TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Fatih Intekin 194231IASM

Real-Time Availability Prediction of Electric Vehicle Charging Spots

Master's Thesis

Supervisor: Sadok Ben Yahia
Professor
Co-Supervisor: Wissem Inoubli
Post-Doc Researcher

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Fatih Intekin 194231IASM

Elektrisõidukite Laadimiskohtade Reaalajas Saadavuse Ennustus

Magistritöö

Juhendaja: Sadok Ben Yahia
Professor
Kaasjuhendaja: Wissem Inoubli
Post-Doc Researcher

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Fatih Intekin

03.01.2022

Abstract

This paper introduces a novel concept to predict the availability of electric vehicle charging stations with high accuracy by both utilizing pre-existing machine learning and deep learning models, techniques, tools and by developing new ones and to present the outcome of prediction to the client systems, to vehicles' multimedia systems, in a reliable manner. The sample dataset used to train the machine learning and deep learning models contains real-life electric vehicle charging point data from Tallinn, Estonia, and Paris, France. Nevertheless, the idea of the system is to make it as generic as possible. Therefore, it allows modifications in the future and can be applied to any city or place easily. Having the destination input taken from the driver or the travel route is provided, the prediction engine created predicts the availability of the charging spots on the route. To achieve this outcome, machine learning and deep learning models are developed and trained with real-life datasets in the scope of this research. Besides those, from the software perspective, the system utilizes distributed systems to push prediction results to client applications or services, calculated by consuming datasets, in a reliable manner and with high availability. Although a limited number of existing implementations focus on availability prediction of the electric vehicle charging spots, their prediction accuracy rate is not high that users adopt them practically, or they do not fit in an end-to-end system that can communicate with client applications easily or can be embedded into vehicles' multimedia dashboards. However, according to recent research, electric vehicles become more dominant day by day and bring the booming charging station numbers. Therefore, this system provides value to its users regarding multiple aspects by helping them find the most available charging station on their destination route, especially these days when electric vehicles are trendy than ever, and their number has been increasing rapidly.

This thesis is written in English and is 42 pages long, including 5 chapters, 14 figures, and 7 tables.

List of abbreviations and terms

ANN	Artificial Neural Network
API	Application Programming Interface
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
CSV	Comma-separated Values
DL	Deep Learning
DNN	Deep Neural Network
EV	Electric Vehicle
GB	Gigabyte
GHZ	Gigahertz
GRU	Gated Recurrent Unit
kNN	K-Nearest Neighbour
KWH	Kilowatt-hour
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MLP	Multi-layer Perceptron
RF	Random Forest
RMSE	Root-Mean-Square Error
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
SMAPE	Symmetric Mean Absolute Percentage Error
SVM	Support Vector Machine
URL	Uniform Resource Locator
XGB	XGBoost

Table of contents

List of figures

List of tables

1 Introduction

As the variety of discussions regarding the environmental issues, pollution of the atmosphere and the harmful effects of fossil fuels are growing altogether, electric vehicle adoption by countries and by users all around the globe has also been rapidly increasing day by day. According to the 2021 Global EV Outlook, there were 10 million electric vehicles on the roads at the end of 2020. Moreover, electric vehicle registrations increased by 41% in 2020, despite global car sales dropping 16% due to the COVID-19 pandemic. Additionally, Europe became the world's largest EV market for the first time, overtaking China [1]. The world's most prominent car manufacturers have already set specific dates for producing only electric vehicles. For instance, BMW's top-selling models, including X3, X5, 3 Series and 5 Series have electric versions, and the company foresees that by 2030, half of their global sales will be electric cars. Another giant manufacturer General Motors plans to stop selling gas and diesel vehicles by 2035. Mercedes and Volvo also aim for going entirely EV by 2030 [2]. Advancements in manufacturing technologies and having electric vehicles performances improved consistently, create a snowball effect in the market which eventually increases consumer demand and drives manufacturers for more innovative and economical solutions.

That being the case with electric vehicles, inevitably, the whole ecosystem surrounding them also grows along the way. Charging infrastructure is one of the most significant components of the electric vehicle ecosystem. Recent studies demonstrate a strong link between EV adoption and the surrounding charging infrastructure. Charging stations not being easily accessible, deficient in numbers, and inadequate quality or quantity, are some of the main reasons preventing EV ownership growth. Researchers concluded that even in a country such as Norway, where among the highest home charging availability worldwide, public charging station infrastructure strongly affects increasing EV ownership [3]. Another research carried out in Sweden, shows that an increased number of public charging points, especially in urban areas, also increases the adoption rate of EVs [4].

With a growing number of EVs and charging stations in both rural and urban areas all around the globe, it becomes more and more challenging for EV drivers to find the most convenient charging spots for their needs quickly and timely when they are on the road. EV drivers either have to rely on external services and applications that display the statuses of the charging spots' that they operate or built, or they are supposed to find one by trial and error. Moreover, those services displaying the status of their charging stations can only provide the value of a current status, which is generally not adequate for drivers. Forecasting the charging station's future availability is vital, especially in today's EV ecosystem, where a car's battery level is critical for a decent user experience. EV drivers want to have the comfort of finding an available charging station suiting their needs not only in the current moment but also a few moments later when they need it.

The work in this thesis ultimately aims to build an end-to-end system that consists of a prediction engine that forecasts the availability of EV charging stations in real-time, a distributed software components that provide the prediction outcomes to client applications in a reliable and resilient manner. And the system built in the scope of this thesis can be easily integrated within the EV's multimedia systems.

1.1 Problem Definition

The work in this thesis essentially focuses on finding the answer to the following question: "Having taken the EV driver's route (start and destination) of the trip as an input, which charging station is going to be the best one on the route, in terms of availability?".

For example, a simplified model of the problem and the aimed outcome can be seen in Figure 1. The system predicts the availability rate for each charging station on the driver's route and displays it to the driver. Station 3 is considered the best in the above example due to the highest availability rate.

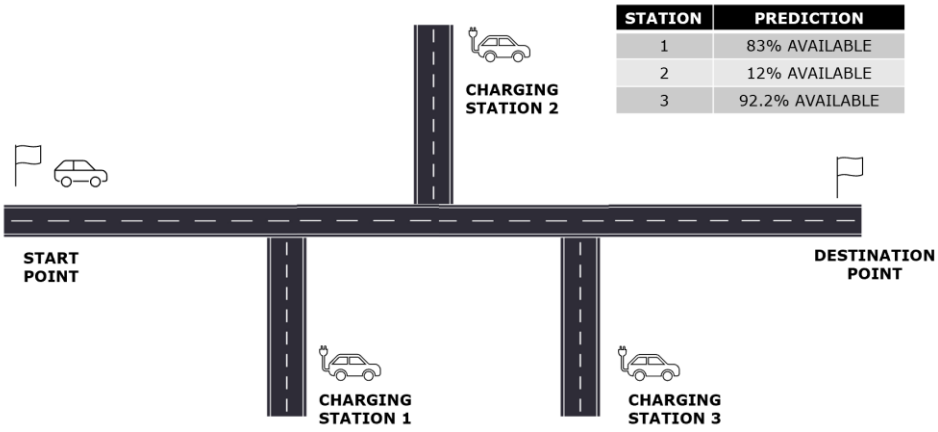


Figure 1. Simplified Problem Model

To answer the question above, each possible status value probability of each station on the drivers' travel route must be calculated. And to be able to calculate the availability probabilities, a prediction engine based on an ML/DL model must be built, and the prediction results must be delivered to client systems.

1.2 Architectural Overview

Architectural overview of the entire system can be seen in Figure 2. The system as a whole constitutes an end-to-end solution for end users, in this context, EV drivers. The parts prior to Web Scraper, are not implemented or their details has not been discussed in the scope of this thesis. Instead, ready-made solutions are utilized for those tasks. Web Scraper, fetches the charging station's data from server, preprocess the data, and generates datasets. Those datasets feeds into the prediction engine, which is a trained ML/DL model that actually performs the prediction. Afterwards, upon a request, results are transmitted to a distributed software application, which is planned to be implemented with Kafka consumers and producers. And finally, client applications, which can be either a mobile application, web application or an application integrated into EV's multimedia system, request prediction results over a web.

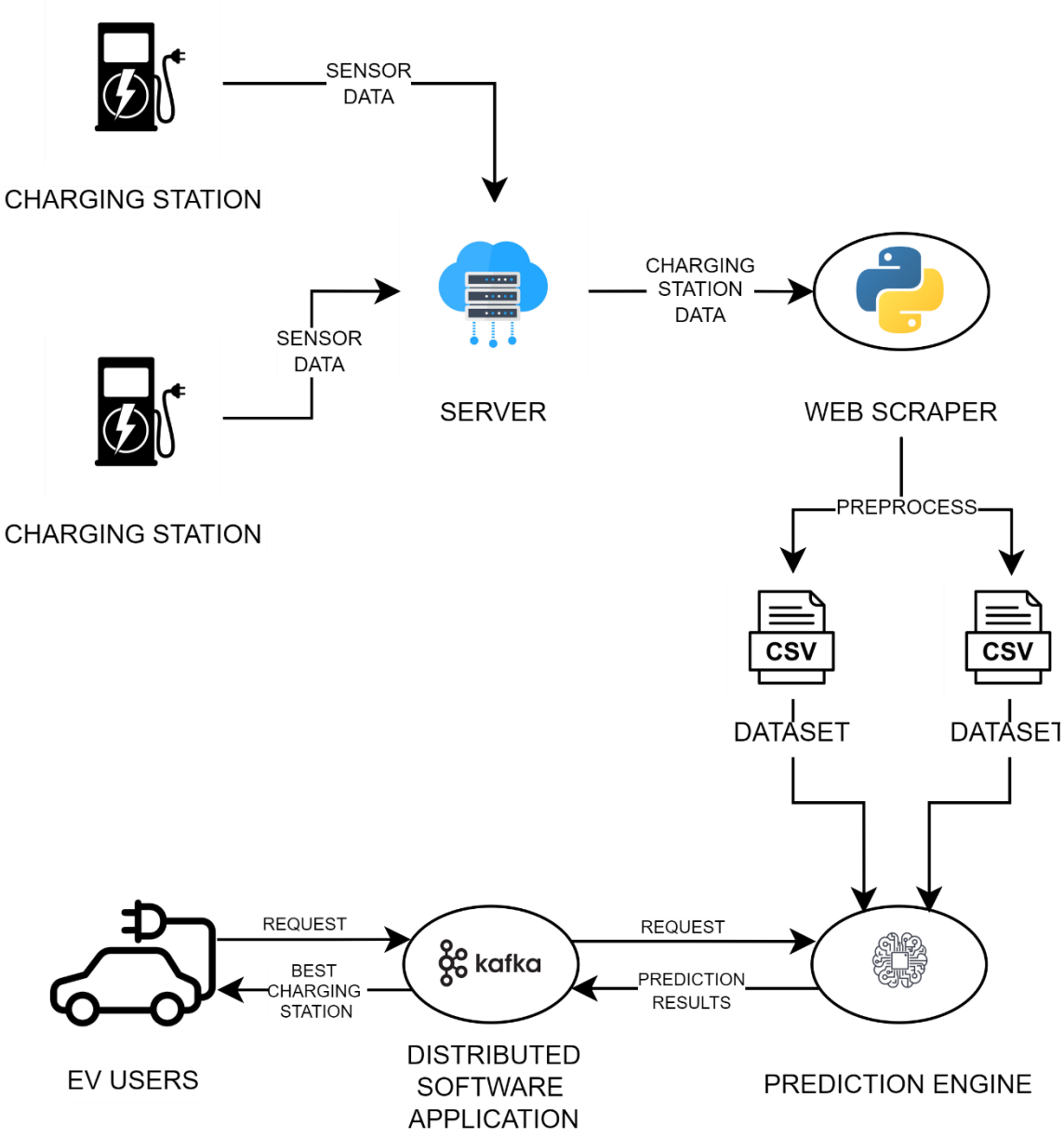


Figure 2. Architectural Overview of the End-to-End System

1.3 Main Contributions

The main contribution of this thesis is that it proposes timely research that correlates with increasing rates of EV adoption. Existing research gap on predicting EV charging stations' availability and the growing need to forecast the results are two of the few factors which make this research valuable and competent.

Moreover, this research provides a high accuracy prediction rate that outperforms existing works and baselines. Besides, it enables a generic solution that can be easily applied in any given location or any given context. Scraping the usually not publicly available data of charging stations in Estonia, a large dataset is made available for any machine to work on.

And last but not least, the proposed solution is integrated into an end-to-end system and can be used in practical, real-world scenarios, bringing value directly to the user experience and the EV ecosystem.

1.4 Structure of the Manuscript

The rest of this paper is organized as the following. Chapter 2 presents the state-of-the-artwork regarding the EV charging station availability prediction and related topics. Also, it discusses prevalent algorithms and techniques used in machine learning and deep learning. Chapter 3 describes the actual implementation steps of the dataset generation, feature engineering, training, and testing machine learning/deep learning models and developing prediction engines. Chapter 4 discusses the results obtained from the implementation part, and finally, Chapter 5 consists of the paper's conclusion, the summary, and the future possibilities.

2 Literature Review

In this section, the state of the artwork for EV charging station availability prediction and a few other relevant prediction/forecasting studies are examined and discussed. Secondly, an overview is given for well-established ML and DL methods which are majorly used for building prediction models. And lastly, the literature review is concluded.

2.1 State of the Art

There has been ongoing research and experimentation regarding electric vehicles and their surrounding ecosystem in recent years. Several of them focus particularly on EV charging infrastructure and are heavily rely on machine learning and deep learning methods. In their study, F. Soldan et al. employed a similar approach to short-term forecasting of EV charging stations occupancy probability, using big data streaming analysis [5]. Researchers propose a big data streaming architecture for providing electric charging station availability forecast after a certain number of times from the present time. To train a streaming logistic regression model, batch data of past changes and real-time data streams are used, to consider recurrent past situations and unexpected current events.

