To appear in Proceedings of the 2014 IEEE PES General Meeting, 27-31 July. National Harbor, MD, USA. (in press).

Fast Prediction for Sparse Time Series: Demand Forecast of EV Charging Stations for Cell Phone Applications

Mostafa Majidpour, *Student Member, IEEE*, Charlie Qiu, Peter Chu, Rajit Gadh, Hemanshu R. Pota, *Member, IEEE*

Abstract— **This article describes the core cellphone application algorithm which has been implemented for the prediction of energy consumption at Electric Vehicle (EV) Charging Stations at UCLA. For this interactive user application, the total time of accessing database, processing the data and making the prediction needs to be within a few seconds. We analyze three relatively fast Machine Learning based time series prediction algorithms and find that the Nearest Neighbor (NN) algorithm (k Nearest Neighbor with k=1) shows better accuracy. Considering the sparseness of the time series of the charging records, we then modify the dissimilarity measure in the NN algorithm to improve the accuracy and processing time. Two applications have been designed on top of the proposed prediction algorithm: one predicts the expected available energy at the station and the other one predicts the expected charging finishing time. The total time, including accessing the database, data processing, and prediction is approximately one second for both applications. The granularity of the prediction is one hour and the horizon is 24 hours; data have been collected from 15 charging stations.**

Index Terms— **Cellphone Applications, Electric Vehicles, Kernel methods, Nearest Neighbor Searches, Prediction Methods, Sparse time series.**

I. Introduction

Electric Vehicles (EVs) (or Plug-in Hybrid Electric Vehicles (PHEV)) will be an important part of the Smart Grid. The big challenge for EVs is their charging within the existing distribution system infrastructure. According to the EV33 rule (33 miles driving range for a single charge [5]), the minimum battery size in EVs varies from 8.6 kWh to 15.2 kWh [6]. Charging batteries with the aforementioned size will take between four and eight hours in a Level 1 household charger (120V, 16 A). As an alternative, EV owners can charge their vehicle at their place of employment, provided that the employer has installed the chargers in the parking lot. Other than workplace parking lots, cities and private operators have invested in charging stations for public use. As of October 2013, the number of these stations in the US has reached 19,730 which is roughly a sixth of the number of gas stations in the US [11]. Electric power has to be consumed simultaneously upon generation. As a result, the EV charging station needs to have a good idea of how many vehicles will need charging at each moment. On the other hand, since a reasonable amount of charging will take tens of minutes, it would be useful for a customer to know when and how much energy is expected to be available at a given charging station in a given time window. Both the customer and moderator will substantially benefit from an algorithm that can predict the power consumption at charging stations.

With emerging smart phones, it would be beneficial to customers if they could receive the prediction information on their cellphone, giving them an estimate of the charging time. This paper discusses prediction algorithms that can run in less than a few seconds, so that EV users can query the system and get the results in a reasonable time on their cellphones. To our knowledge, our work is the first research that discusses fast prediction of the available energy and/or expected charging finishing time at the charging station for use in a cellphone application. The data used in the algorithm is obtained from the charging records. A charging record in this paper is a datum that contains the start and end time of a charging transaction as well as the total amount of received energy (in kWh) by the EV. The charging records database consists of the historical charging data for a given charging station.

The work described in this paper differs from other previous works in literature: 1) we have used just one type of recorded data, Charging Records, which only contains the start and end of the charging transaction and the total amount (a scalar value; not time dependent) of energy received in the charging transaction rather than geographical or any driving habit related data; 2) our predictions is at the charging station level (not parking lot or the whole building level); 3) our method is online and fast with the whole process taking about a second. Indeed, the reason that we do not use other sources of data is our speed requirement and the fact that adding more data will slow the process.

The rest of this paper is organized as follows: Section II provides a brief review of existing literature, Section III formulates the problem, Section IV explains the methods that

This work has been sponsored in part by grant from the LADWP/DOE fund 20699 & 20686, Smart Grid Regional Demonstration Project.

M. Majidpour, C. Qiu, P. Chu, and R. Gadh are with Smart Grid Energy Research Center, University of California, Los Angeles 90095 USA (emails:mostafam@ucla.edu)

H. R. Pota is with the School of Engineering & Information Technology, The University of NSW, Canberra ACT 2610 Australia (e-mail: h.pota@adfa.edu.au).

are evaluated in this paper in order to compare with the proposed method. Section V reports and analyzes the result of applying the algorithms on the University of California, Los Angeles (UCLA) parking structures' data. Section VI explains the proposed algorithm via modifications to discussed algorithms, Section VII talks about the implementation of the cellphone applications using the proposed algorithm and Section VIII provides the conclusion and future work.

II. Literature Review

Time series prediction (forecasting) methods predict the future of a certain variable given its past history. There is a rich literature on different methods of time series prediction that has evolved from statistics, mathematics, computer science, economics, and engineering [13]-[15]. Probably, the most famous model in time series prediction is the ARIMA model with Box-Jenkins approach [14] which has been used widely in economics and statistics [15] and is considered to be the traditional approach in time series prediction. Selecting the correct parameters for the ARIMA model is not a trivial task, and, depending on the approach, it might be time consuming.

Prediction algorithms are part of most of the modern smart grid technologies [34] such as Photovoltaic systems [36], Smart Buildings [32], and Wind Turbines [35]. Extensive research has focused on EV charging algorithms and charging station infrastructures [7]-[10], and, as a next step, EV related research has started to utilize forecasting algorithms [16]-[22]. Some papers apply forecasting algorithms to EV driving habits and predict the State of Charge (SOC) of a particular EV and when it needs to be charged [17][18]. Authors in [16] have applied Artificial Neural Network (ANN) forecasting algorithms to predict the charging profile of the EV within the Building Energy Management System (BEMS) in order to improve the overall energy efficiency of the building. Reference [19] and [20] discuss the prediction of the EV charging profile while taking into account various sources of data, such as vehicle driving/usage data. Authors in [21] and [22] consider forecasting at the charging station level based on the EV user classification and Monte Carlo simulations method. Reference [33] discusses an energy management system for EVs that takes advantage of prediction in different levels through hierarchical Model Predictive Control.

