Liu *et al. Hum. Cent. Comput. Inf. Sci.* (2020) 10:34
https://doi.org/10.1186/s13673-020-00239-5

Human-centric Computing and Information Sciences

RESEARCH Open Access

Cloud spot instance price prediction using *k*NN regression

Wenqiang Liu[†], Pengwei Wang[*†], Ying Meng, Caihui Zhao and Zhaohui Zhang

*Correspondence:
wangpengwei@dhu.edu.cn
[†]W. Liu and P. Wang—co-first author
School of Computer Science and Technology, Donghua University, 201620 Shanghai, China

Abstract

Cloud computing can provide users with basic hardware resources, and there are three instance types: reserved instances, on-demand instances and spot instances. The price of spot instance is lower than others on average, but it fluctuates according to market demand and supply. When a user requests a spot instance, he/she needs to give a bid. Only if the bid is not lower than the spot price, user can obtain the right to use this instance. Thus, it is very important and challenging to predict the price of spot instance. To this end, we take the most popular and representative Amazon EC2 as a testbed, and use the price history of its spot instance to predict future price by building a *k*-Nearest Neighbors (*k*NN) regression model, which is based on our mathematical description of spot instance price prediction problem. We compare our model with Linear Regression (LR), Support Vector Machine Regression (SVR), Random Forest (RF), Multi-layer Perception Regression (MLPR), gcForest, and the experiments show that our model outperforms the others.

Keywords: Cloud, Amazon EC2, Spot instances, Price prediction, Machine learning, AI, *k*-Nearest Neighbors (*k*NN) Regression model

Introduction

With the amount of data rapidly increasing, many applications need higher-performance hardware to support their running. However, for most individuals and organizations, these hardware are too expensive, and their budgets are limited. Consequently, the cloud computing price scheme based on pay-as-you-go is a very suitable choice for users to migrate their applications to the cloud, and using cloud can reduce the cost of purchasing and maintaining hardware. There are three major service models for cloud computing: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [1]. Amazon Elastic Compute Cloud (EC2) [2] is one of the representatives of IaaS, which can provide users with basic hardware resources, such as CPU, network, memory and storage.

Amazon EC2 provides users with many purchase options. They can be divided into three categories: reserved instances, on-demand instances and spot instances [2]. Reserved instances allow users to purchase a long-term right to use instances at a lower price, which are cost-effective for long-term applications. On-demand instances cost more than reserved instances, however users can purchase it according to the actual

running time of their applications, so they are suitable for short-term applications. Due to the fluctuation of user requirements, Amazon sets a lot of redundant resources to respond to the peak of users' requirements, but in most of time, many resources are idle. Therefore, in December 2009, Amazon proposed a new type of instances called spot instances [3] to sell these idle resources in order to improve resource utilization.

Spot instances allow users to propose a bid, which is the maximum price user can afford. As we all know, the spot instance price is mainly affected by the fluctuation of market supply and demand. When the price of spot instance is not higher than user's bid, the user can obtain the right to use this instance. In this case, the price paid by user is not his bid, but the actual price of the spot instance. When the price of a spot instance changes to be higher than the user's bid, the user's instance will be interrupted by force. In this case, Amazon will give the user a two-minute warning time [3], during which the user can save or migrate his data. Figure 1 shows the spot price history of c4.xlarge instance in us-east-2a region from October 27 to November 26, 2017, and this spot price is fluctuating all the way. If user's bid is just equal to on-demand price, the bid is usually successful, for the reason that the spot price is lower than bid for most of the time. But at time points a and b in Fig. 1, the spot price is higher than bid, so out-of-bid event will happen and user's instance will be interrupted.

Amazon believes that fair use of spot instances can save up to 90% of the cost of on-demand instances [3]. What's more, we can see from Fig. 1 that the cost of using spot instances is indeed less than on-demand instances. However, due to the large fluctuation of spot instance price, out-of-bid events, like failing to bid, may happen. Therefore, if the price of spot instance can be predicted accurately in advance, users will effectively solve the problems of bid setting and instance selection. Thus, it will help users save a considerable amount of money, and improve reliability obviously. In addition, users can know the price trend of spot instance in the future by predicting the price in advance, and can reasonably arrange their purchase time and give a suitable bid to avoid high cost caused by high bid and instance unavailability caused by low bid. Therefore, the future price needs to be predicted based on the historical price data. The user can obtain the historical price of spot instance on Amazon EC2 dashboard [4], or they can also obtain the

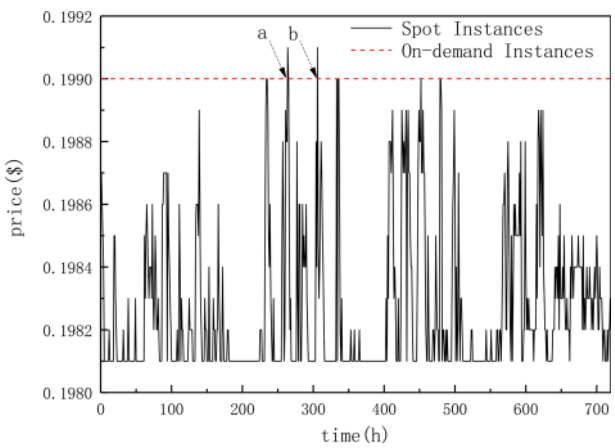

Fig. 1 Spot price in us-east-2a c4.xlarge instance (Oct, 27 to Nov, 26 2017)

historical price through some methods provided by Amazon, like Boto, an AWS SDK for Python [5], and the AWS Command Line Interface (AWS CLI) [6], an open source tool built on top of Boto which provides a consistent interface for interacting with all parts of Amazon Web Services (AWS).

