Meta Data Inference on Building Sensors Data

Master-thesis

in partial fulfillment of the requirements for the degree of

Master of Science (M.Sc) Computer Science

submitted by

Muhammad Hamiz Ahmed

Matriculation Number: 4354150

on July 15th 2019

First Examiner: Prof. Dr. Hannah Bast
Second Examiner: Dr. Fang Wei-Kleiner
Supervisor: Dipl.-Phys. Tim Rist

Chair of Algorithms and Data Structure
Dept. of Computer Science, Faculty of Engineering
Albert-Ludwig-University Freiburg

Fraunhofer ISE
Freiburg

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date: Freiburg im Breisgau, 15-07-2019 Signature: _____

Acknowledgments

I would like to express my deepest appreciation to all those who provided me the possibility to complete this thesis. I would like to convey my gratitude to the thesis examiner Professor Dr. Hannah Bast for her useful remarks during the thesis.

I would also like to thank the supervisor of my thesis, Mr. Tim Rist, for his guidance, continuous support and amazing feedback and suggestions during the challenges faced in solving the problem of this thesis. His contribution in stimulating suggestions and encouragement, helped me to coordinate my project, especially in writing this report.

I would also like to thank Soundarya Palanisamy, for sharing useful insights of her experience in doing her thesis at Fraunhofer ISE.

Finally I would like to thank my family, friends and all my colleagues for their unconditional support throughout the thesis.

Abstract

Modern buildings are equipped with complex heating, cooling and ventilation systems. In the group "Building Performance Optimization" at Fraunhofer ISE, various methods are investigated for detecting faulty or suboptimal operation of such systems (e.g. simultaneous heating and cooling). These methods require data of a multitude of sensors for input (like temperature, pressure, current etc.). For this, a manual labelling of the sensor is required, which is excessively time consuming. The labelling convention called data point naming convention marks the origin and type of the sensor. Every sensor has recorded time series data as well as a small description text describing the nature of the sensor. In this thesis, we create a supervised machine learning system that utilizes the time series and description text data and automates the labelling of the data point name of the sensor. The proposed system is able to predict with an accuracy of 88% with 218 labels in the dataset, outperforming the baseline score of 76% which only used 81 labels.

Contents

List of Figures

List of Tables

1. Introduction

In today's world, when it comes to building construction, it is simply no longer enough for our homes and offices to just provide shelter and keep us warm. The evolution of technology has enabled us to not only provide all the services that occupants need but also make the buildings more efficient by minimizing the cost and reducing the impact of buildings on the environment. To achieve this, buildings are now equipped with high technology, complex equipment and sophisticated controls. The human experience in these smart buildings has been improved by the incorporation of automated systems for cooling, heating and ventilation. However, in order for these systems to operate, the buildings need to have different kinds of sensors installed, which could monitor different quantities like temperature, pressure, control signal, solar radiation, volume, heating energy, cooling energy, etc.

Even for sophisticated system configurations only correct operation of the building can ensure the demanded energy efficiency and hence, in order to optimize the performance of the buildings, there is a constant need to monitor the sensors installed in these buildings. The monitoring of the sensors does not only improve the performance of the building but, in case of malfunctions, also helps in the detection of faults and anomalies to keep the system running.

1.1. Motivation

In the group "Building Performance Optimization" at Fraunhofer ISE, various methods are investigated for detecting faulty or suboptimal states during operation of building automation systems. The buildings, that are investigated by the group, are equipped with a variety of sensors that measure different quantities of the buildings like temperature, pressure, current etc. However, the naming convention of these sensors varies from one building to another and is usually not machine readable. Furthermore, the collaboration of the group at collaborates with different partners also results in a non-uniform naming convention across the buildings. For instance, two temperature sensors of the same behaviour are labelled differently

across various buildings (Example: 033EFA8E-7AD1-4713-A0F1-178CAAE1C2ED and VFZH.HZ01.M172000004). Having different naming schemes make it unfeasible to analyze the measurement values of the sensors in these different buildings. Therefore, in order to make the measurement data usable for the analysis, a common data point naming convention is applied to each sensor, which marks the origin and type of it. The application of a common naming convention makes the analysis of the sensors easier and comprehensible across all buildings. However, applying this data point naming convention has to be performed manually, which is excessively time-consuming.

1.2. Problem Statement

In the data point naming convention for the sensor, discussed in the previous section, each label is comprised of a set of different metadata categories. These metadata categories describe the characteristics and origin of the sensor. For example, under the data point naming convention, an original point name label of a certain sensor can be BuildingX_ ZoneY_AHU_MTR.EL___MEA_I. This label is comprised of a set of individual class labels of different metadata categories described in Table 1.

Every sensor, like the one mentioned above, has a collection of readings, recorded at different points in time which we call the **Time Series Data**. The time series data is sensor specific and hence, the data of each type of sensor shows certain patterns. Figure 1 shows an example of a time series data for a **Temperature** sensor. Apart from this, each sensor is also described by a textual information, which we a call the **Description Text** of the sensor. For example, the sensor, whose label is shown in Table 1 has the description text "ELZ UV-ISP04 RLT 2 + RLT 4 Strom L2" describing the type of sensor to be **Current (I)**. The aim of this thesis is to create a system that utilizes the Time Series Data and Description Text of the sensor and automate the application of data point naming convention for the sensor using different machine learning techniques. The concrete problem formulation would be:

> *Develop a methodology for automatically mapping data sources to a hierarchically structured point name label based on raw time series data together with available texts in a supervised learning manner.*

Metadata Categories	Labels
Building	BuildingX
Zone	ZoneY
System	AHU
Subsystem1	-
Subsystem2	MTR.EL
Medium	-
Position	-
Kind	MEA
Point	I

Table 1.: Individual metadata categories for the original point name label **BuildingX_ ZoneY_AHU_MTR.EL___MEA_I**

1.3. Contribution to the field of Building Performance Optimization

In this work, we first generalize the original point name label of the sensor by removing the building and zone part from the label. Later, instead of using this generalized point name label as the target for classification, we train machine learning models to classify the class labels of each metadata category that make up the final point name label. To achieve this, we use various hand-crafted features, that were inspired from the work of [5], to develop a model, that could classify time series signals, of sensors, obtained from 13 different buildings across Germany, into their respective metadata category class labels. We also review the performance of artificial neural networks (ANN) like LSTM to classify time series signals using raw data points. Similarly, we evaluate different Deep Learning Architectures to classify the same set of sensors using their description texts, into their respective metadata category class labels again.

The predictions of the two types of models, obtained on these two different types of inputs, are combined using a *Meta-Classification Scheme*. Different schemes, like *Stacking* and *Voting* are evaluated for this purpose. The final point name label is then created using the prediction of individual meta-classification models. The performance of the system is examined by conducting Intra-Building evaluation. The results are compared with the results of previous thesis [5] performed at Fraunhofer ISE, to tackle a similar problem.

Finally, we tested the proposed system for its usefulness by creating a scenario where

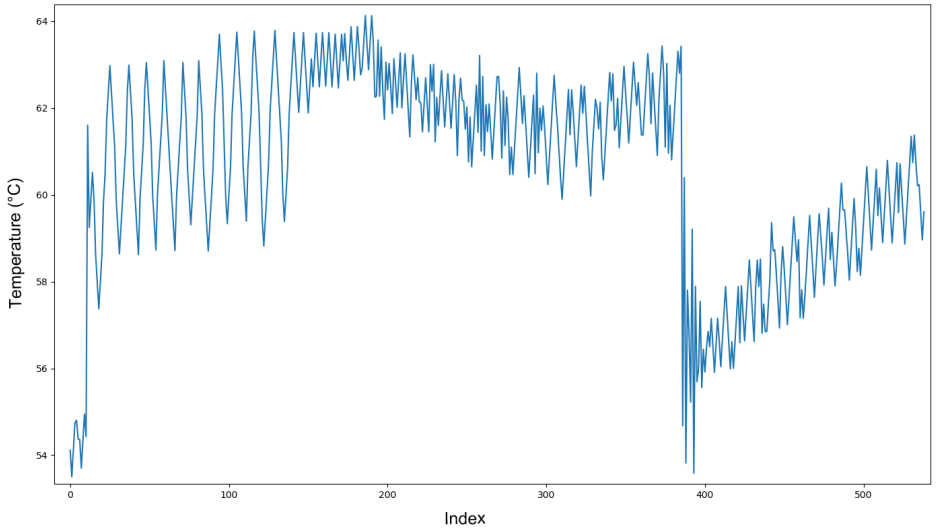


Figure 1.: Time series data of a single temperature sensor

sensors were categorized into different sets, each set containing sensors belonging to one building, and the and the system was tested out on each set building while it was trained on the rest of the sets. We called it the Inter-Building evaluation.

1.4. Structure of the thesis

The thesis is structured in the following way. The previous work by researchers on time series classification for statistical models and neural networks and classification of natural language is discussed in Chapter 2. The background of different machine learning models and concepts that have been used in this thesis are mentioned in Chapter 3. Chapter 4 and 5 discuss about the Data point naming convention, the extraction of features on the dataset and the pre-processing steps that have been taken to obtain quality data for the experiments. Chapter 6 discusses the approach that we took to solve our problem. Chapter 7 discusses the results of different models that were tried for different layers of the system architecture. Finally, the conclusion is discussed along with future steps in Chapter 8.

2. Related Work

Time series classification has always been a challenging problem in data mining. Time series are encountered in many real-world applications such as signature verification, person identification based on keystroke dynamics, detection of cardiovascular diseases and brain disorders (e.g. early stage of Alzheimer disease or dementia). The classification has been used in audio classification in the past and the performances were evaluated on Support Vector Machines (SVMs) and K-Nearest Neighbor [6]. It has also been used to classify the brain's normal and epilepsy activity with intracranial electroencephalogram (EEG) signals using Support Vector machines [7]. Deep Learning methods have also been used with great success in the area of Time Series Classification. H. Ismail Fawaz and G.Forestier [8] compared different deep learning frameworks for Time Series Classification by training 8,730 deep learning models on 97 time series datasets. [9] shows the advantage of using Deep CNN and Long short term memory(LSTM) recurrent networks in a single framework for carrying out the activity recognition task using time series data.

Similarly, the huge success of Deep Neural Networks has motivated the researchers to propose several Deep Neural Network (DNN) Architectures to solve tasks of Natural Language Processing, such as machine translation, learning word embeddings and document classification [10]. [11] proposes an architecture, that uses a combination of Recurrent and Convolution Layers with a pre-trained word embedding, for text classification. [12] describes the advantages of using very deep convolutional neural networks for the text classification, using very small convolutions and pooling operations and observes that the architecture beats the state of the art on many text classification tasks.

There is very few peer work related to this thesis with regard to building automation and sensor classification based on time series as well as natural language features. The thesis performed by Soundarya Palanisamy [5] uses time series data from different sensors to predict the final point name label of the sensors but uses limited number of labels. The proposed method in [13] was to deduce the metadata using trained statistical models like Random forest, KNN, Decision Tree, Naive Bayes, Radial Basis

Function kernels (RBF SVM), Logistic Regression, AdaBoost and Linear Discriminant Analysis (LDA). This thesis work differentiates in many aspects with [13] since it did not use the data from many buildings. This is significant because of the heterogeneity in the building infrastructure. Furthermore, it performs the classification using only Time Series Data of the sensors as the input. In 2015, Hong et al., [14] proposed a classification method which classified the different kinds of sensors streams in two buildings. They also described a method to find misclassified sensors. In [15], a similar problem that we have to solve in this thesis is chosen, yet the described approach uses a structured text source instead of the semi or unstructured text source. In this thesis, we not only classify various sensors to their general types using a well-suited architecture, which utilizes the information from the Time Series Data as well as the Description Text of the sensor but also apply different levels of detailed point names to the sensor.

3. Background

This chapter provides a brief background of the technologies and terminologies that we have used in this thesis. These explanations will aid the reader of this report in having a better understanding of the approach that we have adopted to solve our problem.

3.1. Machine Learning

Machine learning is an application of Artificial Intelligence that offers systems with the capacity to learn and enhance their performance automatically from experience. Machine learning focuses on computer programs that can access information and use it to learn on their own from a set of provided training data and make it generalized when used with unseen data. Nowadays, machine learning methods have been widely adapted to be used in various applications. Search engines like Google or Bing use these algorithms to learn to rank their webpages [16]. Similarly, Facebook uses it to personalize news feed of each of its members [17]. Machine learning algorithms are often categorized as supervised, unsupervised and reinforced.

Supervised Learning In this method, algorithms use the knowledge from the past and apply it to new data to predict the output. The input data provided to the algorithm contains labelled outputs and the model has to learn the patterns in the dataset by computing and adjusting its error for the expected output so that it can apply its learned knowledge on an unseen dataset and correctly predict the target. After a sufficient amount of training, the system is able to provide targets for any new input. An example of this problem is to analyze the sentiments of the people on the reviews of the movies. The model can be trained with a set of positive and negative reviews and then can be tested on a completely new review to predict either the sentiment is positive or negative.

Unsupervised Learning In this method, models are trained on the data that is neither classified nor labelled. The learning is performed by the algorithm on its own

by discovering and adopting, based on the patterns on the input dataset with the goal of discovering groups of similar patterns categorize them. An example of this method is the method used by Google News which groups new stories on the web and puts them into collective news stories.

Reinforcement Learning In this method, the agent learns to take suitable action to maximize reward in a particular situation. The agent tries to learn from its experience, that is, the outcome of its actions and decide its move in future from the information gained. The actions correspond to a reward or a punishment for the agent which makes it possible to achieve its goal by discovering the actions that result in the highest expected reward. A use of this method is to determine the best move to make in a game where, rather than specifying a set of hard-coded rules, the agent plays the game and learns from its experience to become better.

For the thesis, we use supervised machine learning methods only. We use some feature-based models as well as deep learning models which will be discussed in following sections.

3.2. Feature based models

3.2.1. Ensemble Learning

In machine learning, ensemble learning is a methodology in which multiple learners are trained to solve the same problem. These learners are called *Base Learners*. The ability, of ensemble learning methods, to generalize is much stronger when compared to the base learners. The ensemble methods are able to boost the performance of base learners by combining their predictions using different algorithms, thus reducing the variance. The ensembles are created by first creating a number of base learners on the input dataset and then combining their predictions using different combination schemes. The idea is that a diverse group of models are likely to make better decisions than a single one. There are many combination schemes that can be used in ensemble learning, however, for the thesis point of view, we will only discuss, Bagging, Stacking and Voting.

3.2.1.1. Bagging

Bagging is one of the famous ensemble learning techniques. It stands for Bootstrap Aggregating. It was first introduced in [18] by Breiman. The first step of the

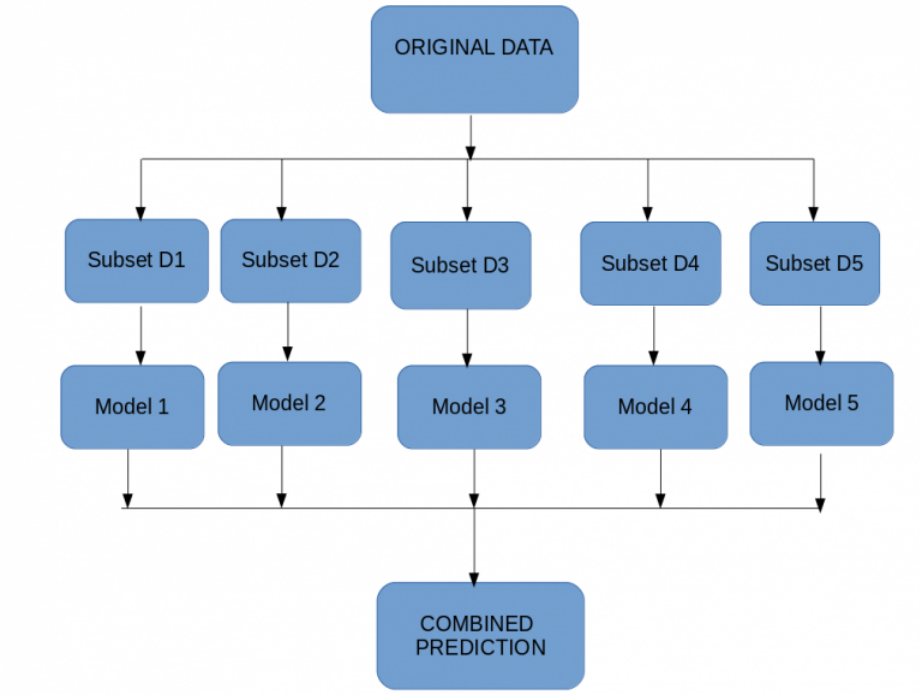


Figure 2.: The bagging procedure where a separate model is trained for each subset of Bootstrapped Data and final prediction is made by combining the predictions of individual models [1]

process involves the creation of subsets of observations from the original dataset, with replacement, also known as Bootstrapping. Once the subsets are obtained, an algorithm (base classifier) is selected to train on each of these subsets, parallel and independent of each other. The final predictions are determined by combining the predictions from all these models. Figure 2 shows explains the procedure. The most commonly used bagging algorithms are Bagging-Meta Classifier and Random Forest.

Bagging Meta Estimator Bagging meta-estimator can be used for both classification and regression problems. It follows the typical bagging technique, described above. However, the subsets of data include all the features. Also, the base learning algorithm and the number of base estimators are specified by the user.

Random Forest Random Forest is the most popular machine learning algorithm which uses the Bagging Technique and Decision Trees as its Base Classifier. Also referred to as forest of trees, random forest model uses the prediction from a number

of these trees to provide a single prediction.

Just like bagging, the algorithm starts by initially creating random subsets of the data from the original dataset, also known as the Bootstrapped Data. A decision tree model is fitted on each of these subsets, resulting in many decision trees. The tree is a binary decision tree which is constructed on the randomly chosen features from the Bootstrapped Data. The tree is split using a Top-Down Greedy approach called Recursive Binary Splitting. The construction starts from the top and each node is split recursively to obtain the next two branches in the tree. All input variables and all possible split points are evaluated and chosen in a greedy manner based on the cost function and at every split, a new feature is selected such that it has the lowest cost for that split.

Gini Index is used as the cost function for the decision tree split. Often referred to as 'gini impurity', the metric is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset [19]. The value ranges between 0 and 1 with 0 indicating that all cases in the node fall into a single target category. All the trees grow until stopping criteria of a specific number of data points in a leaf node are met. This helps to prevent overfitting the model.

All the trees are grown to the largest extent possible during which random forest optimizes the parameters at every node of the respective trees. Every tree is then a classifier where the vote of each tree is considered and the majority vote in the forest is considered as the predicted label for the data point.

3.2.1.2. Stacking

Stacking is a way to ensemble multiple classification or regression model. The idea of the technique is to improve the reliability of the overall prediction by exploring a space of different models for the same problem [20]. Different models can be used that are capable of learning some part but not the complete problem. Therefore multiple models can be trained on the data to create an intermediate prediction and then a new model is trained on these intermediate predictions which is referred to as the Stacked Model. The procedure for creating the Stacked Model involves the following steps:

1. Specify a base learning algorithm
2. Split the training data into k different folds. Perform k-fold cross-validation on each of these folds and collect the cross-validated predicted values for the base

learning algorithm. The idea is to obtain predictions by a version of the base model that is not trained on those samples.

3. Concatenate the predictions obtained from each fold, as shown below, such that the concatenation of all the prediction make up the whole training dataset. The prediction p_{11} describes the prediction obtained on fold 1 of the training dataset from base model 1.

$$\begin{bmatrix} p_{11} \\ p_{12} \\ . \\ . \\ . \\ p_{1k} \end{bmatrix}$$

4. Repeat step 1 to step 4 for each base learning algorithm. Combine the predictions from all the J base models as shown in the matrix below

$$\begin{bmatrix} p_{11} & p_{21} & ..... & p_{J1} \\ p_{12} & p_{22} & ..... & p_{J2} \\ . & . & ..... & . \\ . & . & ..... & . \\ . & . & ..... & . \\ p_{1k} & p_{2k} & ..... & p_{Jk} \end{bmatrix}$$

5. Use the matrix shown in step 4 to train the Stacked Model algorithm.

6. To obtain the predictions on the test set, train each base learner on whole training dataset. Use it to obtain predictions on the test set.

7. Feed the predictions obtained from the test set to the Stacked Model to obtain final predictions

Stacked Model helps in combining predictions and improving the overall performance. For the thesis point of view, Stacking is used to combine the predictions obtained from models trained on the time data and description text data of the sensors.

3.2.1.3. Voting

In general, voting can be described as a way of expressing decisions from a classifier. The decision, for classifying a sample in to a particular target can be expressed in the form of a prediction by a classifier which indicates the preference of the machine learning model. A voting method can then be used to integrate the votes of multiple classifiers in to one final decision according to two different strategies.

Hard voting In this case, the class that gets predicted most number of times by the classifiers C_j is chosen as the final label $\hat{y}$

$$\hat{y} = \text{mode}\{C_1(\mathbf{x}), C_2(\mathbf{x}), \ldots, C_m(\mathbf{x})\} \tag{3.1}$$

This approach works well when there are odd number of base classifiers. However, it doesn't take classifier's certainty into account while making the prediction.

Soft Voting In soft voting, the class labels are predicted based on the predicted probabilities p of the classifier. The probabilities for every class are averaged out and the class with the highest probability is selected as the final class label $\hat{y}$

$$\tilde{y} = \text{argmax} \frac{1}{N_{\text{Classifiers}}} \sum_{\text{Classifier}} (p_1, p_2, \ldots, p_n) \tag{3.2}$$

This approach works with any number of classifiers and uses classifier's confidence while making final prediction.

3.2.2. Logistic Regression

Logistic Regression can be defined as a special case of a generalized linear regression model and is based on the concept of probability. The linear regression technique uses a best fit straight line to set up a relationship between the dependent variable (Y) and one or more independent variables (X). The dependent variables need to be continuous while the independent variables can either be continuous or discrete. However, since this linear model outputs a numerical value, is not appropriate to quantify the qualitative response. The logistic regression uses a similar equation as linear regression but outputs the probability for the input data point of being a specific class. The equation below defines logistic regression.

$$y = \frac{e^{(\beta_0 + \beta_1 X)}}{1 + e^{(\beta_0 + \beta_1 X)}} \tag{3.3}$$

where y is the predicted output, β_0 is the bias or intercept term and β_1 is the coefficient for the single input value x.

The input data β coefficients are learned from the training data using an algorithm called Maximum Likelihood Estimation. In the end, the best coefficients would result in a model that will predict a number close to 1 for the default class and a number close to 0 for the other class. Hence, the values for the coefficients are searched in a way that would minimize the error in the model predictions to those in the data.

3.3. Deep Learning Models

In the feature-based models, that we have seen in the previous sections, one of the major issues is to select an appropriate feature space from the input data. In classification tasks, if the classes are not separable by a hyperplane, one can map the features into an intermediate feature space where the classes become linearly separable. However, mapping the features is an expensive procedure since it requires high computational time and expert knowledge [21]. This is where Deep Learning is helpful.

Deep learning is part of a broader family of machine learning. Deep learning networks are neural networks containing multiple layers and have the ability to learn to extract adequate features for a specified problem [22]. Yoshua Bengio, one of the pioneers of deep learning, describes in his paper [23], the ability of deep learning models to efficiently learn good representations of data using feature learning.

> "*Deep learning algorithms seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels, with higher-level learned features defined in terms of lower-level features.*"

For large datasets, parallel computation on the GPU has enabled efficient training of the deep learning networks. Although there are many types of deep learning algorithms. However, for this thesis, we will limit our discussion to Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN).

