


## Article

# Performance Evaluation of Machine Learning and Neural Network-Based Algorithms for Predicting Segment Availability in AIoT-Based Smart Parking

Issa Dia <sup>1</sup>, Ehsan Ahvar <sup>1</sup> and Gyu Myoung Lee <sup>2,\*</sup> <sup>1</sup> Learning, Data and Robotics Laboratory, ESIEA Graduate Engineering School, 75005 Paris, France; dia@et.esiea.fr (I.D.); ehsan.ahvar@esiea.fr (E.A.)<sup>2</sup> School of Computer Science and Mathematics, Liverpool John Moores University, Liverpool L3 3AF, UK

\* Correspondence: g.m.lee@ljmu.ac.uk

**Abstract:** Finding an available parking place has been considered a challenge for drivers in large-size smart cities. In a smart parking application, Artificial Intelligence of Things (AIoT) can help drivers to save searching time and automotive fuel by predicting short-term parking place availability. However, performance of various Machine Learning and Neural Network-based (MLNN) algorithms for predicting parking segment availability can be different. To find the most suitable MLNN algorithm for the above mentioned application, this paper evaluates performance of a set of well-known MLNN algorithms as well as different combinations of them (i.e., known as Ensemble Learning or Voting Classifier) based on a real parking datasets. The datasets contain around five millions records of the measured parking availability in San Francisco. For evaluation, in addition to the cross validation scores, we consider resource requirements, simplicity and execution time (i.e., including both training and testing times) of algorithms. Results show that while some ensemble learning algorithms provide the best performance in aspect of validation score, they consume a noticeable amount of computing and time resources. On the other hand, a simple Decision Tree (DT) algorithm provides a much faster execution time than ensemble learning algorithms, while its performance is still acceptable (e.g., DT's accuracy is less than 1% lower than the best ensemble algorithm). We finally propose and simulate a recommendation system using the DT algorithm. We have found that around 77% of drivers can not find a free spot in their selected destinations (i.e., street or segment) and estimated that the recommendation system, by introducing alternative closest vacant locations to destinations, can save, in total, 3500 min drivers searching time for 1000 parking spot requests. It can also help to reduce the traffic and save a noticeable amount of automotive fuel.

**Keywords:** machine learning; deep learning; Internet of Things; Artificial Intelligence of Things; smart parking; smart city; neural network; performance evaluation



**Citation:** Dia, I.; Ahvar, E.; Lee, G.M. Performance Evaluation of Machine Learning and Neural Network-Based Algorithms for Predicting Segment Availability in AIoT-Based Smart Parking. *Network* **2022**, *2*, 225–238. <https://doi.org/10.3390/network2020015>

Academic Editor: Amitava Datta

Received: 21 December 2021

Accepted: 22 March 2022

Published: 8 April 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Finding an available on-street parking place has been considered as an old and noticeable challenge in large size cities. Drivers looking for vacant parking places can negatively impact traffic conditions and the environment (e.g., pollution and automotive fuel consumption) [1,2]. Studies of cruising in several congested downtowns between 1927 and 2001 showed that the average time to find a curb space ranged between 3.5 and 14 min and also between 8 and 74 percent of the traffic was cruising for parking [3]. Results of a recent study [4] also indicates that between 9 and 56 percent of the traffic is cruising for parking and the average search time is around 6.03 min.

The combination of Artificial Intelligence (AI) technologies with the Internet of Things (IoT) infrastructure refers to Artificial Intelligence of Things (AIoT). AIoT can achieve more efficient and autonomous IoT operations, improve human-machine interactions and enhance data management and analytics [5]. In a smart parking, AIoT can help drivers to save searching time and automotive fuel by predicting short-term parking place availability.

In this work, we consider parking segment prediction rather than an individual on-street parking place. Where a street (or part of that) with the possibility of car park is considered as a segment. We decided to consider the parking segment prediction because predicting situation (i.e., vacant or occupied) of an individual on-street parking place is technically very difficult, inaccurate and even not possible. In addition, the drivers are usually willing to know parking availability of segment rather than an individual parking place on a street.

Machine Learning and Neural Network-based (MLNN) algorithms can be applied to large datasets of different applications in order to extract relevant information and for making predictions. However, the results of the algorithms may vary depending upon the datasets and application being used. Therefore, determining the most suitable algorithm for specific datasets and application can be considered as a key advantage.

In order to find the the most suitable MLNN algorithm for predicting parking segment availability, we compare several well-known MLNN algorithms (e.g., K-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF), and Voting Classifier) based on a real parking datasets. The datasets that we use contain around five millions records of the measured parking availability in San Francisco.

We implement the algorithms using Scikit-Learn and Pytorch and compare them considering the well-known evaluation metrics: Precision, Recall, F1-Score, and Accuracy.

In addition, we propose a recommendation system including an algorithm to predict parking segment availability (i.e., utilizing the selected MLNN model) in a desirable time interval (e.g., 30 min later) and, in case of unavailability, suggest alternative segments.

To our knowledge, there are quite a few works (see Related Work section) which tried to evaluate performance of different MLNN algorithms for predicting car park place availability. Different from these works, this paper evaluates a wider range of MLNN algorithms including a large number of various combinations of them (i.e., more than 50 combinations). We also consider some additional metrics that previous works have not considered such as execution time of the algorithms and the walk forward validation technique (instead of k fold cross validation) as well as the three-level parking availability technique to improve performance of our recommendation system.

Our main contributions are listed below with respect to the main objective of predicting the availability of parking spaces:

- performance evaluation of various MLNN algorithms (e.g., KNN, RF and DT);
- performance evaluation of the Ensemble Learning approaches (i.e., with various configurations)
- comparing performance of different MLNN and Ensemble Learning to find the best one for the problem of predicting parking segment availability;
- introducing and simulating our recommendation system to offer the most suitable parking segment to drivers based on their requested destination and prediction time.