The spot instance was firstly proposed by Amazon, and then, other cloud service providers also proposed spot instances, such as Alibaba and Tencent. Amazon's spot instance is still the most popular, widely used and representative in the cloud market. Thus, in this paper, we propose a price prediction method for spot instance, and taking Amazon as the representative to elaborate the method. The whole price prediction process, especially the data preprocessing technologies such as resampling and sliding window, and the evaluation indexes for the price prediction of spot instance, can be completely and easily applied to other spot instances.

The contributions of this paper are mainly as follows:

1. A price prediction model based on *k*-Nearest Neighbors (*k*NN) regression is proposed to predict the future price of cloud spot instances.
2. The representative Amazon AWS is taken as a testbed, and the historical price data of spot instance is obtained through AWS CLI [6]. An innovative sliding window method is adopted to preprocess the data.
3. Taking the different real cases into account, we implement and discuss the spot instance price prediction in two scenarios: 1-day-ahead and 1-week-ahead. The accuracy of our model is verified by comparing with several other models.

The rest of this paper is organized as follows. In "Related work" section, we introduce the related work on price prediction. A mathematical description of the spot price prediction problem is given in "Problem definition" section. Then in "Proposed method" section, we elaborate on the price prediction model based on *k*NN regression. In "Experiments and discussions" section, we describe the experiment setup and perform some experiments to verify the effectiveness of our model. Finally, some conclusions and future works are presented in "Conclusions" section.

Related work

Due to the high complexity of cloud data centers, Fernández-Cerero et al. [7, 8] demonstrate that we are often unable to predict the performance of a data center. In contrast, prices are often predictable, especially for spot instances. Since this kind of instance was proposed by Amazon, more and more researchers are trying to analyze and predict its price. The purpose is to help users to understand the price characteristics of spot instances, and then design a reasonable bidding scheme and a combination of instance selection solutions. Low monetary cost is one of the most important metrics and driving forces for users to using cloud services and hosting their data into cloud, which has been widely studied in cloud instance selection [9–11], cloud storage [12–14], and scientific workflow scheduling [15, 16]. Therefore, it is important to understand and predict the prices of cloud instances, especially the spot instance.

Agmon et al. [17] analyze the actual price of spot instance and build a price model that could be consistent with the existing price trajectory by designing the price reversely.

The result shows that the price of spot instance is likely to be generated most of the time at random within a tight price range via a dynamic hidden reserve price mechanism. During the time of the authors' study, Amazon may set the spot price via a random AR(1) hidden reserve price mechanism.

Kumar et al. [18] presented a survey of spot instance pricing in cloud ecosystem. An insight into the Amazon spot instances and its pricing mechanism has been presented for better understanding of the spot ecosystem. A large amount of important research papers related to price prediction and modeling, spot resource provisioning, bidding strategy designing etc. are summarized and categorized in this survey. There have been many studies about the prediction of spot instance price. We divide them into two major categories, based on the methods that are used.

Price prediction models based on statistical time series analysis

Javadi et al. [19] study the features of Amazon's spot instances, and analyze the historical price in hour-in-day and day-in-week. At the same time, a statistical model based on Gaussian distribution is proposed to fit these two distributions. The model contains three to four components to better capture the dynamic changes in price and the duration of price changes for each instance. It is proved through simulation experiments that it has better accuracy in real work environments.

Cai et al. [20] think spot instance price usually has switching regimes, and traditional autoregressive models are not suitable for their forecasting. So they propose two Markov regime-switching autoregressive models: DMRA-AR-L and DMRA-AR-SW. They use 144 days of spot instance price history to do some experiments, and the results show that DMRA-AR-L performs the best when the forecast period is shorter than 24h in most cases, while DMRA-AR-SW is best when the forecast period increases.

In [21], a SARIMA model is established by analyzing historical prices of spot instance. By comparing this model with other price prediction models like mean and naive, using 11 months of data, SARIMA has better accuracy.

Price prediction models based on machine learning

Mishra et al. [22] use linear regression to deal with the spot price prediction problem. The price history length they used is 90 days. Wallace et al. [23] and Agarwal et al. [24] use artificial neural network to predict the price of spot instance. Wallace et al. [23] use 7 mouths of historical data to predict spot price based on a standard multi-layer perception, but only predict the price for one ahead. Agarwal et al. [24] use 90 days of historical price and establish LSTM model to predict the spot instance price. The experimental results show that the effect of [24] is better than [22] and [23].

Khandelwal et al. [25] use 12 months of Amazon EC2 spot historical price to predict the price of 1-day-ahead and 1-week-ahead prices for spot instance by establishing a random forests regression. Experiments are performed and compared with neural network, support vector machine regression, regression tree and other methods. The results indicate that the effect of random forests regression is better than other methods.

Neto et al. [26] proposed a heuristic model that uses checkpoint and restore techniques, and takes price change traces of spot instances as input in a machine learning and statistical

model to predict time to revocation. By using a bid strategy and the observed price variation history, their model can be able to predict revocation time with high levels of accuracy.

Singh et al. [27] use the data of each month as a global trend and the previous day's data as a local periodic change to establish a price prediction model. Simultaneously, they use the gradient descent method to adjust the parameters of the model and use 9 months of spot instance price data to do some experiments. This paper separately forecasts spot price examples for short-term (1 h) and long-term (1 day and 1 week). The results show that the prediction effect in short-term is obviously better than the long-term prediction.