3.3.1. Convolutional Neural Networks

A feedforward network, in general, is trained on labelled data until it minimizes the error when guessing the category to make accurate classification. Convolutional Neural Networks (ConvNets or CNNs) fall in the category of feedforward deep neural

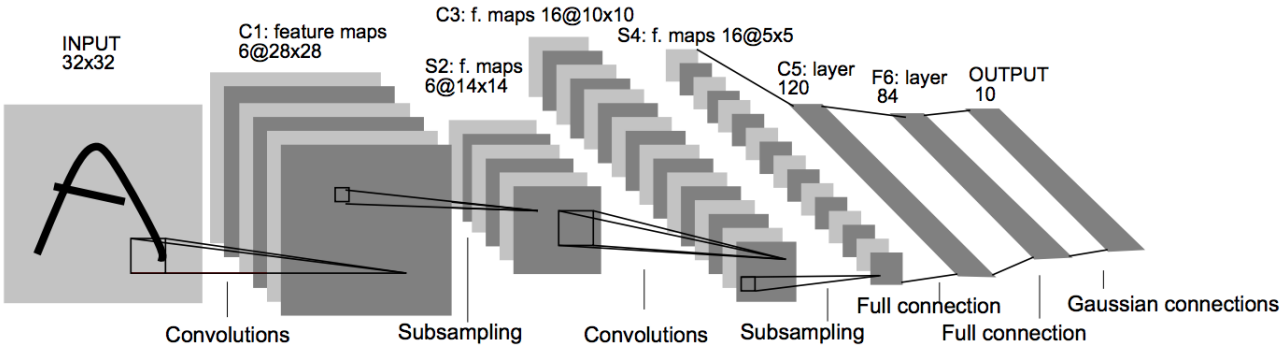
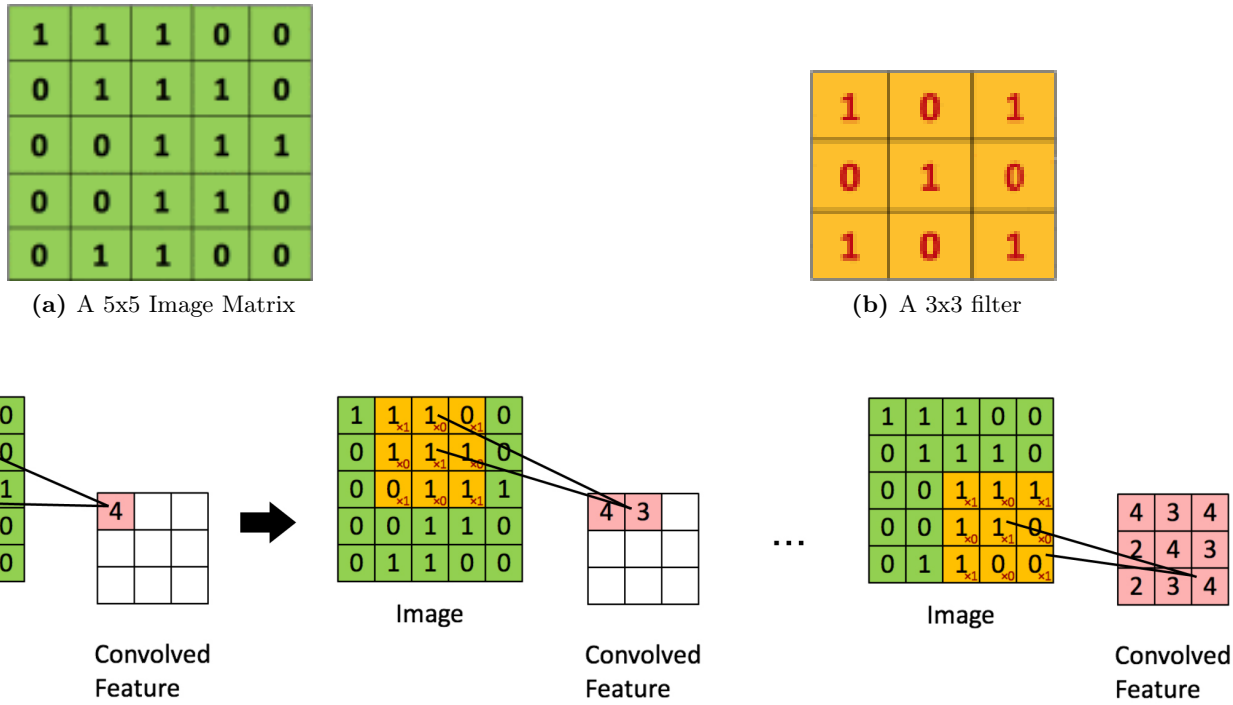


Figure 3.: LeNet-5 [2] architecture describing an example of Convolutional Neural Network

networks and have been found to be very effective in the areas of image recognition and classification. However, recently, ConvNets have been found to be very effective in several Natural Language Processing tasks as well [24].

ConvNets are similar to the connectivity pattern of neurons in the human brain and was inspired by the organization of Visual Cortex [25]. They treat each input as images which provides them with an advantage of constraining the architecture in a more sensible way. A typical ConvNet architecture usually comprises of multiple convolution, pooling and fully connected layers stacked together. Figure 3 shows a famous LeNet-5 convolutional neural network architecture proposed by LeCun et al. [2]. All the components that make up a ConvNet are introduced below.

Convolution Layer The convolution layer extracts features from the input image. An image is a matrix of pixel values. The layer divides the image into small squares which helps to preserve the spatial relationships between pixels. Consider a 5 x 5 image whose pixel values are only 0 and 1 shown in Figure 4a. Normally, even for grayscale images, pixel values range from 0 to 255. The figure is a special case which is created to better understand the concept of convolution. Consider another image shown in Figure 4b.



(a) A 5x5 Image Matrix

(b) A 3x3 filter

(c) Obtaining convolution feature through matrix multiplication and summing the toal

Figure 4.: The process of obtaining Feature Maps through Convolution Layer

The image matrix in Figure 4a is multiplied by the matrix in Figure 4b, also known as filter, to obtain the matrix which we call Convolved feature. The procedure is shown in Figure 4. The filter slides over the original image, 1 pixel at a time (also called 'stride') and for every position, an element-wise multiplication between the matrices is performed. The final outputs are then added to obtain the final integer output which forms a single element of the matrix shown in C. The value of the filter, shown in Figure 4b, is learned by the network on its own during the training process through Backpropagation. The size of the Convolved feature is controlled by the following two parameters that are decided before the convolution step is performed.

1. **Depth**: Depth indicates the number of filters that are used. High depth would correspond to a higher number of feature maps.

2. **Stride**: This can be defined as the number of pixels, a filter slides at each step, over the original image to compute the feature map.

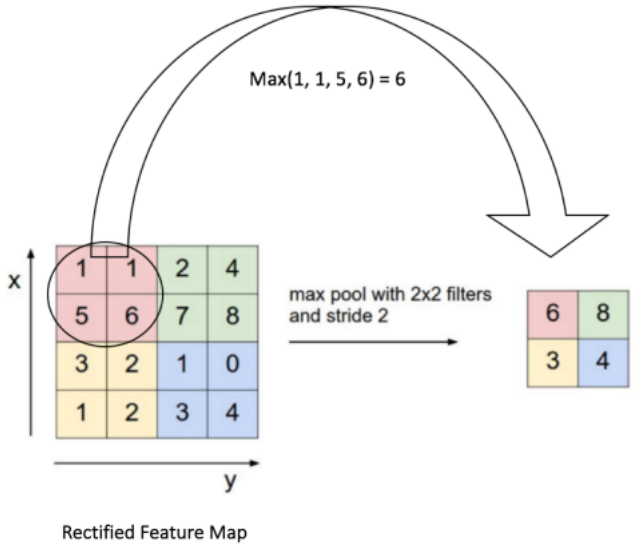


Figure 5.: An example of max pooling [3]

Pooling Layer The feature maps obtained from the Convolution Layer are high in dimension. Pooling layer reduces the dimensionality of each feature map retaining the most important information. A pooling window is defined of a specific size, for example, 2x2, which slides across the feature map with a fixed stride and uses a computing method, such that, no information is lost from the feature map. The most common methods are by computing the mean value, max value and L2-norm of a specific size. Figure 5 shows an example of Max Pooling. The window of size 2x2 slides by 2 cells (also called 'stride') in every step and the maximum value in each region is chosen to make up the reduced matrix.

Fully Connected Layer The output from the pooling layer can be provided to the Fully Connected Layer. The fully connected layer is a Multi-Layer Perceptron in which every neuron in the previous layer is connected to every neuron in the next layer. It uses a Softmax Activation function in its output layer. The features extracted from the convolution and pooling layer are utilized by the Fully Connected Layer to make the final classification.

3.3.2. Recurrent Neural Networks

The feed-forward networks, like Convolutional Neural Networks, described in the previous section, have a limitation of remembering the previous predictions and do

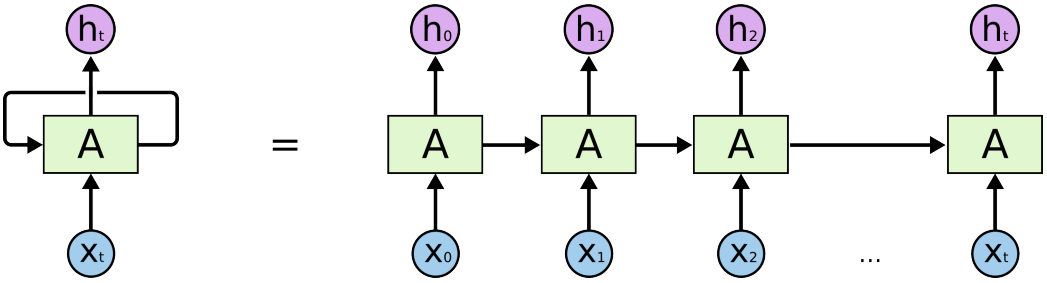


Figure 6.: RNN contain loop. The loop can be unrolled to go back to the states in time [4].

not make future predictions considering the predictions they had already made. For example, if CNN is trained to classify images of animals, the first photograph, it is exposed to, will not affect how it classifies the second image. Seeing the image of a dog would not lead the network to perceive a cat next. Therefore, while making predictions, feedforward networks only consider the current example it is exposed to. In reality, human brains do not work in the same way. Our future decisions involve the knowledge of past events as well. Recurrent Networks (RNN) solve this problem. While making prediction, they not only take the current input into account but also what has been processed previously in time. The network structure includes a loop, which allows the information to persist and pass from one step of the network to the next. Figure 6 shows an example of such a network. As shown in the figure, the network has a chain like structure and has two sources of input, the present and the recent past. The decision which is made at time t will be affected by the decision they have already made in the previous time step, *t-1*. This concept is also illustrated in the equation 3.4 which defines that the current hidden state h_t is a function f of the previous hidden state h_{t-1} and the current input x_t, having θ as the parameter of the function f.

$$h_t = f\left(h_{t-1}, x_t; \theta\right) \tag{3.4}$$

This makes them ideal to use for problems involving sequences and list. These networks are successfully used in problems like language modelling and speech recognition. The weights of the layers in RNN are learned through an extended variant of Backpropagation algorithm called Backpropagation through time or BPTT. The feedforward backpropagation algorithm cannot be directly used by RNNs because the algorithm assumes the connection of units to be cycle free. Therefore, the BPTT algorithm works by unfolding the RNN network in time through stacking different identical copies of the RNN with redirecting connections within the network to create

connections between the copies to create a network similar to feed forward [26]. The backpropagation algorithm can now be applied to the network.

However, RNNs are not suitable to capture long term dependencies. This would mean that the network would be unable to capture interactions between values, that are several time steps apart. This is because of the Vanishing Gradient Problem. In general, the weights of the neurons are updated proportionally to the partial derivative of the error function with respect to the current weight. However, for very large time steps, the gradient becomes very small (close to zero) and thus the value of the weight does not change resulting in stoppage of the training process. This problem was discovered in [27] by Yoshua Bengio in 1994. A solution to this problem is using Long Short Term Memory (LSTM) Networks proposed in [28].

3.3.2.1. Long Short Term Memory

LSTMs belong to the category of Recurrent Neural Network that has the capability of learning long term dependencies thus, they can remember information for long periods of time. Figure 7 shows the basic difference between a standard RNN and an LSTM network. All the RNNs form a chain like structure of repeating modules of the network. The repeating module in the standard RNN has a very simple structure, shown in Figure 7a. However, LSTMs, comprise of multiple neural network layers in the repeating module, enhancing their capability of remembering long term dependency.

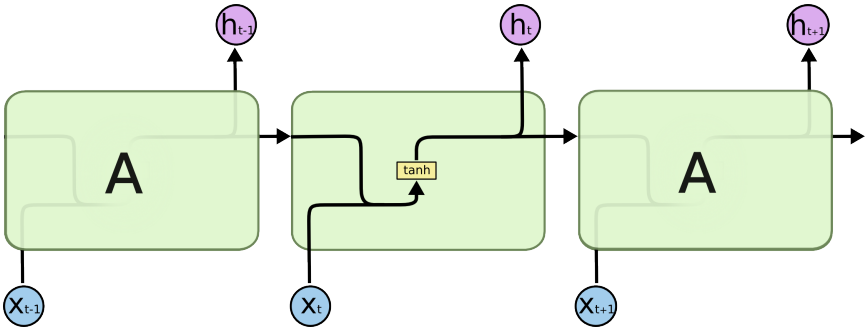
LSTMs maintain the flow of information through a gated cell. Just like a computer's memory, information can be read from or written to a cell. The cell controls the flow of information making decisions such as when to allow reads or when to allow information to be erased or written, with the help of the gates which are analog. The analog signal is differentiable and hence suitable for backpropagation. Equations below describe how the memory cells are updated in an LSTM at every time step t.

Initially, the values for input gate, i_t and the candidate value $\tilde{C}_t$ for the state of memory cell at time t are recorded.

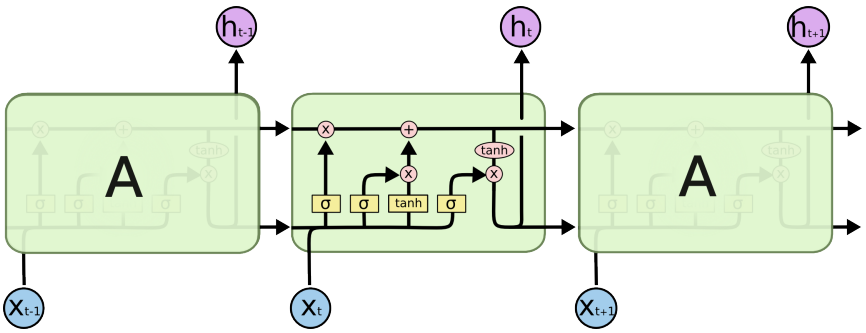
$$i_t = \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right) \tag{3.5}$$

$$\tilde{C}_t = \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right) \tag{3.6}$$

After computing i_t and $\tilde{C}_t$, the value for f_t is computed. f_t represents the activation of the memory cells forgotten at time t



(a) The repeating module in a standard RNN contains a single layer



(b) The repeating module in an LSTM contains four interacting layers.

Figure 7.: Comparison between a stadnard RNN and LSTM [4]

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \tag{3.7}$$

Once the value of input gate activation i_t, forget gate activation f_t and candidate state value $\tilde{C}_t$ are obtained, we can compute the value of C_t, the new state of memory cells at time t.

$$C_t = i_t * \tilde{C}_t + f_t * C_{t-1} \tag{3.8}$$

The output with the new state of the memory cell will then be as follows. o_t represents output gate's activation vector while h_t represents output vector of the LSTM unit.

$$\begin{aligned} o_t &= \sigma\left(W_o x_t + U_o h_{t-1} + V_o C_t + b_o\right) \\ h_t &= o_t^* \tanh\left(C_t\right) \end{aligned} \tag{3.9}$$

In all the equations described above, x_t is the input to the memory cell layer at time t, W_i, W_f, W_c, W_o, U_i, U_f, U_c, U_o and V_o are weight matrices while b_i, b_f, b_c, b_o are the bias vectors.

3.3.3. Bidirectional LSTM

The standard LSTM preserves information from the past since it only runs in one direction, from past to future. Bidirectional LSTMs run on input in two directions, one from past to future and the other from future to past. Therefore, at any point of time, the information from both the past and future is preserved. This information helps in many problems where information from the future is also necessary to consider before making any decision. For example, in word classification tasks, they can see the past and future context of the word resulting in much better classification of the word then the unidirectional LSTM.

3.4. Transfer Learning

Transfer Learning can be defined as a machine learning method where the knowledge of the model, trained on one task, can be used as a starting point for the second task. Usually, this is performed by first training a model on a base dataset and task, and then transfer the learned features to a second target network to be trained on a target dataset. Recently, this methodology has been used with great success on deep learning tasks which have limited data to train the model. However, the approach produces better results if the model features learned from the first task are general [29]. In our system, we use transfer learning by using pre-trained Word Embedding models for the classification of sensor's description text.

3.5. Word Embeddings

Word Embedding is a method of language modelling and feature learning technique where words from the vocabulary are mapped to vectors of real numbers. The reason for it becoming successful in the area of Natural Language Processing is its ability to capture the context of a word in a document, semantic and syntactic similarity, relation with other words. Thus the vectors created, represent similar meaning words, close to each other. The representation of words in the vector is learned through their usage in the document. This can be compared with the bag of words model where different words have different representations irrespective of how they are used.

The Word Embedding vectors are represented in a predefined vector space are learned in a way that resembles neural networks, hence, making them suitable to be used with deep learning architectures. There are different techniques that can be

used to learn a word embedding from text data. However, we will review only two techniques that are relevant, within the scope of this thesis.

3.5.1. Embedding Layer

An embedding layer has the ability to learn a word embedding jointly with a neural network mode on a particular NLP task which requires the input to be tokenized. The size of the vector space is specified manually and the vectors, which are the weights of the embedding layer, can be initialized with random numbers. The vectors are then learned in a supervised manner and are updated through Backpropagation Algorithm. This approach of learning embedding requires a lot of data but the vectors learned a specific to a certain task. However, if the weights of the embedding layer are initialized using a pre-trained embedding model, this method produces very good results.

3.5.2. GloVe

GloVe is a word embedding technique, designed by Pennington, et al. at Stanford [30]. The technique uses an unsupervised learning algorithm to obtain vector representation of the words. It has a set of available pre-trained models of different dimensions, trained on a very large corpus, that can be used either directly, or as a starting point for any NLP task. For example, the weights of the pre-trained model can be used to seed the weights of the Embedding Layer. GloVe constructs word context using statistics across whole text corpus resulting in a better word embedding.

3.6. Evaluation Metrics

The measures used to evaluate the performance of models in this thesis are Accuracy and F1-Score.

3.6.1. F-1 Score

F1-Score is a weighted average of both precision and recall. Precision is defined as the number of true positives divided by the sum of true positives and false positives. It defines the ratio of instances that are correctly classified over all the elements that were classified for that particular class.

Recall is defined as the number of true positives divided by the sum of true positives and false negatives. It defines the fraction of relevant instances that have

been retrieved over the total amount of relevant instances

For example, a computer program for recognizing dogs in photographs identifies 8 dogs in a picture containing 12 dogs and some cats. Of the 8 identified as dogs, 5 actually are dogs (true positives), while the rest are cats (false positives). The program's precision will then be 5/8 while the recall will be 5/12.

The F1-Score can then be defined as follows

$$F1\ Score = 2 \times \frac{\text{precision } \times \text{recall}}{\text{precision} + \text{recall}} \tag{3.10}$$

The F-1 score can be calculated individually for every class in the dataset. However, in a multiclass setting, the average of these scores need to be taken. There are different averaging methods that can be used. For the thesis, we used 'micro' averaging method since it takes into consideration the imbalance among the classes. In order to calculate the average micro F-1 score, the average score of precision and recall need to be calculated. They can be calculated as shown in Equation 3.11

$$\begin{aligned} Micro\ Average\ Precision &= \frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FP_c} \\ Micro\ Average\ Recall &= \frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FN_c} \end{aligned} \tag{3.11}$$

where c is the class label, TP, TN, FP, FN stands for True Positive, True Negative, False Positive, and False Negative respectively.

The Micro-F1 score can then be calculated with micro average values of precision and recall using equation 3.10.

3.6.2. Accuracy

Accuracy describes the closeness of the predicted value and true value.

$$Accuracy = \frac{TP + TN}{P + N} \tag{3.12}$$

4. Dataset Description

The datasets, in Fraunhofer ISE, are analyzed using a tool called *DataStorage* [31]. The tool not only helps in the storage and visualization of the time series data from different sensors but also helps in performing statistical analysis on it. This chapter discusses the details about the Data Point Naming convention and the dataset that we have used to analyze the system proposed in this thesis.

4.1. Data point naming convention

The input signals from different buildings, recorded at different points in time, are saved as time series data. Similarly, every sensor is manually assigned a small description text, which provides little information regarding its nature. In order to effectively analyze the sensor's data, the origin and type of sensor have to be marked. Also, the group at Fraunhofer ISE works with various partners, therefore, the original labelling of the sensor differs, for various buildings and systems.

Hence, in order to efficiently analyze a sensor's data to help in fault detection and use the *DataStorage* tool on the data of all buildings, the **data point naming convention** was created, which marks the origin and type of sensor. Applying the convention ensures that a unique name is assigned to each individual sensor.

The data point name for a single sensor, in a hierarchical order, starts with it's building name and ends with its sensor type. The meta-categories of data point naming convention are shown in Table 2.

The categories are separated by an underscore '_'. The specifications are appended after a dot in-order to be more specific. If a category is not used, it's left empty. For example, the original data point name for a given supply temperature sensor of an air handling unit in a building has the format, as shown in table 1.

4.2. Properties of the dataset

In the dataset used for this thesis, there are thirteen buildings from which data of different sensors is collected. These buildings are located at different parts of

Metadata Category	Description
Building	User defined Name of the building or abbreviation of name
Zone	User defined Name of Zone to which sensor corresponds (e.g. supply air, temperature)
System	main system to which the sensor belongs
Subsystem1	If appropriate: subsystem of system
Subsystem2	If appropriate: subsystem of subsystem1
Medium	Medium in which the sensor is placed
Position	Position of the sensor
Kind	Kind of the data point
Point	The physical quantity which is measured by the sensor

Table 2.: Metadata Categories Labels with their descriptions

Germany. The details regarding the sensor information present in each building, is shown in Table 3. All these buildings have the data of around 3300 sensors in total with each sensor comprising of just single description text and time series data of multiple days. The sensor names are referred to as original_point_name_label which consists of all the categories of the data point naming convention as shown in Table 2. The time series data of all the sensors, obtained from these buildings, is recorded at every minute of the day. This means that in a single day, a maximum of 1440 (60 minutes x 24 hours) raw sensor measure values, are recorded. Sometimes, there also occur some faults in the system during which the values are not recorded, resulting in missing values. From the thesis point of view, values recorded from **20.02.2015 to 01.08.2016** has been used, for the raw time series data.

