AWS PredSpot: Machine Learning for Predicting the Price of Spot Instances in AWS Cloud

Alejandro Baldominos*, Yago Saez, David Quintana, Pedro Isasi

Computer Science and Engineering Department, Universidad Carlos III de Madrid, Leganés, Madrid (Spain)

Received 30 July 2020 | Accepted 25 January 2022 | Published 8 February 2022

Abstract

Elastic Cloud Compute (EC2) is one of the most well-known services provided by Amazon for provisioning cloud computing resources, also known as instances. Besides the classical on-demand scheme, where users purchase compute capacity at a fixed cost, EC2 supports so-called spot instances, which are offered following a bidding scheme, where users can save up to 90% of the cost of the on-demand instance. EC2 spot instances can be a useful alternative for attaining an important reduction in infrastructure cost, but designing bidding policies can be a difficult task, since bidding under their cost will either prevent users from provisioning instances or losing those that they already own. Towards this extent, accurate forecasting of spot instance prices can be of an outstanding interest for designing working bidding policies. In this paper, we propose the use of different machine learning techniques to estimate the future price of EC2 spot instances. These include linear, ridge and lasso regressions, multilayer perceptrons, K-nearest neighbors, extra trees and random forests. The obtained performance varies significantly between instances types, and root mean squared errors ranges between values very close to zero up to values over 60 in some of the most expensive instances. Still, we can see that for most of the instances, forecasting performance is remarkably good, encouraging further research in this field of study.

Keywords

Cloud Computing, Machine Learning, Prediction, Price Forecasting.

DOI: 10.9781/ijimai.2022.02.003

I. Introduction

Amazon Web Services (AWS) is an Amazon ecosystem comprising a large number of cloud services. This ecosystem is in a process of continuous growth, with new services or functionalities added every few months.

One of the most well-known AWS services is EC2 (Elastic Cloud Compute), an application that provides Infrastructure-as-a-Service (IaaS) for cloud computing. These services allow users to launch on-demand instances (virtual machines) in order to satisfy certain computational needs. This option is interesting when a user or company has a variable computing load, thus avoiding the need to acquire specific infrastructure whose administration and maintenance can become very expensive.

Besides on-demand instances, EC2 allows users to bid for computing capacity that is not in use. This enables users to establish a maximum bidding price and, in case they be the winner of the bid, then they are able to use the corresponding computational capacity. In EC2, these instances are called "spot instances". The hourly cost of spot instances can be significantly lower than on-demand instances; however, the instance will only belong to the user as long as the bid is higher than the spot price. In other case, the instance will be terminated and the user will not be able to access it anymore.

* Corresponding author.

E-mail address: abaldomi@inf.uc3m.es

AWS allows to query the price of spot instances in real time [1]. Additionally, users can study the historic evolution of EC2 instances of a certain type, up to three months in the past, as it can be seen in Fig. 1.

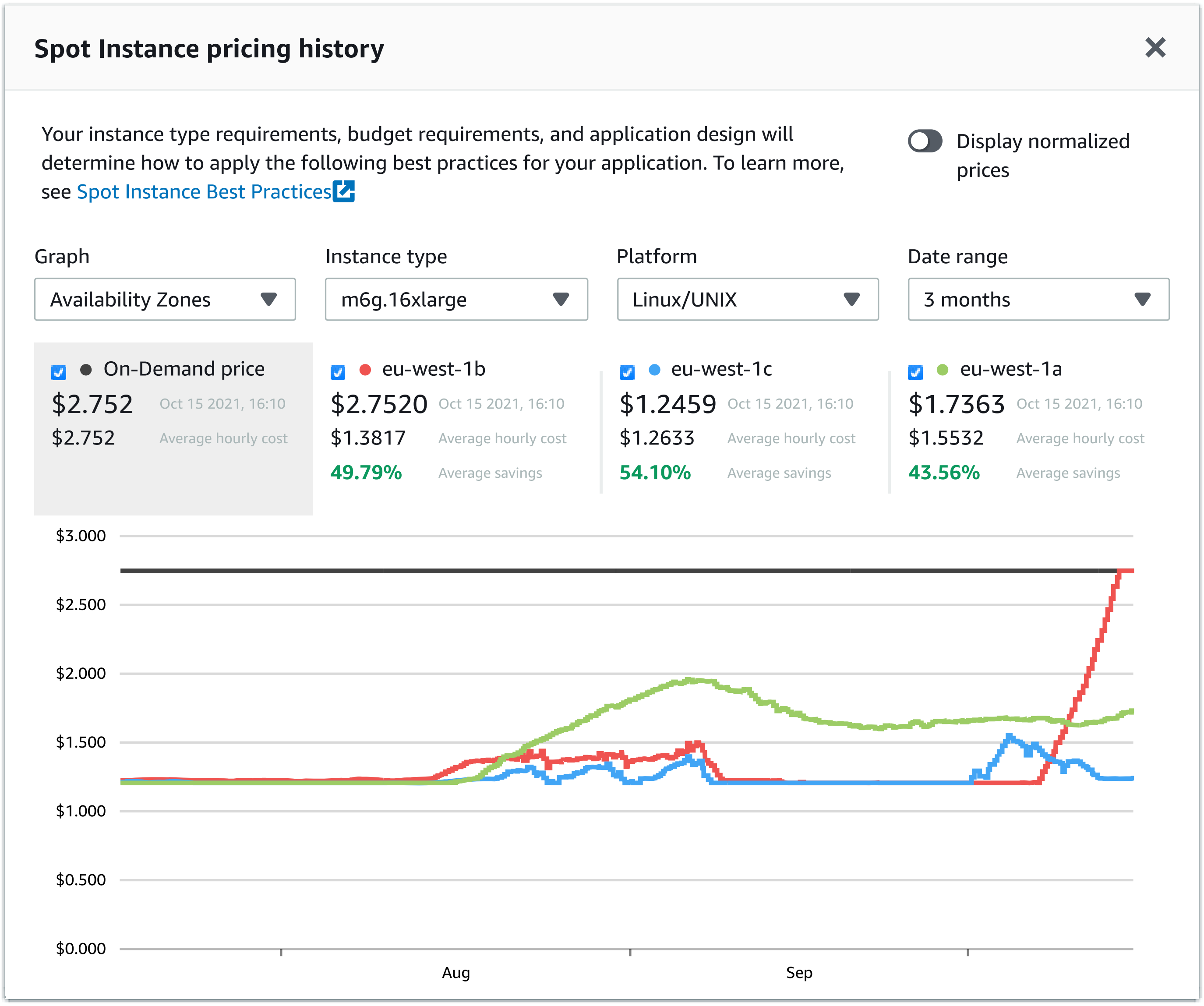
In this paper, we aim at designing and developing a system able to predict the future price of a spot instance in EC2, with the final objective of easing the optimization of the bidding procedure. To do so, we will rely on historic information in the spot instances prices.