As mentioned above, the aim of this work is to focus on relatively fast time series forecasting algorithms in which the whole process of prediction (including pre-processing) takes a reasonably short amount of time for a user that sends prediction-related queries from a cellphone. Machine Learning (ML) based heuristic methods have been shown to provide a good performance in forecasting [12]. Some of these methods such as k-Nearest Neighbor (kNN) and weighted kNN depending on the number of neighbors could be pretty fast. However, the selection of parameters for these ML methods still remains a challenge.

For our predictor application, we are interested in the ML based algorithms that have low computation requirements and are relatively faster. A brief introduction of these methods will follow in Section IV.

III. Problem Formulation

The objective is to predict the available energy in the next 24 hours at each charging station with a minimum time for processing. Formally, we assume there is some function relating future available energy and the past consumed energy:

$$\hat{E}(t) = f\big(E(t-i), \varepsilon(t)\big) \quad i \in \{1,2,\dots\} \qquad (2),$$

where $E(t)$ is the actual energy consumption at time t, $\hat{E}(t)$ is the prediction of the energy consumption at time t, $\varepsilon(t)$ is the set of all variables (such as noise) other than past energy consumption records ($E(t-i)$) that f might depend on.

As the usual practice in forecasting, we are interested to find an estimation of $E(t)$ according to a particular performance (or error) criteria. For the error measurement, we have chosen Symmetric Mean Absolute Percentage Error (SMAPE). For the day i, the SMAPE is defined as:

$$SAMPE(i) = \frac{1}{H}\sum_{t\in Day(i)} \frac{\left|E(t)-\hat{E}(t)\right|}{E(t)+\hat{E}(t)} \times 100, \qquad (2),$$

where H is the horizon of prediction in a given day H(=24 in this paper).

Since there is no access to future data in real life, the last portion of the data (last 10% in this paper) is set aside as the test set to evaluate the performance of the algorithm. Thus, our goal is to find an algorithm that minimizes the error between the actual value and its prediction in the test set. Note that the test set is not used in either the parameter selection or training phase.

We use the notation $\underline{\hat{E}}(t)$ as the vector of prediction for the next 24 hours ending at t (Fig 1. a). Also, $t_r = \{1,2,\dots,N_{tr}\}$ and $t_s = \{N_{tr}+1,\dots,N\}$ are the set of indices for the training and test sets, respectively. Later on, in the parameter selection phase, parts of the training set will be treated as the validation set. The different methods used to select the validation set are further explained in the parameter selection section.

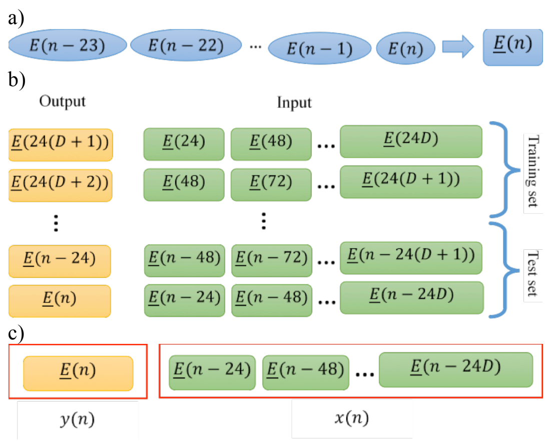

Figure 1. a) energy consumption vector ($\underline{E}$) for 24 hours , b) input-output pairs and division of data into training and test sets, c) labeling inputs as x and outputs as y.

IV. Applied Algorithms

Four prediction algorithms have been briefly described here. These algorithms were employed to compare and demonstrate the effectiveness of the proposed approach. A detailed description can be found in [25].

A. Historical Average

This algorithm is one of the simplest algorithms that is sometimes referred to as naïve approach and is used only for comparison with other methods. According to this approach, the predicted charging energy in the future is the average of the charging energy consumption in the past. Formally:

$$\hat{E}(t) = \frac{1}{D}\sum_{d=1}^{D} E(t-24d) \qquad (3),$$

where D is the depth of the averaging. For instance, the predicted energy consumption for 3pm on the next day is equal to the average of the energy consumed today, yesterday…, and up to the past D days at 3pm. D is a parameter that needs to be selected before the evaluation on the test set.

B. K-Nearest Neighbor

This algorithm is a well-known algorithm in the machine learning community [26]. Based on the k-Nearest Neighbor (kNN) algorithm, each sample (training, test or validation) is composed of input and output pairs. In our application, as seen in Fig. 1. c, the output is the predicted energy consumption for the next 24 hours:

$$y(t) = \underline{E}(t) \qquad (4),$$

and the input is the concatenation of the consumption records for up to D previous days:

$$x(t) = \{\underline{E}(t-24), \underline{E}(t-48), \dots, \underline{E}(t-24D)\} \qquad (5).$$

This concatenation repeats for all days: if there are N days in the data set, there will be $N\text{-}D\text{+}1$ of these input-output pairs (Fig 1.b). The total number of data points is $n = 24N$. Now, in order to find an estimate for $y(t_{s^*})$ where $t_{s^*} \in t_s$ is an instance of test set indices, first, the dissimilarity between $x(t_{s^*})$ and all other $x(t_r)$ that belong to the training set is computed. We have used the Euclidian distance as the measure of dissimilarity here. After determining the k closest $x(t_r)$ to $x(t_{s^*})$, the average of their corresponding $y(t_r)$ is generated as $y(t_{s^*})$. In this algorithm, the parameter k needs to be determined. Fig. 2 illustrates the algorithm where $dis[j]$ refers to dissimilarity between $x(t_{s^*})$ and $x(j)$.

k-Nearest Neighbor Algorithm

```
Inputs: $x(t_r), y(t_r), x(t_{s^*}), k$
Output: $y(t_{s^*})$
1. for $j \in t_r$
2.    $dis[j] = \|x(t_{s^*}) - x(j)\|$
3. for $i \in \{1, \dots, k\}$
4.    $idx[i]$ = index of $i^{th}$ smallest$(dis)$
5. $y(t_{s^*}) = \frac{1}{k}\sum_{i\in\{1,\dots,k\}} y(idx[i])$
```

Figure 2. k-Nearest Neighbor Algorithm.

C. Weighted k-Nearest Neighbor

Weighted kNN is based on the idea that closer points to the query should contribute more to the output and in this way improve the accuracy of the prediction compared to kNN [1].