No.	Building Names	Description
1	01_BZR_Ddorf	Office building of the district government Düsseldorf
2	02_DKB_Berlin	Office building of the Deutschen Kreditbank AG (mainly offices and server rooms; 9873 m^2)
3	03_KuP_Zentrale_Berlin	Headquaters of "Kieback & Peter" in Berlin (offices, cantina; 3716 m^2)
4	04_Lanuv_Essen	Property of the State Agency for Nature, Environment and Consumer Protection NRW (offices, labs)
5	EADS_88	LMT Group production site in Schwarzenbek near Hamburg (factory halls, offices, storage)
6	06_KPB_Mettmann	Police Department in Mettmann (offices, control center, lock-up, storage, shooting range; 5839 m^2)
7	07_Mendelsohn_Hamburg	Elementary school at Mendelsohn Street Hamburg (Offices, laboratories, canteens and class-rooms; 5839 m^2)
8	08_Sterntaler_Hamburg	Elementary school at Sterntaler Street (Offices, laboratories, canteens and class-rooms)
9	DVZ	Service and administration center Barnim (Offices; 8000 m^2)
10	Grosspoesna	Headquarters of ennovatis GmbH in Großpösna (Offices; 436 m 2)
11	Kraft_Foods	European RnD Center of Kraft Foods in Munich
12	VFG	Building of the University of Stuttgart, 8,200 m^2 office, laboratory
13	MWME	Ministry of Economic Affairs and Energy of the State of North Rhine-Westphalia, Düsseldorf (Offices; 30000 m 2)

Table 3.: Building Names and Description

5. Data Preprocessing

This chapter discusses the steps involved in preprocessing the data before using it for our supervised machine learning models. These steps are necessary to optimize the performance of the models. We begin with generalizing the original point name labels of the sensors, in order to avoid building specific classifications. We also discuss the handling of missing time series data and how we improved the quality of the whole dataset to improve the learning, during the training phase of the model.

5.1. Generalization of sensor names

The data is collected from various sensors in different buildings and the original point name label is specific to each building. For example, the original point name label consists of **building** and **zone** as it's first two entries. Since the behaviour of all the sensors is the same, we can obtain generalization by omitting these two categories from the original point name label. We refer these generalized point name labels as the final target classes for our classification task. Generalizing reduces the number of classes from 3300 to around 900. For example, the point name label of a current measuring sensor of an air handling unit AHU_MTR.EL___MEA_I, is obtained after generalization of the following original point name labels.

1. *MWME_ Office_AHU_MTR.EL___MEA_I*
2. *MWME_ R.3.407_AHU_MTR.EL___MEA_I*
3. *MWME_ LAB_AHU_MTR.EL___MEA_I*
4. *KPB_ SCHIESS_AHU_MTR.EL___MEA_I*

In the above original point name labels the building specific information like MWME, KPB and zone specific pieces of information like OFFICE, R.3.407, LAB, SCHIESS are excluded to generalize the sensor names to the point name label, AHU_MTR.EL___MEA_I.

5.2. Improvement of Data Quality

5.2.1. Time Series Data

As discussed in the above section, the generalization to the point name label helps to ease the classification task. However, missing time series data in the dataset poses a major challenge. Since the data from every sensor is recorded every minute, maximum of 1440 raw values are gathered in a single day. In some cases, the measurement procedure in the building is designed in a way that the sensor values are being recorded only when the change is noticed in the behaviour of the sensor. In case the values remain constant compared to a threshold then the data is not recorded in the system and appears as missing values. The missing values may also occur due to some faults in the system.

Also, sensors belonging to the same metadata category often show inconsistent behaviour. Figure 8 shows the recorded values of 10 different sensors belonging to the class **T** of metadata category **Point**. It can be observed that even the data belonging to the same class **T**, not only have missing values but also behaves differently depending upon the building it belongs to. The inconsistency is a result of multiple factors. The way in which the buildings are operated often varies from one location to another and it gets reflected in the measured values. Similarly, for some sensors, values are not measured continuously, but rather, in steps, adding to the inconsistent behaviour of the sensors.

In order to improve the quality of the time series data, we perform the following two pre-processing steps:

1. There should exist at-least 30 raw measured values collected in a single day for a single sensor. If not, then the values from that day are discarded

2. During the peak working hours in a day which is considered to be 6 am to 6 pm, there should not exist eight hours of continuous missing data.

After performing these pre-processing steps, we obtain minimum quality time series data which can be used for classification purpose. However, still, the inconsistency of the time series data makes it very unreliable to perform this classification, solely on it.

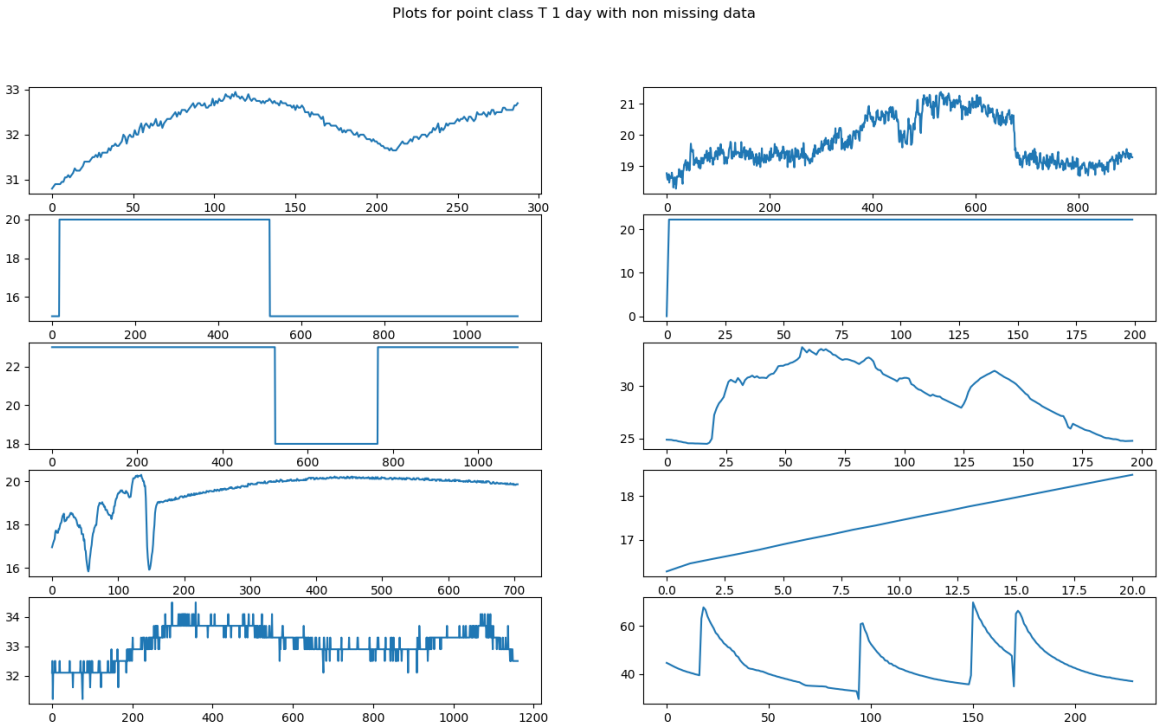


Figure 8.: Time series signals belong to the meta-data category of point for **T**. Y axes represent the measured temperature in $°C$ while X axes represent the indices of the data points.

5.2.2. Description Data

Beside a unique identifier, each sensor also has a description text assigned which is used to aid technical personnel at the building in understanding the nature of the sensor and its type. For example, a sensor belonging to the *Air Handling Unit (AHU)* system, measuring current can have a description text like *'ELZ UV-ISP03 Schaltschrank RLT 03 Strom L3'*. It can be observed that the terms used in description text are difficult to understand. However, they do exhibit some relation with the metadata category specifying the type of the sensor to be current. All available description texts associated with a sensor are written down manually making them very different from one building to another. Therefore, solely using description text to perform the classification task is also not a good idea. All the description texts are in German and all the word processing APIs that we use, handle the data in English

only. The first step involves the translation of the text to English. Once translated, every description text is tokenized and the stopwords are filtered out before it can be used further used for classification purpose.

5.3. Data Extraction

After performing the pre-processing steps, the raw time series data of sensors, from all the building databases, is extracted, using the *DataStorage* tool's API in python. Once the raw data is obtained, various handcrafted features are extracted from the data **(discussed in section 6.1.1.1)** and are saved in one large hdf5 file. Each sensor has multiple days of handcrafted features, with each row, corresponding to one day. The description texts of sensors, extracted from the same set of databases, are stored in a csv file along with their raw names, point name labels and individual metadata category labels (which make up the point name label). The description text of the sensor can then be directly accessed from the csv file while the handcrafted features for time series data can be looked up from the hdf5 file.

6. Approach

As discussed earlier, the point name label of the sensor is the target, that we want to predict. However, both the time series data and description text data are not good enough to be used independently for this prediction task. This chapter discusses the proposed architecture to solve the problem that we used and how and where it differs from the approach of the previous thesis [5] that was done in Fraunhofer ISE to solve a similar problem.

6.1. System Architecture

While performing classification tasks, it is important to make use of every available information that can help in obtaining better results. The approach that was used in the earlier thesis [5] utilized only time series data to classify the sensors. However, we have already discussed that time series data is inconsistent and poses a problem of missing values making it not comprehensive enough to be used independently. Therefore, in this thesis, along with the time series data, we also use sensor's description text data, to assist the classification of the sensors to point name labels. However, before the classification can be performed, there are two problems that need to be resolved.

1. Although, we have generalized the point name labels, as discussed in **Chapter 5**, the target still has over 900 classes. The dataset consists of information comprising from 3300 sensors only, not sufficient enough to train a model that could classify around 900 labels.

2. Both the input data (time series data and description text) are different in nature. Time series data is a sequence of recorded values taken at equally spaced points in time while description data is just textual information. Therefore, it would not be efficient to apply a single machine learning model on both data inputs.

Metadata Categories	Classes
System	78
Subsystem1	74
Subsystem2	26
Medium	24
Position	39
Kind	13
Point	52

Table 4.: Number of classes in each meta data category

As discussed earlier, the point name label is just the concatenation of seven different metadata categories. The first problem can be resolved by treating each metadata category of the point name label as an independent target. If each category is considered as an independent target and a separate model is trained to predict the class of each metadata category, the problem of too many classes with too few training data will be resolved. For example, let us assume that a sensor has a point name label of *AHU_PV_MTR.EL_SUPA_SUP_MEA_**T*** and another sensor has a point name label of *AHU_PV_MTR.EL_SUPA_SUP_MEA_**P***. The only differentiating entry between the two labels is the last, **point**, category, thus making them two different classes. However, if we treat AHU, PV, MTR.EL, SUPA, SUP, MEA and P as seven independent targets, we would only have to consider the classes in each of these metadata categories. This would mean that a significantly low number of classes have to be considered while training the machine learning model. Table 4 shows the number of classes present in each metadata category.

The second problem can be resolved if we train separate models for time series data and description texts, independent of each other. Therefore, we create an architecture, that not only treats each metadata category as an independent target but also makes use of time series data and the description text as two independent input data by training separate models for them, keeping in mind final prediction target of point name label. Figure 9 shows the architecture that we created for this thesis. The architecture comprises of three layers of models, named, **Base Models**, **Meta-Classification Models** and **Top Level Model**, each providing the input to the next layer. All these models are discussed in detail in the following subsections.

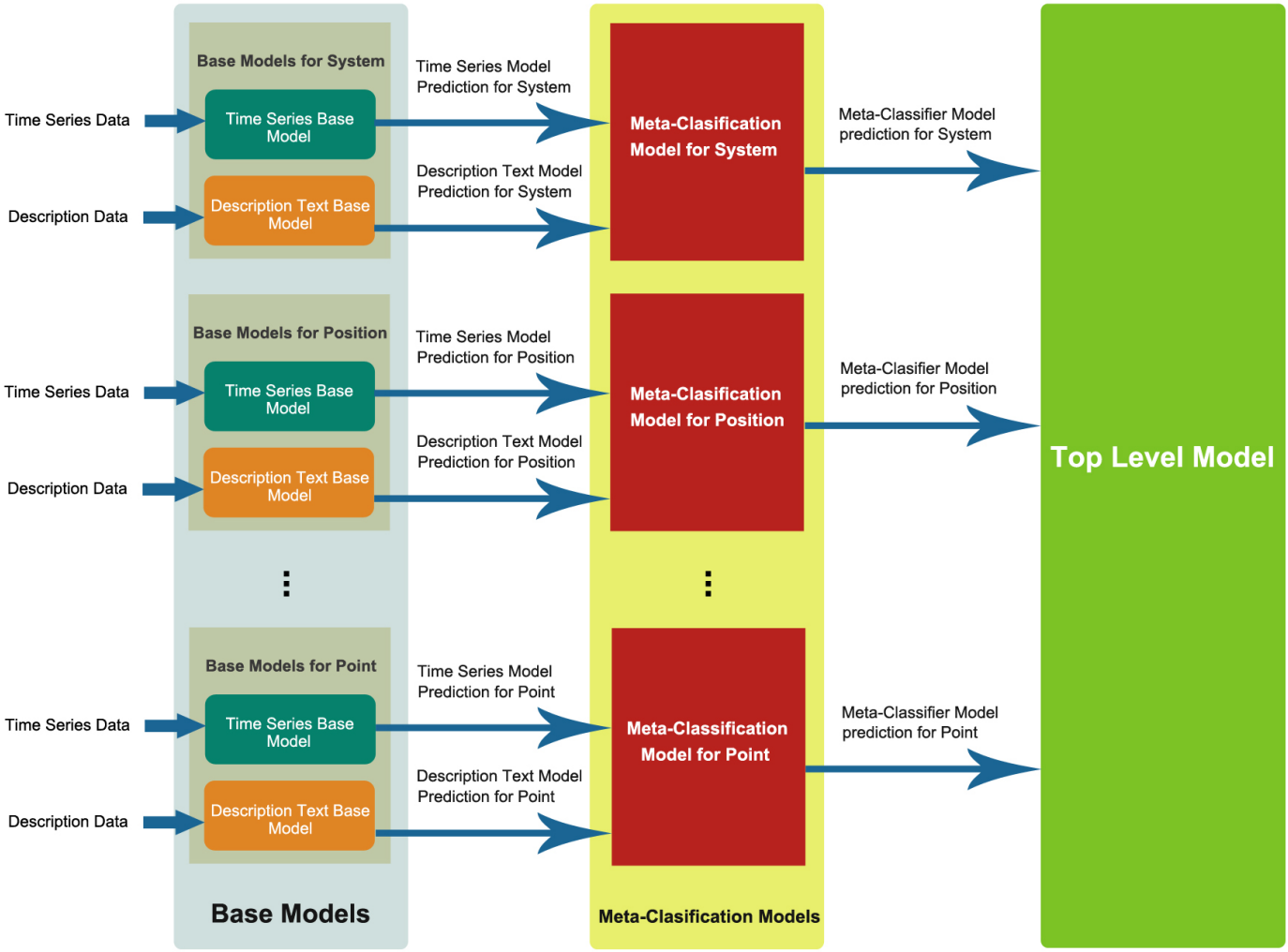


Figure 9.: The High Level Architecture of the System

6.1.1. Base Models

Base model layer is the first layer of the architecture of our system. It receives time series data and description text as input and trains two independent models on these for each metadata category as the target to predict, that is, for each metadata category, we obtain two models, one model is trained on the time series data, called the Time Series Base Model and the other is trained on the sensor's description text, called Description Text Base Model. This process is repeated for all seven different metadata categories. In the end, each base model predicts the class label of the metadata category, it is trained on. Figure 9 explains the use of base models in our

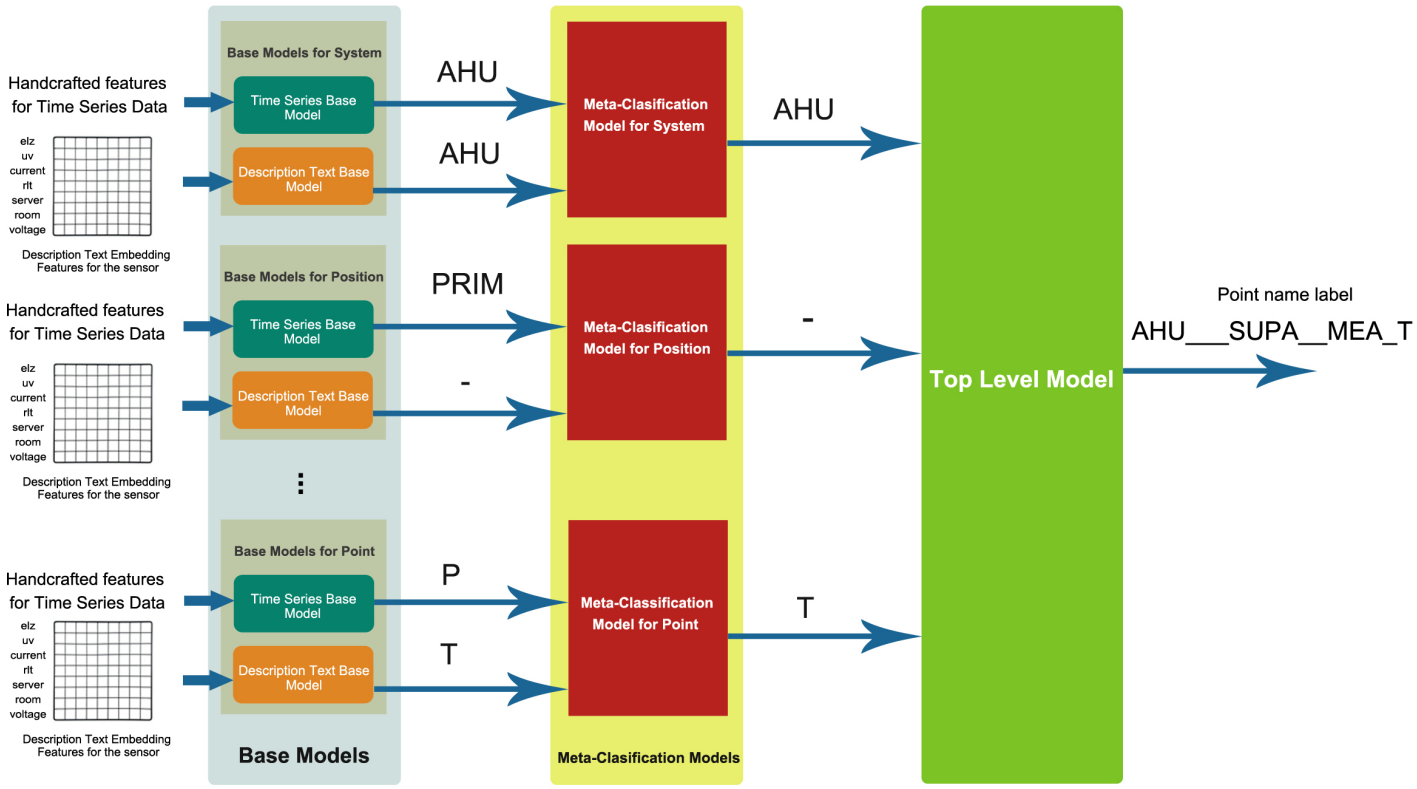


Figure 10.: Example of a data from a Sensor X, demonstrating the functioning of the architecture. The sensor has the description "ELZ UV Current RLT server room room voltage".

system. Figure 10 shows an example of the functioning of architecture for the system. As shown in the example in figure 10, the base model for each respective metadata category predicts its class label. The Time Series Base Model for system predicts AHU as its output based on the features of raw time series data. Similarly, a description text base model for the same metadata category predicts AHU as its output based on the embedding features of the description text. Similarly, for the metadata category point, the Time Series Base Model predicts T, while the Description Text Base Model predicts P. Once the predictions are obtained from Time Series and Description Text Base Models for all seven metadata categories class labels, they are forwarded to the second layer of the architecture, called the Meta-Classification Models.

6.1.1.1. Time Series Base Model

The raw time series data consist of the values recorded by sensors throughout the day. Every sensor has time series data of multiple days. Our aim is to use the time series data to train the Time Series Base Model which could predict the class labels of all the metadata categories of the sensor. We have seen in Section 5.2 that the sensors, even belonging to the same class, show inconsistency in their behaviours and have missing values. The difference in behaviour and missing information poses a major challenge for the prediction of each metadata category label using the input of raw data points. Therefore, it is required to project the data into a meaningful representation which makes it easy for the model, to classify the sensor.

Hence, in order to serve this purpose, various handcrafted features are extracted from the raw time series data, which are inspired from the work of [5]. These features handle the complexity of the data and help the classifiers to distinguish between different classes. In [5], these features were directly used to classify sensors in to point name labels. However, in this work, these handcrafted features are used by Time Series Base Models which classify sensors based on the class labels of their respective metadata category, which ultimately helps in the prediction of point name labels. All these handcrafted features were originally inspired from the work of [32].

The extracted features have been measured for each single day of the sensor data. This, not only helps to preserve the dynamics of the building in the extracted features, since the building operations follow a daily pattern but also keeps the model more robust. The daily operations in the building follow a specific pattern The features also help in overcoming the problem of missing data. Following features are extracted from the raw time series data of the sensors:

1. Mean of the Day The mean of all the raw time series values, recorded in a single day for a particular sensor

$$\mu = \sum_{t=1}^{N} \left(\frac{x_t}{N} \right), \quad N \leq 1440 \tag{6.1}$$

where x_t represents the sensor value at time t and N represents the number of data points of a single day.

2. Standard Deviation of the day The standard deviation of all the raw time series values of a particular day

$$\sigma = \sqrt{\frac{\sum (x_t - \mu)^2}{N}}, \quad N \leq 1440 \tag{6.2}$$

where x_t represents the sensor value at time t, N represents the number of data points of a single day and μ is the mean of the day

3. Minimum in a Day This is the minimum raw time series data value recorded in a single day

4. Maximum in a Day This is the maximum raw time series data value recorded in a single day.

5. Standard deviation of difference between consecutive elements in a day The difference between consecutive elements in a day can be defined as the difference of two consecutive elements together. For example, lets assume that $[x_1, x_2, x_3, x_4 \ldots\ldots, x_n]$ is a time series data measured in a single day. The difference between consecutive elements in a day (D) can then be defined as $[x_2 - x_1, x_3 - x_2, x_4 - x_3, \ldots\ldots, x_n - x_{n-1}]$. The standard deviation of the difference can then be defined as:

$$\sigma(D) = \sigma\left([x_2 - x_1, x_3 - x_2, x_4 - x_3, \ldots\ldots, x_n - x_{n-1}]\right) \tag{6.3}$$

6. Minimum of difference between consecutive elements in a day The minimum value of the difference of consecutive elements in the day. This can be defined as

$$\text{Min}(D) = \min\left([x_2 - x_1, x_3 - x_2, x_4 - x_3, \ldots\ldots, x_n - x_{n-1}]\right) \tag{6.4}$$

7. Maximum of difference between consecutive elements in a day The maximum value of the difference of consecutive elements in the day. This can be defined as

$$\text{Max}(D) = \min\left([x_2 - x_1, x_3 - x_2, x_4 - x_3, \ldots\ldots, x_n - x_{n-1}]\right) \tag{6.5}$$

8. Mean of hourly Standard Deviation Hourly standard deviation (Y) can be defined as the standard deviation computed on the raw time series data of every hour.

$$Y = \sigma(h_1), \sigma(h_2), \sigma(h_3), \sigma(h_4), \sigma(h_5), \ldots\ldots\ldots, \sigma(h_H), \quad H \leq 24 \tag{6.6}$$

where H represents the number of hours and $\sigma(h)$ represents standard deviation of data recorded in an hour.

Using this equation, the mean of hourly standard deviation can be defined as follows:

$$\mu(Y) = \sum_{h=1}^{H} \left(\frac{\sigma(h)}{H} \right), \quad H \leq 24 \tag{6.7}$$

9. Standard Deviation of Hourly Standard Deviation From equations 6.6 and 6.7, this can be defined as:

$$\sigma(Y) = \sqrt{\frac{\sum(\sigma(h) - \mu(Y))^2}{H}} \tag{6.8}$$

10. Maximum of hourly standard deviation

$$\text{Max}(Y) = \max\left[\sigma(h_1), \sigma(h_2), \sigma(h_3), \sigma(h_4), \sigma(h_5), \ldots\ldots\ldots, \sigma(h_H)\right] \tag{6.9}$$

11. Minimum of hourly standard deviation

$$\text{Min}(Y) = \min\left[\sigma(h_1), \sigma(h_2), \sigma(h_3), \sigma(h_4), \sigma(h_5), \ldots\ldots\ldots, \sigma(h_H)\right] \tag{6.10}$$

12. Standard deviation of absolute/real values of Discrete Fourier Transform (DFT) The time domain signal of a single day is transformed to frequency domain using Discrete Fourier Transform (DFT). The standard deviation is then taken of the real value coefficients of obtained frequency domain signal.