They discovered that their streaming model performs better than a model trained using only historical data because the forecast model trained just using historical data can result in accuracy errors, especially in the case of unexpected events such as a match for an EV charging station that is close to a stadium. Researchers used Logistic Regression as a classification model and increasingly updated the model using real-time data from the actual occupancy of EV charging stations. The classification model and threshold 0.5 mark occupations as occupied (if greater than a threshold) or not occupied (lower than a threshold).

They used precision, recall and F1-score metrics to evaluate their model's result. The latter have shown that occupancy probabilities from the streaming model are generally lower than those from the batch model. However, the streaming forecasts display a higher increase if a charging station is occupied. This increase is more evident with long charges,

whenever occupancy probabilities reach above 0.8. Researchers concluded that a model with a better recall than precision would be chosen in the case of the need to forecast the highest number of charges, with the risk of forecasting as a charge an event that will not be confirmed as an actual charge.

On the contrary, a model with better precision than recall will be chosen when it is necessary to forecast only correct charges, with the risk of losing some charge predictions [5]. Besides those, there are a few notable weaknesses of this research. Firstly, the dataset is not rich as it only consists of temporal variables. Moreover, only one model was applied to the dataset. Finally, a proposed strength of mixing batch and real-time data is performed randomly, without calibration.

In another research by A. Sao et al. [6], the authors established a novel deep learning approach, Deep Fusion of Dynamic and Static Information model (DFDS), to forecast charging station occupancy effectively. DFDS exploits the typical static patterns of the individual charging stations, such as regular occupation rates or means occupation concerning the time of the day, and the dynamic information, such as the current occupation, daytime, and weekday to facilitate occupation predictions. Significant contributions of the paper include that those researchers proposing a novel architecture that effectively combines dynamic and static information. The model efficiently fuses dynamic and static information to facilitate accurate forecasting. Also, the model has an effective prediction rate, outperforming baselines in F1-score. DFDS uses a Gated Recurrent Unit (GRU) based dynamic information encoder to capture the dynamic occupancy of the charging stations.

Further, researchers use statistical features to capture the individual station's typical occupation pattern in the static information component. Finally, they fuse the dynamic and static information and use a GRU-based decoder, called *the fusion component*, to forecast charging station usage. Researchers again used a real-world dataset from Lower Saxony, Germany, between August 2020 and December 2020. To evaluate their results, the metrics used are Precision, Recall, and F1-Score. Researchers observed that dynamic information helps achieve high precision, while static information enables a high recall. As baseline ML and DL models, kNN, Random Forest, Logistic Regression, Support Vector Machine, GRU + Fully Connected, and Sequence2Sequence models are used along with the naïve statistical historical average. Experiments demonstrate that DFDS

outperforms the baselines by 3.45 percent points in F1-score on average. In addition, the DFDS model has a Precision value of 73.12 percent, a recall value of 64.53, percent and an F1-score value of 68.55 percent.

Besides the availability prediction of EV charging stations, several pieces of research focus on EV charging behaviors and predicting future charging demand, using similar machine learning and deep learning approaches. For example, in their paper, F. Qiao et al. [7] aimed to predict future charging demand by building predictive models to characterize behaviors of both registered long-term users and unregistered short-term users. Even though the focus is not the charging station's availability prediction, the research has importance as it considers different user behaviors. For example, registered users tend to use the system for longer terms. In contrast, unregistered users typically are short-term users who use charging stations occasionally. Prediction design includes one predictive model for registered and one for unregistered users. Working on a real-world charging record dataset collected in Caltech, the study applies supervised learning-based algorithms, specifically XGBoost, Support Vector Regression (SVR), and Gradient Boost Decision Tree (GBDT) to predict sequences of future availability. Registered/unregistered users used XGBoost to train two predictive models separately and then combined middle prediction results to obtain final results. Researchers compared their proposed prediction model with SVR, GBDT, and XGBoost and concluded that for RMSE and MAPE, XGBoost performs the best in each time granularity. For MAPE, GBDT performs better than SVR when time granularity is 15, 30, and 80 minutes, while SVR performs better than GBDT in other cases [7].

With similar purposes, S. Shahriar et al. [8] used popular machine learning algorithms to predict charging behavior, more specifically EV session duration and energy consumption, mainly to provide smart scheduling and solve the strain on power grid infrastructure due to the high-power requirements of the EVs. ACN (Adaptive Charging Network) dataset [9] considers input features such as traffic and weather conditions and local events. ACN is a public dataset that contains charging records from stations in Caltech and JPL university campuses. Random Forest, SVM, XGBoost, and Deep Artificial Neural Networks are taken as baselines, along with two additional ensemble learning methods, Voting Regressor and Stacking Regressor. Researchers aggregated three best performing models in the training phase into two ensemble models, which resulted in improved cross validation scores. Results are compared based on RMSE,

MAE, R^2 and SMAPE metrics. The work suggested that results outperform previous works that report similar evaluation metrics.

Authors of [10] [11] [12] also worked on developing models to predict charging demand. Y. Zhao et al. built a novel data-driven framework to ensure the safety of EVs and provide reliable inputs for grid-load calculations. The framework is developed by individually controlling the strongly linear and weakly non-linear contributions. The proposed framework concurrently addresses the overfitting of non-linear networks using a low proportion of training data and the poorly descriptive ability of linear networks under complex environments. To validate the performance of the proposed prediction model, actual real-world EV data and five existing high-performance prediction models (Linear Regression, XGB, RF, and kNN) are employed. China's national big data platform for EVs: Dataset of National Monitoring and Management Center for New Energy Vehicles is used as a real-world data source. Compared with existing prediction models (such as the random forest, XGBoost, and neural network), the proposed framework persists with evidently higher accuracy and stability over a wide range of the ratio between the number of EVs used for testing and training; its mean absolute percentage error (MAPE) is maintained at 2.5–3.8% when the ratio ranges from 0.1 to 1,000. However the prediction model in this research may not obtain the accurate charging energy of an EV when the SOC (state of charge) variation is tiny, i.e., 1–3%. There are many reasons for this, such as unstable charging power at the beginning, sensor errors, etc [10].

A. Almaghrebi et al. also employed Linear Regression, XGBoost, RF, and SVM methods to build a charging demand prediction model. They evaluated their results using the Coefficient of determination (R^2), RMSE and MAE metrics. In addition, researchers used a real-world dataset containing charging sessions data from Nebraska, USA. Their result demonstrates that XGBoost outperforms other methods with 6.68 kWh and 51.9% R^2 [11]. Y. Kim et al. [12] study compared various modeling techniques, including trigonometric exponential smoothing state space (i.e., Trigonometric, Box-Cox, Auto-Regressive-Moving-Average (ARMA), Trend, and Seasonality (TBATS)), autoregressive integrated moving average (ARIMA), artificial neural networks (ANN), and long short-term memory (LSTM) modeling, based on past values and exogenous variables. In addition, the models are evaluated based on MAPE. Researchers concluded that privacy issues regarding driver information play an essential role in predicting charging demand. They must be resolved to forecast power supply effectively, as in a

single station, exogenous variables do not significantly influence accuracy because individual behavior is essential in determining consumption.

2.2 Algorithms and Techniques Overview

This section provides an overview and theoretical information, for the most prevalent ML/DL algorithms and techniques mentioned in the previous state-of-the-art section. It is worthy of note that the algorithms and techniques discussed in this section are chosen based on the relevancy of the work provided in this thesis.

2.2.1 Supervised Machine Learning Algorithms

Machine learning can be defined as a branch of the Artificial Intelligence discipline. ML focuses on using data and algorithms to imitate the way humans learn, gradually improving its accuracy. Through various statistical methods and techniques, algorithms are trained to make classifications and predictions. Generally, ML algorithms use historical data to forecast the future values of desired output variables [13].

There are several machine learning methods, and they can usually be categorized under four primary categories namely Supervised ML, Unsupervised ML, Semi-Supervised ML and Reinforcement ML. However, since the algorithms used in the scope of this thesis are all Supervised ML algorithms, other categories are not discussed in this chapter.

In supervised learning, the dataset being used has been pre-labeled and classified by users to allow the algorithm to see how accurate its performance is [14]. Usually, in Supervised learning, the person training the data knows a lot more about the training data than the machine, so the person can feed labeled data into the machine which can be easily classified later. Supervised ML has many use cases in real-world applications, including classifying spam mails in a separate mail folder, self-driving cars classifying different objects through image processing or, a web application of tourism agent forecasting how many of the hotel rooms will be available on a specific date. A typical machine learning algorithm consists of roughly three components. The first component, the decision process, refers to the steps ML algorithm produces an estimate about a pattern in a data, based on the input data provided. In addition, it can be defined as any calculations that take in the data and return it into a guess at the kind of pattern the ML algorithm is looking to find. The second component is called an error function. It’s a technique of measuring

how good the guess was served to evaluate the model's prediction. It aims to quantify the miss rate, regarding the guess performed by an algorithm. The last component is the model optimization process. In this step, the algorithm looks at failures and then updates it's decision process and means to come to the final decision thus next time miss will be lower. If the model can fit better to the data points in the training set, it will adjust the weights to reduce the discrepancy between the known examples and its estimations. This evaluation and optimization process will be repeated autonomously until a predefined accuracy threshold has been met [13] [14].

2.2.2 Classification Algorithms

Given that the ultimate goal of the research in this paper is to predict the probability of each possible output of the EV charging stations, and since this output will be equal to one of the finite numbers of status values, it can be stated that the model aimed to be built trying to solve a multi-class classification.

Multiclass classification algorithms classify given input into one of the N possible classes. These are supervised ML algorithms. Unlike binary classification, where you have only two possible outcomes, multi-class classifications are not limited to or does not restrict itself to any number of classes (See the comparison scheme in Figure 2). Therefore, two classes are dependent (target) variables when the problem is a multi-class classification problem. Multi-class classification assumes that each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time [15]. Multi-class classification algorithms have a broad scope of usage including image classification, handwritten digit recognition, intent classification in NLP, and so on. In the scope of the work of this thesis, the desired output consists of multiple classes of EV charging stations availability statuses, such as "available," "busy," "unknown" etc. In addition to that, the probability rate of each status value is intended to be calculated, so that the availability percentage can be provided to the end-user. For instance, "Station 1 will be 96% available."

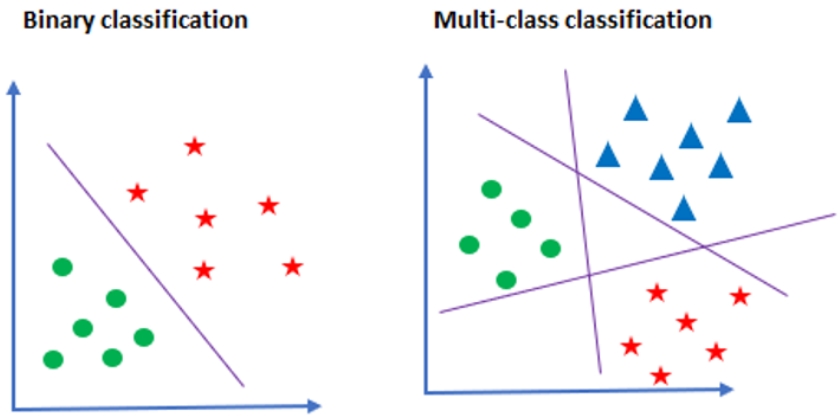


Figure 3. Comparison of Binary Classification and Multiclass Classification [16].

2.2.3 K-Nearest Neighbours (kNN)

K-Nearest Neighbour has always been one of the most popular supervised-learning algorithms for multi-class classification and regression problems. In the simple sense, kNN is based on estimating the class of the vector formed by the independent variables of the value to be assessed, based on the information in which class the nearest neighbors are dense. It is an easy-to-implement algorithm with simple usage; however, it performs lazy learning and can be computationally expensive significantly as the number of independent variables increases because it measures the distance between each data point [17] [18].

In the kNN classifier, the distances between test data and the training data can be identified by different measures. Euclidian distance function, the most common one, can be seen in Equation 2.1. Others include the Minkowsky distance function in Equation 2.2 and the Manhattan distance function in Equation 2.3 [19].

$$d(x,y) = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} \tag{2.1}$$

$$d(x,y) = \sqrt{(\sum_{i=1}^{n}|x_i - y_i|^P)^{1/p}} \tag{2.2}$$

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \quad (2.3)$$

K in kNN is a hyperparameter that can be set to achieve the best possible fit of the algorithm. It corresponds to the number of neighbors on which the calculation will be performed. There are no predefined methods to find the best K value. Lower K values might increase the chance of overfitting and leads to unstable decision boundaries, while higher K values might result in overgeneralized results. One of the methods can be deriving a plot between error rate and K, denoting values in a defined range. Then K value is chosen as having the minimum error rate. However, small K values do not always suit small datasets and big K values do not always serve big datasets [20] [21].

2.2.4 Logistic Regression

Logistic Regression is a binary classification algorithm intended for datasets with two classes of categorical or numerical target variables. Therefore, it cannot be directly used for multi-class classification. Instead, it requires a transformation of the model beforehand.

One of the popular approaches for adapting Logistic Regression to multi-class classification problems is to split the multi-class classification problems into multiple binary classification problems and apply a standard logistic regression on each split subproblem. Another approach is directly changing the logistic regression model to support numerous class labels' predictions. The probability distribution that defines multi-class probabilities are called a multinomial probability distribution. A LR model adapted to learn and predict a multinomial probability distribution is called Multinomial Logistic Regression [22].