This algorithm is very similar to the original kNN algorithm with the exception that instead of averaging the k closest training outputs (Fig. 2, step 5) their weighted average is used, where the weights are a function of the dissimilarity between input pairs. The weights are defined based on Dudani's weights [1] which seems to give better results compared to other similar weight assigning methods according to both literature [2] and our simulations. Thus, the following two steps will substitute for Step 5 in Fig. 2:

$$w_p = \frac{dis[k+1] - dis[p]}{dis[k+1] - dis[1]} \quad p = 1, .., k \qquad (6),$$

$$y(t_{s^*}) = \frac{1}{\sum_{q\in idx} w_q} \sum_{p\in idx} w_p y(p) \qquad (7).$$

D. Lazy Learning

Lazy Learning (LL) is a generic term which refers to algorithms that postpone the learning until a query is submitted to the system. In fact, weighted kNN and kNN are simple forms of LL. A version of LL according to [27] has been implemented in this paper. This version of the LL algorithm is similar to the kNN algorithm in principle, with the difference that for each query, the optimum number of neighbors (k) is not fixed and is estimated separately. The idea is that for some queries it seems better to look at more neighbors and for some others, fewer neighbors would be enough. For each query, kNN is performed k_{max} times for $k = \{1,2,\dots,k_{max}\}$. Then, based on Leave-one-out (LOO) cross validation, the error for each k is estimated, and the output corresponding to the k with a lower LOO cross validation error is selected. We use the PRESS statistic [27] to estimate LOO for each k. For a fixed $k \in \{1,2,\dots,k_{max}\}$, suppose $j^* \in \{1,2,\dots,k\}$ indicates the index of the jth closest neighbor to the query $x(t_{s^*})$. For each j^*, we define an error term

$$e_k(j^*) = k\frac{y(j^*) - y(i)}{k-1} \qquad (6),$$

which gives the LOO cross validation error for the specific k:

$$e_{LOO}(k) = \frac{1}{k}\sum_{j^*=1}^{k} (e_k(j^*)^T * (e_k(j^*)) \qquad (7).$$

The k with smallest e_{LOO} will be selected as the optimum k. The steps are detailed in Fig. 3.

Lazy Learning Algorithm

```
Inputs: $x(t_r), y(t_r), x(t_{s^*}), k_{max}$
Output: $y(t_{s^*})$
1. for $j \in t_r$
2.    $dis[j] = \|x(t_{s^*}) - x(j)\|$
3. for $i \in \{1, \dots, k_{max}\}$
4.    $idx[i]$ = index of $i^{th}$ smallest$(dis)$
4. for $k \in \{2, \dots, k_{max}\}$
5.    $y(k) = \frac{1}{k}\sum_{i\in\{1,\dots,k\}} y(idx[i])$
6.    Calculate $e_{LOO}(k)$ according to Eq. (7)
7. $k^* = \arg(\min_{k\in\{2,\dots,k_{max}\}} e_{LOO}(k))$
8. $y(t_{s^*}) = y(k^*)$
```

Figure 3. Lazy Learning Algorithm.

V. Simulation Setup and Preliminary Results

A. Data

The algorithms described above are applied to charging stations located on the UCLA campus. The data used in this paper were recorded from December 7, 2011 to October 16, 2013; however, for each day, not all stations were in use. Among all the charging stations at UCLA, 15 stations have charging data for more than 60 effective days (days that some nonzero charging has been reported); these stations have been used in our implementation. The number of effective days for each station is reported in Table I.

Data for each station comes in a format which is referred to as Charging Records. Each charging record contains the beginning and end of the charging time as well as the acquired energy.

B. Preprocessing

The Charging Records are converted to time series by uniformly dividing the acquired energy to the charging interval. For example, if charging interval is 3 hours and the acquired energy is 3kWh, it is assumed that the EV received 1kWh of energy in each hour. In pre-processing the data, if all the values in an input-output pair $(x(i), y(i))$ are zero or not reported, the pair was removed from the data set.

There was no normalization or feature extracting from the data. The only implemented pre-processing was to force energy records that were mistakenly recorded as more than the physical maximum of the charging device (E_{max}) and less than zero to the interval of $[0, E_{max}]$.

C. Parameter Selection

The following parameters need to be determined for our algorithms: Depth, D, for all algorithms, which is the number of previous days considered in the input vector, and the number of neighbors, k, in kNN and weighted kNN.

There are different options for performing parameter selection through validation in time series [3]. One of the popular methods is the k-fold cross validation. In k-fold cross validation, for evaluating a certain set of candidate parameters, the training data is divided into k parts ($P_i, i = 1, \dots, k$) and algorithm trains on k-1 parts ($P_{-i} = Training\ set - P_i$) while the error is calculated on the remaining part (P_i), i.e., the validation set. This process iterates k times, and each time one of the parts will be the validation set and the other k-1 parts will make up the training set. The average error of the algorithm on these k iterations will determine the final error for the current selection of parameters. The whole process is repeated for different combinations of parameters and the combination that yields the least final error is selected as algorithm parameters.

However, [4] discusses how k-fold cross validation in its original form might not be appropriate for time series since it does not respect the order. It proposes a “time series cross validation” in which for cross validation we are only allowed to use training data prior to the validation set, i.e. if validation set is P_i, then the training data would be $P_j, j = 1, \dots, i-1$ instead of P_{-i}. Another approach taken in [29] is to use the last portion of the training data for validation and parameter selection purposes. In this approach, the validation set is the last block of training data, P_k, and the rest of data prior to it, $P_j, j = 1, \dots, k-1$, is used for training; hence, it is less computationally expensive since it evaluates parameters on just one block rather than k blocks.

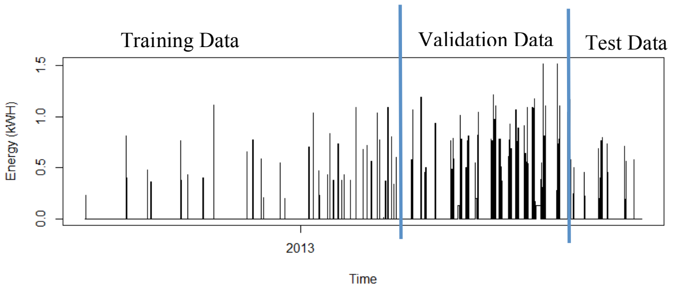

Figure 4. Separating validation data from training data in the last block validation method.

