as $f(x) \leq \theta$, where $f(x)$ is a **probability density function** (**PDF**), and θ is a scalar threshold. Given some training data $\hat{\boldsymbol{x}}$, the true density function $f^*(x) : \mathbb{R}^n \to \mathbb{R}^+$, and a second function $f(x, \omega)$, a supervised learning approach to estimate the probability densities considers a suitable loss function, $L(y, y^*)$, that has to be optimized so to find the best set of parameters ω that minimizes the considered loss:

$$\text{argmin}_\omega \, L(f(\hat{\boldsymbol{x}}, \omega), f^*(\hat{\boldsymbol{x}})) \tag{1}$$

However, this approach cannot work, because usually one does not have access to the true density f^*. Thus, density estimation is an unsupervised learning problem. Such problem can be solves via a number of techniques (e.g. via Kernel Density Estimation). In **Kernel Density Estimation** (**KDE**) the main idea is that wherever, in the input space, there is a sample, then it is likely that there are more samples, so one can assume that each training sample is the center for a density kernel. Formally, the kernel $K(x, h)$ is just a valid PDF, where x is the input variable (scalar or vector), and h is a parameter (scalar or matrix respectively) called *bandwidth*. For example, given a single sample $x = 0.5$, then a Gaussian estimator with $h = 0.1$ will produce the following:



Indeed, in `sklearn`, a Gaussian kernel is given by:

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

which is similar to the PDF of the Normal distribution, where the mean can be interpreted as zero, and h controls the standard deviation of the distribution. However, since the mean is zero, the kernel will be centered on zero. To solve this, one can use an affine transformation, $K(x - \mu, h)$, which gives the value of a kernel computed for the value x and centered on μ. Moreover, the estimated density of any point is obtained as a kernel average:

$$f(x, \hat{\boldsymbol{x}}, h) = \frac{1}{m} \sum_{i=0}^{m} K(x - \hat{x}_i, h) \tag{3}$$

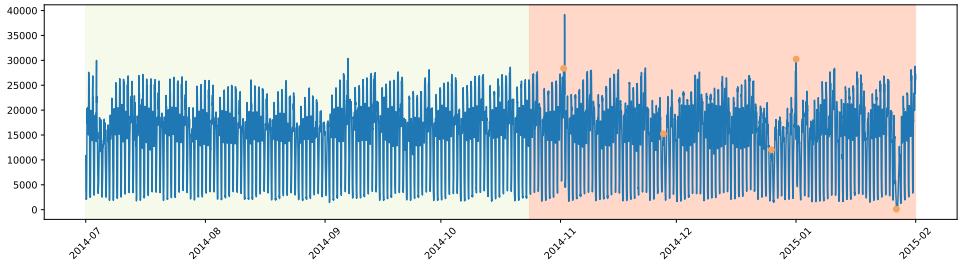
where x is the input for which to compute the estimate, $\hat{\boldsymbol{x}}$ is the matrix containing the training samples, $x - \hat{x}_i$ is the difference between x and the i-th training sample. Thus, KDE models are not trained in the usual sense: the training set is part of the model parameters. The only thing that one needs to train is h. For the univariate case, one can apply the following rule of thumb:

$$h = 0.9 \min\left(\hat{\sigma}, \frac{IQR}{1.34}\right) m^{-\frac{1}{5}} \tag{4}$$

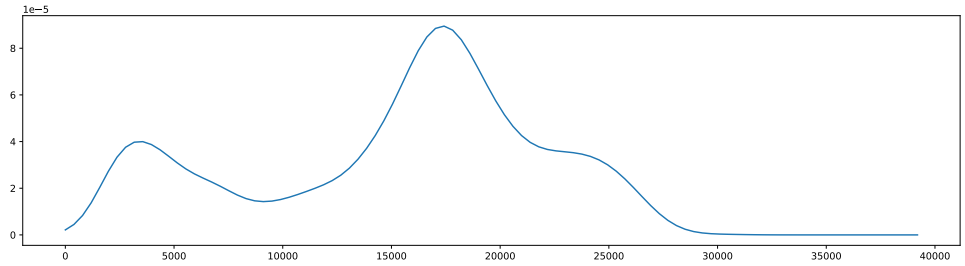
where $\hat{\sigma}$ is the standard deviation computed using the training data, and IQR is the inter-quartile range. Lastly, to avoid taking products, one can work with negated log probabilities, so that the anomaly detection condition becomes $-\log f(x, \omega) \geq \theta$.