The remainder of this document is structured as follows: in Section II we present some basic concepts which are key to understand the current proposal, in Section III we will briefly describe the state of the art and some related work.

Then, in Section IV,we will identify different data sources, explaining the acquisition process, and in Section V we will describe the cleansing and processing stages.

Later, in Section VI we will detail the procedure for learning regression models that fit the instance prices and in Section VII we will provide quality metrics to assess the performance of the learned prediction models and discuss the results obtained.

Finally, in Section VIII we will provide some conclusive remarks regarding the work performed in this paper as well as suggest lines of future research. In , we will present the prediction system delivered as a service, describing the infrastructure underlying the prediction system and an API for accessing it.



Spot Instance pricing history

Your instance type requirements, budget requirements, and application design will determine how to apply the following best practices for your application. To learn more, see Spot Instance Best Practices

Display normalized prices

Graph	Instance type	Platform	Date range
Availability Zones	m6g.16xlarge	Linux/UNIX	3 months

● On-Demand price		● eu-west-1b		● eu-west-1c		● eu-west-1a	
$2.752	Oct 15 2021, 16:10	$2.7520	Oct 15 2021, 16:10	$1.2459	Oct 15 2021, 16:10	$1.7363	Oct 15 2021, 16:10
$2.752	Average hourly cost	$1.3817	Average hourly cost	$1.2633	Average hourly cost	$1.5532	Average hourly cost
		49.79%	Average savings	54.10%	Average savings	43.56%	Average savings

Fig. 1. Panel showing the evolution of the EC2 spot instance price in the AWS console, for a specific instance type and different availability zones, over a period of three months. On-demand (non spot) price is shown in the black line.

II. Structure of the Paper

Before proceeding with the description of the proposal, it is important to introduce some relevant concepts that are required to understand which factors affect the instance price (both in the case of on-demand and spot instances). These factors are the following:

- Region: it refers to the Amazon datacenter where the instance will be launched. Some examples of regions are the following: North Virginia (us-east-1), Ohio (us-east-2), North California (us-west-1), Canada (ca-central-1), Ireland (eu-west-1), etc.
- Availability zone: it is a more precise area within the region. It is identified with a letter after the region codename, for instance, region "us-east-1" contains zones from "us-east-1a" to "us-east-1f". Instances, even those of the same type, can see their cost affected depending on their availability zone.
- Type and size: the instance type determines the compute capabilities it provides. Most often, the instance types adheres to the following convention: <type>.<size>. For example, an instance p3.16xlarge is an instance of type P3 (general use GPU computing) and of size 16xlarge, meaning in this case that it contains 8 GPUs. Amazon provides an updated listing with all the different instance types and their specifications [2]. This factor is the one that affects the most the instance cost.
- Operating system: also called "product" the instance cost can vary depending on whether it runs a Windows environment or a UNIX/Linux one.

III. Related Work

The problem of forecasting the prices of EC2 spot instances is of clear interest, since it allows companies of different sizes to work on optimal bidding strategies that can optimize economic resources spent on cloud computing infrastructure. For this reason, this problem has been observed mostly from two perspectives. The first perspective relies on the study of EC2 spot pricing as an economic problem, using approaches based on econometrics or other financial tools to design bidding models. The second perspective, which is the one followed in this paper, relies on techniques of computational intelligence to frame the approach as a supervised learning problem.

One early work which aims at reverse engineering the EC2 spot pricing scheme is provided by Ben-Yehuda et al. [3], where they build a model concluding that the prices are not fully market-driven, but are most of the times generated randomly within a small range of values. They do not suggest a bidding strategy as such, but rather work on a thorough economic analysis of the pricing models.

Another statistical analysis of the pricing scheme of EC2 spot instances is provided much more recently by Portella et al. [4]. Again, they do not focus on a bidding strategy or price forecasting, but obtain a useful conclusion from the analysis: by bidding at 30% of the on-demand price, availability of over a 90% can be attained, although the specifics vary based on the instance type. Another work by Lumpe et al. [5] focuses as well on a descriptive statistical analysis and econometric study of EC2 spot prices, with authors also devising a bidding strategy that minimises the bidding cost while guaranteeing a certain probability of availability over a defined threshold.

Another early work by Tian et al. [6] suggests a decision model for provisioning computing resources in EC2 by combining different schemes, combining spot instances with the classical on-demand model. In the paper, they introduce a model able to predict the demand and spot prices are expected to vary as a result. Interestingly, they do not focus only on price forecasting, but also in how to diversify instances to deal with potential loss of spot instances (in case their actual price exceeds the bidding price).

