Analytics

Forecasting

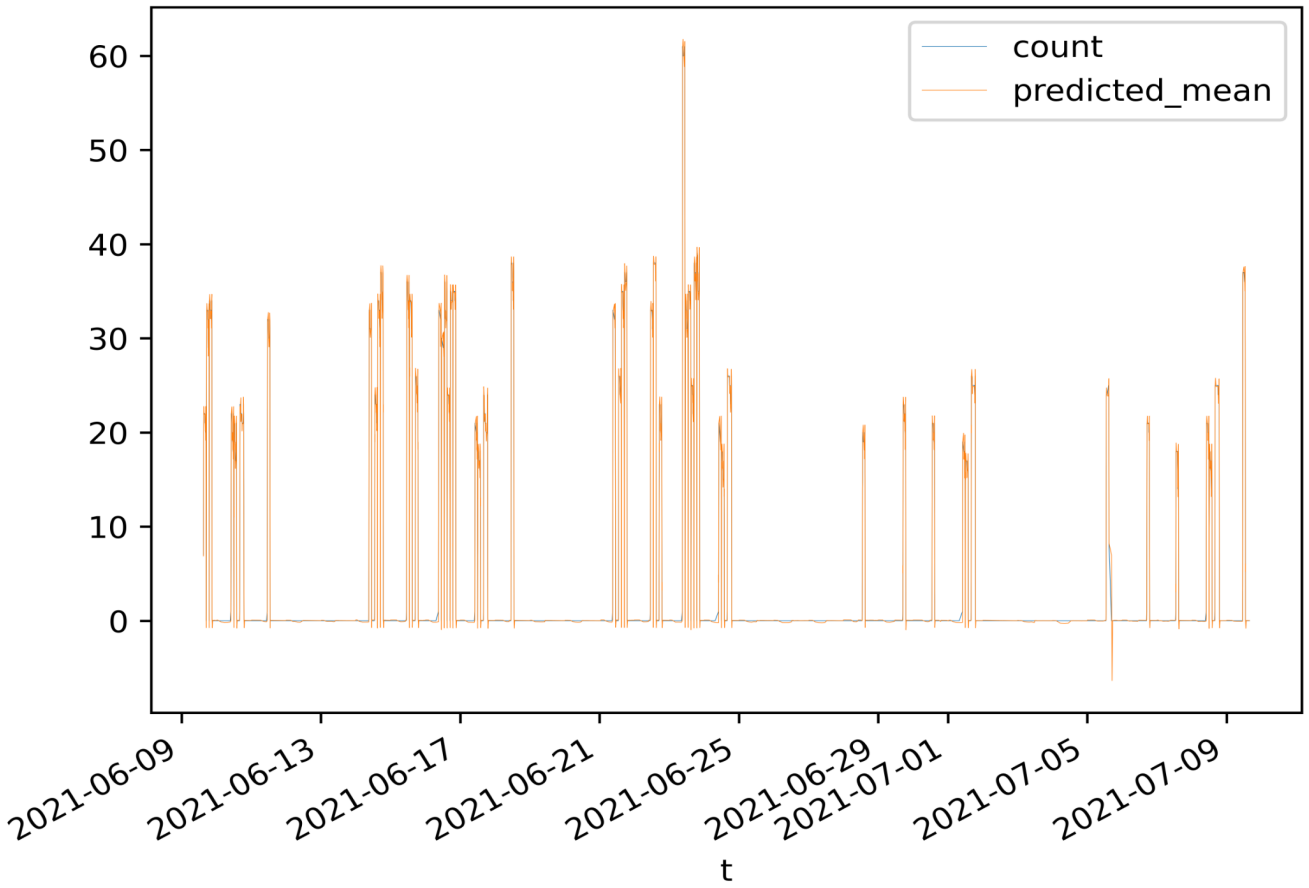
Model 1: Linear regression

For linear regression, we used the **sklearn** implementation of the lasso variant where the objective function is augmented with an L1 regularization term which helps to automatically eliminate uninformative features by assigning them with 0 weight. As input features into the model, we basically choose the following:

1. **Lag-1** and **lag-2** counts,
2. **dT**: current time minus previous time $T_i - T_{i-1}$,
3. **d(count)/dT**: estimated as $\frac{C_{i-1} - C_{i-2}}{T_{i-1} - T_{i-2}}$,
4. **Hour of day** (0-23),
5. **Day of week** (0-6),
6. **Month of year** (0-11).