This paper is organized as follows. Section 2 introduces the related work. In Section 3, we describe our use case and the recommendation system. Section 4 briefly presents the MLNN algorithms under study in this paper. In Section 5, we implement and evaluate the performance of different MLNN algorithms and their combination (i.e., Ensemble Learning) as well as the proposed recommendation system. In Section 6, we will have a discussion and, finally, Section 7 concludes our work.

## 2. Related Work

A large number of studies have been carried out on the solution for parking space problems. Some works (e.g., [1,6–9]) have recently studied and analyzed existing smart parking solutions and provided comprehensive insights into the building of smart parking solution.

One of the solutions to help drivers to save searching time and automotive fuel is predicting the availability of parking places at a particular time based on AI methods. In this case, the data generated by sensors and AIoT devices are proceed by MLNN approaches (i.e., considered as an AIoT system).

Several AI-based studies have been conducted in the literature to find solutions for parking availability problem. Canli et al. [10] have recently proposed a deep learning and cloud-based mobile smart parking method for minimizing the problem of searching for parking places. Ali et al. [11] introduced a framework based on a deep long short term memory network to predict the availability of parking space with the integration of IoT and cloud computing. To this end, they utilized the Birmingham parking sensors dataset. Tekouabou et al. [12] proposed a system that integrates the IoT and a predictive model based on ensemble methods for predicting the availability of parking spots in smart parking. Arjona et al. [13] studied and developed two Recurrent Neural Network (RNN) architectures (i.e., Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU)) for predicting parking availability in urban areas. In order to improve the quality of the models, exogenous variables like hourly weather and calendar effects have been considered. Sonny et al. [14] proposed a method for detecting the occupancy status of an outdoor parking space using Long Term Evolution (LTE)-based Channel State Information and Convolutional Neural Network (CNN). Piccialli et al. [15] presented a Deep Learning-based ensemble technique to predict the parking space occupancy. A genetic algorithm has also been utilized to optimize predictors parameters. The proposed system has been evaluated on a real IoT dataset including more than 15M of collected sensor records. Rahman et al. [16] modified the architecture of CNN to classify parking spaces and increase the work efficiency of the smart parking system in processing parking availability information.

Considering various MLNN algorithms, one technical challenge is to detect the most suitable MLNN model for predicting parking place availability. Because, depending on applications, the performance of each MLNN model can be different.

Chen et al. [17] analyzed and compared four classifiers RF, Support Vector Machines (SVMs), KNN, and Linear Discriminant Analysis (LDA) as well as combination of these them with different features selection methods. To this end, they used three popular datasets. However, this work is not specialized for the smart parking application.

To our knowledge, there are quite a few works which tried to analyze and compare the performance of different MLNN algorithms for predicting car park place availability.

Awan et al. [18] evaluated the performance of several MLNN algorithms using the Santander's parking dataset. In this work, we evaluate the performance of some other MLNN algorithms such as LSTM, Single Layer Perceptron (SLP) and Categorical Naive Bayes (CNB) using a different dataset (i.e., the San Francisco dataset). We also considered some additional metrics such as execution time of the algorithms. In addition, unlike [18] which considered only one voting classifier (Ensemble Learning algorithm), we implement and evaluate a large number of various combinations of different MLNN algorithms (i.e., more than 50 combinations) to offer a comprehensive analysis. Unlike [18], we use the walk forward validation. It provides better performance than k fold cross validation for time series data analysis [19].

In order to get a more accurate result, unlike [18] which considered two availability situations (i.e., free and occupied), we define three parking availability levels to provide a priority for the parking segments with higher free places.

### 3. Use Case and Recommendation System

We consider a smart city including several streets. Here, a street with the possibility of car park is considered as a segment. In order to increase performance, very long streets are divided into several segments.

The drivers have a list of streets and the number of available segments in every street (i.e., for a long street, the segments are numbered starting from beginning of the street).

In order to get a recommendation, a driver first sends a request including his destination location (i.e., street name and segment number) and a time period that he intends to be there. When the system receives the request, it applies a MLNN algorithm to the historical data of the requested parking segment to predict the number of free parking places for the requested time interval.

Our recommendation system considers three levels of parking place availability called L1 (zero or very low available places), L2 (several available places) and L3 (high availability) using two thresholds (i.e., T1 and T2) as follows:

- less than or equal to T1 free places is considered as level L1;
- between T1 and T2 free places is considered as level L2;
- equal or above T2 free places is considered as level L3.

As Algorithm 1 shows, if the level of the selected parking segment by a driver is predicted as L3 for the requested time interval, the recommendation system recommends the selected segment to the driver.

If the level of the selected parking segment by a driver is predicted as L2, the system defines a circle area with center of the requested segment and radius of X (e.g., X = 500) meters. It only considers and evaluates the historical data of those parking segments that are located in the circle area. In this case, the system separately trains the best model on the data sets of each parking segment in the circle area. Finally, among the eligible parking segments (i.e., the segments in the circle area which their availability levels are L3 for the requested time interval), the closest parking segment to the requested destination is recommended to the driver. However, if the recommendation system can not find a segment with level of L3 in the circle, it offers the selected segment itself to the driver.

If the level of the selected parking segment by driver is predicted as L1, the closest L3 level parking segment to the requested destination is recommended to the driver. If there is no segment with level of L3, but there are one or more segments with L2, it suggests the closest segment with L2 to the requested segment to driver. Otherwise, the system sends a warning message to the driver.