Tang et al. [7] address the problem of tackling an optimal bidding strategy. In this case, authors use Markov decision processes, and prove a theorem by which any sequence of bidding decisions can be obtained by a dual-option strategy, either bidding the maximum spot price or giving up at each time. In a more recent work [8], the authors apply this strategy under service-level agreement constraints. In both works, the authors do not focus on price forecasting as an intermediate task to build the bidding policy.

Chhetri et al. [9] have studied the streamlined EC2 spot markets, a different model where prices are softened by using long-term trends in demand and supply. They combine econometric indices as well as computational techniques (logistic regression and principal component analysis) to perform their study. Authors extract interesting conclusions from their analysis: median spot prices have grown in the streamlined model, and sophisticated bidding strategies are less useful in this pricing model. Also, they suggest how to perform bidding price estimation.

When focusing on spot price forecasting, Chhetri et al. [10] use time-series decomposition and look-backs, attaining results that compare or slightly outperform other more classical approaches. For the evaluation, they constrain to eight Microsoft Windows-based instance types in the Sydney region, attaining root mean squared errors that achieve values of 0.559 in c3.xlarge instances.

Another approach using regression random forests has been provided by Khandelwal et al. [11], where they learn models to perform one-day and one-week ahead forecasting. They state that this technique outperforms other methods, reporting a mean absolute percentage error of less than 10% for one-day and less than 15% for one-week forecasting.

A more recent approach has been proposed by Lancon et al. [12], where they use long short-term memory neural networks and claim a reduction of 95% in mean average percentage error as when compared to a baseline model. Unfortunately, absolute errors do not seem to be reported in the paper. We also found this problem in a recent contribution by Malik and Bagmar [13]. These authors discuss a technique to analyze and predict the spot prices for instances using random forests. The authors report mean average percentage errors in the range from 0.15% to 56.2% depending on the instance type.

Recently, Chittora and Gupta [14] explored the feasibility of relying on 2-layer stacked LSTM model for this task using 3 months of spot price data for 5 instances. The results for next day spot price forecast show mean absolute percentage errors under 10% and root mean squared errors below 20%, outperforming the standard LSTM and the 3-layer LSTM considered as alternatives.

Finally, Liu et al. [15] benchmarked kNN regression against linear regression, support vector machine, random forest, multi-layer perceptron and gcForest using the $MAPE_{5\%}$, which represents the number of results whose absolute percentage error is less than or equal to 5% as a percentage of the number of total results, as performance metric. According to their results, kNN regression offered the best performace with a $MAPE_{5\%}$ up to 94% in 1-day-ahead prediction and 94.06% in 1-week-ahead, respectively.

In this work, we will carry out an extensive comparison of diverse machine learning techniques towards forecasting of EC2 spot prices. To the best of our knowledge, this is the most detailed work when it comes to tackling a larger number of EC2 instance types and reporting results in a separate manner for each of them, as well as comprehensive due to the large number of techniques tested.

IV. Data Sources

In this section we will present the different data sources used in order to train and validate the regression models for the price prediction of EC2 spot instances.

A. Approach

The instance price data is modeled as a time series. This series can be seen as a sequence of values for each instant of time, existing a different series for each region, instance type and operating system. Each value in the time serieswill contain a timestamp indicating the moment of time towhich it refers, as well as the instance price at that time.

The data in the time series can be obtained via two different approaches: recovering them from historic archives or querying the prices in real time. In the first case, we would be talking of previously captured data, stored for their later recovery. Meanwhile, in the second case we would refer to new data that is changing as time happens.

The availability of historic data is useful for feeding the models with a large amount of values (e.g., corresponding to several years). Conversely, access to real-time data is useful to provide feedback to the model and updating it periodically to ensure that its predictions are updated to the current characteristics of the time series.

B. History Data Sources

In this work we have used two different data sources providing information of archived historical data of EC2 spot instances prices.