Changing Logistic Regression from binomial to multinomial probability requires a change in the loss function that has been used to train the model. For example, the loss function is changed from Log Loss to Cross-Entropy Loss function. And change is applied to the output from a single probability value to one probability for each class label [22].

Figure 4 demonstrates the logistic regression curve, equation, and linear regression line.

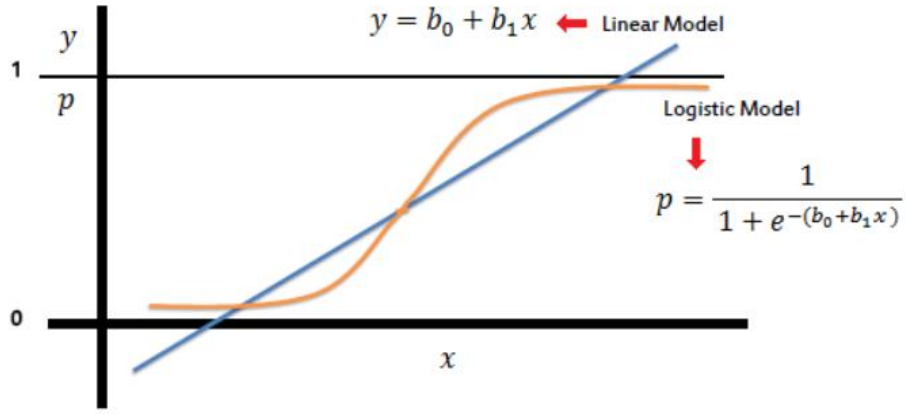


Figure 4. Logistic Regression Curve and Equation [23].

2.2.5 Random Forest

Random Forest is another widely used supervised machine learning algorithm, employed for multi-class classification tasks. In essence, RF consists of many individual "decision trees" that operate as an ensemble. In simpler terms, the model builds multiple decision trees for different parts of the data and selects the final output based on majority voting of individual decision trees in the multi-class classification [24].

RF has a few notable advantages. It has a low chance of overfitting, when sufficient trees are in the RF model. It does not have too many hyperparameters; therefore, it's easy to use and implement. However, there's a trade-off that many trees can make the algorithm too slow and ineffective for real-time predictions. Because in general, RF algorithms can be trained quickly, but they are slow to create forecasts after they are trained. And more accurate predictions require more trees, which eventually results in a slower model [25].

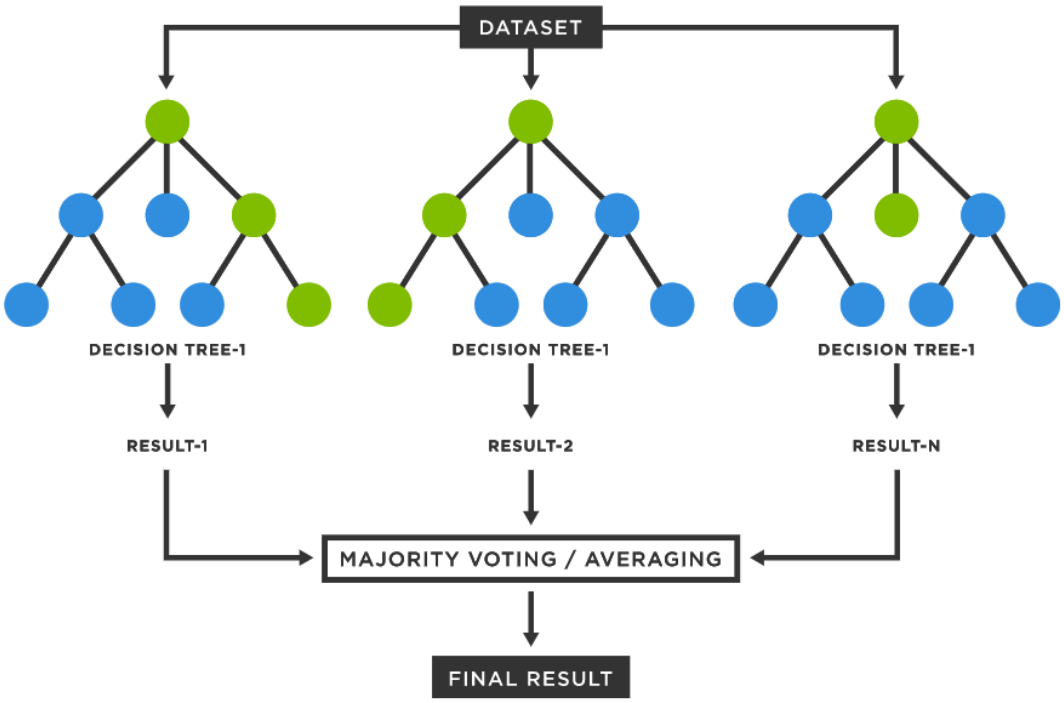


Figure 5. RF is made of multiple individual decision trees.

The aforementioned "ensemble" can use two types of methods. The first of these methods is called bagging, and it creates a different training subset from sample training data with replacement, and the final output is based on majority voting. RF works with this bagging principle. There's also another method called Boosting, and basically, it combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. Models such as AdaBoost and Gradient boosting are based on this principle [26]. For example, XGBoost (Extreme Gradient Boosting) is a widespread high-performance implementation of gradient boosting, consists of an extensive software library and interfaces.

Gradient boosting refers to an approach where new models are created which predict the residuals or errors of their prior models and then added together to make the final prediction with a higher accuracy rate. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models [27]. The gradient descent algorithm also has several subtypes: Stochastic Gradient Descent, Batch Gradient Descent and Mini-Batch Gradient Descent.

2.2.6 Support Vector Machine (SVM)

Support Vector Machine is another supervised ML algorithm used for multi-class classification. The main working principle of SVM is to find a hyperplane that classifies or differentiates the output classes, which comprises data points plotted in an n-dimensional space. Hyperplanes are decision boundaries that help classify the data points, and many possible options of hyperplanes could be chosen, and the SVM algorithm has a feature to ignore the outliers. SVM's main objective is to find a hyperplane with the highest margin value. Margin corresponds to the distance between a hyperplane and the nearest data point. Figure 6 demonstrates a hyperplane with the highest margin. Samples on the margin are called the support vectors [28] [29].

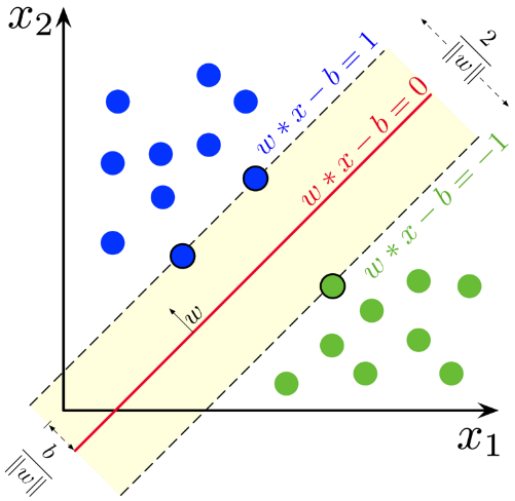


Figure 6. Highest margin hyperplane and support vectors for an SVM with 2 classes [30].

The loss function in SVM that helps maximize the margin is hinge loss, seen in Equation 2.4. The cost is 0 when the predicted value has the same sign as the actual value. The loss value is calculated without the same sign [29].

$$c(x, y, f(x)) = \begin{cases} 0, & if \ \ y * f(x) \ \geq 1 \\ 1 - y * f(x) \geq 1, & else \end{cases} \tag{2.4}$$

The regularization parameter is added to the SVM loss function to balance the margin maximization and loss. After adding the regularization parameter, the loss function equation is given in Equation 2.5 [29].

$$\min_{w} \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+ \tag{2.5}$$

Since the loss function is established, partial derivatives are taken concerning the weights to find the gradients. Weights can be updated using the gradients, as shown in equations 2.6 and 2.7 [29].

$$\frac{\delta}{\delta w_k} \parallel w \parallel^2 = 2\lambda w_k \tag{2.6}$$

$$\frac{\delta}{\delta w_k} (1 - y_i \langle x_i, w \rangle)_+ = \begin{cases} 0, & if \quad y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & else \end{cases} \tag{2.7}$$

Only gradient has to be updated from the regularization parameter When there is no misclassification, as in equation 2.8 [29].

$$w = w - \alpha \cdot (2\lambda w) \tag{2.8}$$

A loss is included along with the regularization parameter to perform gradient update, when there is a misclassification, as seen in equation 2.9 [29].

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w) \tag{2.9}$$

2.2.7 Deep Learning and Artificial Neural Networks

Deep Learning can be considered as a subset of Machine Learning. The primary difference between DL and ML derives from how their algorithms learn. In DL, the majority of the feature extraction part is automated. The classical ML approach is more dependent on manual human intervention to learn. Human experts determine the importance or hierarchy of the features and understand the differences between data inputs, which usually results in requiring more structured data. However, this part is mainly eliminated in DL. DL does not necessarily need a labeled dataset, as it can use unstructured raw data and automatically determine the set of features to distinguish or classify outputs. A DL model can cluster inputs appropriately by observing patterns in the data. It has more complex use cases, i.e., virtual assistants or fraud detection [13] [31].

Artificial Neural Networks, is an Artificial Intelligence branch that tries to mimic the human brain through algorithms. ANN consists of four main components at a basic level: inputs, weights, bias or threshold, and output. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. The “deep” in deep learning refers to the depth of layers in a neural network [13]. And has use cases in areas such as computer vision, natural language processing, and speech recognition. There are several architectural approaches in ANN.

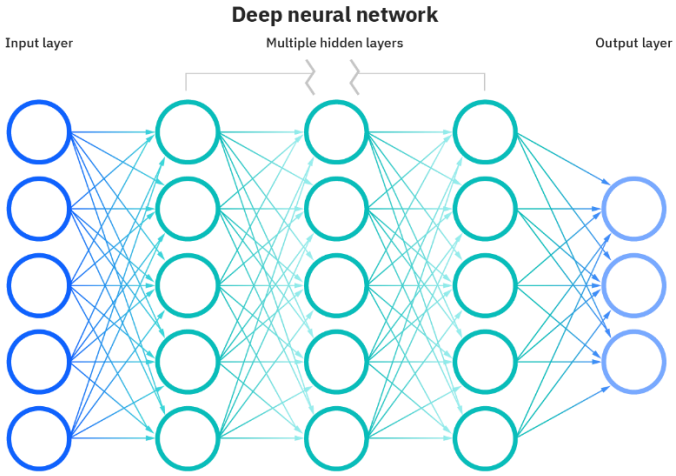


Figure 7. Deep Neural Network (ANN) representation [31].

2.2.7.1 Convolutional Neural Networks

CNN is mainly different from other ANN types by their superior performance with image, speech or audio signal inputs. CNN has three main layers: Convolutional Layer, Pooling Layer and Fully-Connected (FC) layer [32]. Figure 8 shows an example architectural overview of CNN. First, a loss function calculates the model's performance with specific weights and kernels through forwarding propagation on training data. Then, weights and kernels are updated accordingly to the loss value through backpropagation with gradient descent optimization algorithm, e.g., ReLU (rectified linear unit) [33].

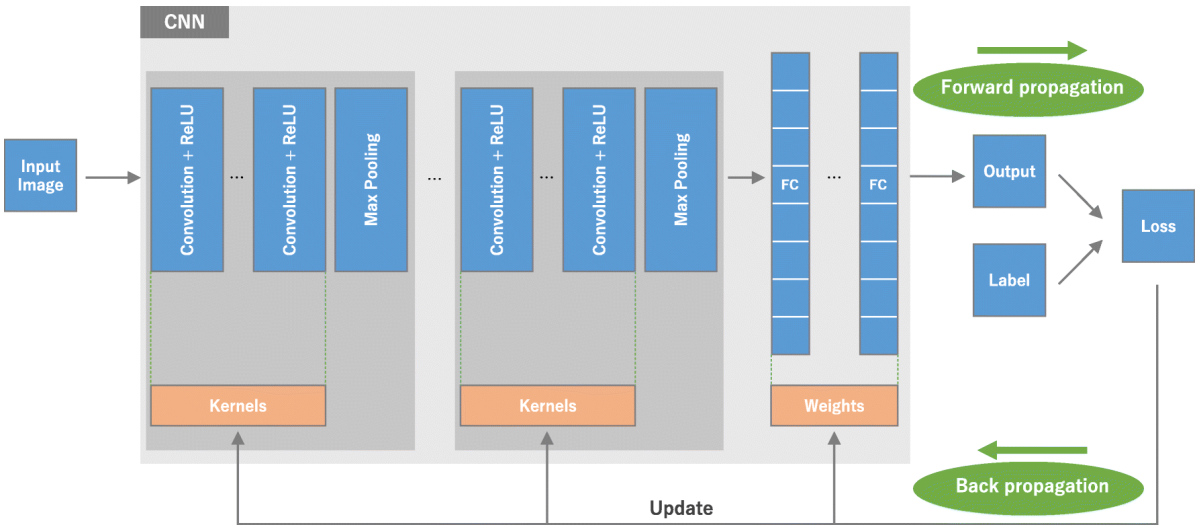


Figure 8. An overview of CNN architecture and training process [33].

The first two layers, convolutional and pooling layer, perform feature extraction. The final FC layer maps the features into the final output, i.e., classification. A convolution layer plays a significant role in CNN due to its mathematical operations, a linear operation [33].

2.2.7.2 Recurrent Neural Networks

RNN is another type of ANN using sequential or time-series data. The main difference between RNN and other ANN models is that RNN have a memory, and they take information from prior inputs to influence the current inputs and outputs. Moreover, they share parameters across each layer of the network. Feedforward networks have different weights across different nodes. RNN, on the other hand, shares the same weight parameter within network layers. An RNN is commonly used for ordinal and temporal problems

such as speech recognition, natural language processing, image captioning, and language translation [34].