We also eliminate all feature columns with constant value (e.g., month of year) in our dataset before training. Lasso regression is a rather simple model with only 1 important hyperparameter, namely the regularization coefficient **α**.

In our implementation, the way we determine the hyperparameter is by using a python package called **optuna** which utilizes an SMBO (sequential model-based optimization) method called TPE (tree-structured parzen estimator). Because there is no clear way of deciding on the regularization coefficient given the dataset just by manual inspection and also it is not possible to use first order or higher order optimization methods (e.g., gradient descent). Hence our choice of using a black box hyperparameter optimizer package called **optuna**. We used the mean squared error over the test data as our objective function to minimize.



The first **90%** of the dataset was taken as the training data and the remaining **10%** as the test data. In the figure above, you can see how well the model was able to predict not only the training but also the test regime. In the training run that is responsible for the figure above, the regularization parameter was optimized to be **0.0006**. Also, following are some accuracy metrics for the model calculated over the test dataset (**last 10%**):

- **Mean absolute error**: 0.22496694
- **Root mean squared error**: 0.38944238

Model 2: Long short-term memory

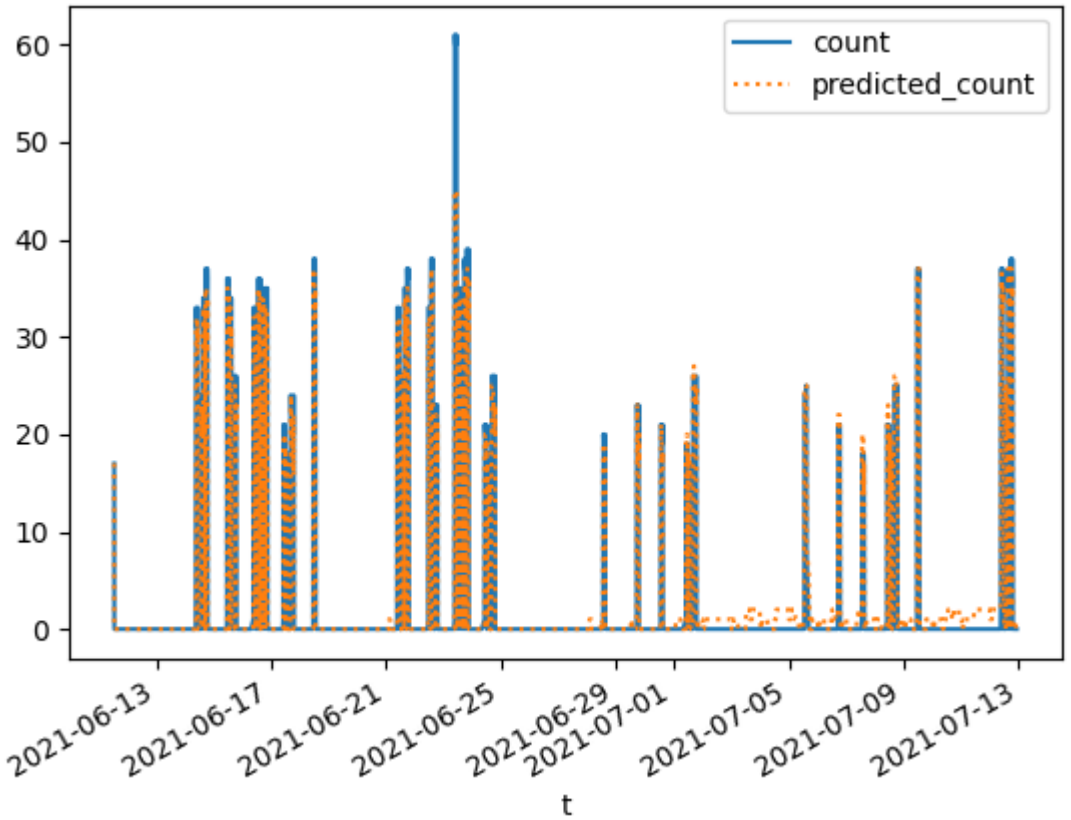
We implemented the LSTM with **PyTorch** and PyTorch Lightning. For the FaaS platform and in our final submission we used only PyTorch to have less package dependencies and a more compact implementation. PyTorch Lightning was essential in the beginning to log and visualize training progress.
Our LSTM consists of (two) stacked core LSTM modules, a linear layer, a ReLU and another linear layer for the output. The ReLU is important to overwrite negative values with 0. As a loss function we use MSE. As an optimizer the Adam optimizer.