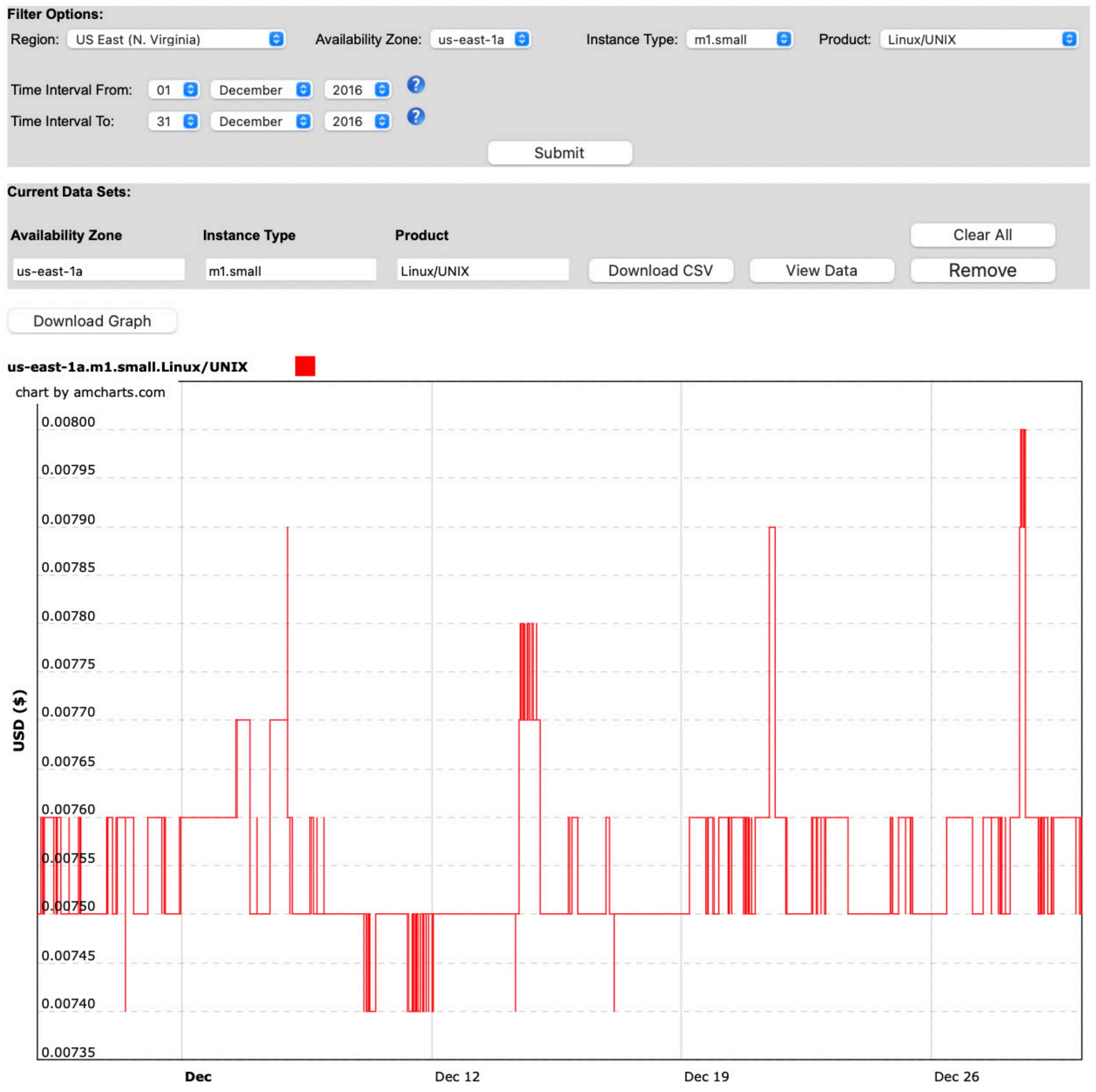
1. AWS Spot Pricing Market

Dataset provided by the Data Science Awards 2017 competition, which is publicly available for download in Kaggle. This involves a CSV file for each of the regions [16]. The structure of these CSV files is shown in Table I.

TABLE I. Structure of AWS Spot Pricing Market CSV Files

Timestamp	Type	OS	Zone	Price
2017-05-06 17:29:01	c4.large	Linux	ca-central-1a	0.0139
2017-05-06 17:29:01	m4.4xlarge	Windows	ca-central-1b	0.8328

This dataset is very complete as it gathers many different types of possible instances, for every operating system and comprising eleven AWS regions. However, the main drawback of this dataset has to do



Filter Options:

Region: US East (N. Virginia) Availability Zone: us-east-1a Instance Type: m1.small Product: Linux/UNIX

Time Interval From: 01 December 2016

Time Interval To: 31 December 2016

Current Data Sets:

Availability Zone	Instance Type	Product
us-east-1a	m1.small	Linux/UNIX

Fig. 2. Graphical user interface of Spot Price Archive.

with its limited of the time period considered, since it only includes instance prices between February and May 2017, not providing time series longer than three months.

This is of course a problem when trying to capture seasonal patterns. To illustrate this with an example, it could be reasonable to hypothesize that in summer season instances are cheaper, because of the low demand (there are fewer companies actively requiring cloud computing services). Conversely, there could be demand peaks at other moments, such as back-to-work period or Christmas campaigns.

Moreover, there could exist inter-annual trends, such as a decrease in the average instance cost, which cannot be detected since the dataset only comprises data from 2017.

2. Spot Price Archive

Spot Price Archive is a historic data archive of EC2 spot instances prices provided by Western Sydney University, Australia. The archive provides a graphical interface for accessing data [17] (see Fig. 2) and was developed for a project aiming at modeling the spot instances price [18].

This dataset is much more complete regarding the length of time series, since it provides data for all years comprised between 2009 and 2016. As a drawback, it imposes some limitations over the previous dataset, since it only comprises certain regions and instance types.

In particular, Data Science Awards dataset provided prices for 68 instance types and 11 regions, whereas Spot Price Archive only contains 15 instance types in 8 regions. Besides, the number of availability zones is also more limited. For example, region us-east-1 comprises six zones (from us-east-1a to us-east-1f), from which the former dataset gathers five and the latter only four.

Despite of being more restricted, this historic dataset will be used to improve the prediction performance over those instance types and regions included in the dataset, providing more information about seasonal behaviors along the year as well as inter-annual trends.

Data from this dataset can be downloaded in CSV files. The acquisition process can be automated because URLs for the CSV files always follow the same convention, with the following base URL:

```
http://spot.scem.uws.edu.au/ec2si/Download.jsp
```

This URL accepts the following query parameters, which can be specified in the GET request: Zone (the availability zone), Type (the instance type), Product (the operating system), IntervalFrom (the start date, formatted as “yyyy-mm-dd 00:00:00.0” and IntervalTo (the end date, with the same format).

The CSV files obtained are structured as shown in Table II. It can be seen howthis structure is equivalent to the previous dataset, since the values in the columns corresponding to the availability zone, the instance type and the operating system are known beforehand.

TABLE II. Structure of Spot Price Archive CSV Files

Timestamp	**Price**
2012-11-05 12:00:00	0.006

C. Real-Time Data Sources

Historic data allows us to learn regression models that can take into consideration inter-annual trends and seasonal factors. Nevertheless, it is important to periodically feedback these models in order to keep them upgraded and get useful and accurate predictions over time. In this work we consider the use of one source of real-time data.

1. EC2 API

The most convenient way to obtain real-time data is to use EC2’s API, which has an endpoint (*describe_spot_price_history*) [19] that returns the history of prices from the current time up to 90 days into the past. By calling this endpoint, we can obtain real-time data.