---

**Algorithm 1:** Segment-Selector (*SegId*, *timeInterval*)

---

```

foreach request do
  run MLNN algorithm for SegId
  if (SegId.availability == L3) then
    return (SegId)
  if (SegId.availability == L2) then
    run MLNN algorithm for the related circle
    if there are some L3 availability in the circle then
      return(closest L3 to SegId)
    else
      return SegId
  if (SegId.availability == L1) then
    run MLNN algorithm for the related circle
    if there are some L3 availability in the circle then
      return(closest L3 to SegId)
    else
      if there are some L2 availability in the circle then
        return(closest to SegId)
      else
        return 0

```

---

#### 4. A Review on Machine Learning and Neural Network Algorithms

This section briefly introduces concept of Machine learning and Neural network, and then presents the MLNN algorithms that we study and analyze in this paper.

Primary AI research considered hard-coded statements in formal languages, which a computer according to logical inference rules can then automatically reason about (i.e., knowledge base approach). This approach has some limitations as humans generally struggle to explicate all their tacit knowledge that is needed to do complex tasks. Machine learning is utilized to overcome these limitations. It can help to make decisions by learning

from previous computations and extracting regularities from large-size databases. As such, it tries to automate the task of analytical model building to do cognitive tasks such as detecting objects. It is conducted by applying MLNN algorithms that learn from problem-specific training data, which lets computers to detect hidden insights and complex patterns without explicitly being programmed. Depending on the learning task, the field offers different types of MLNN algorithms, with each algorithm coming in multiple specifications and variants, including regressions models, instance-based algorithms, decision trees, Bayesian methods, and Neural networks [20].

In the following, we present the MLNN algorithms that we study and analyze in this paper.

#### 4.1. *K-Nearest Neighbors (KNN)*

The KNN algorithm is considered as one of the simplest machine learning algorithms. It is a non-generalizing learning or lazy learning algorithm. KNN keeps all instances corresponding to training data in  $n$ -dimensional space. In order to classify a new data, it considers a simple majority vote of the  $k$  nearest neighbors of each point. In other words, every new data point is classified based on the saved data and similarity measures such as Euclidean distance function. One of the biggest challenges with KNN is to find the optimal number of neighbors. This algorithm can be utilized for both classification and regression [21].

#### 4.2. *Decision Tree (DT) and Random Forest (RF)*

DT is considered as a well-known non-parametric supervised learning method which is used for both classification and regression tasks. It constructs a tree by setting different conditions on its branches. Iterative Dichotomiser 3 (ID3), C4.5, and Classification and Regression Trees (CART) are well known for DT algorithms [18,21].

RF classifier is considered as an ensemble classification technique [21]. It has been proposed in 2001 by Breiman. RF is considered as a collaborative method which works based on the proximity search. For improving the performance, RF makes use of a standard divide and conquer approach [22]. There are a lot of similarities between the DT and RF algorithms. In fact, the RF consists of multiple independent DTs where each tree sets conditional features differently. Once a sample arrives at a root node, it is transferred to all the sub-trees where every sub-tree predicts the class label for the sample. Finally, the class in the majority is assigned to that sample [18].

#### 4.3. *Support Vector Machines (SVMs)*

For classification and clustering objectives, SVMs map data points to high dimensional vectors. If we have data points in a  $n$ -dimensional space, a  $(n-1)$ -dimensional hyperplane can be utilized as a classifier [23].

#### 4.4. *Categorical Naïve Bayes (CNB)*

CNB is a relatively simple probabilistic algorithm based on the Bayes theorem. It naively assumes that the features are independent. They can be trained rather quickly using supervised learning, but are usually less accurate than more complicated approaches [23,24].

#### 4.5. *Single Layer Perceptron (SLP)*

SLP, or simply Perceptron, is a binary classifier. It assigns a weight to every input of a Perceptron and then sums over the products of the weights and their inputs. The result is compared to a threshold in order to determine the (output) label [23].

#### 4.6. *Multilayer Perceptron (MLP)*

MLP is a Perceptron-based system that includes multiple layers: an input layer, one or more hidden layers and an output layer. By increasing the number of hidden layers, the complexity of the model increases. The MLPs can be very powerful and complex [23,25].



#### 4.7. Long Short-Term Memory (LSTM)

LSTM is considered as an artificial recurrent neural network (RNN) architecture. It has been proposed to solve the vanishing and exploding gradient problems of conventional RNNs. LSTMs originally include special units called memory blocks in the recurrent hidden layer. Each memory block has a memory cell with self-connections storing the temporal state of the network and special multiplicative units (i.e., gates). An input gate is used to control the flow of input activations into the memory cell and an output gate to control the output flow of cell activations into the rest of the network [26].

#### 4.8. Ensemble Learning (Voting Classifier)

An ensemble learning technique combines multiple learning models in order to make a better prediction than running the algorithms separately. It takes the training data and trains each model. In the next step, it feeds the testing data to the models where each model predicts a class label for each sample in the testing data. A voting process is then performed for each sample prediction. There are two ways for voting: hard voting and soft voting. The hard voting acts based on majority where it assigns a class label, voted by majority, to the sample. The soft voting approach averages the probability of all the expected outputs (i.e., the class labels) and, after that, the class with the highest probability is assigned to the sample [18].

### 5. Performance Evaluation

This section includes three main parts. We first introduce the utilized dataset and describe how we prepare and specialize it for our work. In the second part, we present technical details of our implementation including utilized tools, features, hyper-parameters, validation technique and evaluation metrics. In the third part, we first talk about our methods for implementing single algorithms, combining different algorithms (as ensemble learning) and measuring the execution time by introducing main utilized Python functions. We then define three main scenarios to evaluate single and ensemble learning algorithms as well as our proposed recommendation system.

#### 5.1. Dataset Description

We utilized San Francisco's dataset [27,28] which contains around five millions records of the measured parking availability covering a total of 420 parking segments in San Francisco. Parking availability observations were recorded at approximately 5-min intervals for each parking segment during 6 weeks.

Each line of the dataset keeps the situation of a parking segment at a specific timestamp. We consider the columns correspond to the following content:

- timestamp: timestamp reporting when the SFPark API was polled, rounded to the closest minute;
- segmentid: ID of the parking segment;
- capacity: total number of parking places in the segment (the capacity may vary over time, due to parking restrictions);
- occupied: current number of occupied places in the segment.