In this paper, we innovatively use a sliding window method to preprocess the historical price data which is obtained by using AWS CLI [6]. Then, we build a price prediction model based on *k*-Nearest Neighbors (*k*NN) regression to predict the future price of spot instance. Some experiments are performed to discuss the spot instance price prediction in two scenarios: 1-day-ahead and 1-week-ahead. The accuracy of our model is verified by comparing with several other models.

Problem definition

The historical price of spot instance s is represented as a vector $\boldsymbol{p} = [p_1, p_2, \ldots, p_{l_p}]$, $\boldsymbol{p} \in \boldsymbol{R}^{l_p}$, where l_p is the length of $\boldsymbol{p}$, in other words, the length of historical price. For example, in Fig. 2a, the historical price is displayed when the sampling time is 1 day (24 h) and the time interval is 1 h, and in this case, the historical price is represented as a vector ($\boldsymbol{p} = [p_1, p_2, \ldots, p_{24}]$) and its length is 24 ($l_p = 24$).

In this paper, we use sliding window to divide the price data. We use l_{sw} and l_{pt} respectively to denote the length of sliding window and the length of time window to be predicted. In order to ensure the accuracy of data division, each sliding length of the sliding window is set to l_{pt}. So the number of samples we get by sliding the window is:

$$n_s = \lfloor \frac{l_p - l_{sw}}{l_{pt}} \rfloor \tag{1}$$

In Eq. 1, we get the sample number by rounding down the result, because there is excess of data, and it is necessary to move the sliding window in reverse. Like in Fig. 2b, the number of samples is 3 ($n_s = 3$) when the length of sliding window is 12 ($l_{sw} = 12$) and the length of time window to be predicted is 4 ($l_{pt} = 4$). However, in Fig. 2c the number of samples is 2 ($n_s = 2$) when the length of sliding window is 10 ($l_{sw} = 10$) and the length of time window to be predicted is 5 ($l_{pt} = 5$), and in this case, there is excess of data. We use $\boldsymbol{D} = \{D_1, D_2, \ldots, D_{n_s}\}$ to denote the sample set after sliding, where $D_i = (\boldsymbol{x_i}, \boldsymbol{y_i})$ is the sample data formed after sliding $n_s - i + 1$ times, and $\boldsymbol{x_i} \in \boldsymbol{R}^{l_{sw}}$ is the sample D_i's vector, which is the data in sliding window, and $\boldsymbol{y_i} \in \boldsymbol{R}^{l_{pt}}$ is sample D_i's label vector. Like in Fig. 2b the sample set is $\boldsymbol{D} = \{(\boldsymbol{x_1}, \boldsymbol{y_1}), (\boldsymbol{x_2}, \boldsymbol{y_2}), (\boldsymbol{x_3}, \boldsymbol{y_3})\}$, and in Fig. 2c, $\boldsymbol{D} = \{(\boldsymbol{x_1}, \boldsymbol{y_1}), (\boldsymbol{x_2}, \boldsymbol{y_2})\}$.

$$\boldsymbol{x_i} = [p_{l_p-(n_s-i+1)l_{pt}-l_{sw}+1}, \ldots, p_{l_p-(n_s-i+1)l_{pt}}] \tag{2}$$

$$\boldsymbol{y_i} = [p_{l_p-(n_s-i+1)l_{pt}+1}, \ldots, p_{l_p-(n_s-i)l_{pt}}] \tag{3}$$

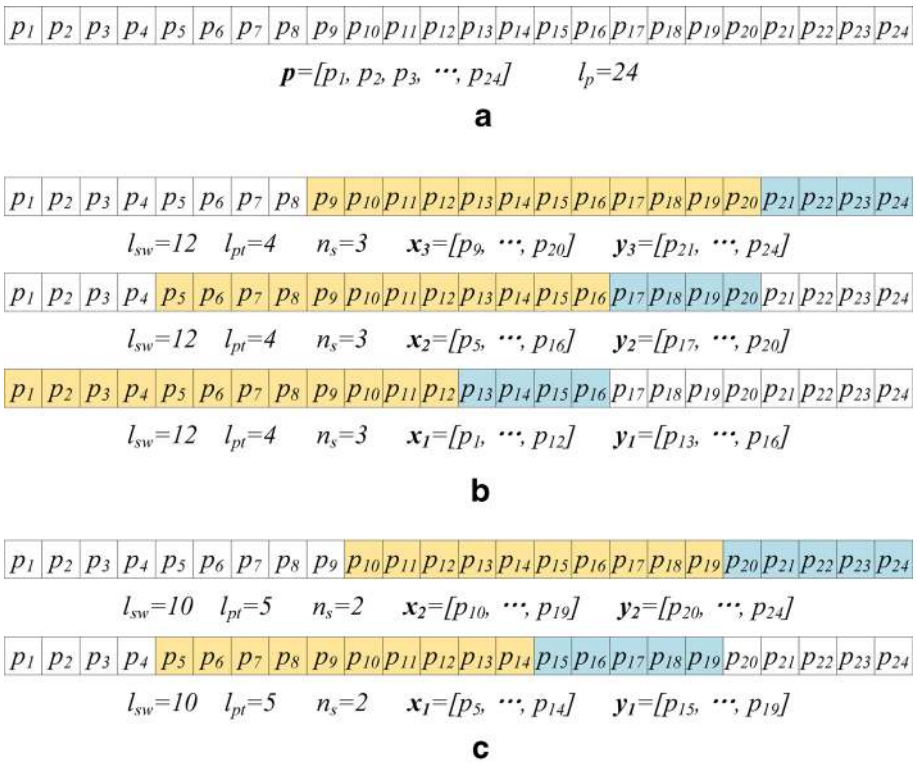

Fig. 2 An example of using sliding window to divide the price data. **a** Description of price data when the time interval is 1 hour and the sampling time is one day ($\boldsymbol{p} = [p_1, p_2, \ldots, p_{24}]$, $l_p = 24$). **b** Description of the sample set generation ($l_{sw} = 12, l_{pt} = 4$), and in this case, the data is just being used completely. **c** Description of the sample set generation ($l_{sw} = 10, l_{pt} = 5$), and in this case, there is excess of data