13. Max of absolute/real values of DFT From the obtained frequency choose the maximum real value of computed fourier transform.

14. Min of absolute/real values of DFT From the obtained frequency choose the minimum real value of computed fourier transform.

15. Min frequency obtained in DFT Compute the discrete fourier transform sample frequencies and choose the minimum value.

16. Max frequency obtained in DFT Computed the discrete fourier transform sample frequencies and choose the maximum value.

17. Median frequency obtained in DFT Choose the median value from computed the discrete fourier transform sample frequencies.

18. Spectral Entropy Spectral Entropy defines the complexity of the system. To calculate the spectral entropy, following steps are taken:

1. Calculate spectrum $S(x_t)$ of the time series signal.

2. Calculate the the power spectral density of the signal

$$P(x_t) = \frac{1}{N} |Y(x_t)|^2 \tag{6.11}$$

3. Normalize the power spectral density

$$p_t = \frac{P(x_t)}{\sum_t P(x_t)} \tag{6.12}$$

4. Calculate the spectral entropy (SE) with the following equation:

$$SE = -\sum_{t=1}^{N} p_t \ln p_t, \quad N \leq 1440 \tag{6.13}$$

19. Number of peaks Find values of local maxima in a set of data containing measured values from a sensor of a single day.

20. Power The power of raw data points is calculated as:

$$Power = \frac{\sum_{t=1}^{N} x_t^2}{N} \tag{6.14}$$

where x_t is a raw data point in a single sensor, N is the total number of data of points.

Figure 11 shows the representation data from sensors after the handcrafted features are extracted from them. The features belong from the sensors of class label **T** for metadata category **Point**.

Popular machine learning algorithms like Logistic Regression, Random Forest and MLP have gained great interest on feature based prediction. For the Time Series Base Model, initially, we start our experiments by training machine learning models on these handcrafted features and evaluated the results. However, the performance of

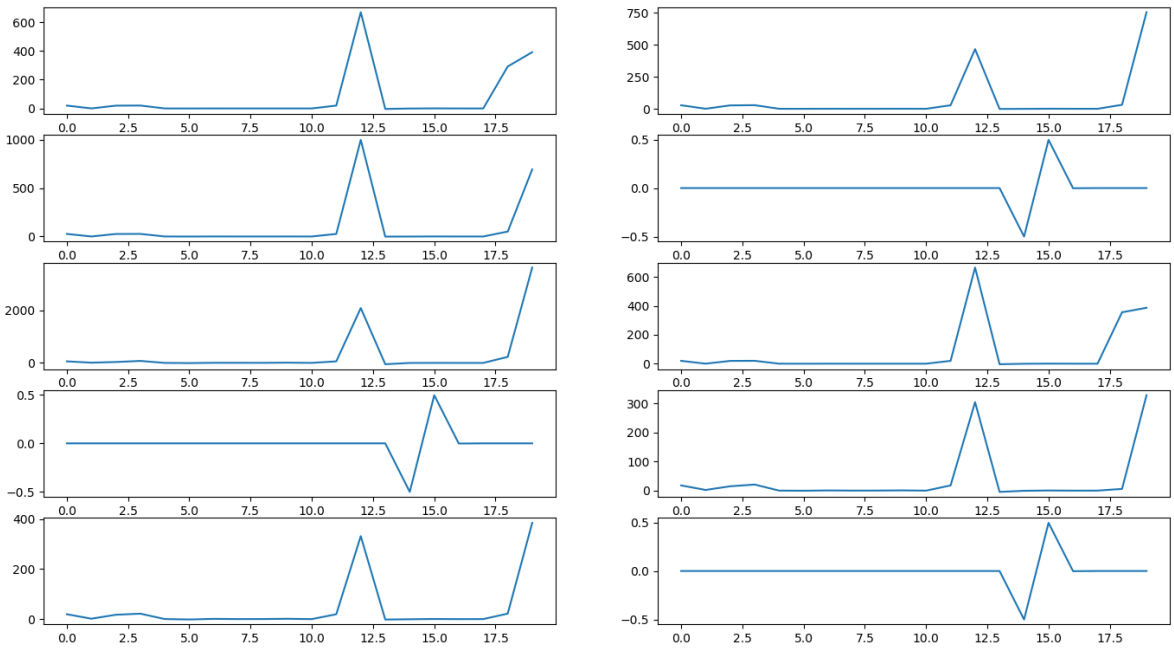


Figure 11.: The representation of handcrafted features for meta-data category **Point** with class label **T**. Y axes represent the measured values while X axes represent the feature index.

these models is highly dependent on these extracted features. Hence, we later analyse whether the performance of a model would further improve when the features are extracted automatically from the raw data using a feature learning technique like deep learning. The results of both these experiments are discussed in Chapter 7.

6.1.1.2. Description Text Base Model

The description text of a sensor specifies information about the metadata category it belongs to. Description Text Base Models train on the description text of sensors to predict the class labels of different metadata categories. However, to train the model, we need a way to represent text data, to effectively extract information out of it. For example, consider the following sensor descriptions:

1. ELZ UV server room Room E033 Voltage L1

2. ELZ UV server room Room E033 Output L3om E033 Output L3

	elz	uv	server	room	e033	voltage	output	l1	l3
Description 1	1	1	1	2	1	1	0	1	0
Description 2	1	1	1	2	1	0	1	0	1

Table 5.: Count Vectors of each description

It is not possible for the computer to understand these descriptions directly. They need to be represented into a numerical form for any machine learning model to process them.

One of the ways to represent this is to learn the vocabulary from all the documents, then represent each document by counting the number of times each word appears, also known as Count Vectors. The two descriptions in our example have the vocabulary as *[ELZ, UV, server, room, E033, voltage, output, L1, L3]*. The count vector matrix for these descriptions is shown in Table 5.

However, these frequency based approaches have several drawbacks. First, there is a problem related to the dimensions of the vector. The size of the vector increases with the size of the vocabulary. Large document set will have a larger vocabulary which will also increase the size of the vector. Secondly, since, the words are independent of each other, the semantics and the relationship between the words cannot be established by this approach. This would mean that phrase like 'ELZ UV server room' would not be treated as a continuous sentence but rather, four separate words. This would also mean that sentences, where same words have a different context, would not be reflected in the vector. For example, sentences like "Apple is a fruit" and "Apple's products are expensive" have different meaning of the word 'Apple' here. In the first sentence, Apple is used as the name of the fruit, while in the second, Apple is used as the name of the brand.

In recent times, to tackle this problem, there has been an increasing trend in the use of word embeddings. Word embeddings are versatile tools that help in solving many basic problems in the areas of Information Retrieval and Natural Language Processing [33]. Currently, they are state of the art in NLP [34]. In general, word embedding maps the words of a text data, into a continuous low dimensional vector space such that the internal semantic and syntactic information of the words can be captured [35]. Each word is represented by a fixed size vector, no matter how large the corpus is. Taking the example from above, using word embeddings, the word 'Current' in description 1, can be represented as a vector $[0.1, 0.3, \ldots, 0.2]^k$. k here defines the size of the pre-defined vector space. Unlike the frequency based approach, not only do these vectors have fixed dimensions, but they also represent the semantic

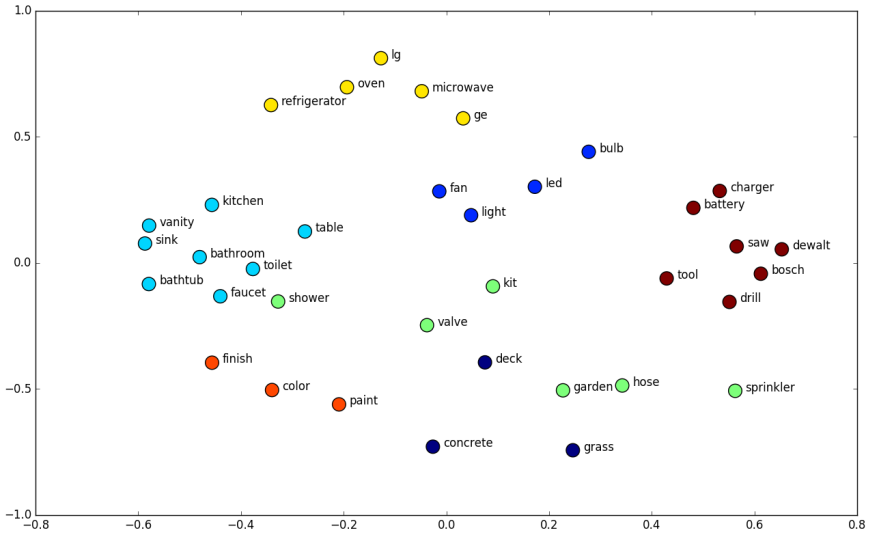


Figure 12.: The relationships among words obtained through word embeddings on a general vocabulary. Similar meaning words are close to each other

relationship among words. Hence, the words, that have similar meanings, will have vectors that will be close to each other. Figure 12 shows a general plot of the word vectors obtained after embedding. Notice that the words with similar meanings are closely plotted.

Deep learning models, combined with Word Embedding Vectors have been found to be increasingly effective in the field of Natural Language Processing. [36] showed in his work that a simple deep learning framework outperforms most state-of-the-art approaches in several NLP tasks. In general, the process of learning these vectors is either joint with the neural network model on some task, such as document classification, or is an unsupervised process, using document statistics. For each process, there are multiple methods, which can be used to learn these embeddings. Two of the most popular methods are the GloVe (Global Vectors for Word Representation) [30] and Word2Vec [34]. GloVe is an algorithm developed by Jeffrey Pennington, Richard Socher and Christopher D. Manning at Stanford, that uses unsupervised learning to obtain vector representations for words. Similarly, word2Vec, developed at Google, is an embedding creation algorithm that uses two-layer neural networks that are trained to reconstruct linguistic contexts of words.

However, learning useful embeddings, in either of these algorithms, require a large amount of text data, such as millions or billions of words to capture the semantics. In our dataset, unlike raw time series data, where each sensor has several days of measured values, description texts are limited in numbers. There are only 3300 sensor descriptions in total, each corresponding to one sensor, not sufficient to train the word embedding from scratch.

Transfer learning with pre-trained word embeddings has been used with great success in text classification [37] to tackle the problem of small training datasets. The idea is to obtain word embedding vectors from a model, that has already been trained on a large-scale dataset and then use those vectors to conduct learning for another target task. Since the model is usually trained on a diverse dataset, the resulting word vectors are high in quality.

Because of huge success in the field of NLP, we use deep learning architecture to train different Description Text Base models in our system combined with pre-trained vectors from Glove [38] that have been trained on the vocabulary of Common Crawl data [39]. These deep learning models are trained for each metadata category. All the vectors for sensor description text vocabulary are extracted from the pre-trained vectors which are then used as weights to seed the embedding layer of the deep learning architecture of Description Text Base Model. Figure 13 shows the result of using pre-trained Glove Embedding on the vocabulary of description texts of sensors. The words similar to 'temperature' for the vocabulary of description texts are highlighted in the figure.

Figure 14 shows the use of embedding features in the model. Initially, all descriptions are tokenized and a word index of the vocabulary is created such that each unique word of the corpus is assigned a unique number. Consider the example of descriptions from above. In order to extract the embedding vectors, these descriptions are first encoded by assigning each word a unique integer number. Table 6 shows the word index obtained from the vocabulary of the two descriptions mentioned above. The two descriptions could then be encoded as [1, 2, 3, 4, 4, 5, 9, 6] and [1, 2, 3, 4, 4, 5, 8, 7] respectively, based on their created index. These description vectors are then passed on to the embedding layer whose weights had already been initialized using the pre-trained Glove embedding. The weights of the embedding layer is a table of embedding vectors with each row representing the index of the word from word index. Table 7 shows an example of these vectors that are relevant for our example. The embedding layer looks up the table and maps the integers in the description to

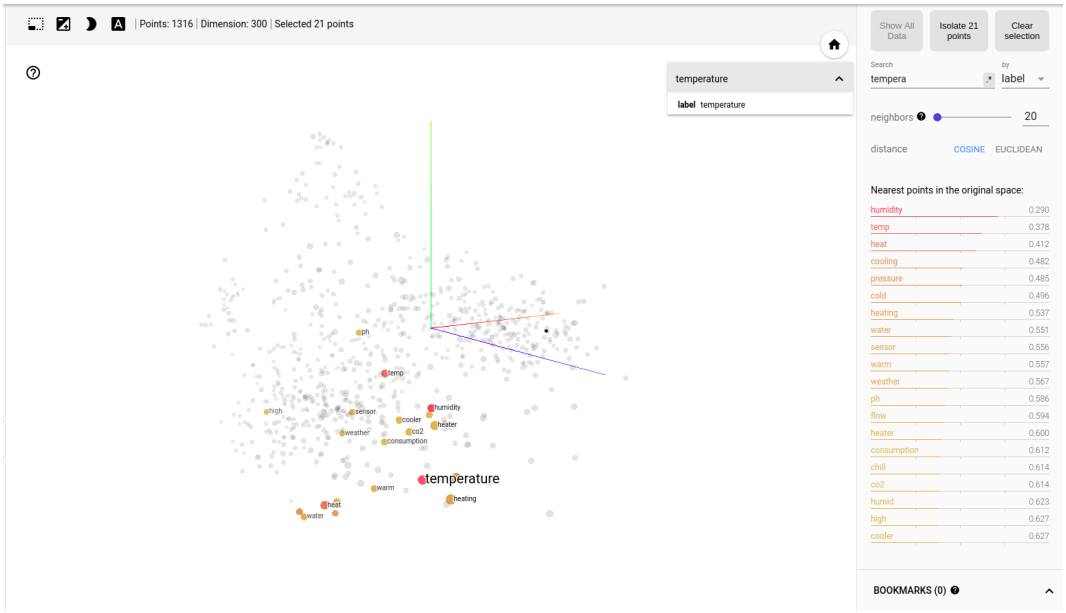


Figure 13.: Demonstration of word embeddings using pre-trained vectors on the vocabulary of description texts for the training Description Text Base Models

embedding vectors. Therefore, the first description will be represented as follows:

$$[[1.2, 3.1], [0.1, 4.5], [1.4, 2.1], [2.0, 1.0], [2.0, 1.0], [4.7, 1.5], [1.7, 2.6], [3.2, 1.9]]$$

These embedding vectors are passed on to the next layers like CNN or LSTM for training.

The Embedding layer has the ability to either freeze its pre-initialized weights, or update the weights based on Backpropagation. There are some special words in the vocabulary whose vectors are not found in the pre-trained embedding, for example, words like 'ELZ' and 'L1' depict characteristics of metadata categories, and thus these words cannot be discarded. The vectors of words like these are initialized by zero and since the weights of the embedding layer are the pre-trained vector of words in the vocabulary, the model would produce better quality results if the weights further get adjusted with respect to the training target. Hence, as the model is trained, the weights also get updated and are adjusted to minimize the loss of the model.

Once the training is completed, the Description Text Base Models, for each metadata category, can be used to get the prediction of the class label.

Words	Index
elz	1
uv	2
server	3
room	4
E033	5
l1	6
l3	7
output	8
voltage	9

Table 6.: Word Index for descriptions mentioned above

index	embeddings
0	[1.2, 3.1]
1	[0.1, 4.5]
2	[1.4, 2.1]
3	[2.0, 1.0]
4	[4.7, 1.5]
5	[3.2, 1.9]
6	[0.1, 1.5]
7	[1.0, 2.0]
8	[1.7, 2.6]

Table 7.: An example of word embeddings for descriptions

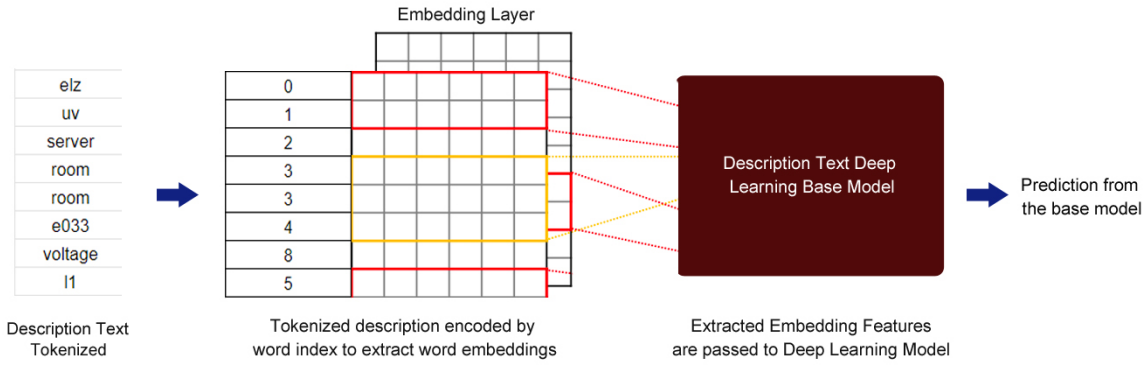


Figure 14.: The description is tokenized and embedding vectors are extracted before feeding it to the Deep Learning Base Model

6.1.2. Meta-Classification Models

Meta-Classification scheme can be defined as an ensemble technique which, based on the characteristics of dataset or other simpler learning algorithms, can be used to predict the right algorithm for a particular problem [40]. In our architecture, meta-classifiers are used on the predictions of base models. For each metadata category, we obtain class label predictions from two base classifiers, the Time Series Base Model and the Description Text Base model. The aim is to select the correct metadata category class label obtained from these two base models. The output from this layer of models is a single metadata category class label prediction. Combinations of multiple base classifiers decrease variance and produce reliable classification. Taking the example from Figure 10 for the metadata category point where the Time Series and Description Text Base Model, both predicted their output to be T and P respectively, the Meta-Classification Model for the category point would use a meta-classification scheme to determine which output is correct and chose the correct class label based on it. In the example, it chose T to be the correct output.

There are multiple meta-classification schemes that can be used. For the thesis point of view, we performed our experiments using Stacking and Voting.

6.1.2.1. Stacking

Stacking can be defined as an ensemble technique which determines the reliability of the classifiers using a meta-learner. In stacking, a number of base classifiers are combined using one meta-classifier which learns their outputs and determines which base classifier prediction is more reliable on the type of input [41]. The technique uses predictions from multiple models to build a new model. This model is used for making predictions on the test set. A more detailed explanation of the Stacking Algorithm can be found in section 3.2.1.2.

In our system, the stacked model is trained on the predictions of base models. The idea is to make the Stacked Model learn which classifier is more reliable on the type of data it receives. There are multiple machine learning algorithms like Bagging, Random Forest or Logistic Regression that can be used as Stacked Classifier Algorithms. The process starts by obtaining the predictions from base models on the training and testing sets of raw time series data and description text, for each metadata category. It is important that the Stacked Algorithm is trained on a separate dataset to the examples used to train the base level models to avoid overfitting. One way of achieving this is to divide the training dataset into two parts, that is training and validation set and train the base models on training dataset while train the stacked model on the predictions obtained from validation dataset. However, the problem with this approach is that the Stacked Model is not trained on the whole training dataset. A more concrete approach is to use k fold cross validation to create training dataset for the Stacked Model. We adopted a similar approach where we divided our training dataset into 2 folds. Both base models are first trained on the first fold while the predictions are obtained on the second fold. These predictions are saved and the process is repeated again where the base models are trained on the second fold and predictions are obtained from the first fold. Therefore, the predictions are made on the unseen data which provides a better generality for the Stacked Model to be trained on. Table 8 shows an example of a training dataset for training a Stacked Classifier for the metadata category *Point*. The features for training the Stacked classifier are the predictions from the Time Series and Description Text Base Model while the target is the original metadata category class label of the sensor.

6.1.2.2. Voting

Voting is a meta-classifier for combining the results of different base classifiers for classification via majority voting. There are two types of voting methods named

Time Series Base Model Prediction	Description Text Base Model Prediction	Original Label
U	U	U
I	T	T
P.EL	P	P.EL
VOL	VOL	VOL

Table 8.: Example of Stacked Classifier Training Dataset for Point

The Time Series Base Model Prediction and Description Text Base Model Predictions are used as features for the Stacked Classifier with Original Label of the sensor for a particular meta-data category as the target

"hard" and "soft" voting. Hard voting predicts the final class label as the label which has been most frequently predicted by the base classifiers. On the other hand, soft voting predicts the final class label based on averaging out the prediction class probabilities of the base classifiers. Table 9 shows a general example of the difference between Soft and Hard Voting. Hard Voting predicted the class label as A while soft voting, based on the confidence of the classifiers, predicted the final class label as B.

Since, we have even number of base classifiers, using hard voting would result in conflicts. For example, for any metadata category, if the output of one base classifier is different then the other, then it is impossible to determine the final class label. Hence, soft voting works best for our system. The prediction class probabilities from the time series models and the description text base models are combined and then averaged out. The class label with the highest probability is then taken as the predicted output. This method ensures that the confidence of base classifiers, with which they predicted the class label for the metadata category, is taken into account while making the final prediction and hence, the final output doesn't have to rely on any external classifier.

6.1.3. Top Level Model

Once the predictions from Meta-Classifier Models are obtained, they can be used in the following two ways:

1. The predictions from the meta classifier model of each metadata category can be concatenated with an underscore '_' to form a point name label

2. The predictions from the meta classifier model of each metadata category can be used to train another machine learning model, which we call the Top Level Model.

Classifiers	Prediction
Classifier 1	A
Classifier 2	A
Classifier 3	B
Voting Classifier	A

(a) Hard Voting Predicting A as the predicted class

Classifiers	Class A Prediction Probability	Class B Prediction Probability
Classifier 1	0.6	0.4
Classifier 2	0.5	0.5
Classifier 3	0.1	0.9
Soft Voting (Mean Probabilities)	0.4	0.6

(b) Soft Voting predicting Class B to be the predicted class

Table 9.: Comparison between Soft and Hard voting

Meta-Classifier Model	Predicted Class Label
System	AHU
Subsystem1	-
Subsystem2	-
Medium	SUPA
Position	-
Kind	MEA
Point	T

Table 10.: Combining predictions result in point name label of AHU___SUPA___MEA_T

The output of both these methods will result in a point name label for the sensor.

6.1.3.1. Combining Meta Classifier Predictions

We obtain a class output from each meta-classifier of the metadata category. We already know that the point name label is a concatenation of class labels of single metadata categories, separated by an underscore '_'. Therefore, combining the output of meta-classifiers provides us the final point name label of the sensor as the output. Table 10 shows how the meta-classification predictions of example shown in Figure 10 can be concatenated to form the final point name label.