In addition, there is a complementary file where each line of that corresponds to a parking segment and its columns correspond to the following content:

- segmentid: ID of the parking segments;
- streetname: name of the related street;
- startx, starty, endx, endy: WGS84 coordinates of start and end point of parking segment.

##### 5.1.1. Data Cleaning

The obtained dataset has been already cleaned from implausible sensor values and the segments with issues have been excluded [28]. As a result, 420 segments from the original

579 road segments have been kept and the dataset contains a total of more than 5 million observations.

In addition, we verified that:

- the obtained dataset does not contain empty cells, duplicates or wrong format;
- Observations are regularly and continuously recorded every 5 min for each parking segment during the considered 6 weeks.

#### 5.1.2. Dataset Individualization

In order to provide prediction with 30 min validity, we recreated the dataset in a way that it contains availability observations at approximately 30-min intervals.

In addition, we separated the obtained dataset into 420 datasets where each new dataset contains availability observations at approximately 30-min intervals for only one individual parking segment. There are two following reasons behind this decision. First, based on the use case under study (see Section 3), we need to predict parking place availability for a requested segment by drivers individually. In addition, for the prediction of parking availability in one special segment, we found that utilizing the whole dataset (i.e., including information of all segments together which are usually unrelated) can make a negative effect on the trained models, and therefore the results of the prediction of parking availability for that special segment. As a result, this idea could help us to reduce complexity, processing time and improve performance.

### 5.2. Implementation Technical Details

#### 5.2.1. Tools and Software

The machine learning and ensemble algorithms have been implemented using Scikit-learn library [29] and the deep learning algorithm using PyTorch [30].

#### 5.2.2. Input Features

According to our dataset information and the use case requirements, we define the following input features for our MLNN model:

- segmentid: ID of the parking segment;
- observation time: including *hour* and *minute* features which keep the hour and minute for each observation;
- availability level: is calculated based on the number of free parking spots in each segment. In order to define our three availability levels, in this work, two T1 and T2 threshold values are considered equal to 3 and 10 respectively. We selected these values after doing several tests with different values. In this case:
  - less than or equal to 3 free places is considered as level L1.
  - between 3 and 10 free places is considered as level L2.
  - equal or above 10 free places is considered as level L3.
- streetname: name of the related street in the segment;
- startx, starty, endx, endy: WGS84 coordinates of start and end point of parking segment.

#### 5.2.3. Hyper-Parameters

In order to improve performance of the models, we found the most appropriate hyper-parameters using the Scikit-learn's RandomSearch Cross Validation [31]. Recall that CNB does not have hyper-parameters to tune. We considered around 16% of the dataset (i.e., one week out of 6 weeks) for tuning the hyper-parameters. Tables 1 and 2 show the utilized hyper-parameter values in this work.

**Table 1.** hyper-parameter values- Part I.

KNN		Decision Tree		Random Forest	
Parameter	Value	Parameter	Value	Parameter	Value
n_neighbors	12	max_depth	7	max_depth	29
metric	manhattan	criterion	gini	criterion	entropy
weights	distance	min_samples_leaf	1	min_samples_leaf	2
				n_estimators	160

**Table 2.** hyper-parameter values- Part II.

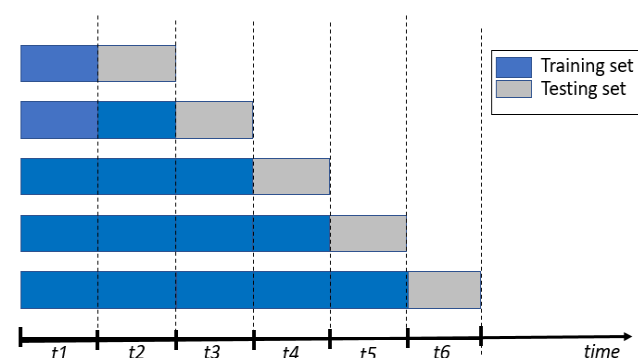
SVM		SLP		MLP		LSTM	
Par.	Val.	Par.	Val.	Par.	Val.	Par.	Val.
kernel	rbf	loss	perceptron	hidden_size	60	input_size	3
C	1	alpha	0.1	solver	adam	hidden_size	20
gamma	0.01	penalty	l2	activation	tanh	layer_dim	1
-	-	-	-	solver	SLP	-	-
-	-	-	-	learning_rate	invescaling	-	-

#### 5.2.4. Validation Technique

For the time series data, using k-fold cross validation does not work perfectly. This is because it does not consider the order of data [19].

In this work, we use a validation technique which is called walk-forward (i.e., a technique preserving the order of data).

As Figure 1 shows, the walk-forward technique divides dataset into several units (e.g., in our work, one week as a unit). After that, units are chronologically ordered, and in each run, all data available before the unit to predict is considered as the training set, and the unit to predict is used as test-set. Finally, the model accuracy is considered as the average among runs. The number of runs is equal to one less than the number of data units [19]. We considered around 84% of the dataset (i.e., five continues last weeks/units out of 6 weeks) for training and testing our models.

**Figure 1.** The walk-forward technique for 6 units.

#### 5.2.5. Evaluation Metrics

We evaluate the performance of different algorithms via Accuracy, Precision, Recall and F1 Score metrics. We also measured execution time of every algorithm (i.e., including both training and testing times).

#### 5.3. Implementation and Evaluation

For implementing the algorithms (except LSTM), we use the Scikit-learn built-in Machine Learning classifiers (e.g., DecisionTreeClassifier for DT algorithm). For ensemble



learning (i.e., combination of algorithms), we implement the hard voting technique considering the same weight for inputs (i.e., algorithms). As a result, we combine odd numbers of algorithms (i.e., 3 and 5 algorithms). To this end, we utilized the Scikit-learn voting classifier (i.e., VotingClassifier). We also use the random search function available in Scikit-learn to find appropriate values for hyper-parameters.