We have tried all of the three mentioned validation methods: k-fold cross validation, time series cross validation, and the last block validation. We found that they lead to similar behavior in the final results; therefore, in this paper, we are going to dedicate the last block of training data to validation data which is relatively less computationally expensive, and therefore more appropriate for our fast prediction application. This method basically partitions the data to non-overlapping training, validation and test data sets as depicted in Fig 4.

The length of the validation block is the next matter of importance. We must determine how much of the last portion of the available data is a good representative of the behavior of all the data for specific parameters. In order to answer this question, we picked different percentages of the last part of the training data for each station and each algorithm, and we compared it with the whole dataset in terms of similarity in response to change in parameters.

The smallest percentage of data that could represent the whole training dataset for all algorithms is selected as the final validation set for each station. Table I shows the percentage of the data that is used as validation set for each station. The candidate percentages were: 10%, 15%, 20%, 25%, and 30%.

TABLE I. Number of Effective Days for Each Station And Percentage of Data used for Training, Validation and Test sets

No	Station	Effective Days	Training Set (%)	Validation Set (%)	Test Set (%)
1	PS3L401LIA3	95	85	15	10
2	PS8L201LIA1	84	85	15	10
3	PS8L201LIA3	97	85	15	10
4	PS8L201LIA4	171	90	10	10
5	PS8L202LIIA1	163	85	15	10
6	PS8L202LIIA2	168	90	10	10
7	PS8L202LIIA3	151	90	10	10
8	PS9L401LIA1	178	80	20	10
9	PS9L401LIA2	126	75	25	10
10	PS9L401LIA3	307	90	10	10
11	PS9L401LIA5	189	80	20	10

12	PS9L401LIA6	269	80	20	10
13	PS9L601LIA1	206	70	30	10
14	PS9L601LIA3	178	80	20	10
15	PS9L601LIA4	124	85	15	10

For parameter selection, the depth parameter, *D*, is varied between 1 to 60 (equal to looking only at yesterday or up to the past two months); in order to make the process faster, a subset of 1 to 60, namely {1,2,…,9,10,15,20,…,60} is employed. The number of neighbors varies between 1 to 5 for kNN and 2 to 5 for weighted kNN (weighted kNN with *k*=1 is the same as kNN with *k*=1).

D. Preliminary Results

Table II and III show the depth (*D*) and number of neighbors (*k*) parameters, selected according to the previous section, for each site and each parameter.

TABLE II. SELECTED DEPTH PARAMETER FOR EACH ALGORITHM BASED ON VALIDATION RESULTS

No	Station	Historical Average	kNN	Weighted kNN	Lazy Learning
1	PS3L401LIA3	1	60	60	60
2	PS8L201LIA1	1	55	8	60
3	PS8L201LIA3	1	50	60	60
4	PS8L201LIA4	2	55	4	20
5	PS8L202LIIA1	1	40	1	40
6	PS8L202LIIA2	1	60	1	35
7	PS8L202LIIA3	1	20	1	20
8	PS9L401LIA1	1	15	1	2
9	PS9L401LIA2	1	15	4	15
10	PS9L401LIA3	1	35	15	25
11	PS9L401LIA5	1	45	45	45
12	PS9L401LIA6	1	60	1	60
13	PS9L601LIA1	1	60	60	60
14	PS9L601LIA3	1	60	55	55
15	PS9L601LIA4	1	30	30	50

After selecting the parameters, the union of the training data and validation data are treated as the new training dataset and used for predicting the first day in the test dataset. For predicting the second day in the test dataset, the union of training dataset, validation dataset, and the first day in the test data is used; similarly, for predicting the *i*th day in the test set, all the data prior to it (training dataset, validation dataset, and test dataset up to *(i-1)*th day) is employed.

Interestingly, as can be seen in Table III, for all stations, the optimum number of neighbors is chosen to be equal to 1 for the kNN algorithm. This shows that, regardless of the optimum number of days to forecast, it is always better to look at the most similar event in the past and copy its future energy consumption values as the prediction. Also, the optimum number of neighbors for weighted kNN in all stations is 2; considering that *k* ranges from 2 to 5 and the kNN algorithm with *k*=1 shows better performance, one can conclude that *k*=1 is the optimum parameter.

TABLE III. SELECTED NUMBER OF NEIGHBOURS PARAMETER FOR EACH ALGORITHM ASED ON VALIDATION RESULTS

No	Station	kNN	Weighted kNN
1	PS3L401LIA3	1	2
2	PS8L201LIA1	1	2
3	PS8L201LIA3	1	2
4	PS8L201LIA4	1	2
5	PS8L202LIIA1	1	2
6	PS8L202LIIA2	1	2
7	PS8L202LIIA3	1	2
8	PS9L401LIA1	1	2
9	PS9L401LIA2	1	2
10	PS9L401LIA3	1	2
11	PS9L401LIA5	1	2
12	PS9L401LIA6	1	2
13	PS9L601LIA1	1	2
14	PS9L601LIA3	1	2
15	PS9L601LIA4	1	2

Table IV shows the average and standard deviation of SMAPE on test days for each algorithm and each station.