Meta-classifier for System	Meta-classifier for Subsystem1	Meta-classifier for Subsystem2	Meta-classifier for Medium	Meta-classifier for Position	Meta-classifier for Kind	Meta-classifier for Point	Original Point Name Label
-	MTR.EL	MEA	-	-	-	U	AHU_MTR.EL_MEA___U
EGEN.C	CCH	MTR.EL	MEA	-	-	U	EGEN.C_CCH_MTR.EL___MEA_U
AHU	-	MTR.EL	MEA	-	-	U	AHU__MTR.EL___MEA_T

Table 11.: The training set for the Top level Model
All the predictions from Meta-Classifier are used as features for the model while the original point name label is used as target

6.1.3.2. Training Top Level Model

Although combining predictions from meta-classifiers provides us the point name label that we want but there are still few things which it doesn't take into consideration. The point name label marks the type of the sensor which describes its characteristics. This means that all the metadata categories are correlated with one another, hence, complementing one another. For example, a sensor with the **system** entry water circuit will usually not have a sensor with the system entry water circuit will usually not have a point category of **pressure**. This pattern can only be recognized if a machine learning model like Random Forest or SVM is trained on the outputs of meta-level classifier. Hence, in order to train the top level model, we create a new dataset, in which class outputs from all the meta-classification models are used as features for the model and the original point name label of the sensor is used as the target. Table 11 shows an example of such a training set. It can be seen from the table that there are some instances where the original point name label differs from the point name label which could have been obtained by combining the predictions of meta-classifiers. This is where Top Level Classifier can help in reducing the error in the overall model. Even if any meta-classifier model, somehow predicts an incorrect class label for any metadata, the Top Level Model will ensure that this error is not reflected in the final predicted point name label.

7. Experiments and Results

In this chapter, we present an empirical evaluation of the architecture described in Chapter 6, along with variations of some components. In summary, we have a model with three different layers, and in each layer, there are different machine and deep learning models. Popular algorithms like Logistic Regression, SVM, KNN, Random Forest and MLP have gained great interest on feature based prediction. Similarly, deep learning algorithms like Convolution Neural Network and Recurrent Neural Networks for text classification have been widely used with great success in the field of Natural Language Processing. Initially, we examine different machine learning algorithms, using K fold cross validation, for each layer of the architecture, and chose the best performing set of algorithms. Later, we evaluate the performance of this architecture on Inter-Building cross validated dataset and test the impact of user-simulated feedback on the results.

7.1. Experiments Specifications

7.1.1. Implementation

The implementation of this thesis is performed using Python 3 [42]. All the implementation of statistical machine learning algorithms are imported from scikit-learn [43]. For deep learning models, we use keras [44] with tensorflow [45] as its back-end. For data preprocessing and mathematical computations, libraries like Numpy [46] and Pandas [47] are also used.

7.2. Preliminary Experiments

7.2.1. K Fold Cross Validation

To cross validate, we need to divide the data into k-folds. However, there are two problems which need to be solved in order to properly distribute the data. First, in order to preserve the relative class frequencies in each fold, the data should be proportionately distributed among all folds. However, the dataset is highly imbalanced.

Table 12 shows the distribution of sensors in top 15 point name labels. Since there are over 900 point name labels with around 3300 sensors, there exist many classes which contain data of just a single sensor under them. This problem is resolved by using a variant of k-fold called Stratified k-fold. Stratified k-fold ensures that each fold is a good representative of the whole dataset.

Secondly, for using stratified K-fold, we need to choose a target so that the data is proportionally divided based on it. However, in our architecture, Base Models and Meta-Classification Models classify the sensors into their metadata category and the Top Level Model classifies the output of Meta-Classification models into the point name label. Hence, for the first two layers, metadata categories are the target classes while for the last layer, the point name label is the target to predict. This problem is resolved by choosing point name label as the target class for creating different folds. As discussed earlier, the point name label is just a concatenation of all metadata categories. Therefore, if the point name label is chosen as the target class for creating all the stratified k folds, then all the metadata categories are also proportionately distributed in every fold.

The creation of the folds results in the reduction of classes in the test set. This is because all the point name labels, which had just single sensors under them are put into the training set. This ensures that the classifier sees the data of sensors from all the classes, even if it is not tested on it. Table 13 shows the number of classes for each metadata category in the test set that are obtained after the stratification has been performed. The classes have been reduced from their original number previously shown in Table 4. The total number of point name labels are now 218. Chapter B in the appendix shows the classes that are used in the testing set for each metadata category. The training set includes all the classes of the dataset.

Throughout all experiments, the value of k is chosen to be 3.

7.2.2. Architecture Evaluation

We tested out different machine learning and deep learning models for each layer of the architecture. As discussed earlier, our dataset is highly imbalanced, therefore, evaluating models based only on their accuracy wouldn't be optimal. Therefore, along with accuracy, we use the micro-F1 score as the evaluation metric for the models.

point_name_label	Total number of sensors
_____RA___MEA_T	212
___MTR.EL_____MEA_P.EL	107
_____RA___SEV_T	101
RAD_CTRV___HW___SIG_STAT	95
___MTR.EL_____MEA_I	88
___MTR.EL_____MEA_U	87
WIN_SIG_STAT	80
_SIG_STAT	61
AHU_____SUPA___MEA_T	47
___MTR.EL_____MEA_E.EL	38
AHU_____EXHA___MEA_T	36
WC.H_PU___HW_SEC_SIG_STAT	34
WC.H_____HW_SUP.SEC_MEA_T	32
WC.H_____HW_RET_MEA_T	31
WC.H_____HW_RET.SEC_MEA_T	30

Table 12.: Top 15 point_name_labels based on the number of sensors

Metadata Categories	Classes
System	38
Subsystem1	24
Subsystem2	13
Medium	13
Position	11
Kind	7
Point	27

Table 13.: Number of classes in each meta data category in the test set after performing k-fold cross validation

7.2.2.1. Base Models

We already discussed in Section 6.1 that the purpose of Base Models is to predict the class label for each metadata category of the sensor. Here, we discuss the experiments that were carried out on base models with different statistical machine learning and deep learning algorithms for each base model.

Time Series Base Model

The thesis of Soundarya Palanisamy [5] evaluated different machine learning models for the hand-crafted features extracted from Time Series Data and found Random Forest to be the best performing model. In general, Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, good results most of the time. It is also one of the most used algorithms, because of its simplicity. Similarly, deep learning methods have been widely adopted in the Time Series Data classification tasks. Unlike feature based models, deep learning methods learn and extract the features from the raw data automatically. Recurrent Neural Networks and Convolutional Neural Networks (CNN) have been observed to perform Time Series Data classification with good results. However, not only do these models require a huge amount of training data and time, a lot of hyper-parameters need to be tuned as well.

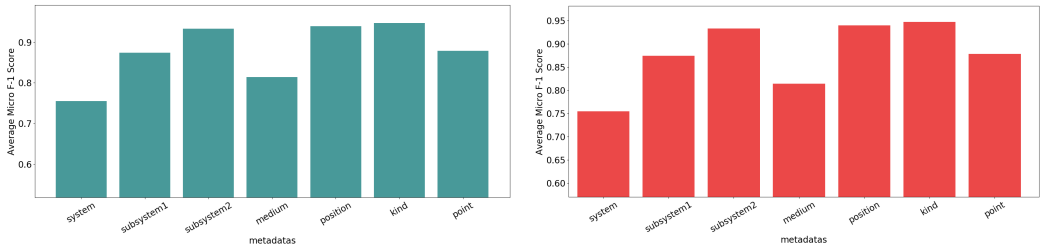
In this thesis, we perform the following two experiments for the Time Series Base Models:

1. Train Random Forest Classifiers on the hand-crafted features of raw-time series data with each metadata category as the target.

2. Train Deep Learning Algorithms like LSTM on the raw-time series data of every sensor with each metadata category as the target.

Training Random Forest Classifier The Random Forest Classifier is trained on the handcrafted features extracted from the raw time series data. Since each sensor has multiple days of measured data and the handcrafted features are extracted on the data of one single day, there are multiple days of handcrafted features that are available for a single sensor. Therefore, in order to predict the final metadata category class label for the sensor, the model is analyzed on the features of all the days and the most frequent predicted label is chosen as the final label for the sensor. For example, let us consider that SensorA had data of 5 days and we are analyzing a Time Series Base Model for the metadata category **system**. All the extracted handcrafted

features for all 5 days would be provided to the model as input and the prediction of the class label for every day would be recorded. Let us say that the model predicted the class label 'AHU' 4 times and 'RAD' 1 time based on the data of 4 days and 1 day respectively. The final class label for sensor_A would then be 'AHU' since it was predicted with majority among all days. The number of estimators that we use for the model are 60, and the criterion is set to 'gini.' The model is trained directly on the hand-crafted features of the sensors with the number of classes mentioned in Table 13 for each metadata category. Figure 15 shows the mean accuracy and micro-F1 score of all the three folds for all metadata categories.



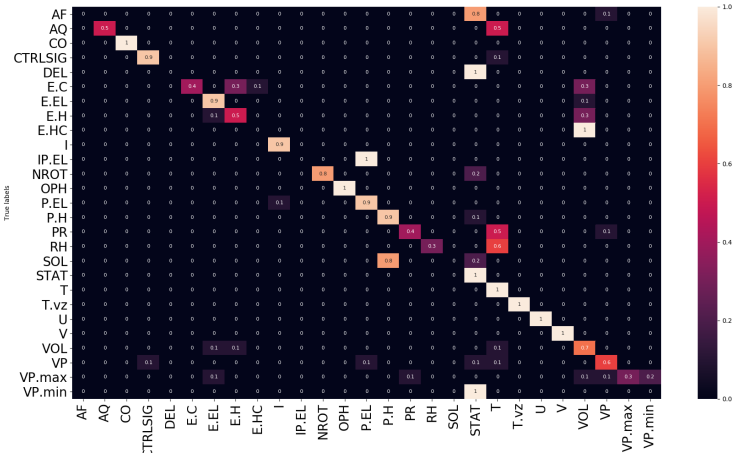
(a) Accuracy Results with RF model for each meta data category

(b) Micro F1-Score Results with RF model for each meta-data category

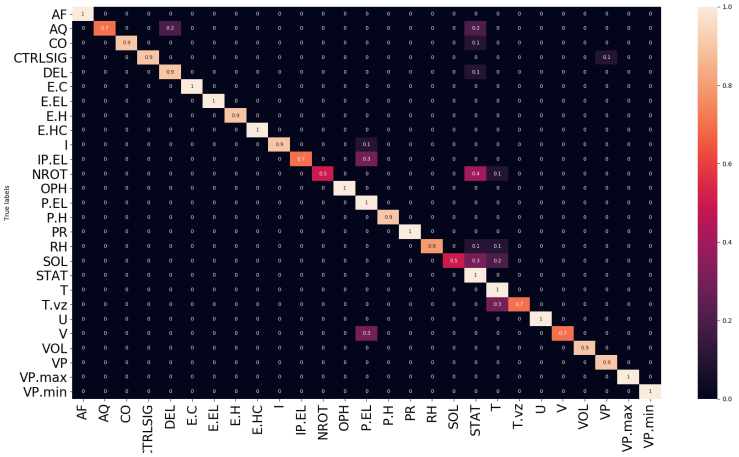
Figure 15.: Time Series Base Model results with Random Forest Classifier

The Random Forest classifier works well with most of the categories with good accuracy and F1 score. The classifier for the metadata category **system** classifies the classes in that metadata category with an average F-1 score of 0.75. Similarly, the classifier for the category **kind** predicts with an average F-1 score of 0.96. However, for all metadata categories, some classes which are highly correlated with each other get wrongly classified sometimes. Figure 16 depicts the heatmap of the classification results for the category point. It can be seen that sensors with a **point** entry of **AF** are often wrongly classified to belong to the **STAT** class. This can be easily understood, since in both cases sensors belonging to the class mainly show discrete values of 0,1. For entries of **PR** (pressure), **RH** (relative humidity) and **AQ** (air quality) the reasons for falling in the **T** (temperature) class are not so clear. While the patterns over time might look similar for this kind of sensors, the magnitude is very different. An explanation maybe is a preference towards magnitude independent features in the handcrafted features. Also, the class **T** contains most sensors which can be very different in nature resulting in a dominance of this class. The wrong classification of **DEL** and **VP.min** entries will most likely stem from their relatively

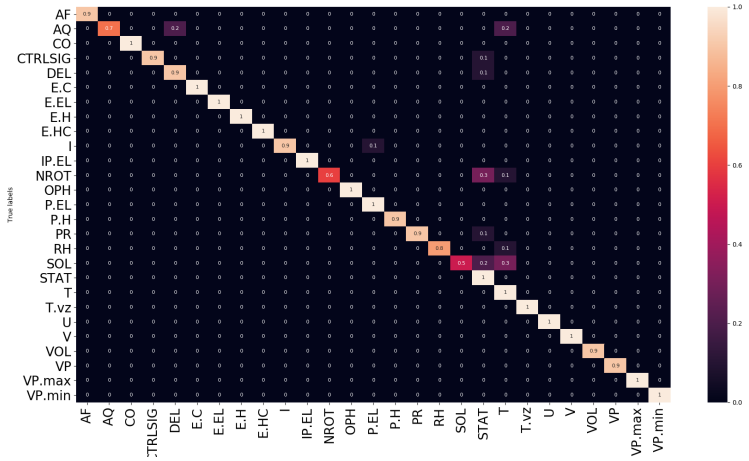
low numbers in the data set. Here the application of the method can also give hints to an improvement of the used data labelling.



(a) Heatmap from Time Series Base Model



(b) Heatmap from Description Text Base Model



(c) Heatmap from Voting Meta-Classifier Model

Figure 16.: The heatmap comparison for the predictions obtained from Random Forest Classifier in Time Series Base Model, CNN-BLSTM architecture in Description Text Base Model and the combination of these predictions using Voting as the Meta-Classification technique for the meta-data category **Point**

Training Deep Learning Model The Deep Learning Model is trained directly on the raw-time series data of the sensors. However, there lies a problem of missing values in the raw-time series data. A lot of missing values in the dataset would make it difficult for the model to distinguish between the classes due to low variance. As

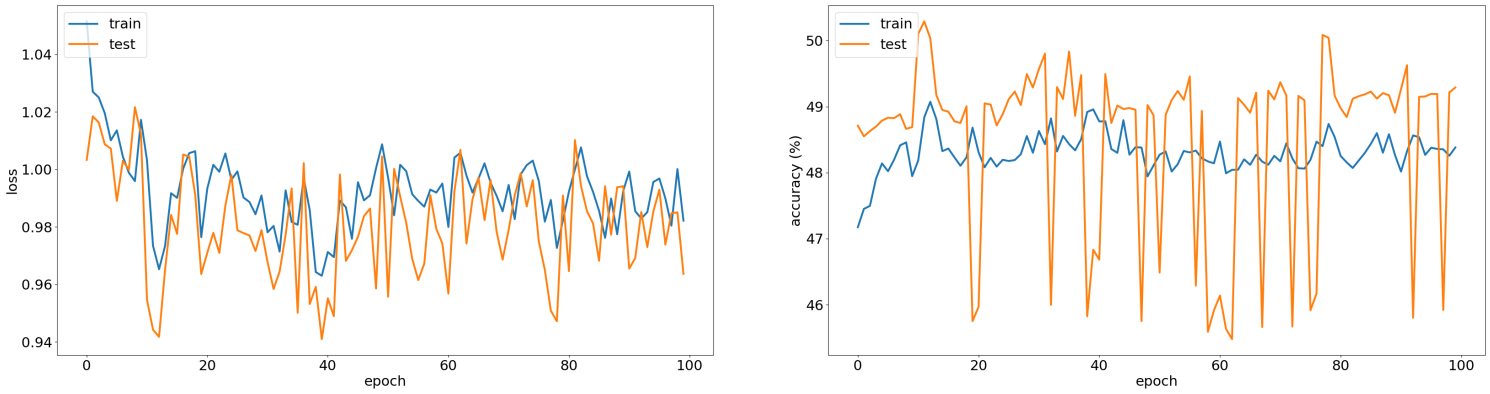
No.	Classes
1	T
2	STAT
3	P.EL
4	I
5	VOL
6	E.EL
7	E.H
8	SOL
9	DEL
10	T.OFF
11	NROT
12	T.ON
13	STAT.H

Table 14.: Classes used for Training Deep Learning Model for Time Series Base Model for the metadata category **Point**

discussed earlier, a maximum of 1440 possible values can be recorded of any sensor in a day. In order to solve the problem of missing values, we append timestamps with the recorded values in the second dimension and add masking of -999 to the missing value. This means that the missing values are represented by a number -999 in the dataset created for this experiment. For this experiment, we started with training the model on only Top 13 labels of metadata category *point* and evaluated the results on it. The data for all 13 classes in the category is balanced out to avoid biased results. For each class, the down-sampling is performed such that there are equal number of sensors with an equal number of days in every class. Table 14 shows the classes that are used for training this deep learning model. Once the data is preprocessed, an LSTM model is trained. The architecture of the model is as follows:

Input Layer → LSTM-256 → Dropout-0.4 → Dense-500 → Tanh → Dropout-0.3 → Softmax

The input layer is followed by an LSTM layer of 256 units. This layer is then followed by a Dropout layer with a rate of 0.4, to prevent overfitting of the model, and then a Fully-Connected dense layer of 500 units. The final layer is the softmax layer. The model is trained for 100 epochs. In every epoch, we use batches of size 64 data points and the training data is shuffled after every epoch. The hyper-parameters used for training the model are shown in Table 16. As depicted in Figure 17, the



(a) Loss plot

(b) Accuracy plot

Figure 17.: Time Series Base LSTM model accuracy and loss comparison, trained on raw data for the meta-data category point on Top 10 Labels **Point**

Activation Function	Tanh
Optimizer	Adam
Loss Function	Categorical Cross Entropy
Learning rate	10^{-3}
Dropout	0.3
Batch Size	64
Shuffle	Batch

Table 15.: Hyper-Parameter Settings for LSTM when used for Time Series Base Model

high inconsistency among the classes in the dataset prevents the loss and accuracy of the model from attaining convergence. The model is unable to extract patterns from the raw-time series data and hence is unable to classify the classes correctly. In the thesis of Soundarya [5], it is also reported that the amount and quality of the available data hinder an application of deep networks on the time series data. Since the focus of this thesis is the overall architecture to combine multiple data sources and a statistical model for classification of time series already exists and work better, further investigations on deep models for time series classification are left for future work.

Description Text Base Model

Description Text Base Model uses deep learning algorithms on the embedding features

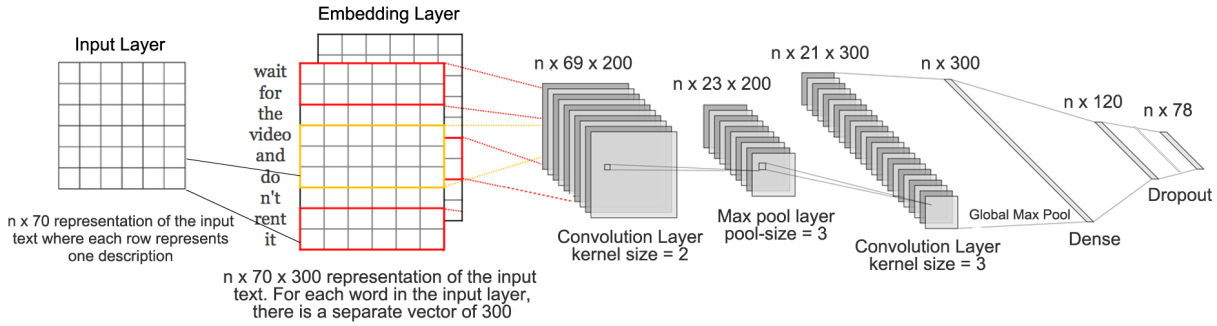
of description text as input and predicts the class label of the metadata category, on which it is trained, as the output. We conducted experiments for the description text base models with different deep learning algorithms. Convolutional Neural Networks have achieved great performances on NLP tasks. However, the architectures required to achieve these performances require many layers to capture long term dependencies and need experts to set up network parameters. Recurrent Neural Networks solve this problem because they are able to capture long-term dependencies in just a single layer but they make predictions based on the past words for a specific task. Therefore, these type of network perform really well in predicting the next word in the context. However, for sentence classification tasks, it's more efficient if the sequence of future words is also known, that is, the words from past and future are already known. Bidirectional Recurrent Neural Networks do this by running two Long Short Term Memory (LSTM), one from left to right and the second one in other direction and then concatenating both the sequences in the end.

For description text base models, initially, we carry out the experiment using Convolutional Neural Networks. Later we use an architecture proposed by [11]. This architecture uses a combination of Convolutional Neural Networks and Bi-directional LSTMs. Both the architectures are described in Figure 18.

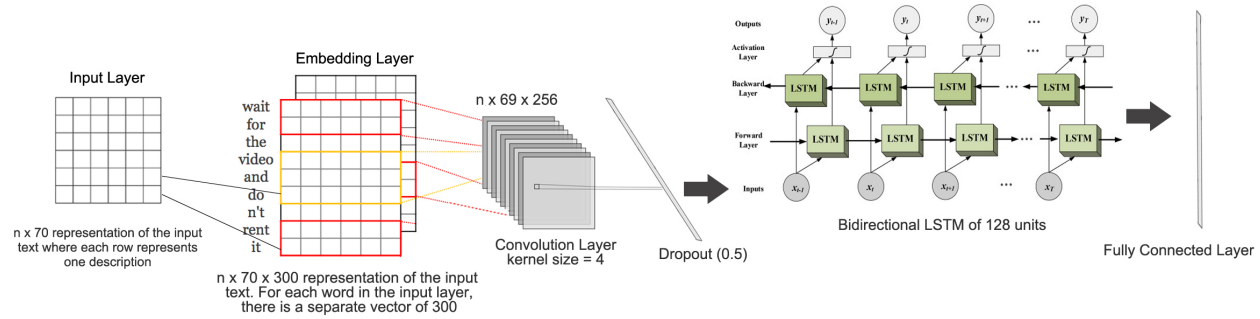
The first architecture consists of convolution and pooling layers. All the convolution and pooling layers are 1 dimensional. The embedding layer is followed by a 1-D convolution layer of filter size 2. This layer is followed by a max pooling layer which is followed by another convolution layer of filter size 3. A Global Max pooling layer is applied after this to reduce the dimensions of the output which is then followed by two Fully Connected Dense Layers, a Dropout layer and then a Softmax Output layer in the end.

In the second architecture, the embedding layer is followed by a convolution layer, a dropout layer with the rate of 0.5 and then a Bi-Directional LSTM layer. The ability of both CNN and RNN to capture long term dependencies make them ideal to use together for text classification. The output of Bi-Directional LSTM is concatenated and a further dropout of 0.5 is applied to avoid overfitting. This output is then passed on to softmax output layer. The hyperparameter settings for both the architectures are described in Table 16. The settings were obtained after performing a simple manual hyperparameter search.

The input to both the architectures of the Description Text Base Model are the word vectors described in Section 6.1.1.2 obtained from the word index. Since description texts vary in their lengths, the length of the vector is padded to a maximum length of



(a) CNN Architecture for Description Text Classification - Architecture 1



(b) Combination of Convolution Layer and Bidirectional LSTM - Architecture 2

Figure 18.: Deep Learning Architectures for Description Text Base Models

70 with zeroes appended to the vectors which are shorter than the maximum length. The input is passed on to the Embedding Layer of the architecture whose weights, as discussed earlier, are initialized by the pre-trained GloVe embedding vectors. The trainable parameter of the embedding layer is set to **True** which optimizes the weights of the embedding layer in a supervised way using the Backpropagation algorithm. The output from the embedding layer is passed on to the Deep Learning Algorithm layers of the architecture which then predict the metadata category class label.

The number of epochs for training the model of each metadata category, for both the architectures, are determined by the Early Stopping Callback method in Keras [44]. This method allows us to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on the test dataset.