1.3 KDE for Anomaly Detection

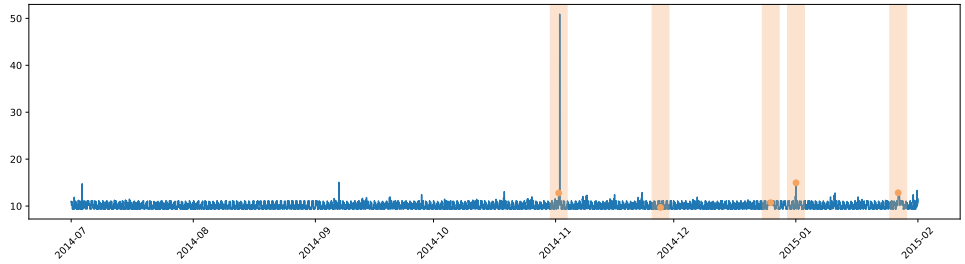
Considering a time-series, one can split it into training set and test set, where the training set only includes data about normal behavior and it will be used to fit a KDE model, while the test is used to asses how well the approach can generalize. If the training set contains some anomalies, then it is fine, as long as these are very infrequent.



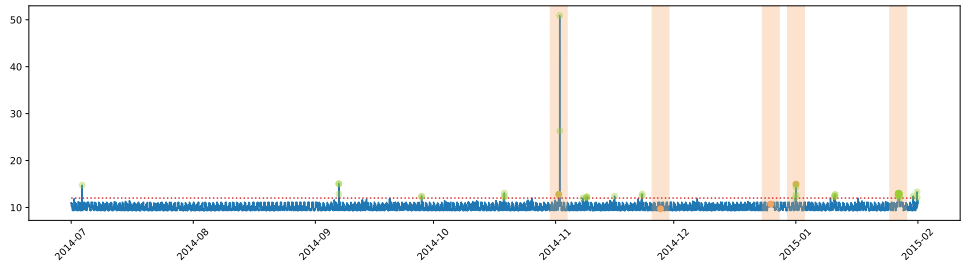
At this point, a univariate KDE is fit to the training data, obtaining the following estimated distribution:



The alarm signal can be then computed from such estimated distribution:



One could also pick a threshold and try to detect some anomalies:



However, the result contains many false positives, which are usually common in anomaly detection.

1.4 Metrics for Anomaly Detection

In order to evaluate costs and benefits of one's predictions, one can rely on a cost model. A simple cost model is based on the concepts of true positives (i.e. windows for which one detects at least one anomaly), false positives (i.e. detected anomalies that do not fall in any window), false negatives (i.e. anomalies that go undetected), and advance (i.e. time between an anomaly and when first it was detected). Then, one can introduce: a cost c_{alarm} for loosing time in analyzing false positives, a cost c_{missed} for missing anomalies, and a cost c_{late} for a late detection. This simple cost model can be used for choosing the threshold to detect anomalies. Indeed, the best threshold is the one that minimizes

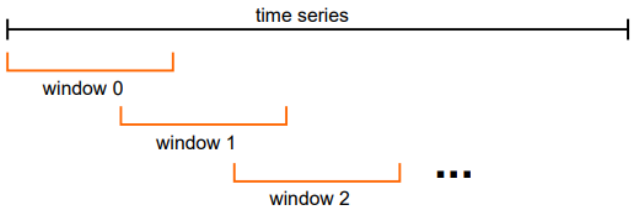
$c_{alarm} \times |FP| + c_{missed} \times |FN| + c_{late} \times |\text{late detections}|$. To do so, one can define a validation set, and apply a linear search, to tune the θ threshold.