RNN uses a backpropagation through time (BPTT) algorithm to determine the gradients. It is slightly different than the traditional backpropagation because it is specific to sequence data. However, BPTT principles are the same as traditional backpropagation. The model trains itself by calculating errors from its output to its input layer. These calculations allow adjusting and fitting the parameters of the model appropriately. But BPTT is different because it sums errors at each time step [34].

RNN has advantages such as processing input of any length and model size is not increased with the input size. But it also has some disadvantages as computation is slow and cannot consider any future input for the current state [35].

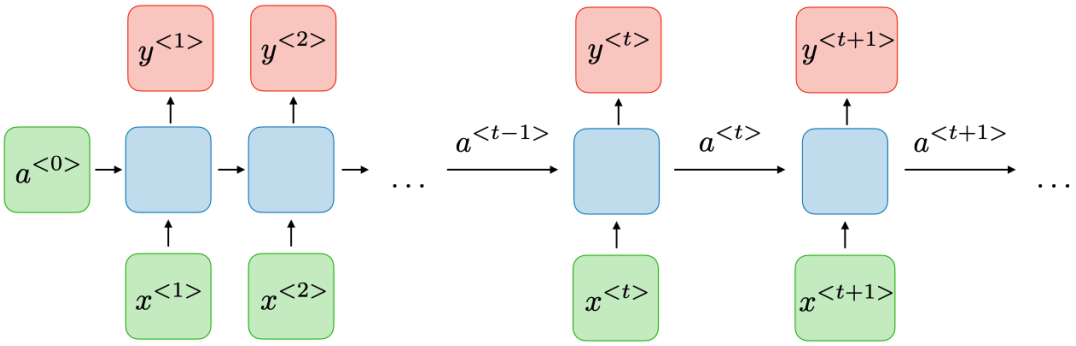


Figure 9. RNN allows previous outputs to be used as inputs while having hidden states [35].

Bidirectional Recurrent Neural Networks (BRNN), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRUs) are some of the popular variants of RNN architecture.

2.2.7.3 Multi-layer Perceptron

MLP is an algorithm that learns a function in Equation 2.10 by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output [36].

$$f(\cdot): R^m \rightarrow R^o \tag{2.10}$$

Figure 10 shows an example of MLP. The layer on the left is an input layer and consists of a set of neurons representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear sum, and a non-linear activation function. Finally, the output layer receives the values from the last hidden layer and transforms them into output values [36].

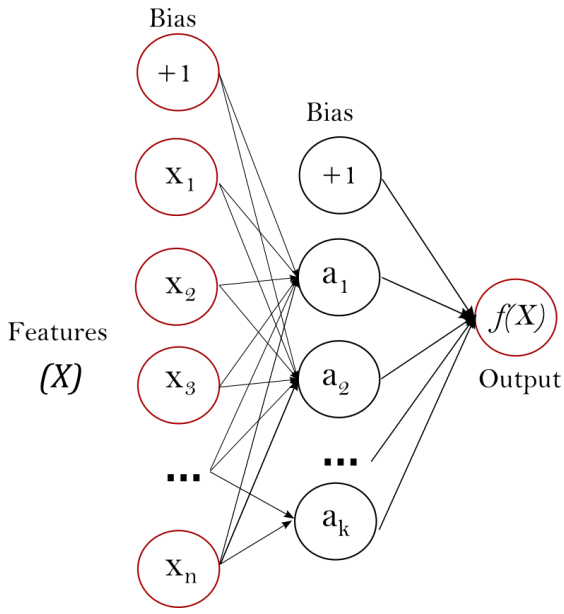


Figure 10. One hidden layer MLP [36]

One of the advantages of MLP is that it is capable of learning non-linear models, and it is capable of learning models in real-time. But it also has some disadvantages, i.e., it is sensitive to feature scaling, and it requires tuning several hyperparameters e.g., the number of hidden neurons, layers, iterations, etc.

2.2.8 Evaluation Metrics

The accuracy and performance of the ML/DL models and algorithms mentioned above are evaluated based on several predefined metrics. Evaluation metrics are used to measure the quality of the model. There are many different types of evaluation metrics available, including but not limited to classification accuracy, logarithmic loss, confusion matrix, etc [37].

Testing a model with multiple evaluation metrics is important because the model can perform well on one measurement but may perform poorly on others from different evaluation metrics.

Three commonly employed evaluation metrics for multi-class classification models are examined.

2.2.8.1 Precision

Precision is defined as the fraction of relevant instances among the retrieved instances. It is used to measure the model's performance on counting the number of true positives correctly out of all positive predictions made by the model [38] [39].

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{2.10}$$

2.2.8.2 Recall

The recall is defined as the fraction of relevant instances that were retrieved. It measures the model's performance regarding the number of true positives correctly out of all the actual positive values [38] [39].

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{2.11}$$

2.2.8.3 F1-Score

F1-Score takes both precision and recall into account. It can be defined as a "harmonic mean" of precision and recall score.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.11}$$

2.3 Conclusion

In the most recent literature regarding the prediction of electric vehicle charging stations availability and related topics, there is a variety of well-proven models, techniques, algorithms, and frameworks of ML and DL. However, some of these methods are used more often than others in researches.

Regarding state-of-the-art, it can be said that there's a research gap that exists regarding charging station availability prediction, as a focal point of many researches is charging demand prediction. Moreover, challenges regarding accuracy and scalability play an essential role in developing and improving of the models. However, reaching high accuracy rates is not straightforward, as datasets can be inadequate or the employed models might not be an optimal fit. Therefore, the main focuses of this thesis are directed towards these weaknesses and deficiencies existing in the literature. That is the main reason high scalability, reliability, and accuracy are emphasised throughout different chapters.

This section provided a fundamental overview and theoretical information of these prevalent models and techniques, most of which are also used during the implementation part in this thesis, to establish the base for the work performed.

3 Implementation

The actual work implemented in this thesis can be summarized as the following. First, data from two different sources is scraped and accumulated frequently to generate datasets. Then, these two datasets are used for building ML/DL models. These models built are then used to predict EV charging spots availability. And last but not least, the prediction system built is integrated into an end-to-end distributed software system.

This section discusses the Data Scraping part and provides detailed info on the dataset used in this research. Afterward, the Baselines part examines the result of base models and methods discussed in the Related Works section applied on the dataset, and their results are compared. Then, the Prediction Model part is the ultimate prediction engine built for real-time prediction. In the final part, the end-to-end software system of which the prediction engine is meant to be integrated is discussed.

3.1 Data Scraping

Data scraping is defined as a process of importing a human-readable form of information from a website or a program into a spreadsheet, local file, or any kind of database [40]. It is one of the most prevalent and most efficient methods for extracting data from any source on the web.

To make any data available for training and testing by a machine, that data must be scraped, cleaned, formatted, and preprocessed. In the scope of this thesis, two different data sources are used and both sources were scraped separately, however with similar techniques, and they were made available for machines to use.

For both data sources, scraping is performed with a predefined frequency by creating a scheduled job on a personal computer and setting the job, so it runs once every half an hour. Hence, two python scripts performing the actual scraping action have been regularly run every half an hour. The scraping cronjob started to run in September 2021. However, there are time intervals when it was stopped due to a technical issue or it failed and had

to be started over again. Except for times that it stopped scraping, data had been scrapped from both data sources once in every 30 minutes since the day it started.

The scraper python scripts have some other similarities, too, as they both use the python requests library [41] to make HTTP requests and parse their responses. In addition, the CSV module [42] in the python standard library is used to write or append results into a CSV file.

The following subsections discuss the actions performed during data scraping of two separate data sources.

3.1.1 Belib – Paris Data Source

The first data source utilized in this research is an open dataset, and it contains data from 1822 different EV charging stations located in Paris, France. It provides geolocated real-time availability data of charging points for EVs. The original title of the dataset in French is: "Belib' - Points de recharge pour véhicules électriques - Disponibilité temps reel". The English translation of the title is Belib - Charging points for electric vehicles - Real-time availability [43]. The entire public network of supervised charging points and providing real-time availability data is called Belib [44]. The Paris council voted to award a service concession to a new operator, Total Marketing France (TMF), for the technical and commercial operation of public charging stations in Paris. Concretely, the network of old Autolib 'terminals will gradually be replaced by new terminals. At the end of the deployment, Paris will then be equipped on the road with 433 charging stations for electric vehicles (one station is equipped with several charging points). The new terminals will be accessible via the new operator as they are installed [43]. The dataset is licensed with Open Database License (ODbL) [45] and is contributed by TMF.

The data source has a public API provided with documentation [46]. Therefore, scraping is performed via sending requests to this public API with appropriate parameters. The complete code of the python script can be seen in Appendix 2. First, an HTTP GET request is sent to the following URL: "https://opendata.paris.fr/api/records/1.0/search/?dataset=belib-points-de-recharge-pour-vehicules-electriques-disponibilite-temps-reel&q=&facet=statut_pdc&rows=2000".
Then, as seen from the URL, the status facet is requested with a maximum of 2,000 rows of EV stations from the Belib data source. Then, results are parsed and saved into a CSV

file each time the scraping is done. In total, more than 700,000 rows of data are parsed, and each row represents a single charging station.

3.1.2 Enefit VOLT – Estonia Data Source

The second data source used in this master thesis is a private company operating in Estonia, called Enefit Volt. Enefit Volt claims to be the largest charging network in Estonia with more than 185 EV Charging stations all over the country. It has a variety of chargers for different needs and constantly growing [47].

Enefit Volt has a "Find the nearest charger" page on their website (see Figure 11) where they display charging spots on a map. Along with charging spots, availability, and a few other information about the spot is displayed on the page. However, since Enefit VOLT does not provide a public API, scraping performed a bit more differently than the previous data source. The complete code of the python script can be found in Appendix 3. Briefly, an asynchronous request that the webpage performs to reload data is imitated in the python code. Request headers are created and a request is sent to a URL with longitude and latitude parameters, covering the whole Estonia map. Like the first data source's script, a response is parsed and then appended to a CSV file. In total, more than 370,000 rows of data are parsed, and each row represents a single charging point.

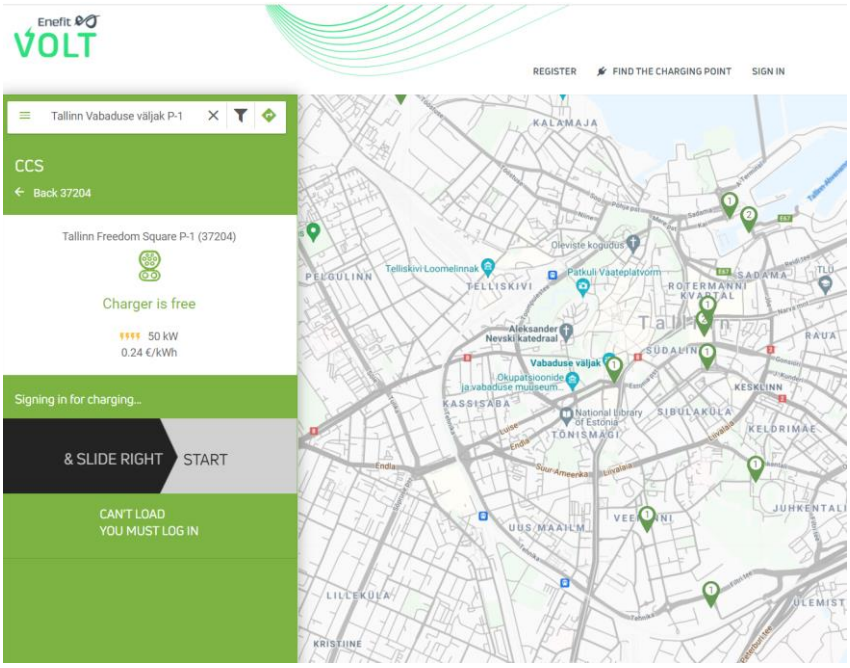


Figure 11. Enefit Volt's Find the nearest charger page.

3.2 Dataset

After the data is scraped from the data sources, rows are appended in a CSV file to generate the actual dataset used for building the model. This section demonstrates what those two datasets look like by providing their fields and descriptions.

The data model of the first dataset, Belib – Paris, France, can be seen in Table 1. It has eight columns, most of which have categorical values.

Table 1. Belib Paris Dataset Model

Field Name	**Description**
ID	Identifier of the charging spot, e.g., FR*V75*EPX12*02*5
Status	The status value of the charging point, the value from the Charging Points Status information table. (see Table 2) e.g., Disponible
Address	Full address (at least, the names of the road or the locality and the town) of the station, e.g., 3, rue de la Gare, Belmont
Postal Code	Postal code of the location of the Charging Point, e.g., 75015
Latitude	Charging Point's geographical coordinate's latitude value, e.g., 48.8512
Longitude	Charging Point's geographical coordinate's longitude value, e.g., 2.2913
Last Updated	Last time the charging station's data is updated, in YYYY-MM-DD'T'HH:mm:ssZZZZ format, e.g., 2021-10 16T10:36:04+00:00
Record Timestamp	The time row is recorded in the CSV file in YYYY-MM DD'T'HH:mm:ss.SSSZZZZ format, e.g., 2021-10-16T10:36:04.525000+00:00

The status field value set of the Belib dataset can be seen in Table 2. There are nine possible values of status that can exist for a charging spot. The original French name for the status value is also given since it is saved in the CSV file.

Table 2. Belib charging spots status values and their description [43].