Activation Function	ReLu
Optimizer	Adam
Loss Function	Categorical Cross Entropy
Learning rate	10^-3
Dropout	0.5
Batch Size	16
Shuffle	Batch

Table 16.: Hyper-Parameter Settings for both architectures of Description Text Base Model

We monitor the loss on test dataset for each metadata category. The patience level is set to 8 epochs with the mode set to "min". This means that once the loss stops improving, the model will be trained for 8 more epochs to see if there is any further improvement in the testing loss and will stop training the model if not found any. We also use a Model Checkpoint Method which monitors and saves the best model only. Therefore, all the predictions obtained are from the best trained model. In every epoch, we use batches of size 16 data points and the training data is shuffled after every epoch. Figure 19 shows the comparison of loss on training and testing dataset of both the architectures for the metadata category **Position**. The loss of testing data for Architecture 1 gradually decreases until 10^{th} epoch after which it starts to increase specifying signs of overfitting. The same happens for Architecture 2 after 15^{th} epoch. It can also be observed that the testing accuracy of both the architectures has already converged until the 10th and 15th epoch respectively. Therefore, the training is stopped and the model obtained after the specified number of epochs is the best performing model for both the architectures which is chosen to examine the test data.

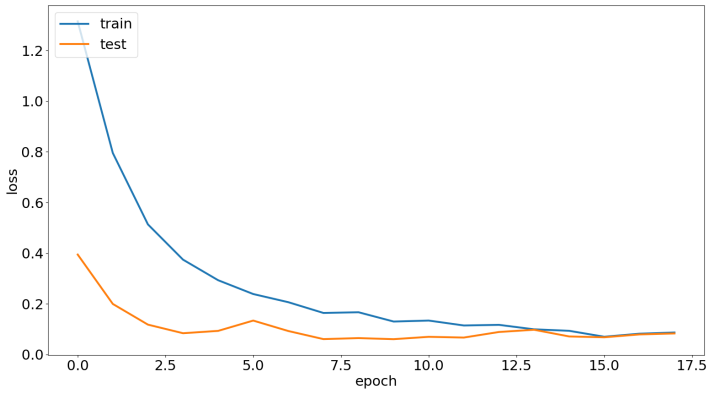
All the seven different models, for each metadata category, are used to obtain the mean micro F-1 score and mean accuracy score on all the folds of the dataset. Table 17 and Table 18 show the comparison of both the architectures on Mean Micro F-1 Score and Mean Accuracy Score of each metadata category. It can be observed that both architectures perform really well on the dataset. However, Architecture 2 works slightly better than Architecture 1 on all the metadata categories. Therefore, we use the results of this architecture for the Description Text Base Model to conduct further experiments.

Meta-data Categories	**CNN (Architecture 1)**	**CNN-BLSTM (Architecture 2)**
System	0.92	**0.94**
Subsystem1	0.96	0.96
Subsystem2	0.97	**0.98**
Medium	0.94	**0.95**
Position	0.98	0.98
Kind	0.97	0.97
Point	0.93	**0.95**

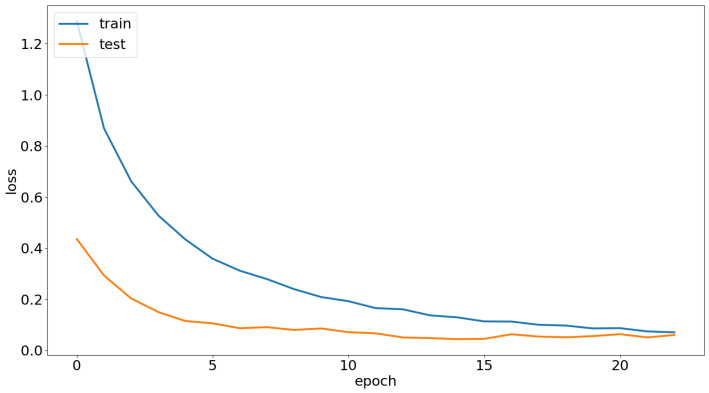
Table 17.: Mean Micro F-1 Score comparison of Architecture 1 and Architecture 2 of Description Text Base Model

Meta-data Categories	**CNN_Architecture (Architecture 1)**	**CNN-BLSTM (Architecture 2)**
System	93.16%	**94.02%**
Subsystem1	95.67%	**96.33%**
Subsystem2	98.22%	**98.48%**
Medium	95.08%	**95.28%**
Position	98.08%	**98.18%**
Kind	96.96%	**97.42%**
Point	94.15%	**94.78%**

Table 18.: Mean Accuracy comparison of Architecture 1 and Architecture 2 of Description Text Base Model



(a) Loss plot for Architecture 1



(b) Loss plot for Architecture 2

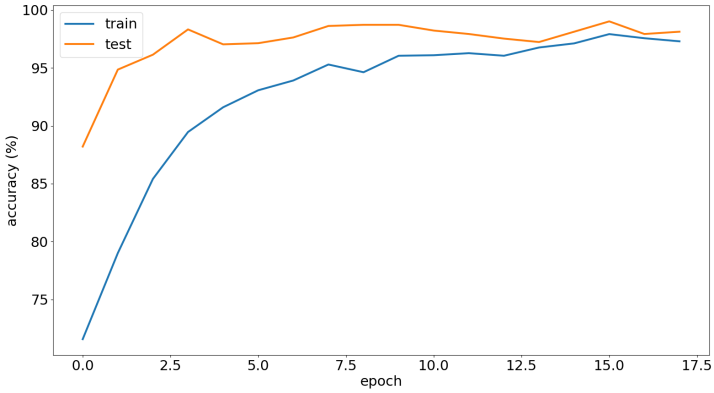
Figure 19.: Description Text Base Model Loss on Training and Testing Data for Architecture 1 and Architecture 2 respectively for the meta-data category **Position**

7.2.2.2. Comparison between the results of Time Series and Description Text Base Model

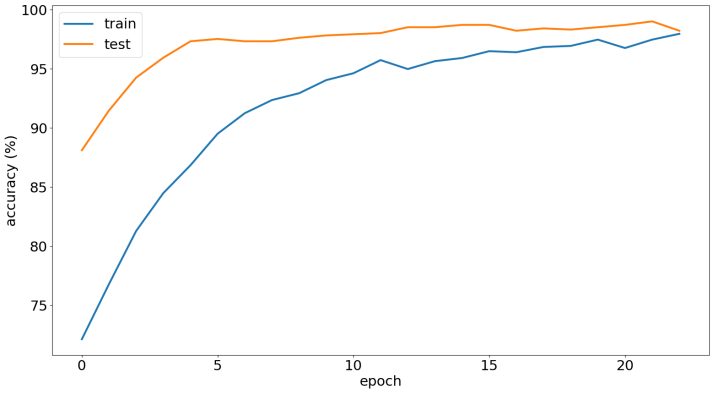
Figure 16a and Figure 16b depict comparison of predictions of metadata category **Point** obtained from two base models. It can be seen from the figure that some classes which get wrongly classified by one model, get correctly classified by the other. For example, as mentioned before, class **AF** gets wrongly classified **STAT** by the time series model 80% of the time. However, the description text base model classifies it correctly every time. Similarly, for class **V**, the description text base model classifies it incorrectly as **P.EL** 30% of the time while the Time Series Base Model classifies it correctly every time. Hence, both the models complement each other well and obtaining final predictions by utilizing the output of both models would produce more reliable and accurate results. The combination of the results is performed by Meta-Classification Models, discussed in the following section.

7.2.2.3. Meta-Classification Models

The meta-classification model, for each metadata category, uses the input of class labels it receives from Time Series and Description Text Base Model and selects the correct metadata class label as the output. As discussed in Section 6.1.2, we use Stacking and Voting as two types of meta-classification schemes for our system.



(a) Accuracy plot for Architecture 1



(b) Accuracy plot for Architecture 2

Figure 20.: Description Text Base Model Accuracy Plots on Training and Testing Data for Architecture 1 and Architecture 2 respectively for the metadata category **Position**

For Stacking, any feature based machine learning algorithm can be used as the meta-classifier. However, we perform experiments using Logistic Regression, Bagging meta-estimator with Decision Trees as base classifier and Random Forest Classifier for our system. Similarly, for voting, we use soft voting method which combines the prediction probabilities of two Base Models, averages them out and chooses the class with the highest predicted probability as the output. The parameter settings of each model in the experiments are given below:

- Logistic Regression : random_state=0, solver='lbfgs', multi_class='multinomial'
- Bagging Meta-Estimator: n_estimators=100, base estimator = DecisionTreeClassifier
- Random Forest Classifier: n_estimators=80, criterion='gini'
- Voting: method='soft'

Since the aim of meta-classification models is to produce more reliable classification, the classifier should outperform the performance of best performing individual base classifier. Table 19 shows the performance of both base classifiers, for all the metadata categories. The Description Text Base Model performs better for each metadata category and hence, is chosen as the baseline to evaluate the results of meta-classification

Meta-data Categories	Time Series Base Model	Description Text Base Model
System	0.75	**0.94**
Subsystem1	0.87	**0.96**
Subsystem2	0.93	**0.98**
Medium	0.81	**0.95**
Position	0.94	**0.98**
Kind	0.95	**0.97**
Point	0.88	**0.95**

Table 19.: Mean Micro F-1 Score comparison of Time Series Base Model and Description Text Base Model. Description Text Base Model provides better results and is chosen as baseline for Meta-Classification Model Comparison

schemes. This means, that the chosen meta-classification algorithm has to outperform the mean Micro-F1 score and mean accuracy score of Description Text Base Model, for all seven metadata categories.

Figure 21 compares the mean micro-F1 score of different algorithms tried in the meta-classification schemes. As seen in the figure, the logistic regression fails to learn the reliability of the classifiers and works poorly for all the categories. Bagging Meta Estimator and Random Forest Model, to some extent learn the pattern and perform well for the category **Subsystem2** but fail to beat the baseline score for other metadata categories. Finally, voting is observed to outperform all the other algorithms by improving upon the baseline score for every metadata category. It correctly utilizes the confidence of both the base predictors hence providing even more accurate results than the base predictors. Figure 16c shows the results of combining the predictions from the Time Series Description Model, shown in Figure 16a and Description Text Base Model, shown in Figure 16b through voting for metadata category **Point**. It can be observed that using voting as the meta-classification scheme enhances the prediction accuracies of labels which get wrongly classified, initially. For example, the class label **IP.EL** gets wrongly classified every time by the Time Series Base Model and gets correctly classified with the accuracy of 70% with the Description Text Base Model. However, when we combine the predictions of both the models using voting meta-classifier the prediction accuracy of the label increases to 100%.

Since voting is the best performing algorithm, all further experiments are performed using Voting as our meta-classification model.

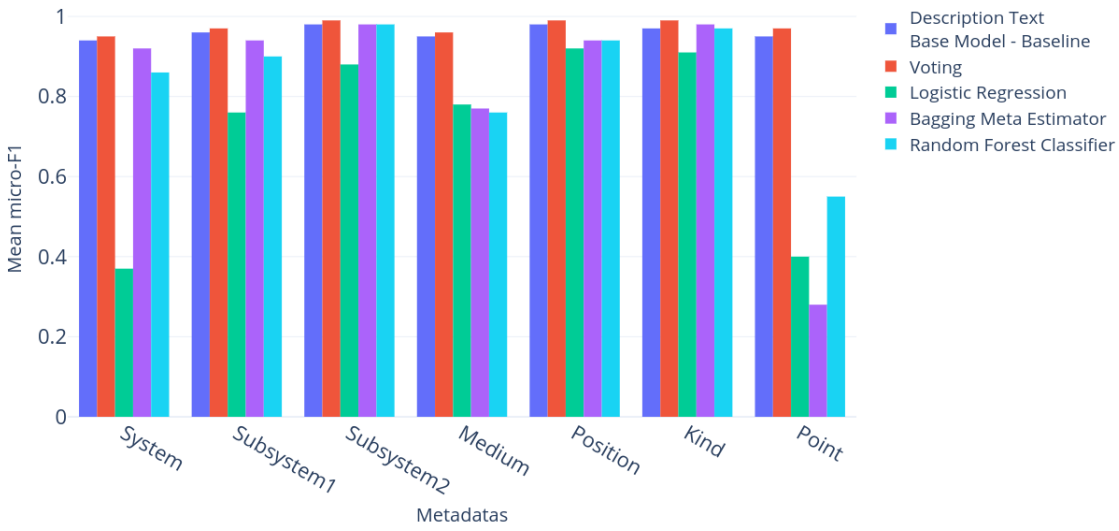


Figure 21.: The Average Micro F-1 Score comparison of different Meta-Classification Algorithms. Random Forest Classifier, Logistic Regression and Bagging Meta-Estimator are used as Stacked Algorithms. Voting Method outperforms every other method.

7.2.2.4. Top Level Model

As discussed in Section 6.1.3, the point name label of the sensor can be obtained by either concatenating the class labels predicted by the meta-classification model of each metadata category, or by training another machine learning model on these predictions. We tried to evaluate the results of both the experiments. Table 20 shows the mean micro F-1 score comparison of both these methods for 218 labels. We tried Random Forest Classification algorithm for the Top Level Model. It can be seen that training a Top Level Model on the outputs of meta-classification models produces slightly better results than just concatenating the outputs. The classifier is able to observe the patterns of the correlated features and hence predicts the point name label with good accuracy. However, concatenating metadata category predicted outputs is a more practical approach since there is a possibility that some new combinations of concatenating metadata categories are created to form new point name labels and since they were not in the training set, it would not be able to predict them correctly.

The thesis [5] performed the classification initially with only 10 point name labels

	Metadata Categories concatenated	**Top Level Model**
F1-Score	0.88	0.88
Accuracy	87.64%	**88.37%**

Table 20.: The mean Micro F-1 score and Accuracy comparison of two methodologies for obtaining point name label prediction for 218 labels after obtaining the meta-classifier predictions through voting

and later with 81 labels. However, we perform our experiments using 218 labels. To create even comparison, we compare the results obtained from our architecture for the 10 labels used in [5] with results of [5] as the baseline. Since training a Top Level Model on the meta-classifier predictions produces slightly better results for our architecture than combining the predictions, we use this approach to perform the comparison with the baseline score. As shown in Figure 22, the proposed architecture produces better results than the baseline on all except 3 labels.

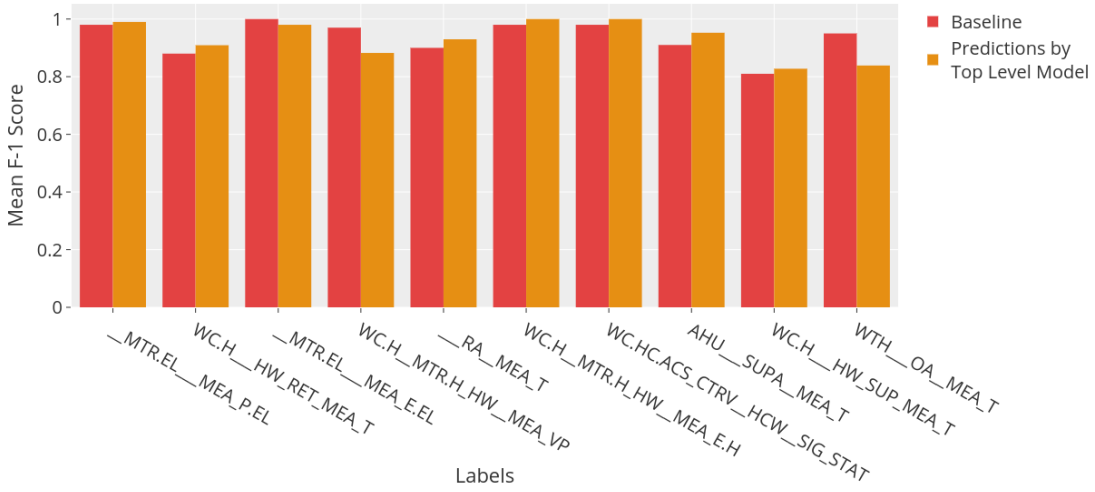


Figure 22.: The Average Micro F-1 Score comparison of 10 labels obtained by two proposed methods of point name label prediction in Section 6.1.3. The baseline score is obtained from previously conducted thesis in Frauhofer ISE [5]

The architecture is unable to beat the baseline score for labels **_MTR.EL__MEA_E.EL**,

Architecture Layer	Algorithms
Time Series Base Model	Random Forest Classifier
Description Text Base Model	CNN with BLSTM
Meta-Classification Model	Voting (Soft)
Top Level Model	Random Forest Classifier

Table 21.: The final chosen algorithms for each layer of models in the architecture

WC.H_MTR.H_HW___MEA_VP and **WTH____OA___MEA_T**. The recorded sensor labels are highly correlated with each other. Since the model in the previous thesis was trained on only 10 labels, it had limited amount of labels to classify compared to our model, and hence for some labels, it was able to classify with better accuracy than our architecture. However, the overall accuracy score depicted in Figure 23 shows that even with an increased number of labels, our proposed architecture outperforms the baseline score with an accuracy of 88%.

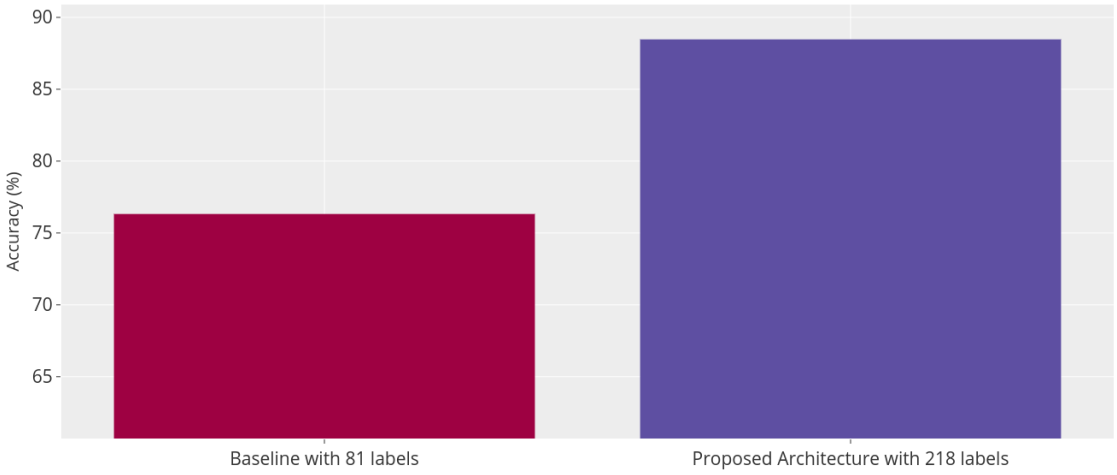


Figure 23.: The accuracy comparison of the proposed architecture with the baseline score

The final list of algorithms and technique that we finalize at each level of the architecture, is depicted in Table 21.

7.3. Inter-Building Cross Validation

The experiments that we discussed in the previous section used the data from a set of selective buildings and the performance of the architecture was also evaluated on the data of the same buildings. In reality, the geographical location of a building as well as the usage characteristics and the detailed control strategies (like time schedules) play a vital role in the measured value of the sensor of that building. Besides of different system combinations and controls, the data logging mechanisms and quality may be different. Each building shows some unique characteristics and hence the performance of the architecture might be greatly affected with the inclusion of an unseen building in the testing set because the behaviour and type of sensor data might be different compared to the data on which the model is already trained. In order to evaluate the performance of the architecture on the unseen data, we create a new methodology to train our architecture. There are 13 buildings in the dataset. We create 13 different sets of sensors, each set corresponding to one building and use one set for testing while using all other 12 sets for training the architecture. The process is repeated until every set is used for testing. We call this process Inter-Building Cross Validation. Figure 24 shows the accuracy score of final point name label prediction, obtained from our architecture when tested on different buildings.

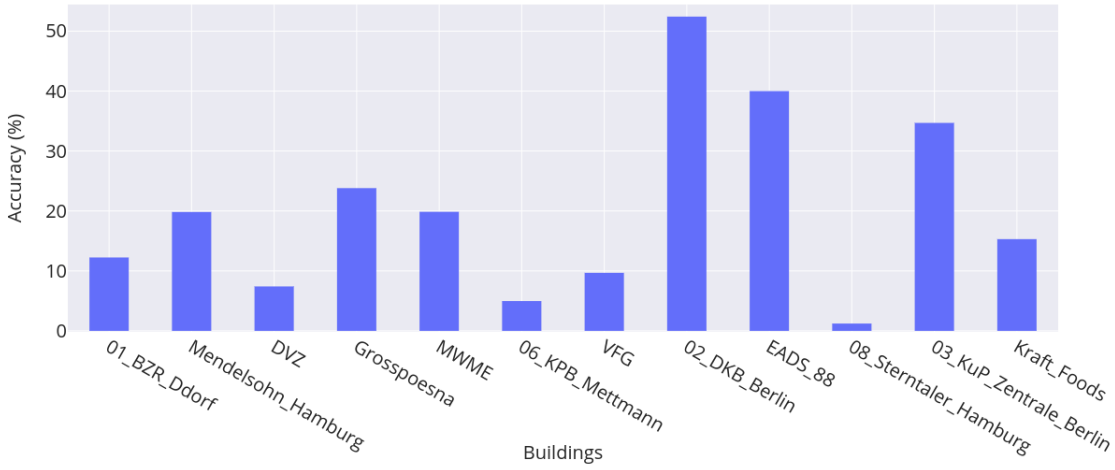


Figure 24.: The accuracy score obtained from the architecture when tested on different buildings

As depicted in the figure, the model performs poorly on the unseen data and is unable to predict point name label correctly in most of the buildings. The highest accuracy obtained is just 52%, obtained on building 02_DKB_Berlin. We have already seen in Section 7.2.2 that the architecture works well with k-fold cross validation experiment. Therefore, if improved quality data is provided to the model, it will produce good results.

7.4. Simulation of User Feedback

In order to solve the problem of the trained model encountering unseen data, discussed in the previous section, we propose a similar approach proposed in the work [5] to improve the prediction results for the architecture. The approach assumes that the knowledge about the ground truth would be provided by a human user in the real world application. In general, the architecture predicts a point name label with a certain probability. We call this probability the confidence of the model. Confidence value describes how much the algorithm is confident while predicting a particular class. Therefore, if a predicted class has high probability then the model has higher confidence in the prediction of that class. The idea is that once the predictions for point name labels have been obtained, the sensors would be sorted, based on their confidence score. The user can then select any specified threshold, for example, top 20 sensors with lowest confidence scores, and then screen the list of sensors at least once to check for the correct classification by the method, by comparing the label with any human comprehensible description. In case a sensor is incorrectly labelled, the user would interact with a hypothetical user interface to correct the metadata category of the sensors. These corrected sensors could then be placed into the training set and the model would then be re-trained. Although, in this procedure a user still has to perform manual work for labeling the data, the amount of work should be decreased since only parts of the sensors would have to be labeled to improve the quality of the classification. Also the system would provide the user with a list of sensors which should impact the results most. This approach differs from the existing approach as it utilizes the confidence of the model to sort the sensors whereas existing approach sorted sensors based on their performance.