1.5 Sliding Windows

Given a time-series, nearby points tend to have similar values, meaning that they are correlated. A useful tool to study such correlation are the autocorrelation plots:

1. Consider a range of possible **lags**.
2. For each lag value l:
   (a) Make a copy of the series and shift it by l time-steps.
   (b) Compute the Pearson correlation coefficient (i.e. linear correlation coefficients) with the original series.
3. Plot the correlation coefficients over the lag values.

Where the curve is far from zero, there is a significant correlation, and where it gets close to zero, no significant correlation exists. These correlations are a source of information and can be exploited to improve the estimated probabilities. Indeed, to take advantage of such information, one could feed one's model with sequences of observations, instead of individual observations. A common approach consist in using a **sliding window**:



In general, let m be the number of examples and w be the window length, the result of this approach is the table

	$\boldsymbol{s_0}$	$\boldsymbol{s_1}$	$\cdots$	$\boldsymbol{s_{w-1}}$
$\boldsymbol{t_{w-1}}$	x_0	x_1	$\cdots$	x_{w-1}
$\boldsymbol{t_w}$	x_1	x_2	$\cdots$	x_w
$\boldsymbol{t_{w+1}}$	x_2	x_3	$\cdots$	x_{w+1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\boldsymbol{t_{m-1}}$	x_{m-w}	x_{m-w+1}	$\cdots$	x_{m-1}

where $\boldsymbol{t_i}$ is the time window index, and $\boldsymbol{s_j}$ is the position of an observation within a window. In general, one can pick the window length to be equal to the number of lags for which the aforementioned correlation is still high.

1.6 Sequence Input in KDE

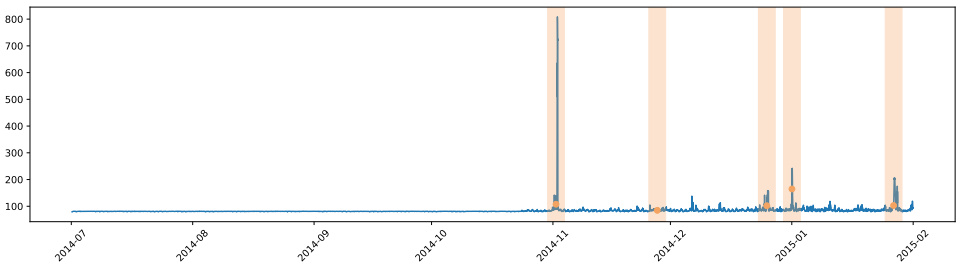
When dealing with KDE, there exists a straightforward approach to take sequences input into account. In particular, this can be done by using multivariate KDE: each sequence is treated as a vector variable, and individual sequences are treated as independent (Markov property). Then, a multivariate KDE estimator is learned. First of all, a suitable bandwidth has to be chosen: i. pick a validation set, ii. tune th bandwidth for maximum likelihood. Formally, let $\tilde{\boldsymbol{x}}$ be a validation set of m samples. Assuming independent observations, its likelihood is:

$$L(\tilde{\boldsymbol{x}}, \hat{\boldsymbol{x}}, h) = \prod_{i=1}^{m} f(\tilde{x}_i, \hat{\boldsymbol{x}}, h) \tag{5}$$

Then, h is chosen to maximize such likelihood:

$$\text{argmax}_h \, \mathbb{E}_{\tilde{\boldsymbol{x}} \sim \mathcal{D}}[L(\tilde{\boldsymbol{x}}, \hat{\boldsymbol{x}}, h)] \tag{6}$$

where $\mathcal{D}$ is the true distribution of samples. However, as many training problems, this cannot be solves in an exact fashion. Instead, one can approximate $\mathbb{E}$ by sampling multiple $\tilde{\boldsymbol{x}}$ and picking the bandwidth h^* leading to the maximum average likelihood. This approximation can be implemented by applying a grid search and searching for h^*. For example, the alarm signal produced when considering sequences input instead of single observations is the following:

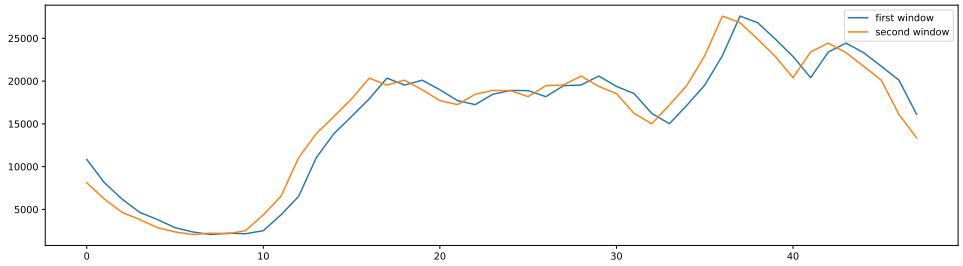


This signal shows much less noise with respect to the one computed when considering single observations.

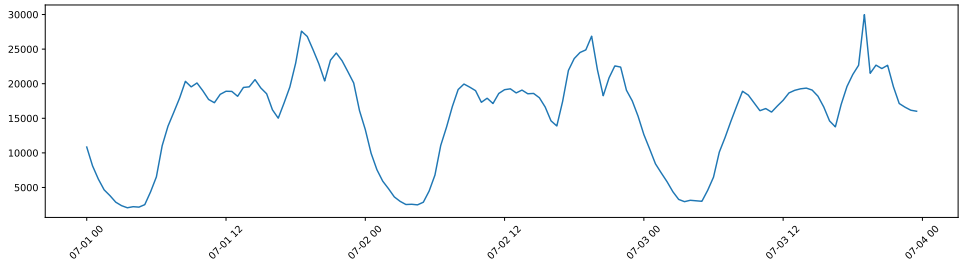
2 Anomaly Detection via Advanced Methods

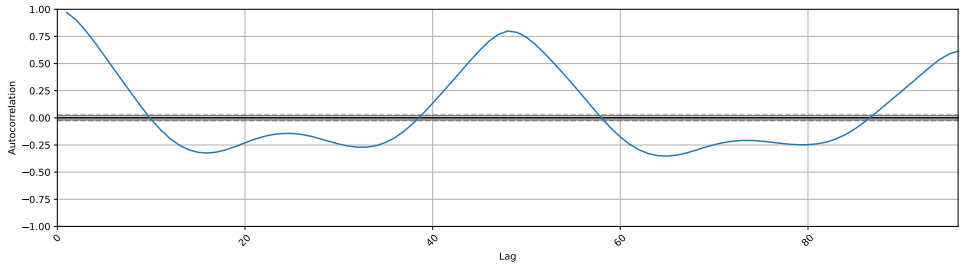
2.1 A Time-Dependent Estimator

Considering a sequence-based estimator, one can notice how this estimator learns from all the training data. This means, for example, that considering two subsequent windows it will learn from both these subsequent sequences:



In particular, in the first window, the observations are x_0, x_1, ..., while in the second window the observations are x_1, x_2, ..., which means that by moving the window forward one learn the distribution of each point (and its correlations) multiple times. Moreover, it could happen that the considered time-series is approximately periodic:



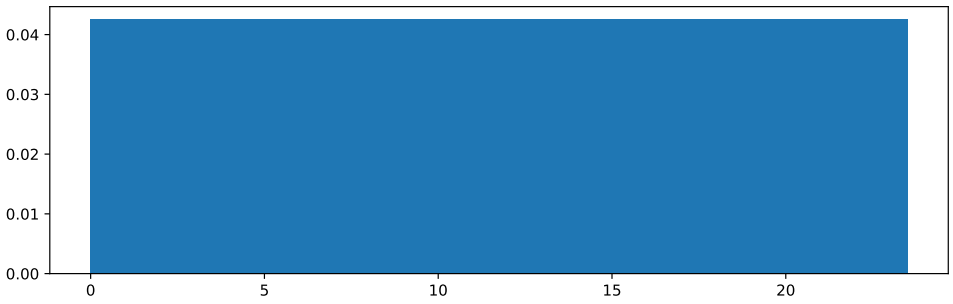


Thus, the previous sequence-based estimator is solving a uselessly complicated problem, and it is not using all the available knowledge. In order to solve these two problems, one could start by adding the

notion of time to the input: $y = (t, x)$, where t represents time and x is the usual input. One can then use this information to build a time-dependent estimator $f(x|t)$. This is a **conditional density**, namely the density value of the observed value of x assuming that the time t is known (indeed, t is a **controlled variable**, i.e. it is completely predictable). Thus, the anomaly detection conditions becomes $f(x|t) \leq \theta$. Considering the definition of conditional probability, $f(t, x) = f(x|t)f(t)$, one can derive:

$$\frac{f(t, x)}{f(t)} \leq \theta \tag{7}$$

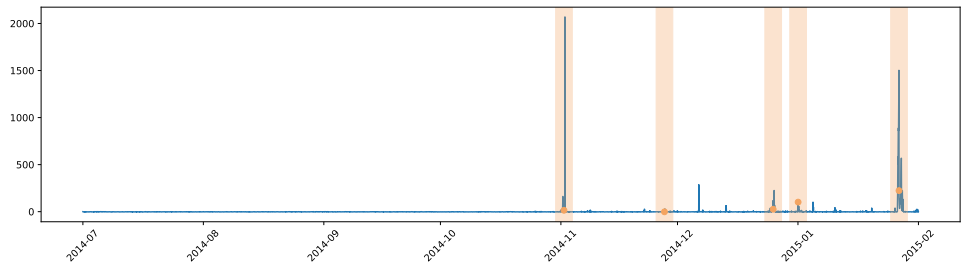
In order implement this, one needs an estimator for $f(t, x)$ (e.g. using KDE), and one estimator for $f(t)$ which one can easily obtain (e.g. using KDE again).



From equation 7:

$$f(t, x) \leq \theta f(t) \quad \xRightarrow{\text{Being } f(t) \text{ constant}} \quad f(t, x) \leq \theta' \tag{8}$$

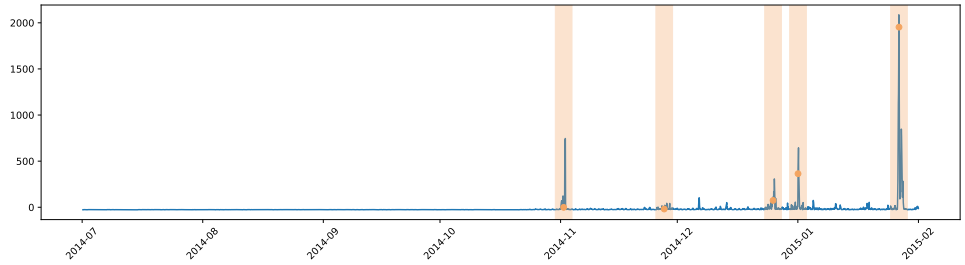
Now, having chosen a specific bandwidth through grid-search, one can produce the specific alarm signal:



Lastly, one can search for the best threshold θ' to minimize the overall cost given by the specific cost model discussed above.

2.2 Time-Indexed Models

A second approach to handle time consists in learning many density estimators, where each estimator is specialized for a given time. Formally, this is an ensemble model. In particular, one can obtain the estimated probabilities by evaluating $f_{g(t)}(x)$, where each f_i function is an estimator, and the $g(t)$ retrieves the correct f_i based on the time value. Each f_i estimator works with smaller amounts of data, but the individual problems are easier. For example, given a range of time of 24 hours, one could learn a different estimator for 00:00, 00:30, 01:00, etc. producing 48 specialized estimators, where each estimator is learned using a different portion of the training set. Lastly, one can estimate the log probabilities for each possible timestamp and concatenate the results so to produce the alarm signal



and then to search for the best possible threshold.