TABLE IV. AVERAGE AND STANDARD DEVIATION (IN PARANTHESES) OF SMAPE (%) ON TEST DAYS FOR EACH ALGORITHM

No	Historical Average	kNN	Weighted kNN	Lazy Learning
1	95.98 (2.36)	3.40 (6.72)	3.58 (7.06)	3.72 (7.48)
2	97.62 (1.64)	16.08 (34.67)	33.83 (45.63)	16.11(29.48)
3	97.45 (1.16)	51.45 (31.40)	49.64 (33.08)	49.69(33.13)
4	94.10 (4.75)	1.64 (3.15)	13.12 (17.45)	14.40(11.89)
5	87.04 (8.91)	28.43 (19.77)	35.08 (23.87)	34.94(15.99)
6	83.19 (10.96)	35.21 (12.60)	26.61 (16.18)	37.82(13.87)
7	87.07 (11.61)	28.63 (17.54)	22.90 (16.05)	38.72(14.80)
8	80.44 (14.54)	21.20 (28.01)	22.16 (21.82)	25.67(27.08)
9	95.95 (2.55)	12.38 (8.61)	18.19 (9.20)	16.05(10.89)
10	92.82 (10.43)	8.66 (14.62)	12.32 (18.69)	12.37(18.10)
11	87.71 (11.71)	7.14 (11.83)	12.70 (13.11)	13.19(13.62)
12	76.69 (12.23)	3.75 (13.34)	18.75 (17.57)	3.75 (13.34)
13	91.99 (4.84)	10.64 (10.26)	15.82 (10.68)	16.11(10.73)
14	90.94 (5.28)	9.78 (12.33)	14.06 (12.24)	14.72(12.22)
15	87.49 (5.91)	8.37 (12.29)	10.12 (14.83)	11.28(16.01)
Mean	**84.86 (7.26)**	**16.45 (15.81)**	**20.59 (18.50)**	**20.57(16.57)**

The historical average has by far the worst performance. For better comparison, the error for other three methods has been depicted in Fig. 5.

Comparing LL results with kNN, it seems that LL was not successful in choosing the best number of neighbors (*k*) for each query; otherwise, it would have better results compared to kNN (which has a constant *k*). Stations no. 3 is the exceptions to this, where LL does (slightly) better than kNN. Also weighted kNN does a better job in stations no 6 and 7. Overall, the accuracy results for kNN with *k*=1 (which is the Nearest Neighbor) nominates it as the best method. It is notable that for station no. 3 all algorithms result SMAPE of around 50%.

All simulations were run with RStudio Version 0.97.551 on an Intel Core i-7 CPU at 3.40 GHz with 16 GB RAM.

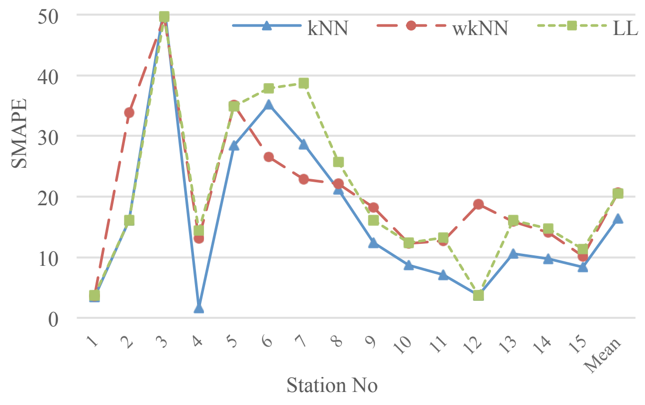

Figure 5. Comparing kNN, wkNN, and LL. kNN gives the better overal accuracy.

VI. Proposed Algorithm

In the previous section, we found that the Nearest Neighbor (NN) algorithm has the best performance as the fast predictor algorithm on the examined data. However, there is still a room to improve the results since the results for some of the stations is not yet satisfactory, e.g. station no 3.

Looking at Fig. 2 for the kNN algorithm, it turns out that after selecting k (which, through parameter selection, is equal to 1), there is not much room for modifying the algorithm except step 2 which is the dissimilarity calculation step. The dissimilarity that have been used in the paper is Euclidean distance. Thus, we are focusing on modifying the dissimilarity measure in step 2 of the kNN algorithm in order to improve the performance.

A. *Dissimilarity measures*

In applying the NN algorithm, we used Euclidian distance to measure the dissimilarity between two data points and determine the nearest neighbor of each input query. However, the EV charging data is sparse, meaning that in significant chunks of time, e.g. during nights in the government or official parking spaces, there is no charging in progress and the consumed power is zero. When the data is sparse, it might be beneficial to employ other dissimilarity measures than Euclidian distance.

One way to define the dissimilarity measure is to use the inverse or negative of a similarity measure. A candidate for similarity measure is the dot product of two vectors since it is zero when two vectors are orthogonal to each other and is maximum when they are equal. Specifically, the dot product of two signals that have non-zero elements in different indices is equal to zero. Fig. 6 illustrates this concept: the dissimilarity between vectors X and Y is equal to $\sqrt{2}$ for both cases when it is calculated through Euclidean distance while the dissimilarity calculated through dot product in Fig.6.a is higher ($-< X, Y >= 0$) than Fig.6.b ($-< X, Y >= -3$). This fits well to sparse time series prediction applications, since the similarity between two pieces of signals with different indices of non-zero elements should be as less as possible.

B. *Kerneleized similarity*

Upon using the dot product as similarity measure, one can use kernelized similarity measures to get more flexibility and to compute similarity in higher dimensions [31]. In particular, polynomial kernels are interesting for us here since it is the natural extension of the dot product. A polynomial kernel for similarity between x and y is often defined as following [25]:

$$K(x, y) = (x^t y + c)^d \quad (8),$$

where $c \geq 0$ is a constant that is trading off the influence of higher-order terms versus lower order ones and d is the degree of the polynomial kernel. Now, we can define the dissimilarity measure based on the polynomial kernel:

$$dis(x(t_1), x(t_2)) = -K(x(t_1), x(t_2)) \quad (9).$$

Another alternative for defining dissimilarity based on kernels is to find the distance of two inputs in the kernel space which can be obtained from the following equation:

$$dis\left(\varphi(x(t_1)), \varphi(x(t_2))\right)^2 = K\left(x(t_1), x(t_1)\right) + K(x(t_2), x(t_2)) - 2K(x(t_1), x(t_2)). \quad (10).$$

It, however, needs more computation because of self-mapping terms, and it does not improve our results in practice.

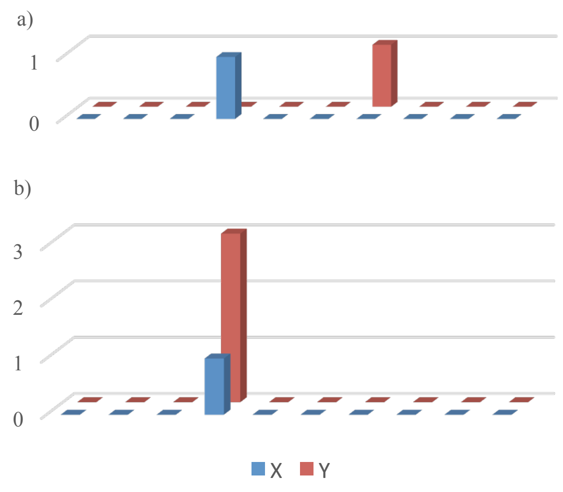

Figure 6. Comparison between Euclidean and dot product similarities of two vectors X and Y. Euclidean distance gives the same dissimilarity for both a) and b) while dot product based dissimilarity, assigns higher disimilarity to a).