The goal of this paper is to predict the spot instance price, namely, it needs to find a function f satisfied the following formula:

$$\boldsymbol{y_i} = f(\boldsymbol{x_i}), 1 \leq i \leq n_s \tag{4}$$

Proposed method

To predict the spot instance price, we construct kNN regression model to predict $\boldsymbol{y}$ based on new input $\boldsymbol{X}$.

For the new input $\boldsymbol{X}$, kNN regression will find k nearest samples in the training set, and the average of their values is used as the predicted value corresponding to $\boldsymbol{X}$, marked as $\hat{\boldsymbol{y}}$. In our model, we have to determine the following two problems.

The first one is the distance function. In our model, when judging the distance between the new input $\boldsymbol{X}$ and the training sample $\boldsymbol{x_i}(1 \leq i \leq n_s)$, we choose Euclidean distance as distance function, which is shown as follows:

$$dist(\boldsymbol{x_i}, \boldsymbol{X}) = \sqrt{\sum_{j=1}^{l_{sw}} (x_i^j - x^j)^2}, 1 \leq i \leq n_s \tag{5}$$

The other one is how to improve computational efficiency when the model searches k nearest neighbors on training data. Since the simplest linear scan is very time-consuming, we use k-dimensional tree(k-d tree) as a fast kNN search algorithm. The k-d tree is a space-partitioning data structure for organizing points in a k-dimensional space. In element search, the average time complexity of k-d tree is $O(logn)$, and it is $O(n)$ in the worst case. However, but the average time complexity of linear scanning is $O(n)$, thus k-d tree is chosen.

In kNN, the model training process is the k-d tree building process. In algorithm **BuildKDTree**, k-d tree is built to complete training process. The dimension of the maximum variance is selected (line 4), where the median sample $\bar{D}$ of $\boldsymbol{D}$ is taken as a current node (line 5 and line 7). Then, the remaining samples $\boldsymbol{D'}$ continue to be divided (line 9–12) until it becomes an empty set (line 1–3). At last, we can get the k-d tree *kdTree* (line 13). Regarding time complexity, the k-d tree is built recursively in algorithm **BuildKDTree**, and the time complexity of recursion is $O(\log n_s)$. Since the dimension of the maximum variance is calculated in each recursion, the time complexity of this calculation is $O(l_{sw}n_s)$. Thus the overall time complexity of algorithm **BuildKDTree** is $O(l_{sw}n_s \log n_s)$.

In algorithm **Search**, the set of k nearest nodes of input $\boldsymbol{X}$ will get from kNN search process. Firstly, for input $\boldsymbol{X}$, we need to find k nodes in $\boldsymbol{D}$ (line 2–4). In nearest nodes set, if the distance between the farthest node ***maxDN*** against $\boldsymbol{X}$ is greater than ***node***, ***maxDN*** should be replaced by ***node*** (line 5 to line 9). Meanwhile, if in *axis* dimension the value of $\boldsymbol{X}$ is greater than ***node***, we should search nearest node in ***node***'s left subtree (line 13–16). In this case, if the distance between $x[axis]$ and $node[axis]$ is less than the maximum distance between k nearest nodes set and $\boldsymbol{X}$, we should search neatest node in ***node***'s right subtree (line 17–19). This is a similar case when $x[axis]$ is more than $node[axis]$ (line 21–27). Finally, this algorithm will return the set of k nearest nodes. In terms of time complexity, because algorithm **Search** needs to get the point that has maximum distance against $\boldsymbol{X}$ in nearest nodes set, its time complexity is $O(kl_{sw})$. The search process for the k-d tree needs to traverse all nodes in the worst case, and its worst time complexity is $O(n_s)$. Thus the overall time complexity of algorithm **Search** is $O(kl_{sw}n_s)$.

Algorithm BuildKDTree: Build k-d tree

Input: the training set $\boldsymbol{D}$
Output: $kdTree$
1: **if** $\boldsymbol{D} = \emptyset$ **then**
2: **return** $NULL$
3: **end if**
4: $axis$=the dimension of the maximum variance
5: $\bar{D}$=the median sample of $\boldsymbol{D}$ in $axis$ dimension
6: $\boldsymbol{D'} = \boldsymbol{D} - \bar{D}$
7: $node.data = \bar{D}$
8: $node.split = axis$
9: $\boldsymbol{D_l} = \{D_l | D_l \in \boldsymbol{D'} \quad and \quad D_l[axis] \leq \bar{D}[axis]\}$
10: $\boldsymbol{D_r} = \{D_r | D_r \in \boldsymbol{D'} \quad and \quad D_r[axis] > \bar{D}[axis]\}$
11: $node.lChild = BuildKDTree(\boldsymbol{D_l})$
12: $node.rChild = BuildKDTree(\boldsymbol{D_r})$
13: **return** $node$

Algorithm *k***NN** aims to realize *k*NN regression. Firstly, we construct a *k*-d tree from $\boldsymbol{D}$ by using algorithm **BuildKDTree** (line 1), and then $\boldsymbol{X}$'s *k* nearest samples can be found via algorithm **Search** (line 2). Finally, by taking the average of *k* samples (line 3), we can get the prediction price $\hat{\boldsymbol{y}}$ of $\boldsymbol{X}$ (line 4). Because the time complexity of line 1 is $O(l_{sw}n_s \log n_s)$, line 2 is $O(kl_{sw}n_s)$ and line 3 is $O(k)$, the time complexity of algorithm *k***NN** is $O(kl_{sw}n_s)$.