2.3 Gaussian Mixture Models

KDE-based approaches work well, but have some trouble with high-dimensional data: with a larger dimensionality, prediction time grows and more data is needed to obtain reliable results. Moreover, KDE has trouble with large training sets, and gives nothing more than an anomaly signal (i.e. determining the cause of the anomaly is up to a domain expert). The first two problems are due to the fact that KDE makes no attempt to compress the information from the training data. Indeed, the size of a KDE model grows directly with the training set size. To solve such problem, another density estimation technique is introduced: the **Gaussian Mixture Models** (**GMMs**). A GMM describes a distribution via a weighted sum of Gaussian components, where the model size depends on the dimensionality and on the number of components, and the number of components can be chosen. Formally, one can assume data is generated by a probabilistic model

$$X_Z \tag{9}$$

where Z and X_k are both variables. In particular, Z represents the index of the component that generates the sample, and X_k follows a multivariate Gaussian distribution. In other words, a GMM is a selection-based ensemble. The probability density function (PDF) of a GMM is given by:

$$g(x, \mu, \Sigma, \tau) = \sum_{k=1}^{n} \tau_k f(x, \mu_k, \Sigma_k) \tag{10}$$

where f is the PDF of a multivariate Normal distribution, μ_k is the vector mean and Σ_k is the covariance matrix for the k-th component, and τ_k corresponds to $P(Z = k)$. When one wants to sample from a GMM, first one needs to sample the Z variable, then one can sample from the corresponding multivariate distribution. Hence, one does not get to know just the sample value, but also which of the Gaussian components it was generated by. Training a GMM to approximate other distributions can be done in terms of likelihood maximization:

$$\text{argmax}_{\mu,\Sigma,\tau}\, \mathbb{E}_{\hat{x}\sim X}[L(\hat{x}, \mu, \Sigma, \tau)] \quad \text{s.t.} \quad \sum_{k=1}^{n} \tau_k = 1 \tag{11}$$

The likelihood function L measures how likely it is that the training sample $\hat{x}$ is generated by a GMM with parameters μ, Σ, τ. This expectation can be approximated by using the training set:

$$\mathbb{E}_{\hat{x}\sim X}[L(\hat{x}, \mu, \Sigma, \tau)] \simeq \prod_{i=1}^{m} g(\hat{x}_i, \mu, \Sigma, \tau) \tag{12}$$

This leads to:

$$\text{argmax}_{\mu,\Sigma,\tau} \prod_{i=1}^{m} \sum_{k=1}^{n} \tau_k f(x, \mu_k, \Sigma_k) \quad \text{s.t.} \quad \sum_{k=1}^{n} \tau_k = 1 \tag{13}$$

From an optimization point of view, this is an annoying problem, mainly due to the presence of a constant, a product and a sum, and the fact that the product cannot be decomposed. So, in order to simplify such formulation, a random variable Z_i is introduced for each example. In particular, $Z_i = k$ if and only if the i-th example was drawn from the k-th component, and the Z_i variables are latent (i.e. their value is unknown). With the new variables, the PDF becomes:

$$\tilde{g}_i(x_i, z_i, \mu, \Sigma, \tau) = \tau_{z_i} f(x, \mu_k, \Sigma_k) \tag{14}$$