Hyperparameters:

- **number of features**: 5
    1. **Lag-1**: predecessor count of the current datapoint
    2. **dT**: current time minus previous time $T_i - T_{i-1}$
    3. **minute of day**: [0, 1339]
    4. **day of week**: [0, 6]
    5. **month of year**: [0, 11]

Note: all data will be scaled to [0, 1]

- **number of layers**: 2
  - found by probing
- **number of hidden layers**: 32
  - found by probing
- **learning rate**: 0.0001
  - found by probing
- **batch size:** 8
  - found by probing
- **sequence length:** 16
  - important parameter to indicate how many data points the LSTM must look back to make conclusion for current value
  - if too low, the LSTM cannot identify patterns
  - if too high, computation effort gets high
- **number of epochs:** 25
  - important parameter that defines the number of training rounds
  - with 25 we get a model that fits to the training dataset well



The first **90%** of the dataset was taken as the training data and the remaining **10%** as the test data. We can see that the LSTM discovers the spike patterns, i.e. students fill the room and leave later again. However, we also see that the LSTM makes no clear decision when the room is in fact empty. Note also that the highest blue peak is not orange dotted since we postprocess outliers that exceed the limit of 45. Following are some accuracy metrics of the model over the test dataset (**last 10%**):

- **Mean absolute error**: 1.0038022813688212
- **Root mean squared error**: 1.3210780250571779