The endpoint returns a JSON-encoded object with the following structure, as described in the specification:

```
{
  "SpotPriceHistory" : [
    "AvailabilityZone"   : <zone>,
    "InstanceType"       : <type>,
    "ProductDescription" : <os>,
    "Timestamp"          : datetime(yyyy, m, d),
    "SpotPrice"          : <price>
  ], ...
}
```

These fields identify the instance type, the zone and the operating system, as well as the timestamp and the price. Therefore, we will be able to easily transform this data into CSV files with the format that we had seen previously in the case of historic data.

V. Data Processing

Before training the regression models for instance price prediction, we will perform some basic processing of the data in order to extract relevant features that can be useful for training the models, as well as to standardize the output format of the different data sources.

A. Feature Selection

After acquiring the data, the available attributes are those characterizing the instance (availability zone, type and operating system) and the timestamp.

The instance features comprise categorical data which will not be subject to any additional processing. However, in the case of the timestamp, it is stored as a text string, which is not particularly useful. We will use it to retrieve some features which can be of interest:

- Year: the year can be relevant to discover inter-annual trends, as whether the price of a certain instance type decreases as new instances are released.
- Month: the month is an important feature to detect intraannual seasonal trends, such as whether the price is lower in summer months, when demand may be lower due to the holidays season.
- Day of month: it is difficult to know whether this is a relevant feature, but it could be in the case of intra-monthly trends.
- Day of week: it can be interesting to detect whether prices change on weekdays versus weekends.
- Hour: it can be of interest because the cost could change based on days as opposed to nights. For this reason, this feature could also depend on the instance region. We have omitted the minute and second since they do not seem to be relevant features affecting the instance price.

B. Output Format Standardization

As we saw in the previous section, all data from different sources is equivalent when it comes to the features (columns). When it comes to rows, they also follow the same format, which is the one provided by Amazon EC2 API: a row is only shown when there is a change in the instance price. This means that the difference between two consecutive timestamps is not constant, and also that there is not explicit information about the price at every timestamp, although this information can be easily inferred.

In this standardization, we reduce the resolution of the timestamp from seconds to hours, as we explained previously. In some cases, the prices can slightly vary within the same hour, in which case we compute the median of the different values. We have decided to use the median instead of the mean to avoid adding values that did not appear originally in the input data.

Finally, we fill the non-existing rows between the start and end date. Since rows only exist when there is a change in the price, newrows will have as the price the last value available immediately before the time of the row being added.

VI. Model Learning

In this section we will explain the machine learning techniques used to train the regression models for instance price prediction. First, we will discuss some design decisions regarding the training process, and later we will detail the training procedure.

A. Design Decisions

When learning a regression model from a time series such as the one described in this paper, we can mainly choose among two different approaches.

In the first approach, we would use the information available in the time series to predict the next value, which is unknown. In this case, the attributes are formed by the price in the n previous times. In other words, given $(t_0, t_1, ..., t_n)$, we want to predict the value at time t_{n+1}.

This approach, despite being very common, does not seem the most appropriate for solving the problem. The first reason is that the quality of predictions degrades when we want to predict values that are far in the future, since we would be using as input some features whose value is unknown and are just an estimation (this means that for predicting t_{n+m}, we will need values (t_{n+1}, t_{n+2}, ..., t_{n+m-1}), which are not known). The second reason, which is domain-dependent, is that this time series is very static, and in some cases an instance can hold the same price during hours or even days. For this reason, a prediction model would like turn to keep the price invariant, missing all the times that the price is actually updated.

The second approach, which we have deemed more interesting in this work, is to introduce as input the parameters that were previously described: year, month, day of month, day of week and hour, as well as availability zone, instance type and operating system. In this case, knowing the values in the time series immediately before the desired prediction time instant is irrelevant, since they are not used for the prediction. As a consequence of this, the approach has the advantage that the quality of the prediction is not affected by how far in the future the desired value is.

Finally, we have decided to train a model for each instance type. This is due to the fact that there is a significant heterogeneity between different instance types, and one model could find difficulties when dealing with such an amount of diverse data. Nevertheless, given a fixed instance type, the price is much more stable, even though it still can vary significantly depending on the availability zone, the operating system, and the date and time.

B. Learning Procedure

In order to train the models, we will use the Scikit-learn library for Python [20].

The first step is to use one-hot-encoding (OHE) to convert categorical attributes into binary features in order to establish a data format which can be accepted by this library. As an example, if we had a sample whose operating system is “Windows”, the OHE encoding would generate three binary features, from which feature “os_windows” would be set to one and features “os_linux-unix” and “os_suse-linux” would be zero.

When it comes to the process of learning a regression model from the available data, there are numerous techniques that could be used, and it can be difficult to determine beforehand which of such techniques could work best. In fact, it could happen that a technique works the best on a certain instance type, but be outperformed by other techniques in other types. In this case, we could not claim that a technique is the best performer.

Because of this, we will test different machine learning techniques with each type of instance. In particular, these techniques are the following:

- Linear regression: a standard algorithm that will learn a hyperplane fitting input data with the least square error.
- Linear regression with ridge regularization: same as the previous one yet imposing a penalty in the size of the regression coefficients.
- Linear regression with lasso regularization: same as the first one, but preferring solutions with fewer parameters.
- Multilayer perceptron: a neural network that can act as a universal function approximator. In the chosen setup, it comprises one hidden layer with 100 units.
- K-nearest neighbors: a geometric model where the K closest instances to the one being predicted are retrieved, and their outputs are averaged to provide a prediction. In the chosen setup, K is set to five.
- Extra Trees: an ensemble grouping several models and weighting their outputs. In particular, in this case these models will be regression trees, where each one will be trained using a random sample of the data and a random subset of the features. In the current setup, the ensemble will be formed of 10 models.
- Random Forests: similar to Extra Trees, yet with less randomness when it comes to choose the attributes to build the decision trees.
- AdaBoost: an ensemble where a regression model is first fitted over the original data and then additional models are trained over this data, but giving a higher weight to instances poorly estimated by previous models.
- Bagging: an ensemble where several regression models are trained over different samples of data.