The PDF is now specific for each example and does not contain a sum. Moreover, the value z_i is now an input to $\tilde{g}_i$ can be used as an index to retrieve the correct τ_k. The likelihood expectation over both X and Z, $\mathbb{E}_{\hat{x}\sim X,\hat{z}\sim Z}[L(\hat{x}, \hat{z}, \mu, \Sigma, \tau)]$ can be computed by using the training set as a single sample so to obtain:

$$\mathbb{E}_{\hat{x}\sim X,\hat{z}\sim Z}[L(\hat{x}, \hat{z}, \mu, \Sigma, \tau)] \simeq \mathbb{E}_{\hat{z}\sim Z}\left[\prod_{i=1}^{m} \tilde{g}_i(x_i, z_i, \mu, \Sigma, \tau)\right] \tag{15}$$

The same technique cannot be used for Z, since the values of the Z_i variables are unknown. To deal with the expectation on Z, another set of variables is added. These variables represent the unknown distribution of the latent Z_i variables. In particular, $\tilde{\tau}_{i,k}$ corresponds to $P(Z_i = k)$. With the new variable, the expectation can be computed in closed form:

$$\mathbb{E}_{\hat{x}\sim X,\hat{z}\sim Z}[L(\hat{x}, \hat{z}, \mu, \Sigma, \tau)] \simeq \prod_{i=1}^{m} \prod_{k=1}^{n} \tilde{g}_i(x_i, z_i, \mu, \Sigma, \tau)^{\tilde{\tau}_{i,k}} \tag{16}$$

Intuitively, $\tilde{\tau}_{i,k}$ samples are generated for each example i and component k. Then, their densities are multiplied as usual. Thus, the training problem is:

$$\begin{aligned} &\text{argmax}_{\mu,\Sigma,\tau,\tilde{\tau}} \prod_{i=1}^{m}\prod_{k=1}^{n} \tilde{g}_i(x_i, z_i, \mu, \Sigma, \tau)^{\tilde{\tau}_{i,k}} \\ &\text{s.t.} \sum_{k=1}^{n} \tau_k = 1 \\ &\text{s.t.} \sum_{k=1}^{n} \tilde{\tau}_{i,k} = 1 \quad \forall i \in \{1, \ldots, m\} \end{aligned} \tag{17}$$

The expectation-maximization algorithm can be now used. This algorithm is an optimization method based on alternating steps:

- In the expectation step:
  - μ, Σ, τ are considered as fixed and one can optimize over $\tilde{\tau}$.
  - The expectation over Z is computed in a symbolic form.
- In the maximization step:
  - $\tilde{\tau}$ is considered as fixed and one can optimize over μ, Σ, τ.

Such method stops when the likelihood improvement become too small. When considering the aforementioned optimization problem, these two steps are defined as follows:

- In the expectation step, the μ, Σ, τ are fixed, so that one needs to solve:

$$\text{argmax}_{\tilde{\tau}} \prod_{i=1}^{m}\prod_{k=1}^{n} \tilde{g}_i(x_i, z_i, \mu, \Sigma, \tau)^{\tilde{\tau}_{i,k}} \quad \text{s.t.} \quad \sum_{k=1}^{n} \tilde{\tau}_{i,k} = 1 \quad \forall i \in \{1, \ldots, m\} \tag{18}$$

  Such optimization problem can be easily decomposed, so one can optimize over each example individually. By substituting $\tilde{g}_i$ for a single example i one has:

$$\text{argmax}_{\tilde{\tau}} \prod_{k=1}^{n} (\tau_k f(x, \mu_k, \Sigma_k))^{\tilde{\tau}_{i,k}} \quad \text{s.t.} \quad \sum_{k=1}^{n} \tilde{\tau}_{i,k} = 1 \tag{19}$$

  Which (since μ, Σ, τ are fixed) is solved by choosing:

$$\tau_{i,k} = \frac{\tau_k f(\hat{x}_i, \mu_k, \Sigma_k)}{\sum_{h=1}^{n} \tau_h f(\hat{x}_i, \mu_k, \Sigma_k)} \tag{20}$$

- For the maximization step the math is a bit more difficult. Each τ_k is optimized by computing relative sum of the corresponding $\tilde{\tau}_{i,k}$ variables:

$$\tau_k = \frac{1}{m} \sum_{i=1}^{m} \tilde{\tau}_{i,k} \tag{21}$$

  In fact, the latent variables represent samples drawn from the Z_k variables. Lastly, the μ_k and Σ_k parameters can be estimated based on classical methods. In particular, each example is given a weight equal to $\tilde{\tau}_{i,k}$, then μ and Σ are estimated via a least square approach.