Original Status Value	Status Value in English	Description
Disponible	Available	The charging spot is available, free to use.
Pas implémenté	Not Implemented	For the moment charging spot is not implemented by the operator. The operator will go back to “In Maintenance” from the moment a maintenance update has been declared.
Occupé (en charge)	Busy (in charge)	The charging spot is busy. There is currently a charging session.
En cours de mise en service	In the process of commissioning	The charging spot has not been commissioned yet.
En maintenance	In maintenance	The charging spot is in maintenance.
Mise en service planifiée	Planned commissioning	Commissioning is planned and it will be available soon. This step follows the status “In the process of commissioning.”
Supprimé	Suppressed	The charging spot is removed from the infrastructure.
Réservé	Reserved	The charging spot is reserved.
Inconnu	Unknown	The status is unknown. Terminal does not communicate with the server, because it is turned off, has no network communication, and is not in maintenance mode.

The data model of the second Dataset, Enefit VOLT, Estonia, can be seen in Table 3. It has twenty-three columns. Columns Socket [1-2] Max Power, Socket Count, and Socket [1-2] Price have numerical values. The rest of the columns have categorical or semi-categorical values.

Table 3. Enefit VOLT Estonia Dataset Model

Field Name	**Description**
ID	Identifier of the charging spot, e.g., 390
Name	Name of the charging spot, e.g., Tallinn Ülemiste Keskus P-1
Status	The status value of the charging point, the value from the Charging Points Status information table. (see Table 4), e.g., Available
Address	Full address of the charging spot, e.g., Akadeemia tee 15, Tallinn
Charging Speed	Speed of the charging station, e.g., Semi Fast
In Maintenance	A Boolean indicator showing whether the charging spot is in maintenance, e.g., True
Latitude	Charging Point’s geographical coordinate’s latitude value, e.g., 59.356
Longitude	Charging Point’s geographical coordinate’s longitude value, e.g., 24.892
Access Level	Charging station’s accessibility, e.g., Public
Socket Count	How many sockets does the charging spot have? e.g., 2
Socket [1-2] Name	Name of the individual socket(s), e.g., CHAdeMO
Socket [1-2] Status	Status of the individual socket(s), e.g., Available
Socket [1-2] Type	Type of the socket(s), e.g., TYPE_2_MENNEKES
Socket [1-2] Charging Mode	Electrical charging mode of the socket, e.g., MODE 3
Socket [1-2] Max Power (kWh)	Max power supplied by a socket in kWh, e.g., 50
Socket [1-2] kWh Price	Price per kWh, e.g., 0.24 euro
Recorded Timestamp	The time row is recorded in the CSV file in YYYY-MM DD HH:mm:ss.SSS format, e.g., 2021-10-16 10:36:04.525000

The status field value set of the Enefit Volt dataset can be seen in Table 4. There are seven possible values of status that can exist for a charging spot.

Table 4. Enefit Volt Charging spots status values and their description

Status Value	Description
Available	The charging spot is available, free to use.
Occupied	The charging spot is busy, occupied by someone else.
Charging	The charging spot is busy. There's currently a charging session.
Paused	Service is paused at the charging spot.
Faulted	The charging spot is faulted and needs to be fixed before being used.
Unavailable	The charging spot is unavailable due to technical problems.
Preparing	The charging spot is preparing to be available.
Unknown	The status is unknown, no connection with the server.

Today, the Paris Belib dataset has 700,000 records and the Estonia Enefit Volt dataset has more than 387,000 records.

3.3 Feature Engineering

A feature engineering process and preprocessing of the datasets are necessary to utilize these generated datasets. Feature engineering refers to the process or pipeline steps that transform raw data into features used in machine learning algorithms. Predictive ML/DL models consist of outcome variables and predictor variables, and during the feature engineering most useful predictor variables are created and selected for the model [48].

Since there are two different datasets with several columns and data types, two separate pre-processors are created for both of them. For feature engineering tasks, some of the

common python libraries are employed. Pandas, being one of those, is an open-source and free library written for python and used for data manipulation and data analysis tasks. It has many capabilities, including reading and writing data from and to CSV files into memory, merging or slicing large datasets, and creating DataFrame objects with integrated indexing [49]. Besides, the Pandas library makes it easy to manipulate and preprocess of time-related and categorical data. Another python library employed for various tasks is scikit-learn. It's similarly an open-source and free library of mainly machine learning. It allows easily and quickly building ML/DL models, running and evaluating them. But it also has many other features regarding data processing and feature engineering. It also works compatible with scientific libraries such as NumPy and SciPy [50].

First, let's examine the preprocessing of the Paris dataset. The complete code of the python script can be seen in Appendix 4. Firstly, raw data is read from the CSV file and assigned to a built-in dataframe object in the memory. Later, the "Last Updated" column, which has a Datetime datatype, is parsed, and the following columns are generated: Year, Month, Day, Hour, Minute, and Second. Then, the redundant columns are dropped from the dataframe. Those dropped columns are: "ID," "Last Updated," "Record Timestamp," "Postal Code," and "Address". Next, normalization is applied to Latitude and Longitude columns. Since those have semi-categorical arbitrary values, they cannot be used directly. Instead, they must be converted to values between 1 and 0 as it is the scale of the numerical data in other columns. Otherwise, an ML/DL model will encounter issues regarding unscaled values. Therefore, Scikit Learn's preprocessing module is used, particularly *MinMaxScaler* function, with range parameter given tuple (0, 1). Pandas *get_dummies* method is used for temporal columns, which were generated by parsing the "Last Updated" column. This method converts categorical variables into indicator variables [51]. A new field is created and filled with 1s and 0s for each minute in the data column. For instance, if the Last Updated value equals "2021-10-15 13:21:53", the minute_21 column would be equal to 1 while all other minute columns would equal 0. Lastly, the processed dataframe is saved to a new CSV file.

Pre-processing of the Estonia dataset also has similar steps. First, raw data is read from the CSV file, and temporal fields are parsed with the same techniques. Then following redundant columns are dropped from the dataframe: "ID," "Name," "Address," "In Maintenance," "Socket Count". And same as before, Latitude and Longitude fields are

scaled. *get_dummies* method in Pandas library is used for One-Hot Encoding on the following columns: “Charging Speed,” “Access Level,” “Socket 1 Type,” “Socket 1 Charging Mode,” “Socket 2 Type,” “Socket 2 Charging Mode”. Later, differently from the Paris preprocessor, sklearn.preprocessing module is used employed for a few additional tasks. Numerical columns had values in different scales in the dataset. For instance, some “Socket 1-2 Max Power (kWh)” fields have value of 50. However, some other “Socket 1-2 kWh Price” fields have a value of 0.24. With scales irrelevant to each other, ML models will result from output values. The complete code of python script can be found in Appendix 5.

To solve this problem, *MinMaxScaler* method of sklearn.preprocessing module is used for scaling the values of the following columns: “Socket 1 Max Power (kWh),” “Socket 2 Max Power (kWh),” “Socket 1 kWh Price,” “Socket 2 kWh Price”. *feature_range* parameter of the *MinMaxScaler* is assigned with a tuple (0,1) so that these fields will get values between 0 and 1. This method, dividing values by the maximum and minimum values of that field, performs scaling in the background. As a result, all the values get a number between 0 and 1.

3.4 Baselines

Baseline ML algorithms are K-NN, Logistic Regression, Random Forest and Support-Vector Machine. The main reason for this is that those are the most prevalent algorithms in state-of-the-art papers, which can for multi-class classification problems. Performances of those baseline algorithms are compared by Precision, Recall, and F1-Score validation metrics. Preprocessed datasets from the previous step are used for training and testing the models. Training and test data are split before the model’s actual fitting. 70% of the data is used as training data, and 30% is used as testing data. Built-in *train_test_split* method in Scikit-Learn library is employed for this task. After the split, baseline models were fit, and the prediction was performed. Apart from the difference of datasets and reading them from CSV into the memory, identical code is used for Paris and Estonia data. Complete python code for running the baseline models can be seen in Appendix 6.

K-NN model is used through Scikit-Learn library’s KNeighborsClassifier. It can take optional parameters, and the *n_neighbors* parameter specifies the number of neighbors to

use during fitting [52]. *n_neighbors* parameter is assigned to 1 to make it the same as state-of-the-art works.

The logistic Regression model is used through Scikit-Learn library's LogisticRegression class. This class applies regularization by default. The optimizer parameter is assigned "lbfsg". In addition, the "max_iter" parameter specifies the number of iterations taken for the solvers to converge, and is set to 1,000 [53].

The Random Forest model is used through Scikit-Learn's RandomForestClassifier. *n_estimators* parameter specifies the number of trees in the forest and is assigned to 500, to match state-of-the-artwork.

Support Vector Machine model is used through Scikit-Learn's LinearSVC (Linear Support Vector Classification). It is called with default parameters.

Once again, the Scikit-Learn library is used for evaluation metrics by importing its metrics module. Built-in *precision_score, recall_score, f1_score* methods are called with the average parameter assigned "weighted". The weighted average calculates metrics for each label and finds their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F1 Score that is not between precision and recall [54].

The system that the baseline methods ran on, including main hardware and software components, can be seen in Table 5.

Table 5. Components used to run baseline models.

Component Name	**Model/Feature**
Memory	16 GB
CPU	Intel Core i7-1065G7 1.50 GHz
Environment	Jupyter Notebook
Programming Language	Python 3.7

3.5 Prediction Engine

Additional approaches than baseline ML algorithms are adopted to implement the actual prediction engine. Because the prediction engine requires prediction outcomes to be provided as probability rates, as opposed to plain ML methods, which provide one final output among the possible classes. Additionally, more advanced and up to date techniques must have been used to make the prediction engine real-time and increase its accuracy as much as possible.

Regarding all those considerations, the Artificial Neural Networks model is built and used for prediction engines. The complete python script used for creating ANN can be seen in Appendix 7. For building the ANN model, some additional libraries and techniques are used. Keras is a popular open-source deep learning library for python, used for defining and training deep learning models. Keras allows users to create DL models easily and quickly, enabling parallelization while running the models. It supports scaling, also can use CPUs and GPUs. Moreover, Keras runs on top of the TensorFlow. TensorFlow is an end-to-end open-source machine learning platform [55].

First encoding is applied on output label, status. Using Scikit Learn's built-in *LabelEncoder* function, the Status column is transformed into integer values starting from 0 and raising by one for each value. Afterward, a sequential model is created using Keras and three layers are added. The first and second layers have a RELU activation function, and the final output layer has a Softmax activation function configured. Then the model is compiled with Categorical Cross entropy loss function and SGD optimizer, all built-in Keras classes or methods. The activation function decides whether to activate a neuron by calculating the weighted sum and adding more bias. The purpose of the activation function is to delinearize the output of a neuron. Those Activation functions, mainly Softmax are chosen because they assign decimal probabilities to each output class in a multi-class classification problem. Those decimal probabilities add up to 1.0. This additional constraint helps the training converge more quickly [56]. Additionally, Softmax presumes that each sample of data is a member of exactly one class. This characteristic makes it a perfect candidate because the outcome aimed to be reached in this thesis matches this idea entirely. Ultimate desired output is sought to be probability rates of possible availability classes, and Softmax enables this.

Since Keras does not have an F1-Score metric directly built-in, it was created manually. Then, the Keras backend module is imported to calculate the number of true positives and its rate to all positive predictions to extrapolate Precision. Finally, similar calculations were applied for calculating the recall, and they are used to calculate F1 Score ultimately.

In the last part, Matplotlib is used to visualize the results. Matplotlib is another open-source python library, that provides extensive object-oriented API for graphical plotting and data visualization [57]. In addition, Matplotlib's pyplot module is employed for drawing bar plots to display availability prediction rates through charts.

3.6 Distributed Server-Side Software

The prediction engine is integrated into a distributed software system to transmit the prediction results to the end users of the system, EV drivers. It is integrated through a connector to become a part of the Kafka Producer application. Therefore, Kafka's prediction engine can be part of a distributed system, making it more reliable and resilient. Kafka is an end-to-end event streaming platform that has several capabilities. Kafka writes and reads streams of events, including continuous import/export of your data from other systems, stores streams of events durably and reliably for as long as it's needed, and processes streams of events as they occur or retrospectively [58].

Figure 12 demonstrates the architectural overview of the system. Client applications, which can either be a mobile application, web application, or a built-in application in EV's multimedia system, make a request to receive the final prediction result from the Kafka producer. The prediction engine is connected to Kafka and sends the prediction results whenever requested. Additionally, there's a separate dashboard that can display prediction stats. Kafka stands at the crossroads of the different components of the system. It provides all the event streaming functionality in a distributed, highly scalable, fault-tolerant, and secure manner. It can be deployed on bare-metal hardware, virtual machines, containers, on-premise hardware and the cloud [58].

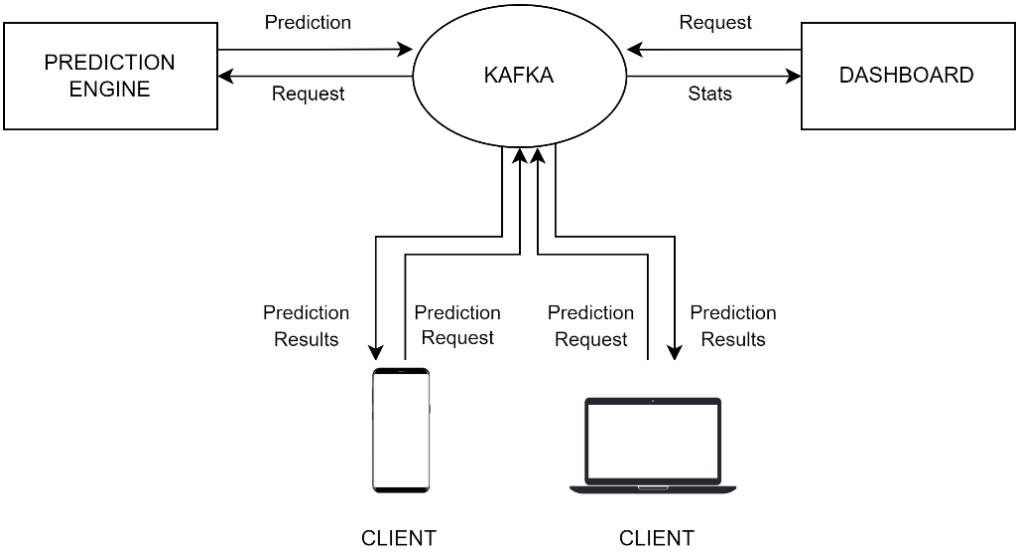


Figure 12. Architectural overview of the distributed software system.