We have chosen these techniques since they capture a large diversity of the machine learning techniques. For instance, linear regression is a simple model aiming at learning a line within the space of features and, along with the variations using ridge and lasso regularization, they are a good representative of linear models. The multilayer perceptron is the best representative of a feed-forward neural network. K-nearest neighbors is a simple model based on analogy, that is able to capture complex frontiers of decision in the space of features, under the hypothesis that similar instances will have a similar output. Finally, we have tried different ensembles based on decision trees, which are models able to learn rules for making a decision based on the features’ values. We have only tested ensembles of decision trees since individual trees will often take longer to train and will rarely obtain better results, as they are more prone to overfitting.

Table III summarize the hyperparameters used for these techniques. These hyperparameters have been chosen after a prior stage of sensitivity analysis.

TABLE III. Hyperparameters of the Different ML Techniques

Technique	Parameter	Value
Ridge / Lasso	Regularization strength	1
MLP	Number of layers	1
	Number of units	100
	Activation function	ReLU
	Optimizer	Adam
KNN	Number of neighbors (K)	5
	Distance metric	Euclidean
RF / ET / Bagging	Number of models	10
AdaBoost	Number of models	50

Once the techniques are chosen, we will follow the next procedure: First, we will split the dataset for each instance type into a training set and a test set. Instead of performing a random division of data, we have decided that data from September 2017 be assigned to the test set, with previous information assigned to the training set.

Later, for each instance type we will train each model ten times. This decision is motivated by the fact that most of the previously described techniques are stochastic, and therefore one single run could introduce an important bias. Finally, for each instance type we will serialize the best model, so we can recover it later when aiming to predict the price of a spot instance in the future.

VII. Model Evaluation

To validate the different models, we will compute three quality metrics for the best model obtained and compare it to a baseline. Such baseline will correspond to the performance of a naive regression model that would always predict the average price. The most improvement over such baseline, the best performance of the model.

The metrics reported in this work are the following:

- Root Mean Squared Error (RMSE) is the square root of the mean of squared errors. Therefore, being yi the real price for sample *i* and

($\hat{y}_i$) the price estimated by the regression model, RMSE is computed as in (1).

$$RMSE = \sqrt{\frac{\sum_{i \in N}(y_i - \hat{y}_i)^2}{|N|}} \quad (1)$$

The closest this value is to zero, the most accurate the model predictions will be.

- Explained Variance Score (EVS) computes to which extent a regression model captures the distribution of the original data. It is computed as shown in (2).

$$EVS = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}} \quad (2)$$

The value will be better as it approaches one. In the case of the baseline, since the mean is always returned, then EVS will be zero.

- R^2 Score, or coefficient of determination, measures to what extent the model will estimate future samples, and is computed as shown in (3).

$$R2 = 1 - \frac{\sum_{i \in N}(y_i - \hat{y}_i)^2}{\sum_{i \in N}(y_i - \bar{y}_i)^2} \quad (3)$$

Again, the value will be better as it approaches one. For the baseline, since y_i is the mean value, R^2 score will be zero. Notwithstanding, a model arbitrarily worse could have a negative score in this metric.

Table IV shows the results of the evaluation in terms of the previously defined metrics, also showing the technique leading to the best obtained model. In the case of the baseline, only the RMSE is shown, since EVS and R^2 score are always zero. The last column displays an estimation of the model quality depending on its R^2 value.

A. Discussion

As it can be seen, results vary significantly depending on the instance type. For some types, such as *c3*, *r3*, *m1*, *m3* or *i3*, we have achieved models that are able to successfully predict prices for almost the entire family. As we suggested earlier, the best model can vary from one instance type to another, although the multilayer perceptron or ensembles often behave well in many cases.

Conversely, some instance types obtain poor results, even if RMSE always improves over the baseline. This happens because the instance type has a small spot offer, therefore turning the market price more unpredictable. This effect is clearly seen in some instance types.

For example, *f1.16xlarge* instances have an on-demand cost of 13.2 dollars per hour, but a baseline RMSE over 65 dollars. The reason is that in a scenario with few offer, the price can be set to the maximum established by AWS, which is ten times the on-demand price, i.e., 132 dollars. This can severely affect the time series, leading to a bumpy landscape that can harden the process or learning a regression model. In such cases, an alternative would be needed to improve these models performance.

Regarding machine learning techniques, there is not a clear winner that shows an outstanding prediction capability for all of the diversity of EC2 instance types. Generally speaking, linear regression is not dominant except for a small set of instances, regardless of whether regularization is used or not. This seems to indicate that most instance types have spot prices that do not have a linear dependency on the input features. Also, KNN is not displaying a good performance except for a couple of instance types. We can also see how MLP seems to be the model of choice for those instance types where prediction is more difficult and leads to worse result. This can be due to the fact that MLP is able to approximate the series of prices better than any other model, but it is still insufficient for considering the result as a good prediction. Conversely, ensembles of decision trees are found most often among the best models to achieve successful regression. Given this information, it seems that price prediction is rarely a linear problem, except for a few cases of instance types.

VIII. Conclusions

In this paper we have described all the steps carried out to tackle the problem of predicting EC2 spot instance prices. In this problem, we are interested in knowing the price of a certain spot instance at some point in the future, in order to be able to bid consequently. In order to solve this problem, we have used two different historical datasets, as well as data extracted in real time from the EC2 API, providing data from the last three months. Once data is retrieved, we have transformed then in order to extract relevant features from the timestamp and have later trained a regression model for instance type. The rationale beyond training separate models based on the instance types is that there is a very high variability in the prices depending on the type.