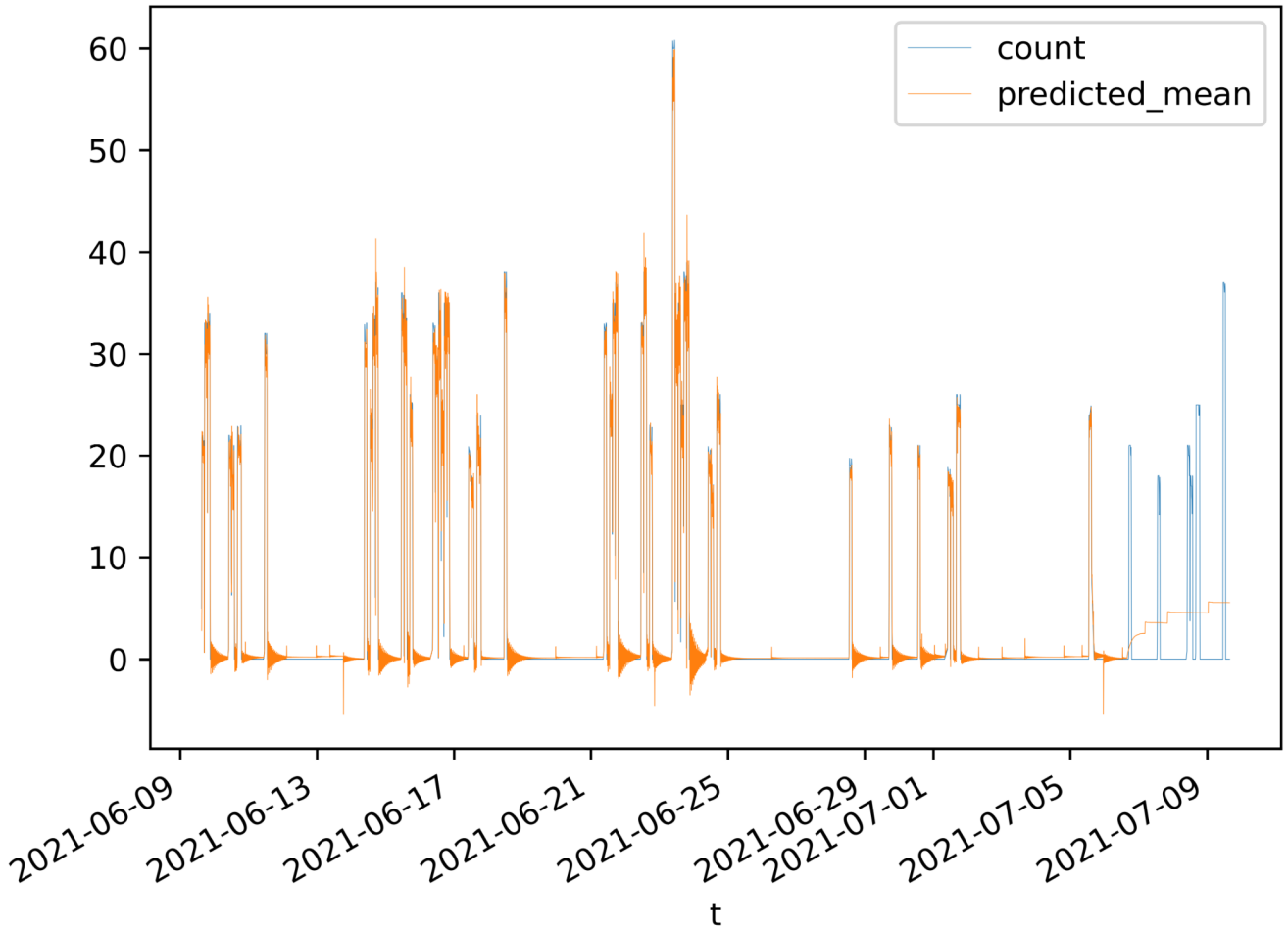
<u>Model 3: SARIMAX</u>

We used the **statsmodel** python package for the implementation using the SARIMAX variant. In comparison to the SARIMA, it provides the possibility to input exogenous (external) regressors aside from the time series data. We chose the following as our exogenous regressors:

1. **Hour of day** (0-23),
2. **Day of week** (0-6),
3. **Month of year** (0-11).

The SARIMAX model has many parameters in its implementation. But from a theoretical standpoint, the model only offers the following parameters: **p**, **d**, **q**, **P**, **D**, **Q** and **s**. The (p, d, q) are the order of the model for the number of AR parameters, differences, and MA parameters. The (P, D, Q, s) are the order of the seasonal component of the model for the AR parameters, differences, MA parameters, and periodicity.

Due to the time constraints, we had at the time, we were not able to use the conventional method of determining these hyperparameters using ACF and PACF plots. Instead, once again, we used the **optuna** library to optimize them. We experimented with a wide range of upper and lower bounds for each parameter and found out that if the seasonality parameter is high enough, the training process demands huge amounts of RAM which causes out of memory errors. Keeping in mind that our time series data has a period of less than 15 minutes, any reasonable seasonality period such as 1 day or 1 week leads to a big enough seasonality parameter that we had to stop using these 4 seasonality parameters, (P, D, Q, s). Instead, as a band aid to this problem, we provided the model with the exogenous variables as described above in order to inject some information on seasonality. We used the mean squared error over the test data as our objective function to minimize.



The first **90%** of the dataset was taken as the training data and the remaining **10%** as the test data. In the training run that is responsible for the figure above, the (p, d, q) parameters were optimized to be **(4, 0, 4)**. In our experience, the out-of-sample prediction performance of SARIMAX has always

been problematic even as compared to a model as simple as the linear regression. Following are some accuracy metrics of the model over the test dataset (**last 10%**):

- **Mean absolute error**: 6.452881304983806
- **Root mean squared error**: 8.787062148931863

Model Selection for Forecasting

- all three models are used on the edge to forecast values
- for bestOnline forecasting we use the strategy of “Selecting the Most Accurate” using the accuracy values to find the winner model and send its forecast also to the ESP device
- “Majority Rule” is implemented as well but only tested and not used in production

Model Comparison:

Linear Regression	Long short-term memory	SARIMAX
● (+) fast and lightweight training ● (+) easy to implement	● (-) heavy training ● (+) extremely flexible ● (+) helpful documentation in the internet ● (-) not robust against anomalies in training	● (-) heavy training ● (-) results are unsatisfying

Conclusion: In the context of predicting the value for fifteen minutes later, linear regression is the preferred one. It achieved best results in the forecasting (most of best-online is produced by LR). Also, it was the only one we could train on the IoT platform in the FaaS environment. LSTM also shows good results. We must say that we encountered late that our LSTM model is extremely sensitive to outliers (values > 45) which distort the training and so the outliers must be reduced to 45. Probably, it then would have performed better during the last weeks. Having more computational power on the IoT platform, LSTM would be a good choice and probably for other prediction settings a good choice due to its flexible adaptation of model components. SARIMAX predictions are not satisfying.

Anomaly Detection in Time Series Data

Used dataset

- The data was fetched from sensor 1276 (Index: 48_122_1276).
- The sensor is used as a receiver of the virtualized student room events.
- Recorded data goes from 1622805315 to 1624644558.
  → *4th June 13:15:15* to *25th June 20:09:18* (MESZ)

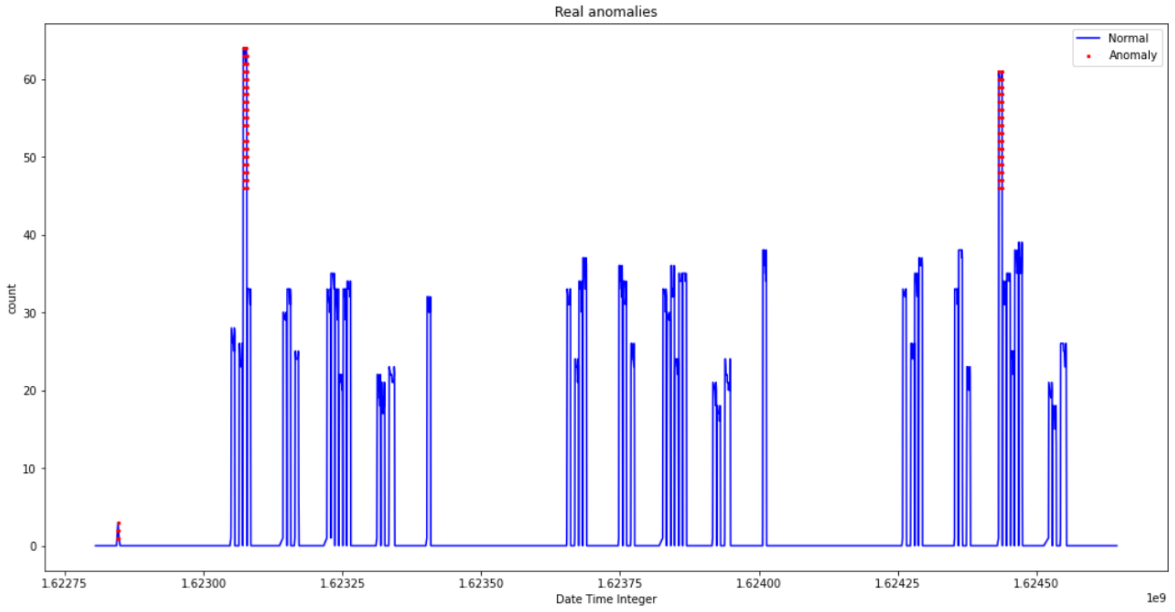
	t	count	hour_of_day	day_of_week	month_of_year
count	4.794000e+03	4794.000000	4794.000000	4794.000000	4794.0
mean	1.623776e+09	14.295160	13.833751	1.993951	6.0
std	5.119805e+05	12.975232	4.664959	1.603565	0.0
min	1.622805e+09	0.000000	0.000000	0.000000	6.0
25%	1.623258e+09	1.000000	11.000000	1.000000	6.0
50%	1.623791e+09	13.000000	14.000000	2.000000	6.0
75%	1.624287e+09	24.000000	17.000000	3.000000	6.0
max	1.624645e+09	64.000000	23.000000	6.000000	6.0

Basic anomaly detection and visualization of the dataset

To label a data point as an anomaly in the dataset (at least) one of the two conditions must hold:

- Any count bigger than 0 at the weekends is an anomaly.
- Any count over 45 is an anomaly.