2.4 Autoencoders for Anomaly Detection

An **autoencoder** is a type of neural network which is usually designed to reconstruct its input vector by first learning an internal representation of the input (encoder), and then by learning how to reconstruct the input starting from such representation (decoder). Formally, an encoder $e(x, \theta_e)$ maps x into a vector of latent variables z, and a decoder $d(z, \theta_d)$ maps z into the reconstructed input tensor. In general, autoencoders are trained for minimum MSE:

$$\text{argmin}_{\theta_e, \theta_d} \| d(e(\hat{x}_i, \theta_e), \theta_d) \|_2^2 \tag{22}$$

Usually, there is a risk that an autoencoder learns a trivial transformation (i.e. $x' = x$), which can be avoided by choosing a small-dimensional latent space, and by encouraging sparse encodings with an L1 regularizer. Moreover, autoencoders can be used for anomaly detection by using the reconstruction as an anomaly signal, e.g.:

$$\| x - d(e(\hat{x}_i, \theta_e), \theta_d) \|_2^2 \geq \theta \tag{23}$$

This approach has some pros and cons compared to KDE. Indeed, the size of a neural network does not depend on the size of the training set, and neural networks have good support for high dimensional data. On top of this, there is a limited possibility of overfitting and time needed for prediction/detection is low. On the other hand, error reconstruction can be harder than density estimation. Moreover, the results obtained by using an autoencoder are similar to ones obtained when using KDE. Indeed, given an autoencoder h, one tries to solve the following problem:

$$\text{argmin}_{\theta} \| h(\hat{x}_i, \theta) - \hat{x}_i \|_2^2 \tag{24}$$

By expanding the L2 norm:

$$\text{argmin}_{\theta} \sum_{i=1}^{m} \sum_{j=1}^{n} (h_j(\hat{x}_i, \theta) - \hat{x}_{i,j})^2 \tag{25}$$

By introducing a log and exp transformation:

$$\text{argmin}_{\theta} \log \exp \left(\sum_{i=1}^{m} \sum_{j=1}^{n} (h_j(\hat{x}_i, \theta) - \hat{x}_{i,j})^2 \right) \tag{26}$$

Rewriting the outer sum using properties of exponentials:

$$\text{argmin}_\theta \log \prod_{i=1}^{m} \exp\left(\sum_{j=1}^{n} (h_j(\hat{x}_i, \theta) - \hat{x}_{i,j})^2\right) \tag{27}$$

Rewriting the inner sum in matrix form:

$$\text{argmin}_\theta \log \prod_{i=1}^{m} \exp\left((h(\hat{x}_i, \theta) - \hat{x}_{i,j})^T I (h(\hat{x}_i, \theta) - \hat{x}_{i,j})\right) \tag{28}$$

Negating the argument of exp and swapping the argmin for an argmax, multiplying the exponential argument by $1/2$, and multiplying the exponential by $1/\sqrt{2\pi}$:

$$\text{argmax}_\theta \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(h(\hat{x}_i, \theta) - \hat{x}_{i,j})^T I (h(\hat{x}_i, \theta) - \hat{x}_{i,j})\right) \tag{29}$$

The term inside the product if the PDF of a multivariate Normal distribution:

$$\text{argmin}_\theta \log \prod_{i=1}^{m} f(\hat{x}_i, h(\hat{x}_i), I) \tag{30}$$

In particular, such distribution is centered on $h(\hat{x}_i)$, and has independent Normal components all having unit variance. Lastly, when the MSE loss is used for training a neural network, such network is trained for maximum likelihood (just like density estimators).