For calculating training/testing execution time of an algorithm, we consider the following approach. As already mentioned, we use the walk-forward technique where we need to execute every algorithm several times using different size of the dataset. We calculate an execution time by measuring the starting time and ending time of execution, and then consider the duration. To do this, we utilize the available “time” function. Finally, the execution time of an algorithm for training/testing is considered as the average of all execution times for training/testing of that algorithm.

In order to evaluate MLNN algorithms, we consider three parts. In Part.I, we implement and evaluate performance of the MLNN algorithms described in Section 4. Notice that we use the one-vs-one method for using binary classification algorithms such as SLP for our three-class classification. Part II evaluates the performance of different combinations of MLNN algorithms, known as ensemble learning algorithms, and finally Part III implements and analyzes the proposed recommendation system introduced in Section 3.

### 5.3.1. Part I-Algorithms

This part evaluates the performance of the well-known MLNN algorithms introduced in Section 4 (i.e., KNN, DT, RF, SVM, MLP, CNB, LSTM and SLP) for predicting parking segment availability in smart cities based on accuracy, precision, recall, F1-Score and execution time.

Table 3 presents the average cross-validation score of the algorithms given 30-min predictions.

We can see that the computationally complex neural networks-based models such as LSTM, MLP and even SLP show generally the lowest performance. The results of neural networks-based models are as expected. Because generally neural networks are poor at time series forecasting. Among neural networks-based models, MLP has the best performance. It can be seen that SVM’s accuracy is the closest to the accuracy of MLP. KNN and DT, as two simple methods, show promising performance. RF can outperform other algorithms. However, it is considered as an ensemble learning method.

**Table 3.** Average cross validation score of models.

Metrics	CNB	DT	KNN	MLP	RF	SLP	SVM	LSTM
Accuracy	77.97	78.96	78.20	75.74	79.15	64.84	75.37	53.56
Precision	80.13	81.09	80.61	77.54	80.94	69.53	78.65	85.41
Recall	77.97	78.96	78.20	75.74	79.15	64.84	75.37	53.56
f1_score	76.18	78.64	77.87	73.72	78.68	61.45	72.17	43.52

Table 4 compares execution time of different models (in s). Training phase for LSTM algorithm is very time consuming in comparison to other algorithms. We can see that simple algorithms such as DT and KNN have better training time than more complex ones. Considering both training and testing execution times together, DT can be considered as one of the quickest algorithms.

**Table 4.** Execution time of different models (in s).

Set	CNB	DT	KNN	MLP	RF	SLP	SVM	LSTM
Training	0.0020	0.0015	0.0014	0.5869	0.2150	0.0046	0.0294	86.5
Testing	0.0009	0.0009	0.0023	0.0016	0.0196	0.0009	0.0155	0.0006

### 5.3.2. Part II-Ensemble Learning algorithms

After analyzing and comparing the performance of MLNN algorithms which are introduced in Section 4, here, we evaluate performance of different combinations of some machine learning algorithms. The combination technique is known as Ensemble Learning or Voting Classifier. Ensemble learning combines two or more learning algorithms in order to get a better performance.

Table 5 shows results of different combination of our 7 algorithms in groups of 3 algorithms. In other words, we implemented 35 different combinations of the algorithms. As LSTM was implemented using Pytorch, due to technical limitation, we did not combine it with other algorithms. The results show that the ensemble learning methods including two neural network-based algorithms (i.e., CNB-SLP-MLP, DT-SLP-MLP, KNN-SLP-MLP, RF-SLP-MLP and SLP-SVM-MLP) have poor performance. However, as expected, a combination of a low-performance neural network algorithm with two high-performance algorithms (e.g., DT, RF and MLP) can give an acceptable performance. In general, it can be seen that there are several ensemble learning algorithms with an accuracy above 79%. Looking again at results of Part I (i.e., Table 4), we have only RF algorithm which its accuracy is above 79%.

According to Table 5, a combination of DT, RF and CNB can provide the best cross validation scores (e.g., with accuracy equals to 79.81%) than other combinations.

Regarding the execution time, we can see that the combinations which contain MLP consume more time than other combinations for the training phase. It is because training phase of MLP is time consuming.