Experiments and discussions

To better evaluate the performance of our model, extensive experiments are conducted in this section. We first introduce the experiment setup, and then describe the experimental results.

Experiment setting

In this section, we first introduce the environments of our experiment, then describe the data acquisition method and data preprocessing process. Finally, the compared algorithms and measurement method will be expressed.

Algorithm Search: Search k-d tree

Input: $kdTree, \boldsymbol{x}, k, nearstNodes = \emptyset$
Output: $nearstNodes$
1: $node = kdTree$
2: **if** $nearstNodes.size < k$ **then**
3: $nearstNodes.add(node)$
4: **end if**
5: $maxDN$=max distance node in $nearstNodes$ against $\boldsymbol{x}$
6: **if** $Dis(maxDN, \boldsymbol{x}) > Dis(node, \boldsymbol{x})$ **then**
7: $nearstNodes.delete(maxDN)$
8: $nearstNodes.add(node)$
9: **end if**
10: $axis = node.split$
11: $value = \boldsymbol{x}[axis]$
12: $median = node[axis]$
13: **if** $value \leq median$ **then**
14: **if** $node.lChild! = NULL$ **then**
15: $Search(node.lChild, \boldsymbol{x}, nearstNodes)$
16: **end if**
17: **if** $node.rChild! = NULL \&\& (median - value) \leq maxDist(nearstNodes, \boldsymbol{x})$ **then**
18: $Search(node.rChild, \boldsymbol{x}, nearstNodes)$
19: **end if**
20: **else**
21: **if** $node.rChild! = NULL$ **then**
22: $Search(node.rChild, \boldsymbol{x}, nearstNodes)$
23: **end if**
24: **if** $node.lChild! = NULL \&\& (median - value) \leq maxDist(nearstNodes, \boldsymbol{x})$ **then**
25: $Search(node.lChild, \boldsymbol{x}, nearstNodes)$
26: **end if**
27: **end if**
28: **return** $nearstNodes$

Experimental environment Experiments are performed on a GUN Linux Operating System with an Intel(R) Core(TM)i5-7500 at 3.40 GHz and 16 GB of RAM memory. Moreover, we use Python3.5 programming language to implement the algorithms.

Table 1 Abbreviations used for different regions

R#	Region	Location
R1	us-east-2a	US East (Ohio)
R2	ap-northeast-2a	Asia Pacific (Seoul)
R3	ap-south-1a	Asia Pacific (Mumbai)
R4	ca-central-1a	Canada (Central)

Table 2 Abbreviations used for different instances

I#	Instance	vCPU	Memory (GiB)	Bandwidth
I1	c4.large	2	3.75	500 Mbps
I2	c4.xlarge	4	7.5	750 Mbps
I3	c4.2xlarge	8	15	1000 Mbps
I4	m4.large	2	8	450 Mbps
I5	m4.xlarge	4	16	750 Mbps
I6	m4.2xlarge	8	32	1000 Mbps
I7	r4.large	2	15.25	Up to 10 GB
I8	r4.xlarge	4	30.5	Up to 10 GB
I9	r4.2xlarge	8	61	Up to 10 GB

Algorithm k**NN**

Input: $\boldsymbol{D}, \boldsymbol{x}, k$
Output: $\hat{\boldsymbol{y}}$
1: $kdTree = BuildKDTree(\boldsymbol{D})$
2: $nearstNodes = Search(kdTree, \boldsymbol{x}, k)$
3: $\hat{\boldsymbol{y}} = average(\boldsymbol{nearstNodes.y})$
4: **return** $\hat{\boldsymbol{y}}$

Experimental data Amazon provides SDK which can help people to get the spot price history from web. Thus, we use AWS CLI [6] to access Amazon EC2 and get 88 days of historical price, from September 1, 2017 to November 27, 2017. The regions and instance types involved are described in Tables 1 and 2.

Because the interval of price change is uncertain, we re-sample the historical price in 1-h unit. Users only need to consider the maximum price per hour to make a bid, so we use the maximum value of selected sampling unit as the re-sampling value. After re-sampling, we get 2112 values in every instance (24×88 (days)) and 76,032 values in total (2112×4(regions)$\times$9(instances)). In this paper, we divide the dataset into 80% and 20% for the training set and test set, respectively.

Evaluated algorithms In this paper, we use 5 algorithms as comparison methods which are Linear Regression (LR) [22], Support Vector Machine Regression (SVR) , Random Forest (RF) [25], Multi-layer Perception Regression (MLPR) [23] and gcForest [28].