Applying these conditions on our dataset, in which we artificially add some >0 counts on the first weekend, we get the following line diagram:



Interpretation:

- We see the artificial weekend anomaly as a first red dotted peak at the beginning.
- 2 high peaks are red dotted since they exceed the maximum room count of 45 → these events are real anomalies in this time period.
- Based on our anomaly conditions we can compute the **outlier fraction θ** which is required for the following detection methods. In our evaluation we get **θ = 0.01773059645390071**

KMeans based anomaly detection

General

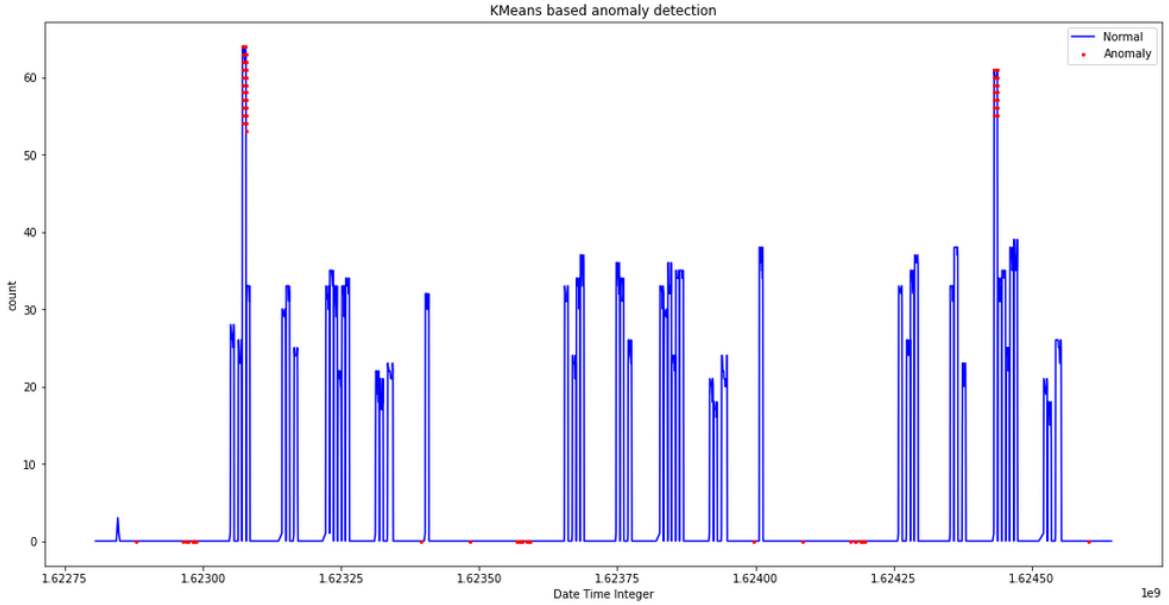
- Generate clusters: Each datapoint is assigned to the cluster with the nearest centroid.
- Compute distance between each point and its nearest centroid.
- Compute outlier threshold by using the outlier fraction θ → distances bigger than threshold are considered as anomalies.

Parameterization

- External parameters
  - Outlier fraction θ
- Internal parameters
  - Number of clusters
    → Using elbow curve to find best number
    - Run k-means multiple times with an increasing number of clusters.
    - Sum of squared distances from each point to centroid as objective function.
    - Take the cluster number that is “bend of elbow”.

Result

- Based on 4 clusters.



Interpretation

- KMeans is able to detect the room count anomaly when threshold of 45 is exceeded. However, not as precise as in our previous basic visualization, as we see some false negatives that exceed the threshold but are not detected as anomalies.
- 0 false positives that appear in some kind of schedule. Apparently, KMeans expects students in the room for these timestamps.

IsolationForest based anomaly detection

General

- Each data point is finally isolated on the isolation tree and has a path beginning at the tree’s root.