To evaluate the performance of the proposed approach, metrics like accuracy, precision or recall do not reflect the problem domain well. The point name label is a concatenation of seven different metadata categories. These evaluation metrics do not incorporate the significance of each specific category, but rather consider all the

categories of the point name label as a single class. Therefore, even if one category of the point name label is incorrectly classified, the whole label will be considered incorrect. Therefore, in order to measure correctly, the number of metadata categories between the predictions and the actual labels, that have been correctly classified, we introduce a new measure called "Usefulness". As the name implies, this provides a measure of the usefulness of a prediction to the user, as classifications that are only slightly off will still give a user useful insight on the sensor and reduce the number of changes needed to get to the correct label. The measure can be defined as the accuracy of individual predicted metadata category class labelled [5].

$$\text{Usefulness Measure} = 1 - (\text{ Number of categories changed / Total Number of Categories}) \tag{7.1}$$

The measure has the highest value of 1.0 when no category needs to be changed and the lowest value of 0.0 when every category needs to be changed. Therefore, the higher value corresponds to more accurate predictions. Consider the sensor with original point name label AHU_____SEV_STAT. Let us consider that the classifier predicts it as RAD_____SEV_T. It can be seen that the first category system 'RAD' and the last category point 'T' of the label has been incorrectly predicted. Therefore 2 categories need to be changed in the label to make it match with the original label and hence the usefulness score would then be calculated as (1 - 2/7 = 0.72)

7.4.1. Feedback Experiment

The feedback simulation experiment involves the following steps:

1. Test the sensor data of the building on the model, already trained on the data of other buildings and obtain predictions of the point name label
2. Compute the usefulness score of all the predictions
3. Sort the list of predictions based on the confidence score of the predicted class by the model, in the ascending order
4. Select top 20 sensors from the sorted list, verify the correctness of the prediction and mark correct label for the sensors
5. Transfer the correctly labelled sensor data back to the training set while discarding it from the test set

6. Retrain the model and test it on the remaining sensors of the building in the test set

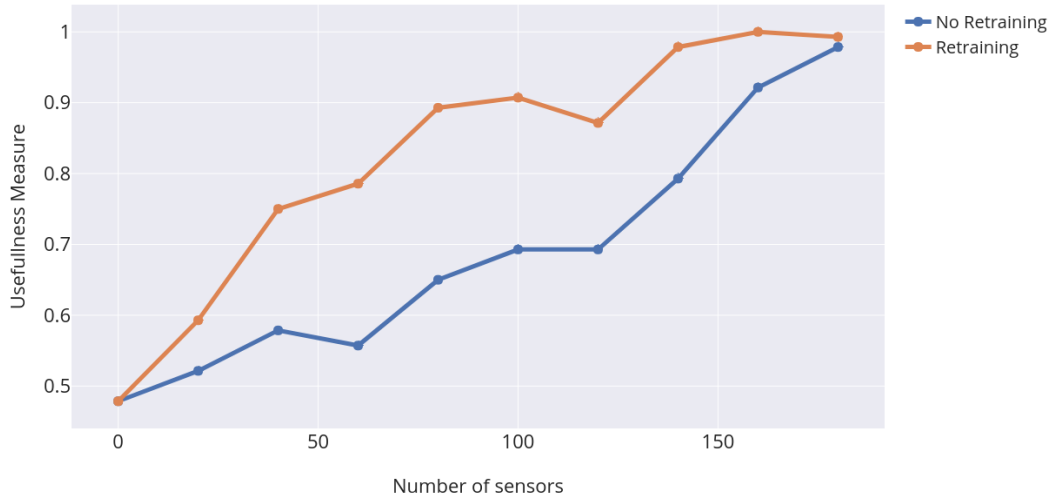
7. Perform step 2 to step 6 in an iterative manner until all the sensors from the building have been added back to the training set

For example, let us consider that BuildingX was chosen for the experiment which contained data from 60 sensors in it. Step 1 will be performed to obtain the predictions of the point name label from the model that would be followed by the computation of the usefulness score and obtaining the model's confidence on the predicted labels. Step 3 would be performed where the list would be sorted and the top 20 sensors with the lowest confidence score would be selected. The user would then mark correct labels for the wrongly classified sensors and put them back into the training set to retrain the model. In order to compare the impact of retraining the model, we create a baseline score where we perform steps 1 to 4, and rather than retraining the model, we discard the sensors from the test set and obtain the predictions on the remaining sensors. This process is also performed in an iterative manner.

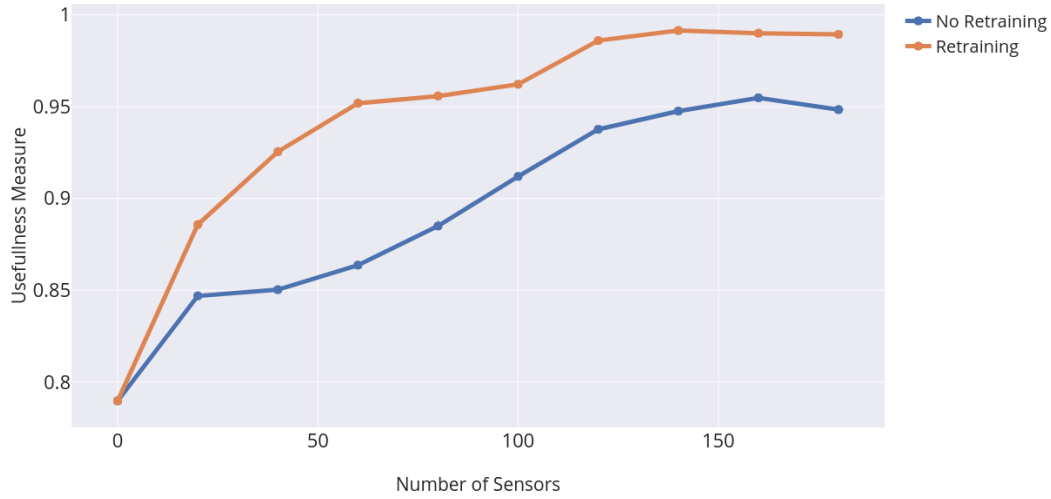
Figure 25a shows the trend of usefulness on the top 20 sensors that are selected every time to put back into the training set for building 03_KuP_Zentrale_Berlin. It can be observed that the average usefulness score on these selected sensors improves faster and to a higher score when the model is re-trained compared to the case where no re-training takes place and the sensors are discarded from the test set. This means that the addition of sensors into the training set improves the overall precision of the classifier and even the predictions made with very low confidence are very accurate. Similarly, Figure 25b shows the impact of retraining the classifier on the usefulness of all the predicted labels excluding the 20 sensors chosen for retraining the model. Since the number of correctly classified sensors is comparatively high in the retraining phase, the average usefulness score of retrained predictions is much higher than the predictions obtained from the non-retrained model. This is also because the measured values of sensors are highly co-related withing a building, therefore, one sensor of a particular building gets highly influenced by the measured value of another sensor of the same class and building. For example, the temperature sensor, when added back to the training set, will have an influence on the other temperature sensors of the same building and therefore repeated retraining of the model on new sensors helps in improving the prediction results. It can also be observed from the graph that removing low sensors predicted with low confidence scores result in an steep rise in the average usefulness of the remaining sensors, thus, justifying the selection of the

sensors chosen to be discarded or added back.

The proposed process serves as a proof of concept which can be implemented to further improve the prediction results.



(a) Average Usefulness on selected 20 labels of building at every iteration



(b) Average Usefulness on all the sensor labels except the 20 selected for retraining the model

Figure 25.: Comparison of usefulness on the selected top 20 labels based on the confidence score versus the rest of the labels for building 03_KuP_Zentrale_Berlin

8. Conclusion and Future Work

8.1. Conclusion

In this work, we developed a system which automates the labelling of various time series and description text data of various sensors, obtained from different kind of buildings like police department, elementary school, housing industries, offices of the district government, etc., in a supervised learning manner. The number of sensors that were used from these buildings were over 3300 in total and included the measured values from distinct air handling units, outdoor air temperature sensor, room air temperature, energy meters, pressure sensors, supply air temperature etc. The labelling convention used in the thesis is called data point naming convention and each sensor's label is called point name label which marks its type and origin. The point name label is a concatenation of seven different meta-data category entries of the sensor, separated by an underscore '_'. The system, that is proposed in this thesis, uses 20 handcrafted-features from the raw time series data of the sensors [5], to provide it with a meaningful representation. Pre-trained Word Embedding vectors are used as features for the description text of the sensors. The aim of the system is to first predict respective meta-data category class labels of the sensor and then use these labels to predict the final point name label. Therefore, a layered architecture is used which takes, both Time Series Data and Description Text of sensors, as input and then predicts a class label for all seven meta-data categories. The output labels are then used to train a Top Level model which predicts the final point name label of the sensor.

For each layer of the architecture, we tested out different machine learning and deep learning models and selected the best performing algorithms. The proposed architecture outperformed the results of previously conducted thesis at Fraunhofer ISE [5]. Our system successfully classifies 218 point name labels with an accuracy of 87% beating the baseline score of [5] which only classified 81 labels with an accuracy of 76%.

In the end, we tested our system towards a more practical simulated real world

scenario where we evaluated the performance of our model on the data of new buildings using a new performance measure, which we called the 'Usefulness Measure'. The results showed that retraining a model in an iterative fashion makes the model more biased but better adapted to the building and thus assist in classifying most of the sensor data of a new building to their corresponding labels.

8.2. Future Work

As we have seen already, the time series data show high inconsistency. The need for restructuring of the meta data system and thus the unified point name labels has to be done by domain experts and is out of scope for this thesis. Nonetheless, the work in the thesis can help the design choices for experts in such an attempt. Also the experiments exposed possibles flaws in the labeling. In future, this thing can be improved by further reviewing the data and dividing already existing classes to further sub-classes. This can also help in improving the performance of deep learning model when applied to raw time series data. Furthermore, the system, proposed in this thesis, can also be tested on the data of more buildings to further evaluate the performance of it which would again raise the need for a review of these buildings data. Also, the system right now uses a combination of deep and machine learning models. Deep Learning models work fast with GPU and training machine learning models with a large amount of data require high RAM. These hardware constraints do not fully automate the operations of the system as both types of models have to be trained on different machines, satisfying the requirement. This constraint can be resolved if the system is operated on machines having GPU and large capacity of RAM.

Additionally, to the task of classifying the sensors, the final use of the sensors data also requires information on the systems or zones a sensor belongs to. This information is always specific to the building at hand. The first three entries of the metadata example shown in Table 1 belong to such a type of information. The name or id of a zone or system could not be inferred from the raw time series data. Yet sensors belonging to the same system might show similar behaviour on a certain time scale. For instance, a system might start its operation at certain times of day or on certain environment events (such as increasing room temperatures with sunrise). All sensors belonging to this system will in some way show this change of operation in their readings. On the other hand systems of the same type might start their operation at the same time and thus their sensors will show different time dependent

behaviour. This effect on the time series data can be one source for the grouping of sensors. An unsupervised machine learning method like 'clustering' might be able to detect such grouping. Another source would be identifiers used in the description text data sources. These would have to be separated from the text fragments that are of general nature and will for instance just denote the type of a related system and not its unique id.

Finally, the system needs to be integrated with a user interface which can also help in aiding the technical personnel with the feedback process.

Parts of the developments performed in this work have been done in the context of the EFRE project **SmartBadenMonitor** [48]. To get comparable results with previous work, up to now no measurement data from that project could be used. The inclusion of new data sets will also respect data of SmartBadenMonitor.

Bibliography

[1] "A comprehensive guide to ensemble learning (with python codes)." https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/. Accessed: 2019-06-24.

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] "Cs231n convolutional neural networks for visual recognition." http://cs231n.github.io/convolutional-networks/. Accessed: 2019-06-24.

[4] "Understanding lstm networks." https://colah.github.io/posts/2015-08-Understanding-LSTMs/?fbclid=IwAR0K5X4LNueP6stb_0Q82N0uuZfooq6jm2H7tC_MTAxrgjbv1qsPjdYnEww. Accessed: 2019-06-24.

[5] S. Palanisamy, "Automated extraction of data point names," Master's thesis, Chair of Machine Learning Dept.of Computer Science, Faculty of Engineering, Albert-Ludwigs-Universität Freiburg, 2017.

[6] L. Lu, H.-J. Zhang, and S. Z. Li, "Content-based audio classification and segmentation by using support vector machines," *Multimedia systems*, vol. 8, no. 6, pp. 482–492, 2003.

[7] W. Chaovalitwongse, O. A. Prokopyev, and P. Pardalos, "Electroencephalogram (eeg) time series classification: Applications in epilepsy," *Annals OR*, vol. 148, pp. 227–250, 11 2006.

[8] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, pp. 917–963, Jul 2019.

[9] F. Ordóñez and D. Roggen, "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, p. 115, 2016.

[10] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, pp. 1188–1196, 2014.

[11] A. Hassan and A. Mahmood, "Efficient deep learning model for text classification based on recurrent and convolutional layers," 12 2017.

[12] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, "Very deep convolutional networks for text classification," *arXiv preprint arXiv:1606.01781*, 2016.

[13] J. Gao, J. Ploennigs, and M. Berges, "A data-driven meta-data inference framework for building automation systems," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pp. 23–32, ACM, 2015.

[14] D. Hong, J. Ortiz, A. Bhattacharya, and K. Whitehouse, "Sensor-type classification in buildings," *arXiv preprint arXiv:1509.00498*, 2015.

[15] A. A. Bhattacharya, D. Hong, D. Culler, J. Ortiz, K. Whitehouse, and E. Wu, "Automated metadata construction to support portable building applications," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pp. 3–12, ACM, 2015.

[16] S. Das, A. Dey, A. Pal, and N. Roy, "Applications of artificial intelligence in machine learning: review and prospect," *International Journal of Computer Applications*, vol. 115, no. 9, 2015.

[17] T. Paek, M. Gamon, S. Counts, D. M. Chickering, and A. Dhesi, "Predicting the importance of newsfeed posts and social network friends," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[18] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[19] P.-N. Tan, M. Steinbach, and V. Kumar, "Classification: basic concepts, decision trees, and model evaluation," *Introduction to data mining*, vol. 1, pp. 145–205, 2006.

[20] A. K. Seewald, "Meta-learning for stacked classification," *audiology*, vol. 24, no. 226, p. 69, 2002.

[21] L. Arnold, S. Rebecchi, S. Chevallier, and H. Paugam-Moisy, "An Introduction to Deep Learning," in *European Symposium on Artificial Neural Networks (ESANN)*,

Proceedings of the European Symposium on Artificial Neural Networks (ESANN), (Bruges, Belgium), Apr. 2011.

[22] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[23] Y. Bengio, "Deep learning of representations for unsupervised and transfer learning," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 17–36, 2012.

[24] O. Abdel-Hamid, L. Deng, and D. Yu, "Exploring convolutional neural network structures and optimization techniques for speech recognition.," in *Interspeech*, vol. 2013, pp. 1173–5, 2013.

[25] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, pp. 574–591, 1959.

[26] H. Jaeger, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, vol. 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[27] Y. Bengio, P. Simard, P. Frasconi, *et al.*, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[28] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[29] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in neural information processing systems*, pp. 3320–3328, 2014.

[30] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[31] C. N. D. J. Prof. Livio Mazzarella, Michele Liziero, "Description of european prototype tool for evaluation of building performance," 2009.

[32] E. R.-V. Lojini Logesparan, Alexander J. Casson, "Optimal features for online seizure detection," *Medical & biological engineering & computing*, vol. 50, pp. 659–669, 2012.

[33] P. P. Peter D. Turney, "From frequency to meaning: Vector space models of semantics," *Journal of Artificial Intelligence Research*, p. 141–188, 2010.

[34] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 3111–3119, Curran Associates, Inc., 2013.

[35] Y. Li and T. Yang, *Word Embedding for Understanding Natural Language: A Survey*, vol. 26. 05 2017.

[36] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[37] A. Mandelbaum and A. Shalev, "Word embeddings and their use in sentence classification tasks," *arXiv preprint arXiv:1610.08229*, 2016.

[38] "Glove: Global vectors for word representation." `https://nlp.stanford.edu/projects/glove/`. Accessed: 2019-05-31.

[39] "Common crawl." `http://commoncrawl.org/`. Accessed: 2019-05-31.

[40] B. H. Pavel Brazdil, João Gama, "Characterizing the applicability of classification algorithms using meta-level learning," *Proceedings of the 7th European Conference on Machine Learning (ECML-94)*, pp. 83–102, 1994.

[41] Y. E. Eitan Menahem, Lior Rokach, "Troika – an improved stacking schema for classification tasks," *Information Sciences, Department of Information Systems Engineering, Ben-Gurion University and Deutsche Telekom Laboratories at Ben-Gurion University*, p. 3, 2009.

[42] Python Core Team (2015), *Python: A dynamic, open source programming language*. RPython Software Foundation, 2015.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,

D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[44] F. Chollet, "Keras: Deep learning for humans." `https://github.com/keras-team/keras`, 27 March 2015.

[45] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[46] T. Oliphant, "NumPy: A guide to NumPy." USA: Trelgol Publishing, 2006–.

[47] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and J. Millman, eds.), pp. 51 – 56, 2010.

[48] "Smartbadenmonitor." `https://smartbadenmonitor.de/`. Accessed: 2019-05-31.

A. Abbreviations of Classes

Tables below describe the abbreviations of some class labels for each meta-data category which make up the final point name label.

No.	Category	Classes	Abbreviation
1	System	DH	District heat
		DC	District cold
		ESUP	Energy supply
		WSUP	Water supply
		WTH	Weather Station
		WC.H	heating circuit (water circuit for heating)
		WC.C	cooling circuit (water circuit for cooling)
		WC.HC	heating / cooling circuit (water circuit for heating / cooling)
		AHU	Air handling unit
2	Subsystem_1	MTR.H	heat meter
		MTR.C	cold meter
		MTR.EL	electricity meter
		MTR.W	water meter
		GLOBSENS	Pyranometer
		PU	Pump
		FAN	Fan
		HC	heating coil
		CC	cooling coil
		PREHC	pre-heating coil
		PREHCC	pre-heating/cooling coil

Table 22.: Table Describing abbreviations of some classes of System and Subsystem_1

No.	Category	Classes	Description
3	Subsystem_2	PU	Pump
		CTRV	Control valve
4	Medium	HW	hot water
		CHW	chilled water
		HCW	hot / chilled waterr
		DCW	domestic cold water
		FUEL	any kind of fuel (gas, oil, wood, etc.)
		OA	outdoor air
		RA	room air
		SUPA	supply air
		EXHA	exhaust air
5	Position	SUP.(PRIM, SEC)	supply (on primary or secondary side of system)
		RET.(PRIM, SEC)	return (on primary or secondary side of system)
		PRIM, SEC	primary or secondary side of system
6	Kind	MEA	measured value
		SEV	set value
		SIG	signal (feedback from component)
7	Point	E	energy
		E.H	heating energy
		E.C	cooling energy
		E.EL	electric energy
		VOL	Volume
		T	temperature
		RH	relative humidity
		SOL	solar radiation
		CTRLSIG	control signal
		STAT	status (1/0)

Table 23.: Table Describing abbreviations of some classes of Subsystem_2, Medium, Position, Kind and Point

B. Classes in the test set

No.	Meta-data Category	Class Labels	Number of Sensors
1	System	”	362
		AHU	239
		WC.H	127
		RAD	53
		WIN	37
		WC.H.RAD	30
		WC.HC.ACS	23
		BL	13
		EGEN.C	13
		WTH	10
		EGEN.H	8
		HP	8
		WC.H.AHU.HC	7
		BHX	7
		OSP	7
		LIFT	7
		WC.H.AHU	6
		WSUP	5
		WC.H.AHU.PREHC	4
		WC.C.Feeder	4
		EGEN	4
		EH	4
		ESUP	3
		SPRS	3
		SUBDIS.TG	3
		SUBDIS.U	3
		WC.C.RCA.C	3
		HX.dc	2
		WIN.AHU	2
		EGEN.C.ADV	2
		DHWP	2
		HSTO	2
		WSUP.CTO	1
		WIN.O	1
		TH	1
		CTO	1
		WIN.W	1
		WC.H.Feeder	1

Table 24.: Classes used for meta-data category System

No.	Meta-data Category	Class Labels	Number of Sensors
2	Subsystem_1	”	735
		CTRV	52
		VFC	40
		PU	27
		FAN	25
		HC	21
		SHD	16
		PRE	13
		PREHC	12
		HCC	11
		CC	9
		SHV	9
		BOI	8
		CCH.AHU	7
		CCH	5
		MTR.W	4
		CTO	3
		PV	3
		FROS	2
		PREV	2
		GLOBSENS	2
		CHOV	1
		HRC	1
		HUM	1
3	Subsystem_2	”	672
		MTR.EL	216
		MTR.H	84
		PU	13
		CTRV	7
		MTR.HC	5
		MTR.C	5
		MTR.W	2
		FC	1
		EVAP.HT.el	1
		CTRV.H	1
		CTRV.C	1
		CHOV	1

Table 25.: Classes used for meta-data category Subsystem_1 and Subsystem_2

No.	Meta-data Category	Class Labels	Number of Sensors
4	Medium	”	330
		HW	266
		RA	201
		SUPA	66
		EXHA	55
		HCW	26
		OA	19
		CHW	19
		SUPA.EXHA	11
		COW	6
		DCW	5
		EXHAO	4
		RCA	1
5	Position	”	865
		RET	42
		SUP	37
		RET.SEC	20
		SEC	19
		SUP.SEC	18
		PRIM	3
		SUP.PRIM	2
		TOP	1
		RET.PRIM	1
		BOT	1
6	Kind	MEA	626
		SIG	234
		SEV	114
		SEV.Nacht	14
		SEV.Tag	12
		CALC	6
		SEV.MAN	3

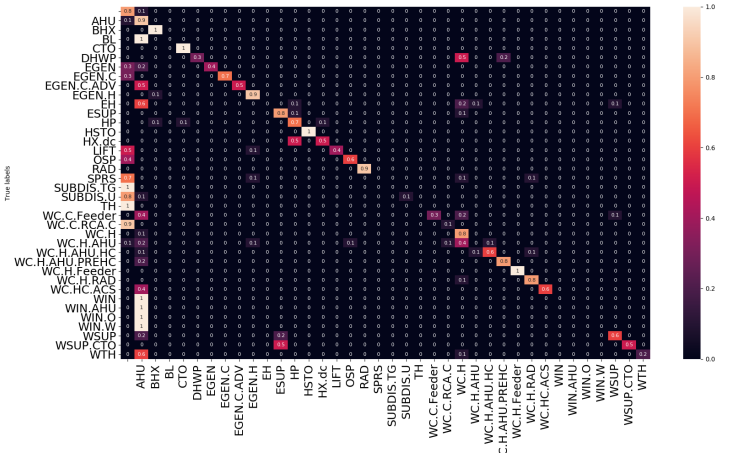
Table 26.: Classes used for meta-data category Medium, Position and Kind

No.	Meta-data Category	Class Labels	Number of Sensors
7	Point	T	370
		STAT	214
		P.EL	72
		U	59
		I	58
		VP	30
		E.EL	27
		VOL	24
		CTRLSIG	23
		RH	23
		E.H	18
		P.H	16
		DEL	13
		AF	11
		CO	10
		VP.max	8
		VP.min	7
		NROT	5
		PR	5
		E.C	5
		OPH	3
		SOL	2
		AQ	2
		IP.EL	1
		E.HC	1
		T.vz	1
		V	1

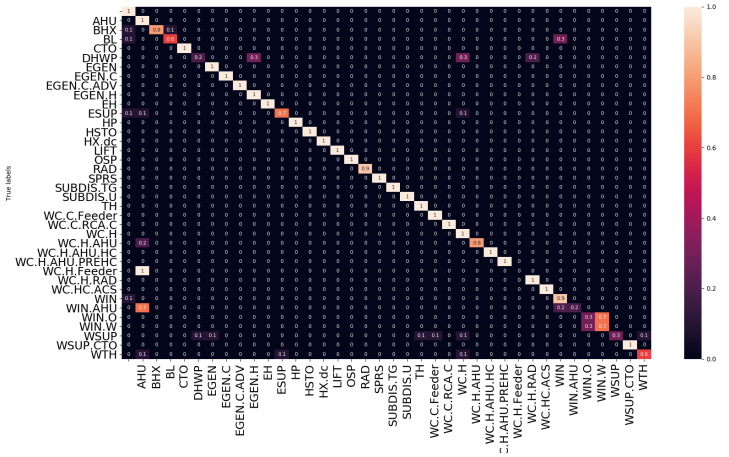
Table 27.: Classes used for meta-data category Point

All the tables above describe the classes used in the test set for each meta-data category to perform K-Fold Cross Validation Experiment. The training set included instances of every class in the dataset.