C. *Time weighted dissimilarity*

Another intuitive modification of the dissimilarity measures could be time weighting; for instance, outputs $y(t_1)$ and $y(t_2)$ are more similar if the recent values of their corresponding inputs $x(t_1)$ and $x(t_2)$ are more similar. In order to weight the recent values for an input that has been defined in (5), we have used linear time weighting:

$$TW = [1 + D, \dots, 1 + 2\Delta, 1 + \Delta, 1]$$

where $\Delta = 1/(24D - 1)$ and D is the depth of input. Also, we have tried other weighting methods such as *exp(TW)* but we did not see improvement in the final results.

Combining all the modifications together, the dissimilarity measure used in kNN algorithm will be substituted with:

$$dis(x(t_1), x(t_2)) = -(x(t_1)^t \, diag(TW) \, x(t_2) + c)^d \quad (10).$$

D. *Results*

We used the same settings as discussed in Section V and the only difference in this section is modifying the dissimilarity measure. This modification however adds two more parameters *c* and *d* according to (10) which, like other parameters, need to be determined with validation. However, in different simulations we did not see much of a difference in the final results of the NN algorithm with the change in *c* or *d*. Therefore, both of these parameters have been set equal to 1. The conclusion is that similarity in higher order terms are not necessary for finding the prediction related similarity in our application, and only the linear terms' contribution is enough for prediction. This is good news computational wise since there is no need to try higher order terms.

Fig. 7 shows a sample test day from Station 1 and its prediction. The SMAPE for this specific day is 16.93%. The results for all stations are presented in Table V. According to the Table V, the average SMAPE has been improved in all methods compared with the Euclidean distance case. Fig 8 displays the kNN, weighted kNN, and LL results with modified dissimilarity measure. It's notable that maximum SMAPE for all methods has been decreased to less than 35% SMAPE.

TABLE V. AVERAGE AND STANDARD DEVIATION (IN PARANTHESES) OF SMAPE (%) ON TEST DAYS FOR EACH ALGORITHM

No	**Station**	**kNN (k=1)**	**Weighted kNN (k=2)**	**Lazy Learning**
1	PS3L401LIA3	3.96 (7.62)	3.65 (6.44)	4.04 (7.04)
2	PS8L201LIA1	0.67 (2.43)	0.67 (2.43)	0.67 (2.43)
3	PS8L201LIA3	0.79 (4.53)	1.49 (5.43)	1.49 (5.43)
4	PS8L201LIA4	9.24 (10.16)	19.54 (13.44)	19.54(13.44)
5	PS8L202LIIA1	20.09 (19.12)	23.08 (21.41)	21.50(18.10)
6	PS8L202LIIA2	30.50 (16.55)	34.73 (9.94)	34.20(10.68)
7	PS8L202LIIA3	24.09 (19.79)	25.61 (19.13)	26.46(20.48)
8	PS9L401LIA1	22.95 (21.37)	32.57 (28.79)	32.28(28.14)
9	PS9L401LIA2	11.46 (7.75)	13.20 (9.10)	13.62(10.19)
10	PS9L401LIA3	5.96 (14.09)	5.96 (14.09)	5.96 (14.09)
11	PS9L401LIA5	14.38 (13.84)	16.17 (14.02)	16.17(14.02)
12	PS9L401LIA6	18.91 (17.90)	13.96 (22.16)	13.96(22.16)
13	PS9L601LIA1	13.89 (13.62)	16.57 (15.06)	16.54(14.94)
14	PS9L601LIA3	7.81 (12.02)	8.34 (11.67)	8.36 (11.69)
15	PS9L601LIA4	6.45 (12.41)	6.45 (11.61)	6.45 (11.61)
Mean		**12.74 (12.88)**	**14.80 (13.65)**	**14.75(13.63)**

The interesting difference in the pattern of SMAPE errors between results from two different dissimilarity measures has been depicted for NN case in Fig. 9. As this figure shows, for stations that the Euclidean dissimilarity has relatively high errors such as station no. 3, the dot product similarity has relatively low errors and vice versa. The fact that SMAPE error in station no. 3 has decreased from 51% (NN with Euclidean distance) to 0.79% (NN with dot product dissimilarity) illustrates that dot product was extremely successful in finding similar points in the time vectors and making the prediction based on that. This phenomenon shows that, depending on the characteristic of the time series in hand, we might need to change our point of view (from measuring Euclidean dissimilarity to dot product one) to be able to see the similar points in the training data to our test query.

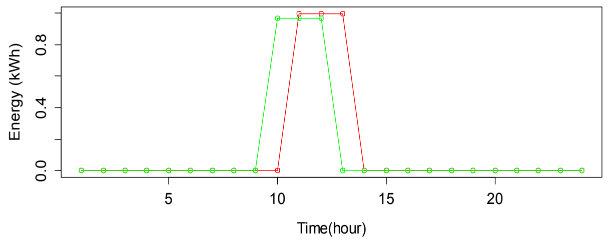

Figure 7. Actual enrgy consumption (green) and its prediction with dot product based NN algorithm (red) for a sample test day in Station 1. The SMAPE for this day is 16.93%.

In fact, the effective characteristic here seems to be sparseness of the time series. Fig. 10 show the percentage of the sparseness of the time series (number of zero entries divided by the total number of entries in the times series) calculated with the optimum depth for each station. Comparing this plot with dot product plot in Fig. 9 shows an interesting relationship: The time series which dot product based dissimilarity does better, are mostly sparser ones.

In order to take advantage of both dissimilarity measures, we implement the best method for each station with the best dissimilarity measure. For example, from Fig. 9, for station no. 3 we use the NN with the dot product dissimilarity while for station no. 12, we use the NN with Euclidean dissimilarity.