**Table 5.** Ensemble Learning Algorithms (combinations of three algorithms).

VC	Accuracy	Precision	Recall	f1_Score	Training (s)	Testing (s)
CNB,SLP,MLP	76.40	78.55	76.40	74.20	0.5928	0.0056
CNB,SLP,SVM	76.21	78.68	76.21	73.53	0.0379	0.0197
CNB,SVM,MLP	76.65	79.05	76.65	74.11	7.3174	0.0206
DT,CNB,MLP	79.42	81.45	79.42	77.94	0.5881	0.0057
DT,CNB,SLP	78.72	80.68	78.72	77.34	0.0094	0.0049
DT,CNB,SVM	79.25	81.41	79.25	77.48	0.0343	0.0196
DT,RF,CNB	79.81	81.65	79.81	79.11	0.2205	0.0239
DT,RF,MLP	79.72	81.58	79.72	79.00	0.7966	0.0245
DT,RF,SLP	79.26	81.13	79.26	78.63	0.2182	0.0233
DT,RF,SVM	79.76	81.60	79.76	78.97	0.2416	0.0380
DT,SLP,MLP	76.84	78.84	76.84	75.06	0.5886	0.0056
DT,SLP,SVM	76.69	78.87	76.69	74.50	0.0373	0.0197
DT,SVM,MLP	76.92	79.12	76.92	74.69	0.6160	0.0206
KNN,CNB,MLP	79.34	81.44	79.34	77.85	0.5870	0.0069
KNN,CNB,SLP	78.28	80.36	78.28	76.88	0.0092	0.0063
KNN,CNB,SVM	79.15	81.41	79.15	77.36	0.0346	0.0212
KNN,DT,CNB	79.60	81.58	79.60	78.98	0.0059	0.0062
KNN,DT,MLP	79.39	81.45	79.39	78.70	0.8575	0.0070
KNN,DT,RF	79.18	81.12	79.18	78.81	0.2245	0.0256
KNN,DT,SLP	78.84	80.96	78.84	78.26	0.0088	0.0062
KNN,DT,SVM	79.44	81.48	79.44	78.68	0.0336	0.0209
KNN,RF,CNB	79.40	81.21	79.40	78.79	0.2171	0.0250
KNN,RF,MLP	79.42	81.18	79.42	78.81	0.7988	0.0259
KNN,RF,SLP	78.93	80.79	78.93	78.37	0.2176	0.0247
KNN,RF,SVM	79.39	81.13	79.39	78.71	0.2427	0.0398
KNN,SLP,MLP	76.58	78.70	76.58	74.81	0.5883	0.0068
KNN,SLP,SVM	76.37	78.53	76.37	74.14	0.0373	0.0211
KNN,SVM,MLP	76.90	79.09	76.90	74.66	4.2762	0.0222
RF,CNB,MLP	79.37	81.44	79.37	77.86	0.7913	0.0245
RF,CNB,SLP	78.57	80.65	78.57	77.09	0.2214	0.0236
RF,CNB,SVM	79.18	81.40	79.18	77.39	0.2439	0.0379

**Table 5.** *Cont.*

VC	Accuracy	Precision	Recall	f1_Score	Training (s)	Testing (s)
RF,SLP,MLP	76.79	78.72	76.79	75.02	5.8749	0.0244
RF,SLP,SVM	76.60	78.79	76.60	74.33	5.3220	0.0380
RF,SVM,MLP	76.92	79.05	76.92	74.65	0.8238	0.0392
SLP,SVM,MLP	75.32	77.49	75.32	72.67	0.6250	0.0207

We can see, in Table 6, results of different combination of the seven algorithms in groups of 5 algorithms. In general, the ensemble algorithms that use RF, DT and KNN together, they can provide a promising performance, especially in aspects of accuracy and recall (i.e., all above 79.5%). In particular, the results show that the combination of five KNN, DT, RF, CNB and SVM algorithms can provide the best cross validation scores (with 79.93% accuracy/recall, 81.79% precision and 79.10% f1\_score) in comparison to other combinations. The second rank is the combination of KNN, DT, RF, CNB and MLP (with 79.92% accuracy/recall, 81.76% precision and 79.14% f1\_score). However, because of using MLP, its training phase is time consuming.

Regarding the execution time, we can see that the combinations which contain SVM consume more time than other combinations for the training phase. It is because training of SVM is time consuming.

**Table 6.** Ensemble Learning Algorithms (combinations of five algorithms).

VC	Accuracy	Precision	Recall	f1_Score	Training (s)	Testing (s)
DT,CNB,SLP,SVM,MLP	77.48	79.70	77.48	75.26	0.6209	0.0227
DT,RF,CNB,SLP,MLP	79.37	81.38	79.37	78.04	0.8037	0.0264
DT,RF,CNB,SLP,SVM	79.31	81.33	79.31	77.79	0.2530	0.0399
DT,RF,CNB,SVM,MLP	79.50	81.59	79.50	77.92	0.8303	0.0414
DT,RF,SLP,SVM,MLP	77.75	79.87	77.75	75.86	2.1170	0.0416
KNN,CNB,SLP,SVM,MLP	77.45	79.74	77.45	75.23	0.6299	0.0236
KNN,DT,CNB,SLP,MLP	79.26	81.25	79.26	77.95	0.5945	0.0088
KNN,DT,CNB,SLP,SVM	79.19	81.28	79.19	77.67	0.0410	0.0229
KNN,DT,CNB,SVM,MLP	79.42	81.57	79.42	77.84	0.6210	0.0238
KNN,DT,RF,CNB,MLP	79.92	81.76	79.92	79.14	3.4872	0.0276
KNN,DT,RF,CNB,SLP	79.62	81.48	79.62	78.90	0.2269	0.0278
KNN,DT,RF,CNB,SVM	79.93	81.79	79.93	79.10	0.2534	0.0421
KNN,DT,RF,SLP,MLP	79.50	81.39	79.50	78.75	0.8036	0.0277
KNN,DT,RF,SLP,SVM	79.56	81.41	79.56	78.74	0.2505	0.0411
KNN,DT,RF,SVM,MLP	79.77	81.63	79.77	78.91	0.8329	0.0428
KNN,DT,SLP,SVM,MLP	77.64	79.68	77.64	75.76	0.6264	0.0239
KNN,RF,CNB,SLP,MLP	79.29	81.21	79.29	77.97	0.8033	0.0280
KNN,RF,CNB,SLP,SVM	79.16	81.07	79.16	77.72	0.2518	0.0412
KNN,RF,CNB,SVM,MLP	79.42	81.50	79.42	77.85	0.8425	0.0429
KNN,RF,SLP,SVM,MLP	77.62	79.63	77.62	75.77	2.9616	0.0426
RF,CNB,SLP,SVM,MLP	77.48	79.83	77.48	75.27	0.8413	0.0418

### 5.3.3. Part III-Recommendation System

The goal of this part is to evaluate the performance of the recommendation system introduced in Section 3. In order to get an accurate result, we implemented and run Algorithm 1 ten times, and then calculated an average result. Every time, 100 drivers sent their requests. For every request, the segment number and requested time interval were selected randomly. To have more accurate results, we consider only day time (i.e., from 8:00 to 18:00). Inspiring from results of Parts I and II of this section, we used the simple but effective DT algorithm in our recommendation system. The results show that 77% of drivers selected L1 or low availability segments. It means, without using the recommendation system, 77% of drivers can not find a free spot in their selected segment and, therefore, they have to search for an available place in other segments. If we assume that finding an

available place in another segment takes 5 min, for our 1000 requests, in total, the drivers should spend  $1000 \times 0.70 \times 5$  min to find a place. In this case, our recommendation system can save this time by predicting (with accuracy around 94%) selected L1 segments, and then suggesting a L3 or L2 segment.