3.7 Conclusion

In this section, the entire implementation part of this thesis was discussed. First, a thorough and meticulous effort is put into implementation starting with creating datasets that constitute the main input for ML/DL models. For example, two countries' EV charging station data is frequently scraped/crawled from various sources and stored in CSV files. Later, feature engineering is applied to those raw stored data to make it compatible and ready to use in ML/DL models. Next, baseline ML models are trained using Python frameworks and preprocessed datasets to make predictions. Then, the creation of the ANN model as an ultimate real-time prediction engine is shown. Finally, the software provides a distributed reliable end-to-end solution for client applications, software application implementation, and prediction engine integration is discussed.

4 Results and Evaluation

In this chapter, the results obtained in the previous implementation chapter is discussed. Multi-class classification ML/DL models are evaluated against Precision, Recall, and F1 Score metrics, to measure and compare their performance.

Results by ML/DL approach for Paris data are given in Table 6. It can be seen that among the ML baselines, RF gave the highest results in all metrics, with over a 95% rate. KNN follows RF with a 94% rate in all metrics. In terms of F1 Score, SVM gave the lowest rate with 59.54%, and Logistic Regression followed that with 60.44%.

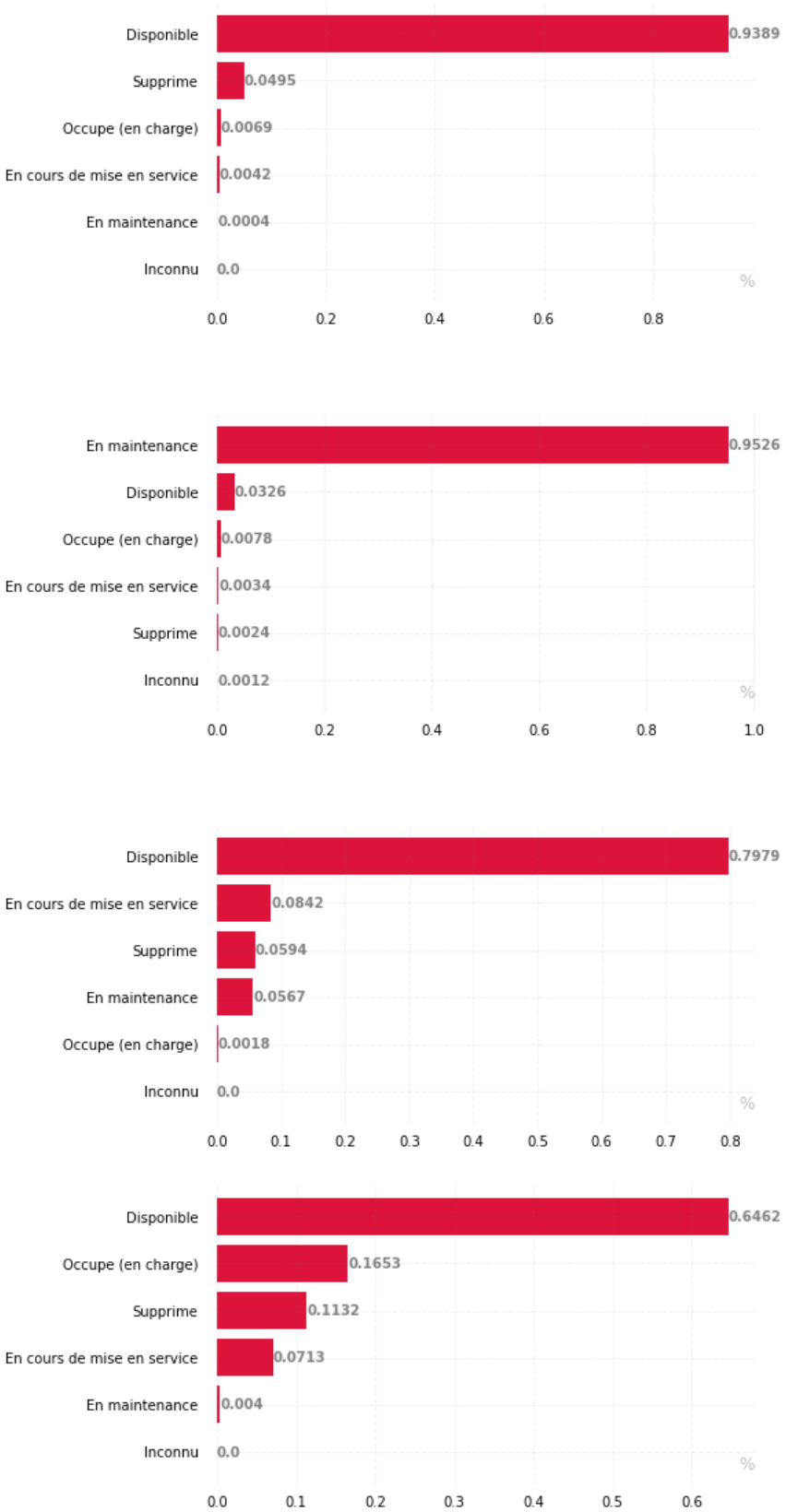
On the other hand, the ANN model has lower scores than KNN and RF models, with 89.46% precision, 87.94% recall, and 88.69% F1 score. However, it should be kept in mind that ANN predicts the rates for each possible outcome, and it can predict results in real-time, whereas kNN has a lazy-learning approach. Therefore it can't be used for real-time predictions, and RF cannot provide output classes rates.

Table 6. Paris Dataset Performance Results (%) by Model.

APPROACH	PRECISION	RECALL	F1-SCORE
K-NEAREST NEIGHBOURS	94.10	94.13	94.11
LOGISTIC REGRESSION	66.65	72.16	60.44
RANDOM FOREST	95.54	95.56	95.41
SUPPORT VECTOR MACHINE	66.41	71.90	59.54
ANN	89.46	87.94	88.69

Figure 13 shows the visualization of the first five prediction results from the ANN prediction engine for the Paris dataset. The values correspond to percentages of status

outputs. For example, the first predicted result can be interpreted as: “Charging station is 93.89% Disponible (Available)”.



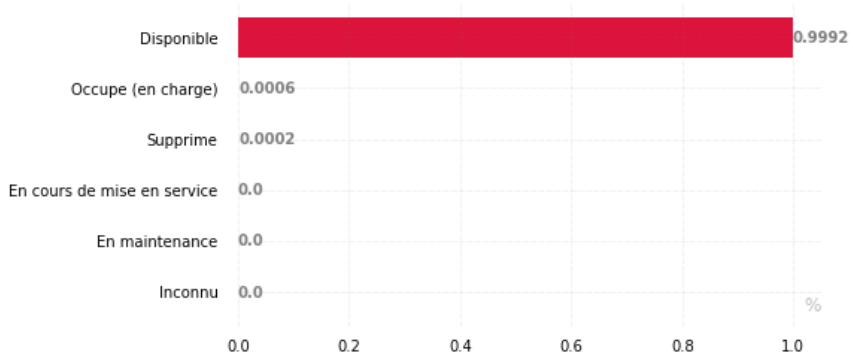


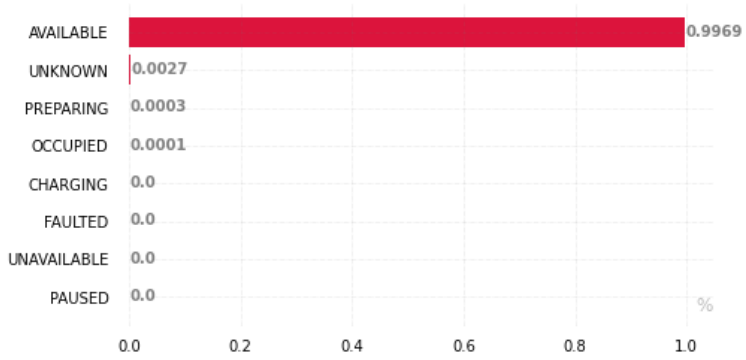
Figure 13. ANN model prediction results for the Paris dataset.

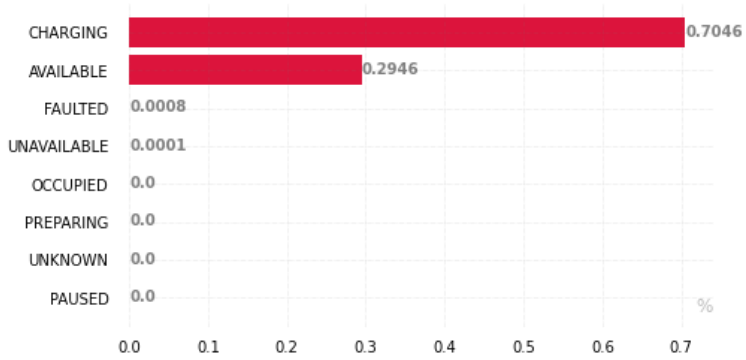
Results by ML/DL approach for Estonia data are given in Table 7. It can be seen that ML baseline models yielded very close results, and they all have high scores. However, the ANN model slightly outperformed all of its competitors in precision and F1 score.

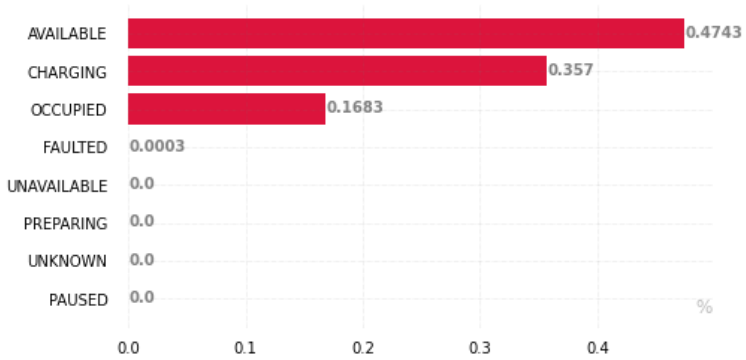
Table 7. Estonia Dataset Performance Results (%) by Model

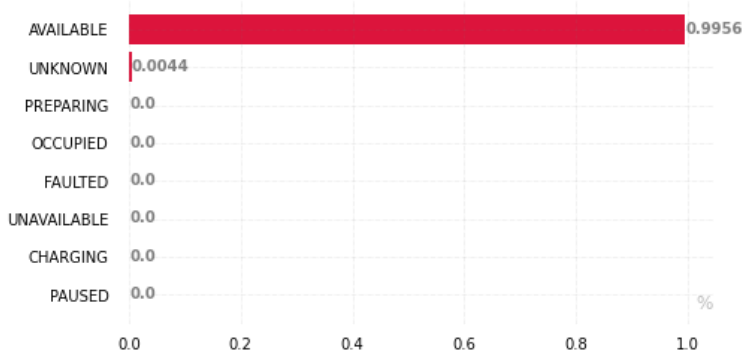
APPROACH	PRECISION	RECALL	F1-SCORE
K-NEAREST NEIGHBOURS	94.11	94.19	94.21
LOGISTIC REGRESSION	94.35	96.63	94.81
RANDOM FOREST	94.78	96.02	95.17
SUPPORT VECTOR MACHINE	95.15	96.32	95.75
ARTIFICIAL NEURAL NETWORK	96.04	95.74	95.89

Figure 14 shows the visualization of the first five prediction results from the ANN prediction engine for the Estonia dataset. The values correspond to percentages of status outputs. For example, the first predicted result can be interpreted as: “Charging station is 99.69% Available”.


AVAILABLE
UNKNOWN
PREPARING
OCCUPIED
CHARGING
FAULTED
UNAVAILABLE
PAUSED
0.9969
0.0027
0.0003
0.0001
0.0
0.0
0.0
0.0
%
0.0
0.2
0.4
0.6
0.8
1.0


CHARGING
AVAILABLE
FAULTED
UNAVAILABLE
OCCUPIED
PREPARING
UNKNOWN
PAUSED
0.7046
0.2946
0.0008
0.0001
0.0
0.0
0.0
0.0
%
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7


AVAILABLE
CHARGING
OCCUPIED
FAULTED
UNAVAILABLE
PREPARING
UNKNOWN
PAUSED
0.4743
0.357
0.1683
0.0003
0.0
0.0
0.0
0.0
%
0.0
0.1
0.2
0.3
0.4


AVAILABLE
UNKNOWN
PREPARING
OCCUPIED
FAULTED
UNAVAILABLE
CHARGING
PAUSED
0.9956
0.0044
0.0
0.0
0.0
0.0
0.0
0.0
%
0.0
0.2
0.4
0.6
0.8
1.0

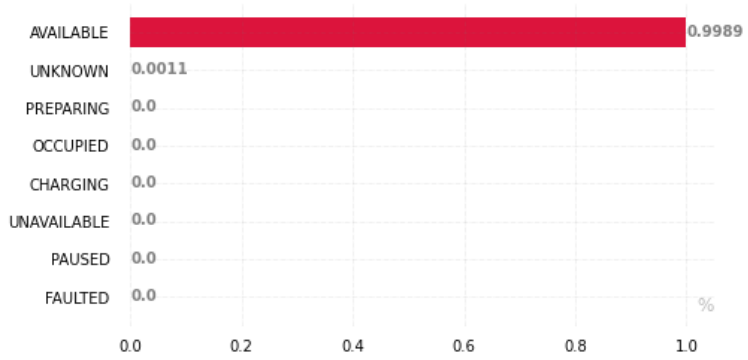


Figure 14. ANN model prediction results for the Estonia dataset.