Performance metrics Mean Absolute Percentage Error (*MAPE*) is a commonly used measurement method for time series forecasting problems. It can measure the outcome of a predictive model. *MAPE* is defined as follows:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n} APE_i \tag{6}$$

where n is the sample number of test set, and APE is absolute percentage error whose definition is as follows:

$$APE = \frac{1}{l_{pt}}\sum_{i=1}^{l_{pt}} \frac{|y_i - \hat{y}_i|}{y_i} \times 100\% \tag{7}$$

In this paper, to make a quantitative estimation, we use $MAPE_{m\%}$ as performance metrics. $MAPE_{m\%}$ represents the number of results whose APE is less than or equal to $m\%$ as a percentage of the number of total results, which is calculated as follows:

$$MAPE_{m\%} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(APE_i - m\%), \mathbf{1}(x) = \begin{cases} 1 & x \le 0 \\ 0 & x > 0 \end{cases} \tag{8}$$

In this paper, the selected value of m is 5.

Experimental results and discussions

Currently, since many applications require a lot of time and money to run, the deployment of local applications to the cloud platform can reduce the cost like hardware purchasing, device cooling, hardware maintenance and so on. For example, if a user deploys a web crawler system, the running time may be several hours or even days. Similarly, in video rendering, videos of different lengths may need different time. Short videos may take several to tens of hours, but larger videos may take several days. Users need to consider the application's possible running time when migrating their own applications to the cloud, and estimate the price in advance during this period to help in successful bidding. Therefore, in this paper, we discuss the spot instance price prediction in two scenarios: 1-day-ahead and 1-week-ahead.

One-day-ahead

Parameter setting In order to maximize the effect of the proposed model, we need to determine the value of k and the sliding window length l_{sw}. We conduct a lot of experiments with different k and l_{sw} respectively. The experimental results are shown in Fig. 3.

In this figure, the result of $k = 1$ is the best obviously. $k = 1$ is the nearest neighbor regression. With the increase of k, the estimation error will increase, because the training sample that is far away from the input $\boldsymbol{X}$ will affect the result to be worse. So we choose the k to equal 1.

With the length of sliding window increasing, higher dimensions will not increase the advantages of different examples, but lead to the reduction of accuracy of the results. From the Fig. 3, we can see that the best effect is when the length of sliding window is equal to the length of prediction length.

Thus we choose the length of sliding window $l_{sw} = 24$.

Experimental result Based on the above settings, we make a forecast for 1-day-ahead. The results are shown in Tables 3 and 4.

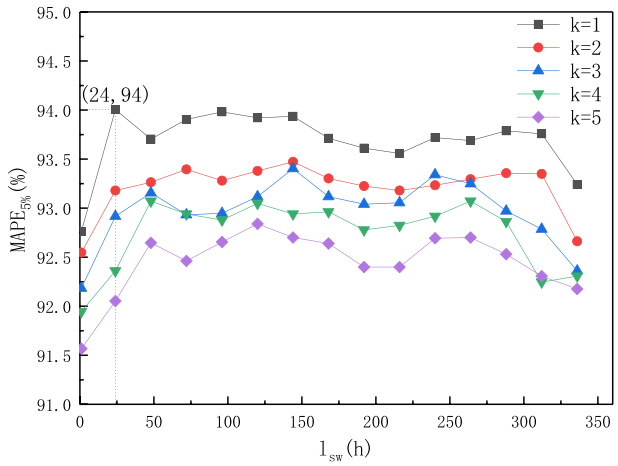

Fig. 3 $MAPE_{5\%}$ variations in different l_{sw} and k on 1-day-ahead

Table 3 Comparison of different methods for regions on 1-day-ahead

R#	*k* NN	LR	SVR	RF	MLPR	gcForest
R1	99.91	94.91	88.89	99.60	44.01	97.62
R2	97.59	81.91	71.45	88.15	45.59	88.52
R3	80.93	41.05	48.30	63.77	22.01	59.48
R4	97.59	79.72	85.71	90.93	37.16	89.81

Table 4 Comparison of different methods for instances on 1-day-ahead

I#	*k*NN	LR	SVR	RF	MLPR	gcForest
I1	94.58	55.76	43.89	73.12	22.08	63.68
I2	85.62	56.60	56.46	61.25	27.22	63.19
I3	93.26	76.18	72.85	87.22	43.61	84.31
I4	99.86	89.38	94.65	97.29	36.32	94.93
I5	94.93	80.35	91.25	94.86	35.21	93.19
I6	93.06	75.76	89.44	92.99	45.00	89.58
I7	99.86	95.76	93.75	99.44	35.69	98.19
I8	90.69	70.00	63.89	83.40	41.46	88.61
I9	94.17	69.79	56.11	80.90	48.12	79.03

We can see from these two tables, *k*NN regression has the best results. RF and gcForest are the better ones among the methods being compared, and the $MAPE_{5\%}$ of them are respectively 85.61% and 83.86% on average, but *k*NN is 94.00% which can achieve about 10% improvement. The effect of MLPR and SVR are both bad, which is like [25].

One-week-ahead

Parameter setting We can see from Fig. 4, like 1-day-ahead prediction, the effect is best when k is equal to 1 and the length of sliding window is equal to the length of prediction window. So we select $k = 1$ and $l_{sw} = 168$ in 1-week-ahead prediction.