## 6. Discussion

Here, we analyze and compare results of the Tables 3–6 together. From Table 3, we found that RF is the best algorithm (with 79.15% accuracy/recall, 80.94% precision and 78.68% f1\_score) and, after RF, DT is ranked second (with 78.96% accuracy/recall, 81.09% precision and 78.64% f1\_score). However, as Table 4 shows, training phase in RF (with 0.215 s) is more time consuming than DT (with 0.0015 s). In Table 5, we have already seen that the combination of DT, RF and CNB can provide 79.81% accuracy/recall, 81.65% precision and 79.11% f1\_score with 0.2205 s training time. And, in Table 6, we can see that the combination of five KNN, DT, RF, CNB and SVM algorithms can provide the best cross validation scores (with 79.93% accuracy/recall, 81.79% precision, 79.10% f1\_score and 0.2534 s training time) in comparison to all other ensemble algorithms.

Table 7 shows the top first single algorithm (i.e., RF), the top first ensemble learning with three algorithms (i.e., DT, RF and CNB) and the top first ensemble learning with three algorithms (i.e., KNN, DT, RF, CNB and SVM). We also put DT, as a simple algorithm, to enrich our discussion. As Table 7 shows, training time of the simple DT algorithm (i.e., 0.0015 s) is much more less than the computing-intensive ensemble algorithm with the combination of five KNN, DT, RF, CNB and SVM algorithms (i.e., 0.2534 s), while there is not a noticeable difference between performance of them (e.g., accuracy difference between them is less than 1%).

Considering the limited resources of AIoT end and even edge devices in smart parkings and also real-time nature of many AIoT applications, using simple algorithms such as DT looks more reasonable than the complex ensemble learning algorithms.

**Table 7.** Algorithms with the best performance in Part I and Part II.

Algorithm	Accuracy	Precision	Recall	f1_Score	Training (s)	Testing (s)
RF	79.15	80.94	79.15	78.68	0.2150	0.0196
DT,RF,CNB	79.81	81.65	79.81	79.11	0.2205	0.0239
KNN,DT,RF,CNB,SVM	79.93	81.79	79.93	79.10	0.2534	0.0421
DT	78.96	81.09	78.96	78.64	0.0015	0.0009

## 7. Conclusions

In a smart parking, AIoT can help drivers to save searching time and automotive fuel by predicting short-term parking place availability utilizing a MLNN algorithms. To find the most suitable MLNN algorithm, this paper evaluated performance of a set of well-known MLNN algorithms (i.e., KNN, DT, RF, SVM, MLP, LSTM and SLP) as well as more than 50 combinations of them (i.e., known as Ensemble Learning) based on a real parking datasets including around five millions records of the measured parking availability in San Francisco. For evaluation, we considered the cross validation scores as well as resource requirement, simplicity and execution time (i.e., including both training and testing times) of algorithms. We saw that some ensemble learning algorithms provide the best performance in aspect of validation score. However, they consume a noticeable amount of computing and time resources. On the other hand, simple algorithms such as DT and KNN could provide a much faster execution time than ensemble learning algorithms, while we could not see a noticeable difference between their performance and the best ensemble learning algorithm (e.g., DT's accuracy is less than 1% lower than the best ensemble algorithm). In a AIoT-based smart parking, the MLNN algorithms should be run on limited-resource edge and even end device. As a result, using simple MLNN algorithms with acceptable performance such as DT can be more appropriate than complex compute-intensive ones.

We finally simulated the proposed recommendation system using the DT Algorithm. For 1000 parking spot requests from drivers, the results showed that around 77% of drivers can not find a free spot in their selected destinations (i.e., street or segment). By predicting this issue and introducing alternative closest vacant locations to destinations to the drivers, the recommended system could save, in total, 3500 min drivers searching time. Therefore, the recommendation system can help to reduce the traffic and save a noticeable amount of fuel.