As can be seen from the results, the results obtained in this thesis work outperform existing literature reviews for all the evaluation metrics: precision, recall and F1 score.

Models have different evaluation scores for Paris and Estonia datasets because those have various features. However, the Estonia dataset has higher scores in general because it can be considered a richer dataset with many additional features i.e., price per kWh, Socket Types, etc.

Although the ANN model has slightly lower scores than RF and KNN in evaluation metrics for Paris dataset, it can perform predictions in real-time, on the contrary of KNN's lazy-learning approach, and it allows results to be provided as a percentage for each output class, which is a highly preferred feature. Moreover, the ANN model is likely to perform better with future improvements, when more records or more complex features are in the dataset.

5 Summary

This thesis paper started with discussing literature review and state-of-the-artwork regarding availability prediction or related topics of EV and its surrounding infrastructure altogether with their main contributions, weaknesses, and models, methods, dataset they used. In addition to that, prevalent and well-proven machine learning and deep learning algorithms and techniques, tools used in those state-of-the-art papers are examined comprehensively in the same chapter. With the extensive literature scan, a base and justification are created to implement real-time availability prediction of electric vehicle charging stations, which consists of a scalable prediction engine integrated into a distributed server-side software application. In the implementation part, data scraping from different sources, creation of datasets, feature engineering applied and the data model is thoroughly discussed. Later, applying baseline machine learning models to preprocessed data is explained, which were discussed in detail in related works chapter. Next, the building and training of the Artificial Neural Networks model, which corresponds to the actual prediction engine, is reported. The last part of the implementation chapter examines integration with distributed server-side software. Finally, in the previous chapter, the results are demonstrated visually, and commonly used metrics evaluates performances of different models. The desired outcome is achieved by a prediction engine built by an artificial neural network, as it can display probabilities of electric vehicle charging station availabilities, with high evaluation scores, outperforming existing works in the literature.

For future work, many possibilities exist regarding improving the work that has been performed in this thesis. For example, the data scraping and feature engineering parts were primarily performed manually. Further improvements are possible for automatizing these processes. For instance, features that constitute importance and are not as critical could be detected automatically, and differences could be logged when a specific feature is excluded. That would, in the end, improve the model's quality, accuracy and reliability even more. Moreover, the prediction engine is only implemented for Paris and Estonia, but it can be easily applied to different cities or countries in the future.

References

[1] IEA, “Global EV Outlook 2021,” IEA, Paris, 2021.

[2] Forbes, “Every Automaker’s EV Plans Through 2035 And Beyond,” [Online]. Available: https://www.forbes.com/wheels/news/automaker-ev-plans/.

[3] F. Schulz and J. Rode, “Public charging infrastructure and electric vehicles in Norway,” *Energy Policy,* vol. 160, 2022.

[4] M. Henlock, “Strong link between charging infrastructure,” 2019. [Online]. Available: https://www.nordicenergy.org/wp-content/uploads/2019/04/Charging-infrastructure-and-adoption-of-electric-vehicles_web.pdf.

[5] F. Soldan, E. Bionda, G. Mauri and S. Celacshi, “Short-term forecast of EV charging stations occupancy probability using big data streaming analysis.,” 2021.

[6] A. Sao, N. Tempelmeier and E. Demidova, “Deep Information Fusion for Electric Vehicle Charging Station Occupancy Forecasting,” 2021.

[7] F. Qiao and S. Lin, “Data-driven prediction of fine-grained EV charging behaviors in public charging stations: Poster,” *Proceedings of the Twelfth ACM International Conference on Future Energy Systems (e-Energy '21),* pp. 276-277, 2021.

[8] S. Shahriar, A. Al-Ali, A. H. Osman, D. Salam and N. Mais, “Prediction of EV Charging Behavior Using Machine Learning,” *IEEE Access,* vol. 9, pp. 111576-111586, 2021.

[9] Caltech, “Adaptive Charging Network Dataset,” [Online]. Available: https://ev.caltech.edu/dataset.

[10] Y. Zhao, Z. Wang, Z.-J. M. Shen and F. Sun, “Data-driven framework for large-scale prediction of charging energy in electric vehicles,” *Applied Energy,* vol. 282, 2021.

[11] A. Almaghrebi, F. Aljuheshi, M. Rafaie, K. James and M. Alahmad, “Data-Driven Charging Demand Prediction at Public Charging Stations Using Supervised Machine Learning Regression Methods,” *Energies,* vol. 13, 2020.

[12] Y. Kim and K. Sahm, “Forecasting Charging Demand of Electric Vehicles Using Time-Series Models,” *Energies,* vol. 14, 2021.

[13] IBM, “Machine Learning,” 15 July 2020. [Online]. Available: https://www.ibm.com/cloud/learn/machine-learning.

[14] Berkeley School of Information, “What Is Machine Learning?,” 26 June 2020. [Online]. Available: https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/.

[15] J. Nabi, “Machine Learning — Multiclass Classification with Imbalanced Dataset,” 22 December 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-multiclass-classification-with-imbalanced-data-set-29f6a177c1a.

[16] “Multi-class Classification - One-vs-All & One-vs-One,” [Online]. Available: https://wadhwatanya1234.medium.com/multi-class-classification-one-vs-all-one-vs-one-993dd23ae7ca.

[17] Wikipedia, “k-nearest neighbors algorithm,” [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[18] O. Harrison, “Machine Learning Basics with the K-Nearest Neighbors Algorithm,” 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[19] L.-Y. Hu, M.-W. Huang, K. Shih-Wen and T. Chih-Fong, “The distance function effect on k-nearest neighbor classification for medical datasets,” *SpringerPlus,* vol. 5, no. 1, p. 1304, 2016.

[20] A. Band, “How to find the optimal value of K in KNN?,” 23 May 2020. [Online]. Available: https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb.

[21] I. Paryudi, “What Affects K Value Selection In K-Nearest Neighbor,” *International Journal of Scientific & Technology Research,* vol. 8, no. 7, 2019.

[22] J. Brownlee, “Multinomial Logistic Regression With Python,” 1 January 2021. [Online]. Available: https://machinelearningmastery.com/multinomial-logistic-regression-with-python/.

[23] “Logistic Regression,” [Online]. Available: https://www.saedsayad.com/logistic_regression.htm.

[24] T. Yiu, “Understanding Random Forest,” 12 June 2019. [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

[25] N. Donges, “A Complete Guide to the Random Forest Algorithm,” 22 July 2021. [Online]. Available: https://builtin.com/data-science/random-forest-algorithm.

[26] S. E. R, “Understanding Random Forest,” 17 June 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/.

[27] J. Brownlee, “A Gentle Introduction to XGBoost for Applied Machine Learning,” 17 August 2016. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/.

[28] S. Ray, “Understanding Support Vector Machine(SVM) algorithm from examples (along with code),” [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

[29] R. Gandhi, “Support Vector Machine — Introduction to Machine Learning Algorithms,” 7 June 2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[30] “Support-vector machine,” [Online]. Available: https://en.wikipedia.org/wiki/Support-vector_machine.

[31] E. Kavlakoglu, “AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What’s the Difference?,” 7 May 2020. [Online]. Available: https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks.

[32] IBM, “Convolutional Neural Networks,” [Online]. Available: https://www.ibm.com/topics/convolutional-neural-networks.

[33] R. Yamashita, M. Nishio, K. Togashi and R. K. G. Do, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging,* no. 9, pp. 611-629, 2018.

[34] IBM, “Recurrent Neural Networks,” [Online]. Available: https://www.ibm.com/topics/recurrent-neural-networks.

[35] S. Amidi and A. Amidi, “Recurrent Neural Networks cheatsheet,” [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[36] “Neural network models (supervised),” [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[37] “Evaluation Metrics,” [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/evaluation-metrics.

[38] “Precision and recall,” [Online]. Available: https://en.wikipedia.org/wiki/Precision_and_recall.

[39] A. Kumar, “Accuracy, Precision, Recall & F1-Score – Python Examples,” 1 October 2021. [Online]. Available: https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/.

[40] “What Is Data Scraping and How Can You Use It?,” [Online]. Available: https://www.targetinternet.com/what-is-data-scraping-and-how-can-you-use-it/.

[41] “Requests: HTTP for Humans,” [Online]. Available: https://docs.python-requests.org/en/latest/.

[42] “csv — CSV File Reading and Writing,” [Online]. Available: https://docs.python.org/3/library/csv.html.

[43] “Belib' - Charging points for electric vehicles - Real-time availability,” [Online]. Available: https://opendata.paris.fr/explore/dataset/belib-points-de-recharge-pour-vehicules-electriques-disponibilite-temps-reel/information/?disjunctive.statut_pdc&disjunctive.postal_code&disjunctive.arrondissement.

[44] “Belib,” [Online]. Available: https://belib.paris/home.

[45] “Open Data Commons Legal Tools For Open Data,” [Online]. Available: https://opendatacommons.org/licenses/odbl/.

[46] “Belib API,” [Online]. Available: https://opendata.paris.fr/explore/dataset/belib-points-de-recharge-pour-vehicules-electriques-disponibilite-temps-reel/api/?disjunctive.statut_pdc&disjunctive.arrondissement.

[47] [Online]. Available: https://enefitvolt.com/en/elektriauto-avalik-laadimine.

[48] “Feature Engineering,” [Online]. Available: https://www.omnisci.com/technical-glossary/feature-engineering.

[49] “About pandas,” [Online]. Available: https://pandas.pydata.org/about/.

[50] L. Buitinck et al., “API design for machine learning software: experiences from the scikit-learn project,” 2013.

[51] “pandas.get_dummies,” [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html.

[52] [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[53] [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[54] [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html.

[55] “About Keras,” [Online]. Available: https://keras.io/about/.

[56] Google, “Multi-Class Neural Networks: Softmax,” [Online]. Available: https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax.

[57] “matplotlib,” [Online]. Available: https://github.com/matplotlib/matplotlib.

[58] “Documentation,” [Online]. Available: https://kafka.apache.org/documentation/.

[59] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way,” 15 December 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Fatih Intekin

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Real-Time Availability Prediction of Electric Vehicle Charging Spots” supervised by Sadok Ben Yahia and Wissem Inoubli
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

03.01.2022

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Data Scraper Code for Paris Belib Dataset

```
import requests
import json
import csv

base_api_url = "https://opendata.paris.fr/api/records/1.0/search/"
url_query = "?dataset=belib-points-de-recharge-pour-vehicules-electriques-
disponibilite-temps-reel&q=&facet=statut_pdc&rows=2000"
url = base_api_url + url_query

translation = {
    233: "e",
    235: "e",
    231: "c",
    201: "E",
    232: "e",
    239: "i",
    226: "a"
}

if __name__ == "__main__":
    response = requests.get(url)
    json_data = json.loads(response.text)

    if "records" in json_data:
        records = json_data["records"]

        with open('output.csv', 'a', newline='') as f:
            writer = csv.writer(f)
            for record in records:
                fields = record["fields"]
                if "ad_station" not in fields:
                    continue
                writer.writerow(
                    [fields["id_pdc"],
                    fields["statut_pdc"].translate(translation),
                    fields["ad_station"].translate(translation),
                    fields["postal_code"],
                    record["geometry"]["coordinates"][1],
                    record["geometry"]["coordinates"][0],
                    fields["last_updated"],
                    record["record_timestamp"]
                ])
    else:
        print(response.text)
```