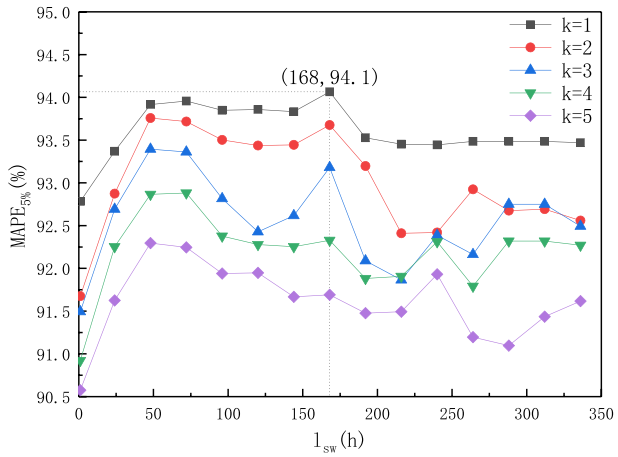

Fig. 4 $MAPE_{5\%}$ variations in different l_{sw} and k on 1-week-ahead

Experimental result Based on the above settings, we make a forecast for 1-week-ahead. The results are shown in Tables 5 and 6.

From these two tables, *k*NN regression can achieve best results except instance *I*7. For *I*7, the best result is RF whose value is 99.93%, 0.15% higher than *k*NN's 99.78%. But in $MAPE_{10\%}$, the results of *k*NN and RF are both 100%. So on the whole, *k*NN is the best method. RF and gcForest are the better ones among the methods being compared, and the $MAPE_{5\%}$ of them are respectively 86.50% and 89.80%. In comparison, *k*NN is 94.06% which can achieve about 6% improvement.

According to the mechanisms of spot instance, the final user is allowed to propose a bid. When the price of spot instance is not higher than this bid, the user can use this instance, otherwise the user fails. In addition, when the user is using a spot instance and the price

Table 5 Comparison of different methods for regions on 1-week-ahead

R#	*k*NN	LR	SVR	RF	MLPR	gcForest
R1	99.83	99.77	95.27	99.40	40.05	96.92
R2	97.45	89.58	86.51	93.55	40.34	92.59
R3	80.95	43.19	68.52	72.26	13.56	73.51
R4	98.02	76.03	95.70	96.03	26.72	96.16

Table 6 Comparison of different methods for instances on 1-week-ahead

I#	*k*NN	LR	SVR	RF	MLPR	gcForest
I1	95.31	58.26	72.25	85.42	13.17	78.87
I2	84.60	51.79	71.88	72.40	17.63	76.56
I3	93.53	80.51	87.43	89.36	31.47	89.73
I4	99.93	99.18	*99.93*	*99.93*	27.23	*99.93*
I5	95.01	80.13	94.57	94.79	37.28	94.79
I6	93.45	60.34	92.71	93.23	33.33	93.15
I7	99.78	99.33	98.96	*99.93*	22.77	99.26
I8	91.15	79.17	81.25	88.76	37.13	86.61
I9	93.82	85.57	79.54	88.99	51.49	89.29

becomes higher than the bid, out-of-bid will happen and the user's instance will be interrupted by force. However, the fluctuation of spot instance price brings great inconvenience and difficulty to final users. Thus, by predicting the future price of spot instance, the proposed method is useful and helpful for final users from following aspects: (a) setting a proper bid or designing a reasonable bidding scheme; (b) choosing the right time to purchase spot instance; (c) selecting the most appropriate spot instance or a combination of them; (d) avoiding out-of-bid events and instance unavailability.

Conclusions

Many cloud providers like Amazon provide users with three main types of instances: reserved instances, on-demand instances and spot instances. Compared with others, the price of spot instance is the lowest, but its fluctuating price is an obstacle for users. Predicting the price of spot instance in advance can help users to know the price trend in the future. Users can reasonably arrange the purchase time and give a suitable bid to avoid high cost caused by high bid and instance unavailability caused by low bid. So it is very important and challenging to predict the spot instance price in advance. In this paper, we give a mathematical description of spot instance prediction problem and use the price history of Amazon EC2 spot instance to predict future price by building a *k*NN regression model. What's more, to better evaluate the performance of our model, we use 88 days of spot instance price of 4 regions and 9 instances to perform many experiments. We compare our model with LR, SVR, RF, MLPR and gcForest. Evaluation results show that the $MAPE_{5\%}$ is up to 94.00% in 1-day-ahead prediction and 94.06% in 1-week-ahead, respectively. In both of these two scenarios, our method achieves better performance than other methods. The method proposed in this paper is applicable to the spot instance price prediction of other cloud providers. Helping users to select appropriate instances [9–11] based on price prediction and combining cloud data storage [12] with cloud instance types selection are two directions for the future work.

Acknowledgements
Our deepest gratitude goes to the anonymous reviewers for their careful work and thoughtful suggestions that have helped improve this paper substantially.

Authors' contributions
Conceptualization, WL, and PW; methodology, WL and PW; software, WL, PW and YM; validation, WL, PW, YM, CZ and ZZ; formal analysis, WL, PW and YM; writing—original draft preparation, WL, PW and YM; writing—review and editing, WL, PW, YM, CZ and ZZ; visualization, WL and PW. All authors read and approved the final manuscript.

Funding
This work was partially supported by the National Natural Science Foundation of China (NSFC) under Grant 61602109, DHU Distinguished Young Professor Program under Grant LZB2019003, Shanghai Sailing Program under Grant 16YF1400300, Fundamental Research Funds for the Central Universities, Shanghai Science and Technology Innovation Action Plan under Grant No. 19511101802, Natural Science Foundation of Shanghai under Grant No. 19ZR1401900, and the Special Fund of Shanghai Municipal Commission of Economy and Informatization under Grant 201801027.

Availability of data and materials
The experimental data is available at: https://github.com/liuwenqiang1202/AmazonSpotInstancePrice/blob/master/data-raw.rar.