In particular, we have used Scikit-learn to test different regression techniques and selected those improving the quality metrics for each instance type: RMSE, EVS and R2. When looking at the results, we notice that some instance types obtain almost perfect models, whereas in others the baseline (a base prediction of the average price) was barely outperformed. This difference can be explained at least partially due to the characteristics of the instance.

Finally, we have developed a Prediction-as-a-Service system which we have deployed in the cloud. The infrastructure underlying this service as well as the API documentation is described in the appendix.

In order to further improve this work, we could add the support for more instance types and availability zones, by retrieving enough data from the EC2 API. Also, we could introduce more features to the problem, taking into account that these features must be known beforehand for those instances we want to predict. An example of such attribute could be whether the day is a national holiday in the region where we want to predict the price.

Appendix: Prediction-as-a-Service

In this appendix, we will detail the backend infrastructure required for storing the models, keeping them updated and enabling real-time prediction using a public endpoint (web service), as well as describe the interface for using the prediction system as a service.

A. Infrastructure

The CSV data and serialized models will be stored in the cloud, in an S3 bucket.

In a periodic fashion, a batch process will update the models. To do so, it will create an EC2 instance that will download the data from S3, include the most recent data using the EC2 API and finally retrain the machine learning model with the new data. This model will be stored in S3 replacing the previous version.

To provide the prediction service, we have deployed the implementation of our API over AWS Lambda. This cloud service allows us to run code in a serverless infrastructure, i.e., without requiring us to manually deal with the server resources, and providing an URL that could be used by the API clients. Moreover, AWS guarantees the service scalability, therefore allowing large number of concurrent requests without increasing the latency or response times.

B. API

The endpoint, available in AWS Lambda and accessible through AWS API Gateway is the following:

TABLE IV. Performance of the Machine Learning Models for Each Instance Type

Type	Baseline RMSE	Technique	RMSE	Best EVS	R^2	Result
t1.micro	0.049	ExtraTrees	0.025	0.741	0.741	+
m1.small	0.117	ExtraTrees	0.005	0.998	0.998	+ +
m1.medium	0.235	RandomForest	0.007	0.999	0.999	+ +
m1.large	0.473	ExtraTrees	0.063	0.982	0.982	+ +
m1.xlarge	0.936	ExtraTrees	0.024	0.999	0.999	+ +
m2.xlarge	0.389	MLP	0.374	0.079	0.079	– –
m2.2xlarge	0.933	Lasso	0.933	0	0	– –
m2.4xlarge	2.707	ExtraTrees	1.620	0.644	0.642	+
m3.medium	0.267	ExtraTrees	0.025	0.991	0.991	+ +
m3.large	0.465	RandomForest	0.098	0.956	0.955	+ +
m3.xlarge	0.927	RandomForest	0.180	0.962	0.962	+ +
m3.2xlarge	1.889	AdaBoost	0.471	0.941	0.938	+ +
m4.large	0.052	KNN	0.006	0.988	0.987	+ +
m4.xlarge	0.110	AdaBoost	0.068	0.634	0.602	+
m4.2xlarge	0.193	AdaBoost	0.067	0.899	0.871	++
m4.4xlarge	2.818	MLP	2.238	0.357	0.357	–
m4.10xlarge	11.703	MLP	7.989	0.558	0.533	+
m4.16xlarge	16.422	MLP	10.793	0.562	0.561	+
c1.medium	0.093	MLP	0.087	0.129	0.129	– –
c1.xlarge	1.723	ExtraTrees	1.212	0.503	0.498	–
c3.large	0.491	RandomForest	0.030	0.996	0.996	+ +
c3.xlarge	1.163	RandomForest	0.014	1	1	+ +
c3.2xlarge	2.117	RandomForest	0.596	0.921	0.920	+ +
c3.4xlarge	4.360	Ridge	2.106	0.762	0.762	+ +
c3.8xlarge	8.990	MLP	2.492	0.932	0.923	+ +
c4.large	0.303	Ridge	0.094	0.904	0.903	+ +
c4.xlarge	0.950	ExtraTrees	0.131	0.981	0.981	+ +
c4.2xlarge	1.122	MLP	0.439	0.847	0.844	+ +
c4.4xlarge	2.989	MLP	2.014	0.549	0.522	+
c4.8xlarge	6.106	KNN	3.412	0.674	0.674	+
x1.16xlarge	44.522	MLP	26.098	0.656	0.656	+
x1.32xlarge	104.019	MP	57.215	0.682	0.675	+
r3.large	0.478	RandomForest	0.141	0.912	0.912	+ +
r3.xlarge	1.198	ExtraTrees	0.061	0.997	0.997	+ +
r3.2xlarge	1.761	ExtraTrees	0.773	0.807	0.807	+ +
r3.4xlarge	4.375	ExtraTrees	1.578	0.869	0.869	+ +
r3.8xlarge	11.143	ExtraTrees	3.108	0.922	0.9	+ +
r4.large	0.046	Ridge	0.007	0.977	0.977	+
r4.xlarge	0.126	Ridge	0.099	0.389	0.388	–
r4.2xlarge	0.725	MLP	0.692	0.089	0.085	– –
r4.4xlarge	3.161	AdaBoost	2.670	0.321	0.285	–
r4.8xlarge	8.448	MLP	6.560	0.398	0.391	–
r4.16xlarge	26.507	MLP	13.310	0.757	0.748	+
p2.xlarge	0.116	LinearRegression	0.105	0.195	0.181	– –
p2.8xlarge	55.079	MLP	27.525	0.751	0.732	+
p2.16xlarge	86.914	MLP	57.599	0.531	0.492	–
g2.2xlarge	0.9	Lasso	0.9	0	0	– –
g2.8xlarge	16.048	MLP	11.418	0.411	0.401	–
cg1.4xlarge	2.120	RandomForest	0	1	1	+ +
f1.2xlarge	0.142	MLP	0.102	0.155	0.142	– –
f1.16xlarge	65.664	MLP	65.393	0	0	– –
i2.xlarge	2.469	RandomForest	0.802	0.894	0.892	+ +
i2.2xlarge	4.741	AdaBoost	3.224	0.598	0.530	+
i2.4xlarge	11.903	Bagging	6.743	0.673	0.673	+
i2.8xlarge	20.549	MLP	11.261	0.707	0.697	+
i3.large	0.590	ExtraTrees	0.153	0.933	0.933	+ +
i3.xlarge	1.010	RandomForest	0.323	0.898	0.897	+ +
i3.2xlarge	1.949	Bagging	0.802	0.831	0.831	+ +
i3.4xlarge	5.789	MLP	4.281	0.454	0.453	–
i3.8xlarge	17.312	ExtraTrees	6.288	0.868	0.867	+ +
i3.16xlarge	34.811	MLP	13.566	0.851	0.847	+ +
cc2.8xlarge	9.748	MLP	8.623	0.100	0.051	– –
d2.xlarge	0.242	Lasso	0.242	0	0	– –
d2.2xlarge	4.871	MLP	4.232	0.207	0.206	– –
d2.4xlarge	8.198	LinearRegression	7.021	0.263	0.262	–
d2.8xlarge	17.020	MLP	12.643	0.451	0.448	–
h1.4xlarge	12.074	MLP	6.681	0.692	0.692	+
cr1.8xlarge	15.786	MLP	2.073	0.983	0.983	+ +