The more accurate algorithm will help the EV charging station owner to increase the profit. Knowing the prediction of the consumption in the future, the station owner can utilize all the capacity of charging stations and therefore obtain more profit; on the other hand, the station owner can reduce the disappointment of the EV owner in the case of all stations being full. Depending on which factor is more important for a certain EV station owner, s/he can penalize the over prediction of consumption (which translates to empty stations in sometimes of the day) or under prediction of the consumption (which translates to disappointed EV owner) in an appropriate way. Here, the SMAPE accuracy measure introduced in (2) is a symmetric one and does not penalize either over prediction or under prediction. However, the algorithms are readily usable for asymmetric error measurement criteria.

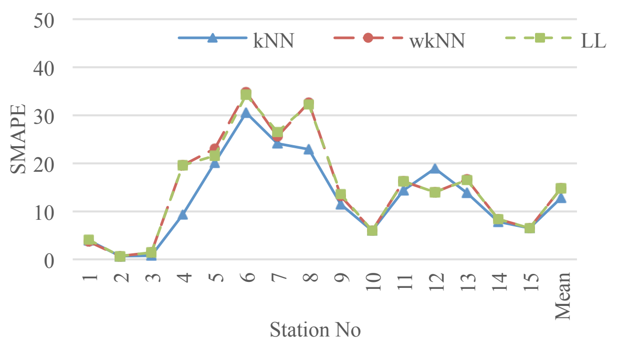

Figure 8. Comparing kNN, wkNN, and LL for dot product dissimilarity measure. kNN gives the better overal accuracy.

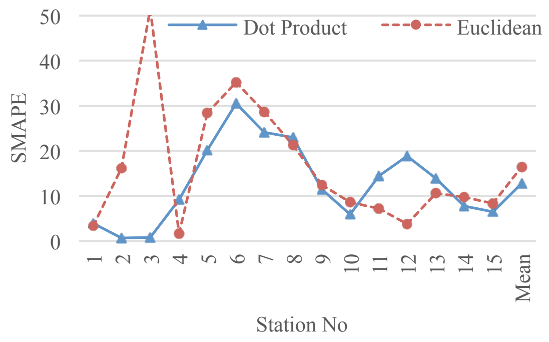

Figure 9. Comparing kNN, wkNN, and LL for dot product dissimilarity

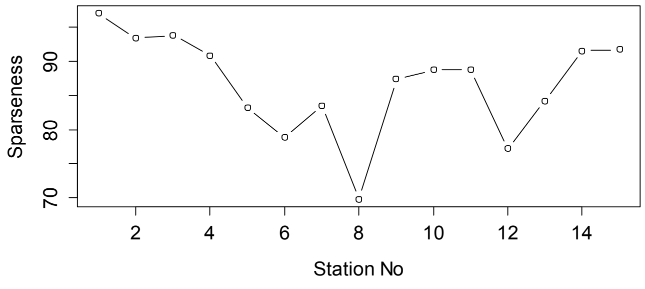

Figure 10. Sparseness of the time series of each station calculated at the optimum depth.

VII. Cellphone Applications Based on Proposed Algorithm

Based on the results, NN with the modified dissimilarity measure (which is simply the dot product) has higher accuracy on most of the stations with sparser time series and NN with the original Euclidean distance dissimilarity is performing better on stations with less sparse time series were selected as the algorithm for the cellphone application. We implemented two algorithms on top of the prediction algorithm:

- One application takes the station name, required energy (in kWh) needed to charge the vehicle, and the start of charging time as input. The output is the predicted end time of charging,
- Another application takes the station name, starting time and ending time for the charging as input. The output is the predicted amount of available energy in kWh.

The total time for the algorithm to run is about a second (less than a second for less crowded stations), which is composed of the time to (a) run in C# under Microsoft Visual Studio 2012, (b) access the database through Microsoft SQL Server 2012, and (c) generate the output. This is well within the acceptable time for a mobile application response time. The algorithm is now running on the mobile application and is available to UCLA EV owners.

VIII. Conclusion

In this paper, we have developed an approach for fast demand prediction of the sparse time series in general, and specifically EV charging stations. We found that, in general, Nearest Neighbor based predictions generate better predictions than kNN and weighted kNN. We modified the dissimilarity measure in NN from standard Euclidean distance in two ways: 1) changing Euclidean distance to (negative) dot product and 2) adding time weightings to the dissimilarity measure so that recent similar indices in time series get more weight than older ones. Each of these modifications improves the accuracy by itself and their combination improves the results more.

We have implemented this method in the cellphone application system that is used by UCLA EV owners.

Acknowledgment

Authors would like to thank Ching-Yen Chung for his inputs.

References

[1] S. A. Dudani, “The Distance-weighted k-Nearest Neighbor Rule,” *IEEE Transactions on System, Man, and Cybernetics*, Vol. SMC-6, pp. 325-327, 1976.

[2] J. Zavrel, “An empirical re-examination of weighted voting for K-NN,” In: Daelemans W, Flach P, van den Bosch A (eds) *Proceedings of the 7th Belgian-Dutch Conference on Machine Learning*, Tilburg, pp 139-148, 1997.

[3] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statist. Surv*., vol. 4, pp.40 -79 2010

[4] http://robjhyndman.com/hyndsight/crossvalidation/

[5] M. Kintner-Meyer, K. Schneider, R. Pratt, “Impacts Assesment of Plug-in Hybrid Vehicles on Electric Utilities and Regional U.S. Power Grids Part 1: Technial Analysis”, PNNL, *Jrnl. of EUEC*, Paper #04, Volume 1, 2007 [Online]. Avilable: http://www.euec.com/getattachment/euecjournal/Paper_4.pdf.aspx

[6] M. Majidpour, W.P. Chen, “Grid and Schedule Constrained Electric Vehicle Charging Algorithm Using Node Sensitivity Approach”, *Proc. 2012 Intl. Conf. Connected Vehicles and Expo (ICCVE)*, pp. 304-310.

[7] C. Chung, A. Shepelev, C. Qiu, C. Chu, R. Gadh, "Design of RFID Mesh Network for Electric Vehicle Smart Charging Infrastructure", *IEEE RFID TA 2013*, Johor Bahru, Malaysia, 4-5 September, 2013.