Competing interests
The authors declare that they have no competing interests.

Received: 11 July 2019 Accepted: 13 July 2020
Published online: 09 August 2020

References

1. Mell P, Grance T (2011) The NIST definition of cloud computing. Natl Inst Stand Technol 6:50
2. Amazon EC2. https://aws.amazon.com/ec2. Accessed 1 Jan 2018
3. Amazon EC2 Spot Instances. https://aws.amazon.com/ec2/spot. Accessed 1 Jan 2018
4. Amazon EC2 Dashboard. https://console.aws.amazon.com/ec2. Accessed 1 Jan 2018
5. Boto3 Documentation. https://boto3.readthedocs.io. Accessed 1 Jan 2018
6. AWS Command Line Interface Documentation. https://aws.amazon.com/documentation/cli. Accessed 1 Jan 2018
7. Fernández-Cerero D, Varela-Vaca ÁJ, Fernández-Montes A, Gómez-López MT, Alvárez-Bermejo JA (2020) Measuring data-centre workflows complexity through process mining: the Google cluster case. J Supercomput 76:2449–2478
8. Fernández-Cerero D, Fernández-Montesa A, Ortegaa J, Jakóbikb A, Widlak A (2020) Sphere: simulator of edge infrastructures for the optimization of performance and resources energy consumption. Simul Model Pract Theory 101:101966
9. Wang P, Zhou W, Zhao C, Lei Y, Zhang Z (2020) A dynamic programming-based approach for cloud instance types selection and optimization. Int J Inf Technol Manag 19:358–375
10. Liu W, Wang P, Meng Y, Zhao Q, Zhao C, Zhang Z (2019) A novel model for optimizing selection of cloud instance types. IEEE Access 7:120508–120521
11. Liu W, Wang P, Meng Y, Zou G, Zhang Z (2019) A novel algorithm for optimizing selection of cloud instance types in multi-cloud environment. In: 2019 25th IEEE the international conference on parallel and distributed systems. IEEE: China, pp 167–170
12. Wang P, Zhao C, Zhang Z (2018) An ant colony algorithm-based approach for cost-effective data hosting with high availability in multi-cloud environments. In: 2018 15th IEEE International conference on networking, sensing and control, IEEE: China, pp 1–6
13. Wang P, Zhao C, Wei Y, Wang D, Zhang Z (2020) An adaptive data placement architecture in multi-cloud environments. Sci Program 1704258:12
14. Wang P, Zhao C, Liu W, Chen Z, Zhang Z (2020) Optimizing data placement for cost effective and high available multi-cloud storage. Comput Inform 39:1001–1032
15. Wang P, Lei Y, Agbedanu P, Zhang Z (2020) Makespan-driven workflow scheduling in Clouds using immune-based PSO Algorithm. IEEE Access 8:29281–29290
16. Sujana J, Revathi T, Priya T (2019) Smart PSO-based secured scheduling approaches for scientific workflows in cloud computing. Soft Computing 23:1745–1765
17. Agmon Ben-Yehuda O, Ben-Yehuda M, Schuster A, Tsafrir D (2013) Deconstructing amazon ec2 spot instance pricing. ACM Trans Econ Comput 3:16
18. Kumar D, Baranwal G, Raza Z, Vidyarthi D (2018) A survey on spot pricing in Cloud computing. J Netw Syst Manag 26:809–856
19. Javadi B, Thulasiramy RK, Buyya R (2011) Statistical modeling of spot instance prices in public cloud environments. In: 2011 Fourth IEEE international conference on utility and cloud computing. IEEE: Australia, pp 219–228
20. Cai Z, Li X, Ruiz R, Li Q (2018) Price forecasting for spot instances in Cloud computing. Future Gen Comput Syst 79:38–53
21. Alkharif S, Lee K, Kim H (2018) Time-series analysis for price prediction of opportunistic Cloud computing resources. 2018 7th international conference on emerging databases. Springer, Singapore, pp 221–229
22. Mishra AK, Yadav DK (2017) Analysis and prediction of Amazon EC2 spot instance prices. Int J Appl Eng Res 21:11205–11212
23. Wallace RM, Turchenko V, Sheikhalishahi M, Turchenko I, Shults V, Vazquez-Poletti JL, Grandinetti L (2013) Applications of neural-based spot market prediction for cloud computing. In: 2013 IEEE 7th international conference on intelligent data acquisition and advanced computing systems. IEEE: Germany, pp 710–716
24. Agarwal S, Mishra AK, Yadav DK (2017) Forecasting price of Amazon spot instances using neural networks. Int J Appl Eng Res 20:10276–10283
25. Khandelwal V, Chaturvedi A, Gupta CP (2020) Amazon EC2 spot price prediction using regression random forests. IEEE Trans Cloud Comput 8:59–72
26. Neto JPA, Pianto DM, Ralha CG (2018) A prediction approach to define checkpoint intervals in spot instances. In: Luo M, Zhang LJ (eds) Cloud computing—CLOUD 2018. Lecture Notes in Computer Science, vol 10967, pp 84–93
27. Singh VK, Dutta K (2015) Dynamic price prediction for amazon spot instances. In: 2015 48th Hawaii international conference on system sciences. IEEE: USA, pp. 1513–1520
28. Zhou Z, Feng J (2017) Deep forest: towards an alternative to deep neural networks. 2017 26th international joint conference on artificial intelligence. AAAI Press, New York, pp 3553–3559

Publisher's Note