Appendix 3 – Data Scraper Code for Estonia Enefit Volt Dataset

```
import time
import requests
import csv
from datetime import datetime
from requests.structures import CaseInsensitiveDict

bounds_url = "https://account.enefitvolt.com/stationFacade/findSitesInBounds"
site_id_url =
"https://account.enefitvolt.com/stationFacade/findStationsBySiteId"
station_id_url =
"https://account.enefitvolt.com/stationFacade/findStationById"

headers = CaseInsensitiveDict()
headers["authority"] = "account.enefitvolt.com"
headers["sec-ch-ua"] = '"Chromium";v="94", "Google Chrome";v="94", ";Not A
Brand";v="99"'
headers["x-csrf-token"] = "f0394462-8ce4-4455-bb13-8da5d9e7f189"
headers["sec-ch-ua-mobile"] = "?0"
headers["user-agent"] = "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.71 Safari/537.36"
headers["content-type"] = "application/json"
headers["accept"] = "application/json, text/javascript, */*; q=0.01"
headers["x-requested-with"] = "XMLHttpRequest"
headers["sec-ch-ua-platform"] = '"Windows"'
headers["origin"] = "https://account.enefitvolt.com"
headers["sec-fetch-site"] = "same-origin"
headers["sec-fetch-mode"] = "cors"
headers["sec-fetch-dest"] = "empty"
headers["referer"] =
"https://account.enefitvolt.com/findCharger?59.7690375,24.5722210,6z"
headers["accept-language"] = "tr,en-US;q=0.9,en;q=0.8,et;q=0.7"
headers["cookie"] =
"_vwo_uuid_v2=D143809F5A14894938F82877C063F27A7|9b131316e37077900c1ba99e1cf24
6fe; _ga=GA1.2.1979870556.1632123935;
_vwo_uuid=D143809F5A14894938F82877C063F27A7;
_vwo_ds=3%3Aa_0%2Ct_0%3A0%241632123934%3A16.35813981%3A%3A%3A2_0%2C1_0%3A1;
_gcl_au=1.1.1498007280.1632123977; _vis_opt_exp_3_combi=1;
cusid=1638032678770; cuvon=1638032678774; _vis_opt_s=8%7C;
_vis_opt_test_cookie=1; _vwo_sn=5908742%3A1; _gid=GA1.2.264020397.1638032680;
_dc_gtm_UA-1116889-58=1; JSESSIONID=3A183C0428FD8664D5B2215C3AA8A14A"
bounds_data =
'{"filterByIsManaged":true,"filterByBounds":{"northEastLat":60.74995507745642
,"northEastLng":35.33882252391387,"southWestLat":55.78451290564244,"southWest
Lng":14.267045180163871}}'

resp = requests.post(bounds_url, headers=headers, data=bounds_data).json()
records = resp['data'][1]
```

```
with open('ee.csv', 'a', newline='', encoding='utf-8') as f:
    writer = csv.writer(f)
    for number, record in enumerate(records):
        site_id_data = '{"filterByIsManaged":true,"filterBySiteId":"' +
str(record['id']) + '"}'
        site_id_resp = requests.post(site_id_url, headers=headers,
data=site_id_data).json()
        stations = site_id_resp['data'][1]

        if number % 8 == 1:
            time.sleep(2)

        for station in stations:
            station_id = station['id']

            url = station_id_url + '?stationId=' + str(station_id)
            station_id_resp = requests.get(url, headers=headers).json()
            data = station_id_resp['data']

            id = data['id']
            name = data['siteDisplayName'].strip()
            address = data['addressAddress1'] + ', ' + data['addressCity']
            charging_speed = data['chargingSpeedId']
            in_maintenance = data['inMaintenance']
            lat = data['latitude']
            lon = data['longitude']
            access_level = data['siteStationAccessLevel']
            station_status = data['stationStatusId']
            socket_count = len(data['stationSockets'])

            socket_1 = { "name": "", "status": "", "type": "",
"charging_mode": "", "max_power": "", "kwh_price": "" }
            socket_2 = socket_1.copy()

            sockets = data['stationSockets']
            if sockets[0]:
                socket_1['name'] = sockets[0]['name']
                socket_1['status'] = sockets[0]['socketStatusId']
                socket_1['type'] =
sockets[0]['stationModelSocketSocketTypeId']
                socket_1['charging_mode'] =
sockets[0]['stationModelSocketChargingMode']
                socket_1['max_power'] =
sockets[0]['stationModelSocketMaximumPower']
                socket_1['kwh_price'] =
sockets[0]['socketPrices'][0]['kwhPrice']
            if sockets[1]:
                socket_2['name'] = sockets[1]['name']
                socket_2['status'] = sockets[1]['socketStatusId']
                socket_2['type'] =
sockets[1]['stationModelSocketSocketTypeId']
```

```
                socket_2['charging_mode'] =
sockets[1]['stationModelSocketChargingMode']
                socket_2['max_power'] =
sockets[1]['stationModelSocketMaximumPower']
                socket_2['kwh_price'] =
sockets[1]['socketPrices'][0]['kwhPrice']

            writer.writerow(
                [id, name, station_status, address, charging_speed,
in_maintenance, lat, lon, access_level, socket_count,
                socket_1['name'], socket_1['status'], socket_1['type'],
socket_1['charging_mode'], socket_1['max_power'], socket_1['kwh_price'],
                socket_2['name'], socket_2['status'], socket_2['type'],
socket_2['charging_mode'], socket_2['max_power'], socket_2['kwh_price'],
                datetime.now()]
            )
```

Appendix 4 – Data Pre-processing Code for Paris Dataset

```
import pandas as pd

# %%
# Read the raw data
data = pd.read_csv("./paris.csv")
data.head(10)

# Datetime conversions
data["Date"] = pd.to_datetime(data["Last Updated"])
data["Date"] = data["Date"].dt.strftime("%Y-%m-%dT%H:%M:%S")
data['Year'] = pd.DatetimeIndex(data['Date']).year
data['Month'] = pd.DatetimeIndex(data['Date']).month
data['Day'] = pd.DatetimeIndex(data['Date']).day
data['Hour'] = pd.DatetimeIndex(data['Date']).hour
data['Minute'] = pd.DatetimeIndex(data['Date']).minute
data['Second'] = pd.DatetimeIndex(data['Date']).second

# Drop redundant columns
df = data.drop(labels=["Date", "ID", "Last Updated", "Record Timestamp",
"Postal Code", "Address"], axis=1)
df.info()

# Geolocation normalization
scaler = MinMaxScaler(feature_range=(0, 1))
cols = ["Latitude", "Longitude"]
df[cols] = scaler.fit_transform(df[cols])

df = pd.get_dummies(df, columns=["Year", "Month", "Day", "Hour", "Minute",
"Second"])
df.head()

df.to_csv('./paris_processed.csv', index=False)
```

Appendix 5 – Data Pre-processing Code for Estonia Dataset

```
import pandas as pd
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler

data = pd.read_csv("./ee.csv")

# Datetime conversions
data["Date"] = pd.to_datetime(data["Recorded Timestamp"])
data["Date"] = data["Date"].dt.strftime("%Y-%m-%d %H:%M:%S")
data['Year'] = pd.DatetimeIndex(data['Date']).year
data['Month'] = pd.DatetimeIndex(data['Date']).month
data['Day'] = pd.DatetimeIndex(data['Date']).day
data['Hour'] = pd.DatetimeIndex(data['Date']).hour
data['Minute'] = pd.DatetimeIndex(data['Date']).minute
data['Second'] = pd.DatetimeIndex(data['Date']).second

# Drop redundant columns
df = data.drop(labels=["ID", "Name",  "Address", "In Maintenance",
                       "Socket Count", "Socket 1 Status", "Socket 1 Name",
"Socket 2 Status", "Socket 2 Name",
                       "Date", "Recorded Timestamp"], axis=1)

# Geolocation normalization
df["Latitude"] = df["Latitude"].apply(lambda x: x/100)
df["Longitute"] = df["Longitute"].apply(lambda x: x/100)
df.head()

le = preprocessing.LabelEncoder()
df["Charging Speed"] = le.fit_transform(df["Charging Speed"])
df["Access Level"] = le.fit_transform(df["Access Level"])
df["Socket 1 Type"] = le.fit_transform(df["Socket 1 Type"])
df["Socket 1 Charging Mode"] = le.fit_transform(df["Socket 1 Charging Mode"])
df["Socket 2 Type"] = le.fit_transform(df["Socket 2 Type"])
df["Socket 2 Charging Mode"] = le.fit_transform(df["Socket 2 Charging Mode"])

scaler = MinMaxScaler(feature_range=(0, 1))
cols = [
    "Socket 1 Max Power (kWh)", "Socket 2 Max Power (kWh)",
    "Socket 1 kWh Price", "Socket 2 kWh Price"]
df[cols] = scaler.fit_transform(df[cols])

df = pd.get_dummies(df, columns=["Year", "Month", "Day", "Hour", "Minute",
"Second"])
df.to_csv('./ee_processed.csv', index=False)
```

Appendix 6 – Baseline Models Code

```
from sklearn import preprocessing, metrics, svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Read the processed data
# df = pd.read_csv('./paris_processed.csv')
df = pd.read_csv('./ee_processed.csv')

# Label encoding and creating of output
le = preprocessing.LabelEncoder()
Y = le.fit_transform(df["Status"])

# Input columns
X = df.drop(["Status"], axis=1).values

# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3) #
70% training and 30% test

# K-NEAREST NEIGHBOUR
knn = KNeighborsClassifier(n_neighbors=1, n_jobs=-1)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Precision:", metrics.precision_score(y_test, y_pred,
average='weighted'))
print("Recall:", metrics.recall_score(y_test, y_pred, average='weighted'))
print("F1 Score:", metrics.f1_score(y_test, y_pred, average='weighted'))

# LOGISTIC REGRESSION
logisticRegr = LogisticRegression(solver='lbfgs', max_iter=15000)
logisticRegr.fit(X_train, y_train)
y_pred = logisticRegr.predict(X_test)
print("Precision:", metrics.precision_score(y_test, y_pred,
average='weighted', labels=np.unique(y_pred)))
print("Recall:", metrics.recall_score(y_test, y_pred, average='weighted',
labels=np.unique(y_pred)))
print("F1 Score:", metrics.f1_score(y_test, y_pred, average='weighted',
labels=np.unique(y_pred)))

#  RANDOM FOREST
clf = RandomForestClassifier(n_estimators=500, verbose=10)
clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)
```

```
print("Precision:", metrics.precision_score(y_test, y_pred,
average='weighted', labels=np.unique(y_pred)))
print("Recall:", metrics.recall_score(y_test, y_pred, average='weighted',
labels=np.unique(y_pred)))
print("F1 Score:", metrics.f1_score(y_test, y_pred, average='weighted',
labels=np.unique(y_pred)))

# SUPPORT-VECTOR MACHINE
lsvc = LinearSVC(verbose=1, max_iter=15000)
lsvc.fit(X_train, y_train)
y_pred=lsvc.predict(X_test)
print("Precision:", metrics.precision_score(y_test, y_pred,
average='weighted', labels=np.unique(y_pred)))
print("Recall:", metrics.recall_score(y_test, y_pred, average='weighted',
labels=np.unique(y_pred)))
print("F1 Score:", metrics.f1_score(y_test, y_pred, average='weighted',
labels=np.unique(y_pred)))
```

Appendix 7 – ANN Model Code

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from keras import backend as K

import matplotlib.pyplot as plt
import tensorflow as tf

# Read the data
df = pd.read_csv('./ee_processed.csv')
df.head()

# Label encoding the output value
encoder = LabelEncoder()
encoded_Y = encoder.fit_transform(df["Status"])
dummy_Y = np_utils.to_categorical(encoded_Y)

# Splitting training and test data
X = df.drop(["Status"], axis=1).values
X_train, X_test, y_train, y_test = train_test_split(X, dummy_Y,
stratify=dummy_Y)

# Recall metric function
def recall_score(y_actual, y_predicted):
    pp = B.sum(B.round(B.clip(y_actual, 0, 1)))
    tp = B.sum(B.round(B.clip(y_actual * y_predicted, 0, 1)))
    return tp / (pp + B.epsilon())

# Precision metric function
def precision_score(y_actual, y_predicted):
    prep = B.sum(B.round(B.clip(y_predicted, 0, 1)))
    tp = B.sum(B.round(B.clip(y_actual * y_predicted, 0, 1)))
    return tp / (prep + B.epsilon())

# F1 score metric function
def f1_score(y_actual, y_predicted):
    recall = recall_score(y_actual, y_predicted)
    precision = precision_score(y_actual, y_predicted)
    return ((precision * recall) / (precision + recall + B.epsilon())) * 2


# Build and compile the model
cce = tf.keras.losses.CategoricalCrossentropy(from_logits=False)
sgd = tf.keras.optimizers.SGD(learning_rate=0.3)
```

```
model = Sequential()
model.add(Dense(256, input_dim=26, activation='relu'))
model.add(Dense(8, activation='softmax'))
model.compile(loss="categorical_crossentropy",
                                optimizer=sgd,
                                metrics=[f1_m,
                                tf.keras.metrics.Precision(),
                                tf.keras.metrics.Recall()])
model.summary()

# Fitting and evaluating the model
model.fit(X_train,y_train,verbose=1,batch_size=10, epochs=20)
model.evaluate(X_test,y_test,verbose=1,batch_size=4000)
y_pred=model.predict(X_test)

# Status dictionary
status_dict={}
status_dict[0]='AVAILABLE'
status_dict[1]='UNKNOWN'
status_dict[2]='OCCUPIED'
status_dict[3]='CHARGING'
status_dict[4]='PAUSED'
status_dict[5]='FAULTED'
status_dict[6]='UNAVAILABLE'
status_dict[7]='PREPARING'

# Visualization of the results
def
percentageOfPredictedClasses(list_,describe_by_text=False,figure=True,limit=5
):
    list_=list_[:limit]
    for rw in list_:
        tempo_dict={}
        for k in range(len(rw)):
            tempo_dict[rw[k]]=k

        rw=np.sort(rw)[::-1]
        x_temp,y_tempo=[],[]
        for k in range(len(rw)):
            x_temp.append(status_dict[tempo_dict[rw[k]]])
            y_tempo.append(rw[k])
            if describe_by_text:
                print(status_dict[tempo_dict[rw[k]]],':',rw[k],'%')

        if figure:
            fig, ax = plt.subplots(figsize=(7,4))

            # Horizontal Bar Plot
            ax.barh(x_temp,y_tempo, color='crimson')

            # Remove axes splines
```

```
            for s in ['top','bottom','left','right']:
                ax.spines[s].set_visible(False)

            # Remove x,y Ticks
            ax.xaxis.set_ticks_position('none')
            ax.yaxis.set_ticks_position('none')

            # Add padding between axes and labels
            ax.xaxis.set_tick_params(pad=5)
            ax.yaxis.set_tick_params(pad=10)

            # # Add x,y gridlines
            ax.grid(b=True, color='grey', linestyle='-.', linewidth=0.5,
alpha=0.2)

            # Show top values
            ax.invert_yaxis()

            # Add annotation to bars
            for i in ax.patches:
                ax.text(i.get_width(), i.get_y()+0.5,
str(round((i.get_width()), 4)),
                        fontsize=10, fontweight='bold', color='grey')

            # Add Text watermark
            fig.text(0.9, 0.15, '%', fontsize=12, color='grey', ha='right',
va='bottom', alpha=0.5)

            # # Show Plot
            plt.show()

        print('#.....................................#')

percentageOfPredictedClasses(y_pred,describe_by_text=False,figure=True,limit=
20)
```