**Author Contributions:** Conceptualization: I.D., E.A. and G.M.L.; Data curation: I.D. and E.A.; Formal analysis: I.D. and E.A.; Writing—original draft: I.D., E.A. and G.M.L.; Writing—review and editing: E.A. and G.M.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Lin, T.; Rivano, H.; Mouël, F.L. A Survey of Smart Parking Solutions. *IEEE Trans. Intell. Transp. Syst.* **2017**, *18*, 3229–3253. [\[CrossRef\]](#)
2. Vlahogianni, E.I.; Kepaptsoglou, K.; Tsetsos, V.; Karlaftis, M.G. A Real-Time Parking Prediction System for Smart Cities. *J. Intell. Transp. Syst.* **2016**, *20*, 192–204. [\[CrossRef\]](#)
3. Donald, C. Shoup. Cruising for parking. *Transp. Policy* **2006**, *13*, 479–486.
4. Zhu, Y.; Ye, X.; Chen, J.; Yan, X.; Wang, T. Impact of Cruising for Parking on Travel Time of Traffic Flow. *Sustainability* **2020**, *12*, 3079. [\[CrossRef\]](#)
5. *Proposal to Develop a Technical Paper on Artificial Intelligence of Things (AIoT)*; Technical Paper; International Telecommunication Union (ITU)-SG20-C979; International Telecommunication Union (ITU): Geneva, Switzerland, October 2021.
6. Biyik, C.; Allam, Z.; Pieri, G.; Moroni, D.; O’Fraifer, M.; O’Connell, E.; Olariu, S.; Khalid, M. Smart Parking Systems: Reviewing the Literature, Architecture and Ways Forward. *Smart Cities* **2021**, *4*, 623–642. [\[CrossRef\]](#)
7. Diaz Ogás, M.G.; Fabregat, R.; Aciar, S. Survey of Smart Parking Systems. *Appl. Sci.* **2020**, *10*, 3872. [\[CrossRef\]](#)
8. Fahim, A.; Hasan, M.; Chowdhury, M.A. Smart parking systems: Comprehensive review based on various aspects. *Heliyon* **2021**, *7*, e07050. [\[CrossRef\]](#)
9. Khalid, M.; Wang, K.; Aslam, N.; Cao, Y.; Ahmad, N.; Khan, M.K. From smart parking towards autonomous valet parking: A survey, challenges and future Works. *J. Netw. Comput. Appl.* **2021**, *175*, 102935. [\[CrossRef\]](#)
10. Canli, H.; Toklu, S. Deep Learning-Based Mobile Application Design for Smart Parking. *IEEE Access* **2021**, *9*, 61171–61183. [\[CrossRef\]](#)
11. Ali, G.; Ali, T.; Irfan, M.; Draz, U.; Sohail, M.; Glowacz, A.; Sulowicz, M.; Mielnik, R.; Faheem, Z.B.; Martis, C. IoT Based Smart Parking System Using Deep Long Short Memory Network. *Electronics* **2020**, *9*, 1696. [\[CrossRef\]](#)
12. Tekouabou, S.C.K.; Cherif, W.; Silkan, H. Improving parking availability prediction in smart cities with IoT and ensemble-based model. *J. King Saud Univ. Comput. Inf. Sci.* **2020**, *34*, 687–697. [\[CrossRef\]](#)
13. Arjona, J.; Linares, M.; Casanovas-Garcia, J.; Vázquez, J.J. Improving Parking Availability Information Using Deep Learning Techniques. *Transp. Res. Procedia* **2020**, *47*, 385–392. [\[CrossRef\]](#)
14. Sonny, A.; Rai, P.K.; Kumar, A.; Khan, M.Z.A. Deep Learning-Based Smart Parking Solution using Channel State Information in LTE-Based Cellular Networks. In Proceedings of the 2020 International Conference on COMMunication Systems & NETWORKS (COMSNETS), Bengaluru, India, 7–11 January 2020; pp. 642–645. [\[CrossRef\]](#)
15. Piccialli, F.; Giampaolo, F.; Prezioso, E.; Crisci, D.; Cuomo, S. Predictive Analytics for Smart Parking: A Deep Learning Approach in Forecasting of IoT Data. *ACM Trans. Internet Technol.* **2021**, *21*, 68. [\[CrossRef\]](#)
16. Rahman, S.; Ramli, M.; Arnia, F.; Muharar, R.; Ikhwan, M.; Munzir, S. Enhancement of convolutional neural network for urban environment parking space classification. *Glob. J. Environ. Sci. Manag.* **2022**, *8*, 315–326. [\[CrossRef\]](#)
17. Chen, R.C.; Dewi, C.; Huang, S.W.; Caraka, R.E. Selecting critical features for data classification based on machine learning methods. *J. Big Data* **2020**, *7*, 52. [\[CrossRef\]](#)
18. Awan, F.M.; Saleem, Y.; Minerva, R.; Crespi, N. A Comparative Analysis of Machine/Deep Learning Models for Parking Space Availability Prediction. *Sensors* **2020**, *20*, 322. [\[CrossRef\]](#)
19. Falessi, D.; Huang, J.; Narayana, L.; Thai, J.F.; Turhan, B. On the need of preserving order of data when validating within-project defect classifiers. *Empir. Softw. Eng.* **2020**, *25*, 4805–4830. [\[CrossRef\]](#)
20. Janiesch, C.; Zschech, P.; Heinrich, K. Machine learning and deep learning. *Electron. Mark.* **2021**, *31*, 685–695. [\[CrossRef\]](#)
21. Sarker, I.H. Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Comput. Sci.* **2021**, *2*, 160. [\[CrossRef\]](#)
22. Manjula, C. Belavagi and Balachandra Muniyal, Performance Evaluation of Supervised Machine Learning Algorithms for Intrusion Detection. *Procedia Comput. Sci.* **2016**, *89*, 117–123.

23. Verbraeken, J.; Wolting, M.; Katzy, J.; Kloppenburg, J.; Verbelen, T.; Rellermeyer, J.S. A Survey on Distributed Machine Learning. *ACM Comput. Surv.* **2020**, *53*, 1–13. [\[CrossRef\]](#)
24. Kaviani1, K.; Dhotre, S. Short Survey on Naive Bayes Algorithm. *Int. J. Adv. Eng. Res. Dev.* **2017**, *4*, 601–611.
25. Marius, P.; Balas, V.E.; Perescu-Popescu, L.; Mastorakis, N.E. Multilayer perceptron and neural networks. *WSEAS Trans. Circuits Syst.* **2009**, *8*, 579–588.
26. Sak, H.; Senior, A.; Beaufays, F. Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling. *arXiv* **2014**, arXiv:1402.1128.
27. Bock, F.; Attanasio, Y.; Di Martino, S. On-street parking data in San Francisco—SFpark sensor data and simulated crowd-sensing data, Harvard Dataverse. *Harv. Dataverse* **2018**. [\[CrossRef\]](#)
28. Bock, F.; Di Martino, S. On-street parking availability data in San Francisco, from stationary sensors and high-mileage probe vehicles. *Data Brief* **2019**, *25*, 104039. [\[CrossRef\]](#)
29. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
30. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 8024–8035.
31. Random-Search in Scikit Learn. Available online: [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.RandomizedSearchCV.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html) (accessed on 1 October 2021).