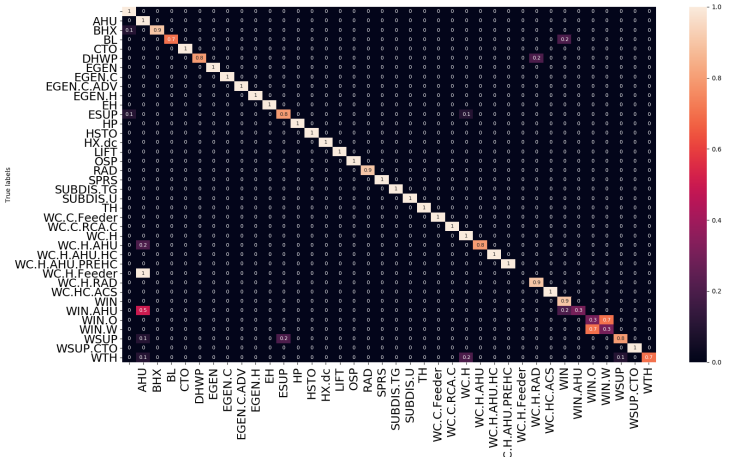
C. Comparison of Heatmaps



(a) Heatmap from Time Series Base Model



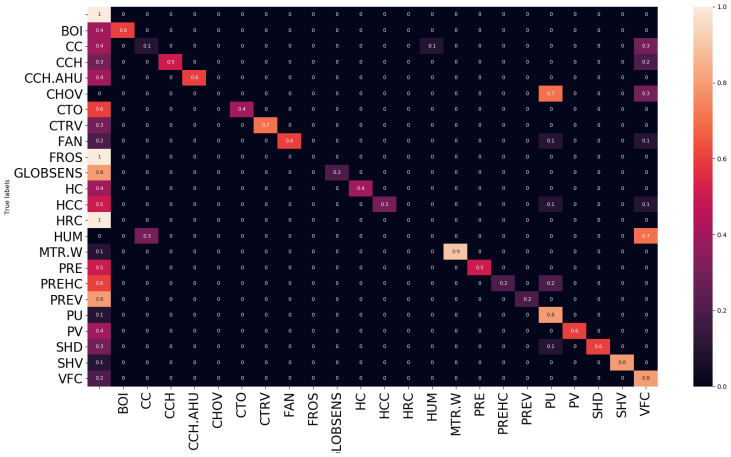
(b) Heatmap from Description Text Base Model



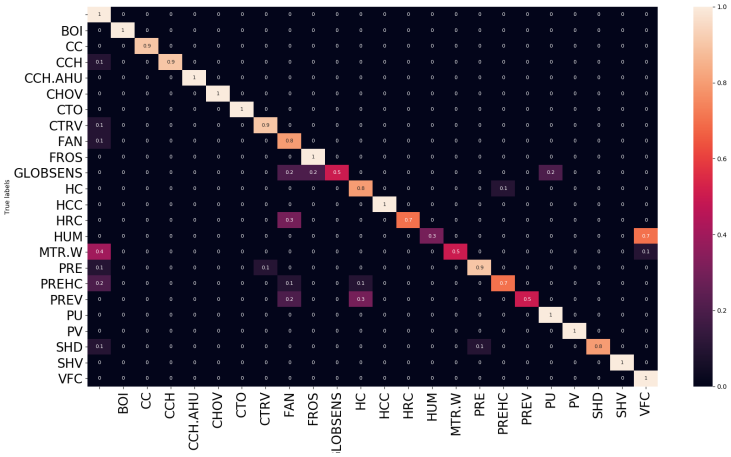
(c) Heatmap from Voting Meta-Classifier Model

Figure 26.: Comparison for meta-data category System

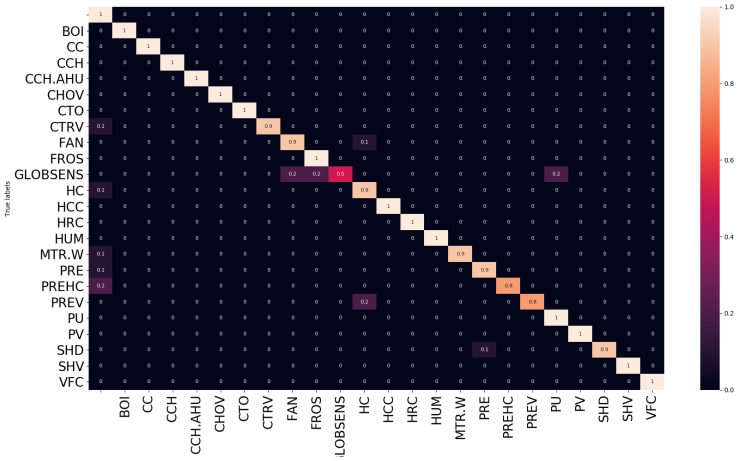
The heatmap comparison for the predictions obtained from Random Forest Classifier in Time Series Base Model, CNN-BLSTM architecture in Description Text Base Model



(a) Heatmap from Time Series Base Model



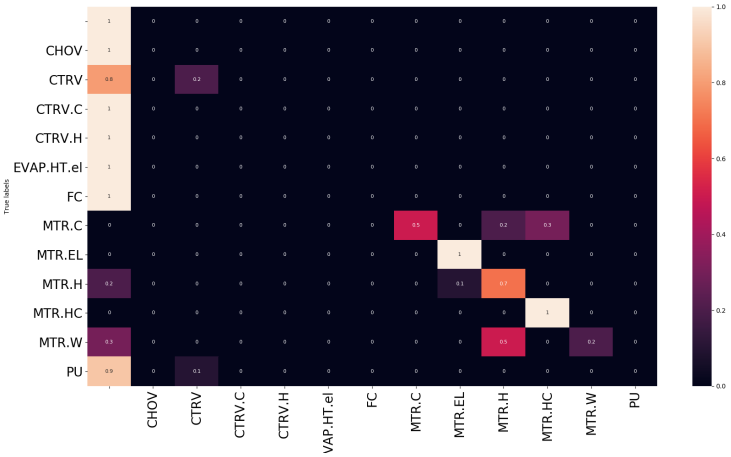
(b) Heatmap from Description Text Base Model



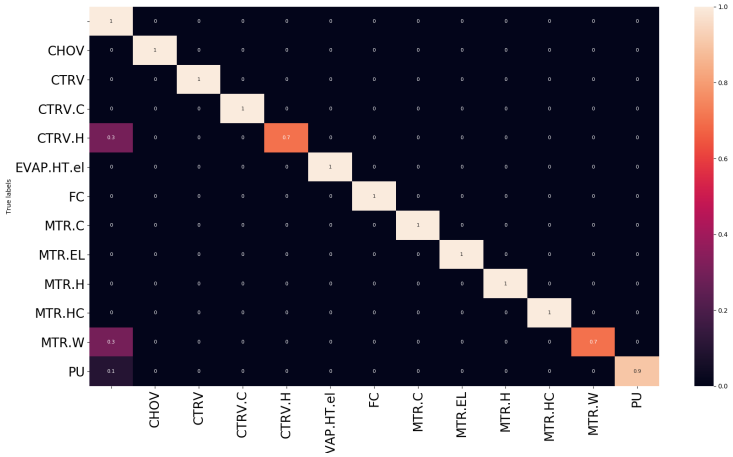
(c) Heatmap from Voting Meta-Classifier Model

Figure 27.: Comparison for meta-data category Subsystem_1

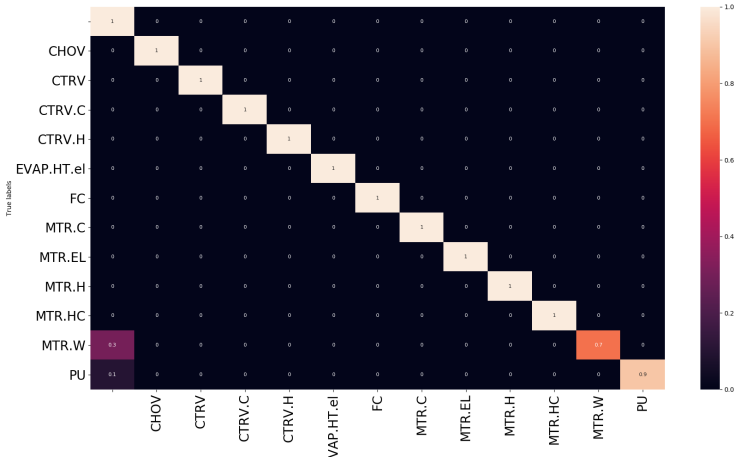
and the combination of these predictions using Voting as the Meta-Classification technique for all meta-data categories.



(a) Heatmap from Time Series Base Model

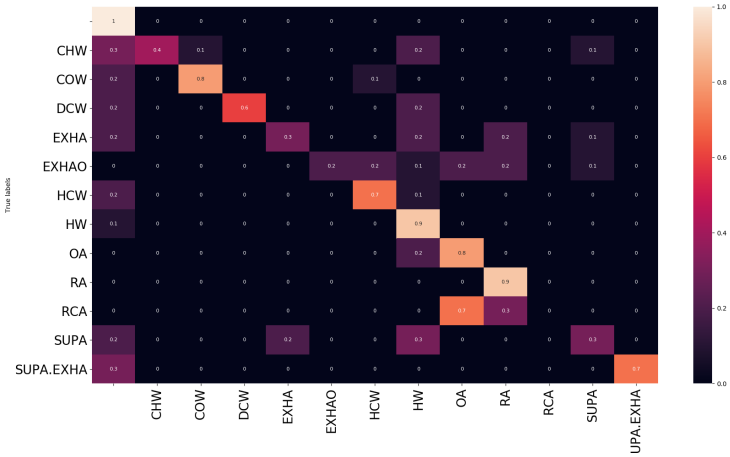


(b) Heatmap from Description Text Base Model

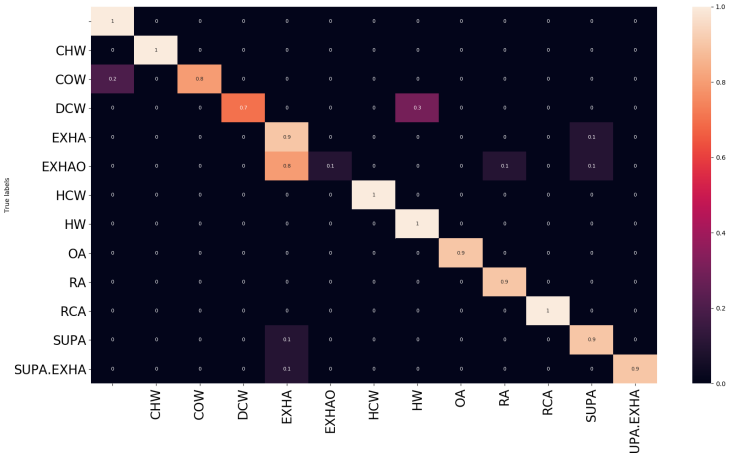


(c) Heatmap from Voting Meta-Classifier Model

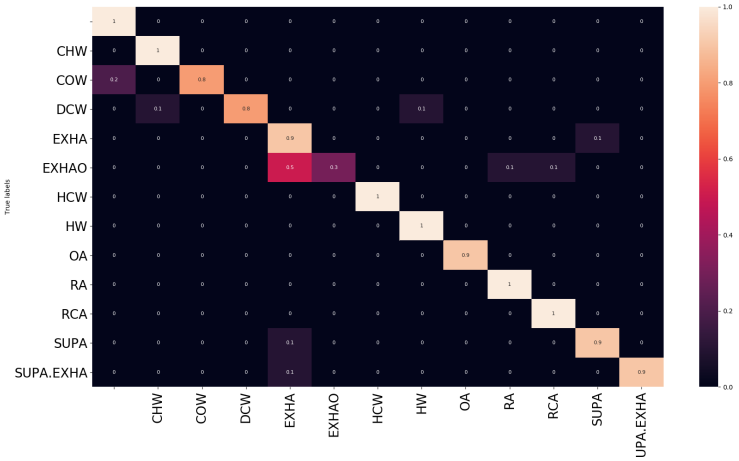
Figure 28.: Comparison for meta-data category Subsystem_2



(a) Heatmap from Time Series Base Model

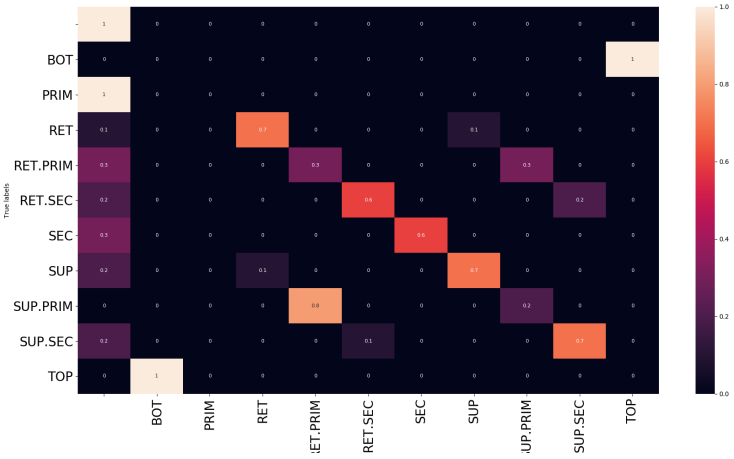


(b) Heatmap from Description Text Base Model

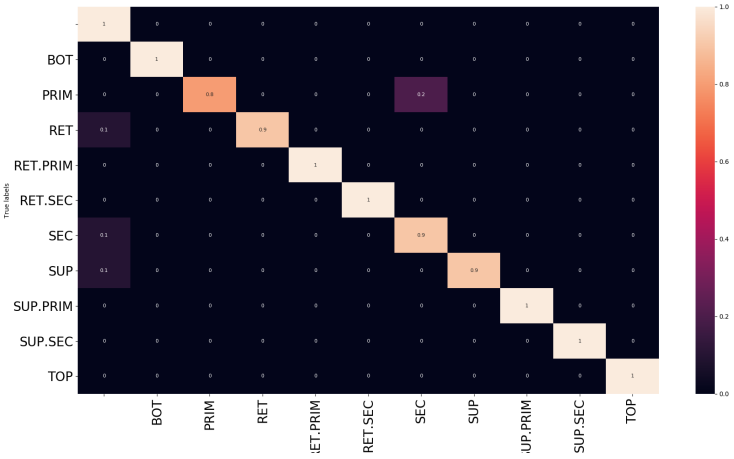


(c) Heatmap from Voting Meta-Classifier Model

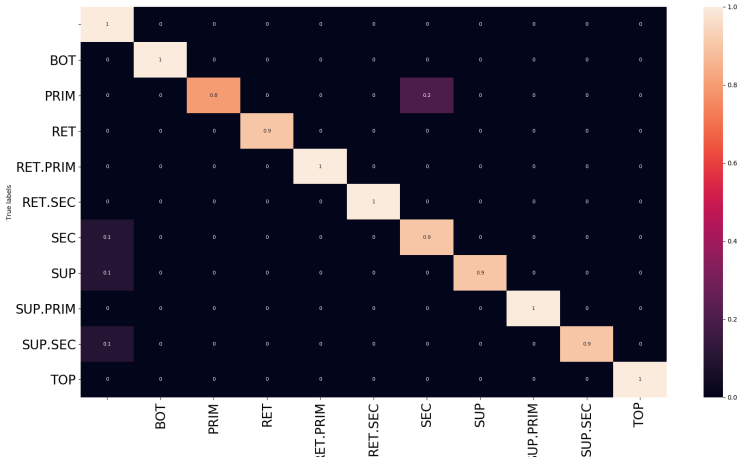
Figure 29.: Comparison for meta-data category Medium



(a) Heatmap from Time Series Base Model

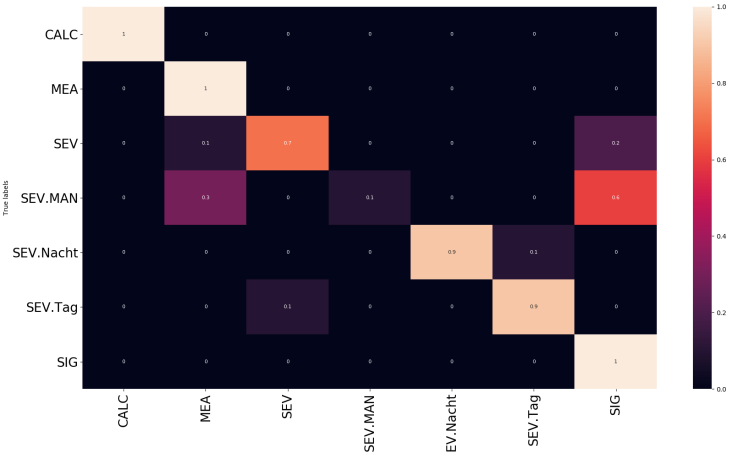


(b) Heatmap from Description Text Base Model

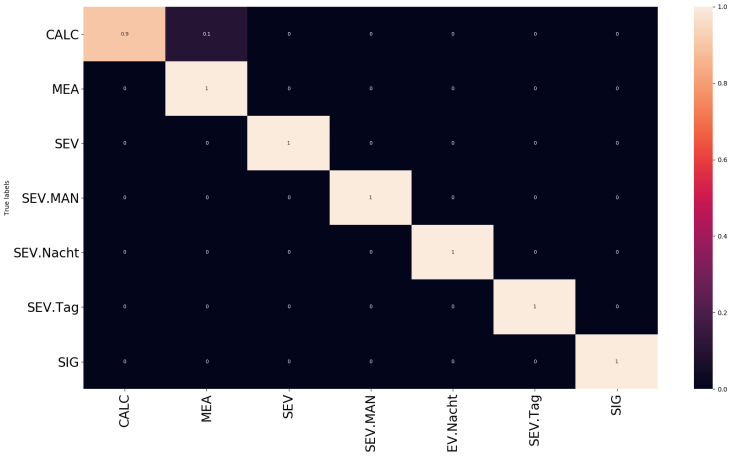


(c) Heatmap from Voting Meta-Classifier Model

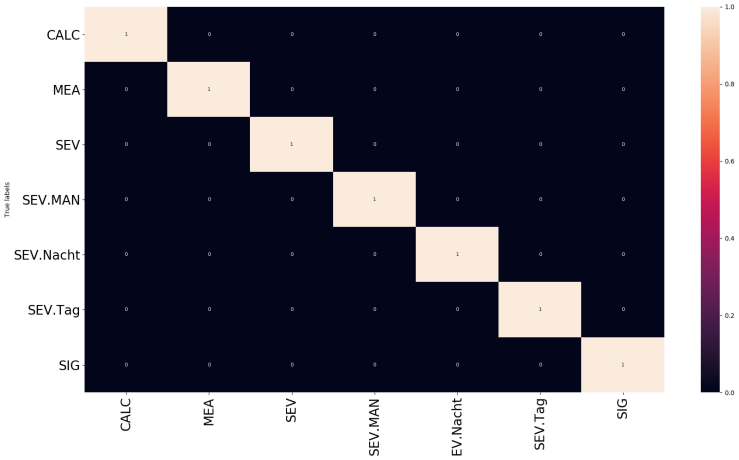
Figure 30.: Comparison for meta-data category Position



(a) Heatmap from Time Series Base Model



(b) Heatmap from Description Text Base Model

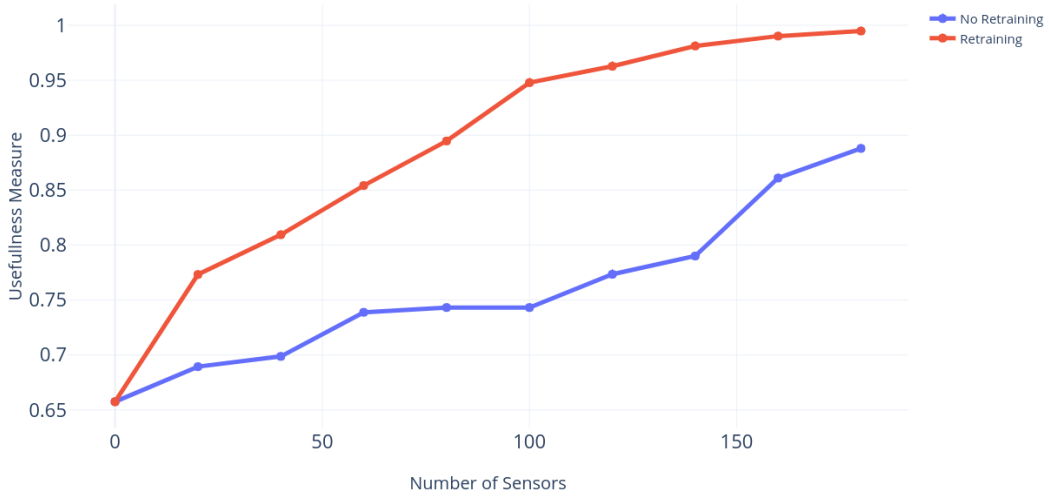


(c) Heatmap from Voting Meta-Classifier Model

Figure 31.: Comparison for meta-data category Kind

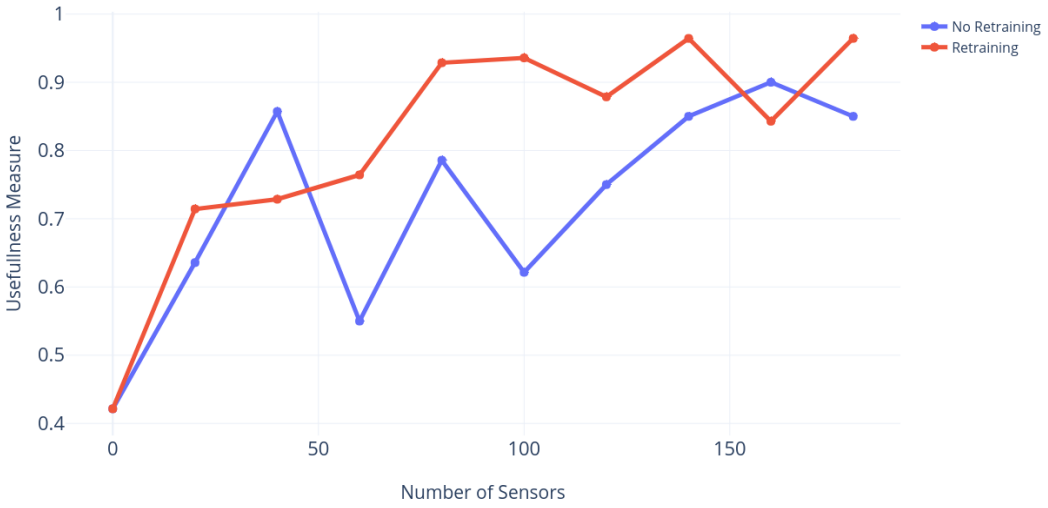
D. Usefulness of other buildings

(a) Average Usefulness on selected 20 labels of building at every iteration



(b) Average Usefulness on all the sensor labels except the 20 selected for retraining the model

Figure 32.: Comparison of usefulness on the selected top 20 labels based on the confidence score versus the rest of the labels for building MWME



(a) Average Usefulness on selected 20 labels of building at every iteration

(b) Average Usefulness on all the sensor labels except the 20 selected for retraining the model

Figure 33.: Comparison of usefulness on the selected top 20 labels based on the confidence score versus the rest of the labels for building 02_DKB_Berlin