- If the path for a data point is long the point is regarded as non-anomalous, if the path is short it is easy to isolate the point, thus the point is regarded as anomalous.

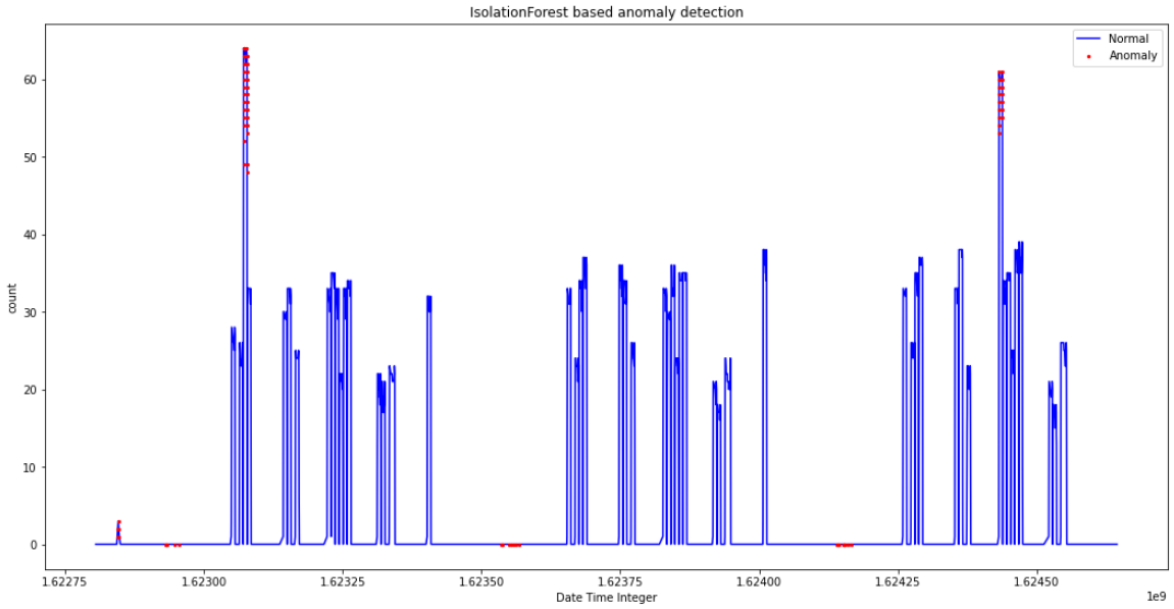
Parameterization

- External parameters
  - Outlier fraction θ
- Internal parameters
  - **number of estimator**s: the total number of isolation trees
  - **max samples**: number of samples used for training an estimator. If value is float, it will be multiplied with the number of rows of the dataset
  - **max features**: number of features. If value is float, it will be multiplied with the number of columns of the dataset
  - **bootstrap**: trees are fit on random subsets of the dataset and sampling with replacement is applied to the dataset
  - **warm start**: reuse gained information by adding result tree to the ensemble

  → Values are found by doing a parameter optimization study using **optuna**.

Result

- Based on: {'n_estimators': 100, 'max_samples': 0.5932156212744166, 'max_features': 0.7148931876176408, 'bootstrap': True, 'warm_start': True}



Interpretation

- IsolationForest is able to detect the room count anomaly when threshold of 45 is exceeded. We see some false negatives that exceed the threshold but are not detected as anomalies. Performs better than KMeans.
- IsolationForest is able to detect the anomaly of students in the room on the first weekend.
- False positives are dotted that come in some kind of schedule. Apparently, IsolationForest expects students in the room for these timestamps.

SVM based anomaly detection

General

- Uses the concept of support vector machine: Separate data points into classes by computing the linear separator with the largest margin possible.
- Use a one class support vector machine to find a function that is able to separate points in those lying in the small region with high density and those lying somewhere else → data points of those in the last group are regarded as anomalies.