[8] S. Mal, A. Chattopadhyay, A. Yang, R. Gadh, "Electric vehicle smart charging and vehicle-to-grid operation", *Intl Journal of Parallel, Emergent and Distributed Systems*, vol. 27, no. 3. March 2012.

[9] C.Chung, E. Youn, J. Chynoweth, C. Qiu, C. Chu, R. Gadh, "Safety Design for Smart Electric Vehicle Charging with Current and Multiplexing Control", *2013 IEEE International Conference on Smart Grid Communications,* Vancouver, Canada, 21-24 October, 2013.

[10] C. Chung, J. Chynoweth, C. Qiu, C. Chu, R. Gadh, "Design of Fast Response Smart Electric Vehicle Charging Infrastructure", *IEEE Green Energy and Systems Conf.*, Long Beach, CA, Nov 25, 2013.

[11] Alternative Fuels Data Center, U.S. Department of Energy, [Online]. Available http://www.afdc.energy.gov/fuels/stations_counts.html

[12] N.K. Ahmed, A.F. Atiya, N. El Gayar, H. El-Shishiny, “An empirical comparison of machine learning models for time series forecasting”, *Econometric Reviews*, 29 (2010), pp. 594–621.

[13] T. G. Dietterich, “Machine Learning for Sequential Data: A Review”, *Springer Lecture Notes in Computer Science*, vol.2396, 2002, pp.15-30.

[14] GEP Box, GM Jenkins, GC Reinsel, *Time series analysis: forecasting and control*, Wiley Series in Probability and Statistics, 2013 (4th ed.)

[15] J. D. Hamilton, *Time Series Analysis*, Princeton University Press, 1994.

[16] K.N. Kumar, P.H. Cheah, B. Sivaneasan, P.L. So, D.Z.W. Wang, “Electric vehicle charging profile prediction for efficient energy management in buildings”, in *Proc. 2012 IEEE Conference on Power & Energy,* pp. 480 – 485.

[17] A. Aabrandt , P. B. Andersen , A. B. Pedersen , S. You , B. Poulsen , N. O’Connell and J. Ostergaard "Prediction and optimization methods for

electric vehicle charging schedules in the EDISON project", *IEEE Power Energy Soc. Innovative Smart Grid Tech. Conf.*, pp.1-7, 2012.

[18] R. Shankar, J. Marco, "Method for estimating the energy consumption of electric vehicles and plug-in hybrid electric vehicles under real-world driving conditions" *IEEE Intelligent Transport Systems*, vol. 7, pp. 138 – 150, March 2013

[19] A. Ashtari , E. Bibeau , S. Shahidinejad and T. Molinski "PEV charging profile prediction and analysis based on vehicle usage data", *IEEE Trans. Smart Grid*, vol. 3, no. 3, pp.341 -350, 2012

[20] M. Alizadeh, A. Scaglione, J. Davies, K. S. Kurani, "A Scalable Stochastic Model for the Electricity Demand of Electric and Plug-In Hybrid Vehicles", *IEEE Trans. Smart Grid*, no. 99, pp.1-13, Sept 2013

[21] Y. Q. Li, Z. H. Jia, F. L. Wang, Y. Zhao, "Demand Forecast of Electric Vehicle Charging Stations Based on User Classification", *Applied Mechanics and Materials*, pp. 291-294, 2013

[22] J. Wang, K. H. Wu, F. Wang, Z. H. Li, Q. S. Niu, Z. Z. Liu, "Electric Vehicle Charging Station Load Forecasting and Impact of the Load Curve", 2012, *Applied Mechanics and Materials*, 229-231, 853

[23] G. Seymour, *Predictive Inference*. NY: Chapman and Hall, 1993

[24] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection". *Proc. of the Fourteenth International Joint Conference on Artificial Intelligence* 2 (12): pp.1137–1143, 1995.

[25] C. M. Bishop, N. M. Nasrabadi, *Pattern recognition and machine learning*. Vol. 1. New York: springer, 2006.

[26] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression". *The American Statistician* 46 (3): pp. 175–185, 1992.

[27] G. Bontempi, S. Ben Taieb, and Y. Le Borgne. "Machine Learning Strategies for Time Series Forecasting." In *Business Intelligence*, pp. 62-77. Springer Berlin Heidelberg, 2013.

[28] D. M. Allen, "The relationship between variable selection and data augmentation and a method for prediction." *Technometrics* 16, no. 1 (1974): 125-127.

[29] Ben Taieb, S., Bontempi, G., Atiya, A.F., Sorjamaa, A. "A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition", (2012) *Expert Systems with Applications*, 39 (8), pp. 7067-7083

[30] M. Majidpour, C. Qiu, C-Y. Chung, P. Chu, R. Gadh, H. Pota, "Fast Demand Forecast of Electric Vehicle Charging Stations for Cell Phone Application", in *Proc. IEEE/PES General Meeting, 27-31 July 2014, Washington, D.C.,USA,* in press.

[31] M. Aizerman, E. Braverman, L. Rozonoer, "Theoretical foundations of the potential function method in pattern recognition learning", *Automation and Remote Control* 25: 821–837, 1964.

[32] M. Sechilariu , B. Wang and F. Locment "Building integrated photovoltaic system with energy storage and smart grid communication", *IEEE Trans. Ind. Electron.*, vol. 60, no. 4, pp.1607 -1618, 2013.

[33] F. Kennel , D. Gorges and S. Liu "Energy management for smart grids with electric vehicles based on hierarchical MPC", *IEEE Ind. Informat.*, vol. 9, no. 3, pp.1528 -1537, 2013.

[34] C. E. Borges , Y. K. Penya and I. Fernandez "Evaluating combined load forecasting in large power systems and smart grids", *IEEE Ind. Informat.*, vol. 9, no. 3, pp.1570 -1577, 2013.

[35] L. Barote , C. Marinescu and M. N. Cirstea "Control structure for single-phase stand-alone wind-based energy sources", *IEEE Trans. Ind. Electron.*, vol. 60, no. 2, pp.764 -772, 2013.

[36] H. Kanchev , D. Lu , F. Colas , V. Lazarov and B. Francois "Energy management and operational planning of a microgrid with a PV-based active generator for smart grid applications", *IEEE Trans. Ind. Electron.*, vol. 58, no. 10, pp.4583 -4592, 2011.