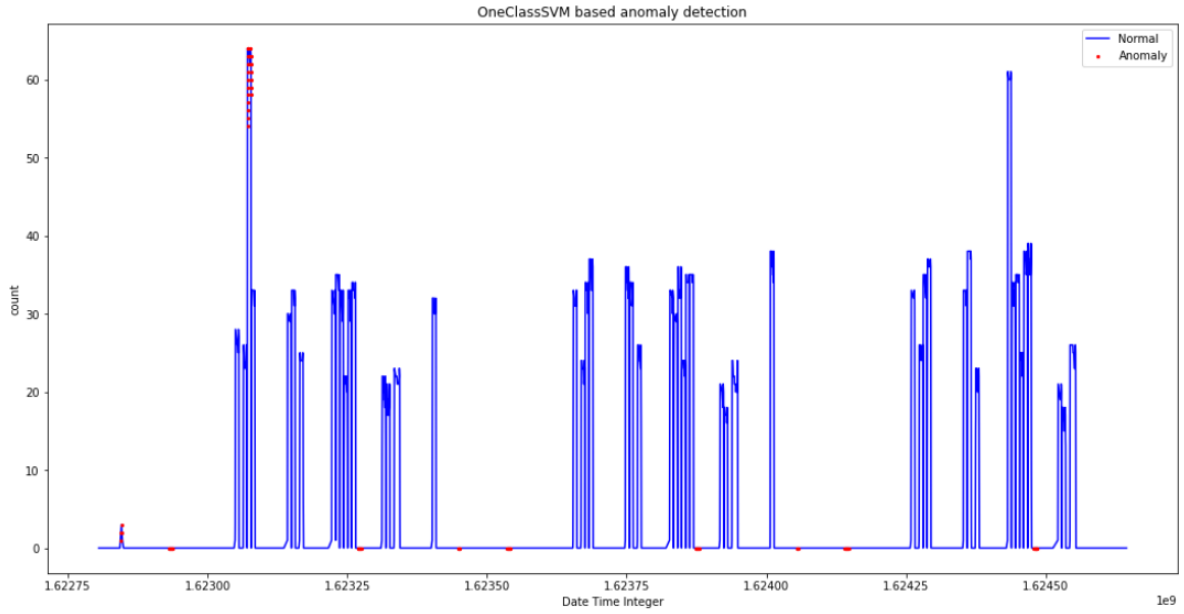
Parameterization

- External parameters
  - Outlier fraction θ
- Internal parameters
  - Kernel: Kernel type to transform the data
    - Selection: linear, polynomial, sigmoid, radial basis function.
  - Degree (if kernel=polynomial): Range [1, 5]
  - shrinking: optimization heuristic

→ Values are found by doing a parameter optimization study using **optuna**.

Result

- Based on: {'kernel': 'sigmoid', 'shrinking': False}



Interpretation

- Detects the first weekend with students as anomalies correctly.
- Detects the first peak that exceeds the threshold of 45 as an anomaly, however with some false negatives similar to the other approaches.
- Does not detect the second peak that exceeds the threshold as an anomaly.

Comparison

Anomaly Detection Method	Advantages	Limitations
KMeans	• easy to understand • #clusters as single internal	• different types of anomalies hard to detect

	variable ● easy to compute for single feature data	● optimal #clusters might change during time
IsolationForest	● no distance or density computations → tree can be computed fast ● low memory footprint: total number of tree nodes is (n + n -1) = 2n -1 where n is the number of data points → tree grows linearly with number of data points	● knowledge about proportion of outliers crucial for training
OneClassSVM	● specialized SVM algorithm for binary decision making	● gives low f1 score in our evaluation

Quick note on anomaly detection on virtual testbed

- Because of evaluation results we use IsolationForest.
- The IsolationForest model can be trained using FaaS on the IoT platform or locally.
- Anomaly detector in the edge uses the model and periodically evaluates the virtual room count data for anomalies.

Sources:

https://towardsdatascience.com/time-series-of-price-anomaly-detection-13586cd5ff46
https://towardsdatascience.com/clustering-metrics-better-than-the-elbow-method-6926e1f723a6
https://towardsdatascience.com/isolation-forest-is-the-best-anomaly-detection-algorithm-for-big-data-right-now-e1a18ec0f94f
https://medium.com/swlh/introduction-to-anomaly-detection-in-time-series-data-and-k-means-clustering-5832fb33d8cb
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html
https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html