The last column shows the quality of the model according to its coefficient of determination, which is directly correlated with the other quality metrics. In particular, the legend for this column is the following: –– means that the model is very poor (R^2 <0.25), – means that the model is poor ($0.25 \leq R^2 < 0.5$), + means that the model is reasonably good ($0.5 \leq R^2 < 0.75$), and finally ++ means that the model is very good ($R^2 \geq 0.75$)).

```
https://slcswaq0e2.execute-api.us-east-1.amazonaws.com
/dsawards/predict
```

This endpoint must be accessed through a POST request, with a JSON body including the following parameters:

- `type`: instance type (required).
- `os`: instance operating system (required).
- datetime: time desired for the prediction, which must be in the format "yyyy-mm-dd hh" (required).
- regions: regions for which a prediction should be returned. A list with one or more regions can be specified, and the service will return the prediction for all zones in each region. This parameter is optional and, if not specified, then all regions will be considered.

An example of a well formed body in the API call would be the following:

```
{
"type" : "c3.xlarge",
"os" : "Linux/UNIX",
"datetime" : "2017-09-13 13",
"regions" : ["us-east-1"]
}
```

Such a valid request will return a JSON object whose keys are the availability zones and the values are the estimated prices. For instance, for the previous call:

```
{
"us-east-1a" : 2.109, "us-east-1b" : 0.155,
"us-east-1c" : 0.155, "us-east-1d" : 0.155,
"us-east-1e" : 0.155, "us-east-1f" : 0.155,
}
```

Acknowledgment

Authors would like to thank Data Science Awards committee for proposing the competition.

This work has been supported by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3MXX), and in the context of the V PRICIT (Regional Programme of Research and Technological Innovation).

References

[1] Amazon Web Services, "Amazon EC2 Spot Instances Pricing." Accessed: Oct. 15, 2021, [Online]. Available: https://aws.amazon.com/ec2/spot/pricing.

[2] Amazon Web Services, "Amazon EC2 Instance Types." Accessed: Oct. 15, 2021, [Online]. Available: https://aws.amazon.com/ec2/instance-types.

[3] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, D. Tsafrir, "Deconstructing Amazon EC2 spot instance pricing," *ACM Transactions on Economics and Computation*, vol. 1, no. 3, p. 16, 2013.

[4] G. Portella, G. N. Rodrigues, E. Nakano, A. C. Melo, "Statistical analysis of Amazon EC2 cloud pricing models," *Concurrency and Computation. Practice and Experience*, vol. 31, no. 18, p. e4451, 2018.

[5] M. Lumpe, M. B. Chhetri, Q. B. Vo, R. Kowalcyk, "On estimating minimum bids for Amazon EC2 spot instances," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Madrid, Spain, 2017, IEEE.

[6] C. Tian, Y. Wang, F. Qi, B. Yin, "Decision model for provisioning virtual resources in Amazon EC2," in *2012 8th Intl. Conf. Network and Service Management and 2012 Workshop on Systems Virtualization Management*, Las Vegas, NV, USA, 2012, IEEE.

[7] S. Tang, J. Yuan, X.-Y. Li, "Towards optimal bidding strategy for Amazon EC2 cloud spot instance," in *2012 IEEE Fifth International Conference on Cloud Computing*, Honolulu, HI, USA, 2012, IEEE.

[8] S. Tang, J. Yuan, C. Wang, X.-Y. Li, "A framework for Amazon EC2 bidding strategy under SLA constraints," *CIEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 2–11, 2014.

[9] M. B. Chhetri, M. Lumpe, Q. B. Vo, R. Kowalczyk, "To bid or not to bid in streamlined EC2 spot markets," in *2018 IEEE International Conference on Services Computing*, San Francisco, CA, USA, 2018, IEEE.

[10] M. B. Chhetri, M. Lumpe, Q. B. Vo, R. Kowalczyk, "On forecasting Amazon EC2 spot prices using time-series decomposition with hybrid look-backs," in *2017 IEEE International Conference on Edge Computing*, Honolulu, HI, USA, 2017, IEEE.

[11] V. Khandelwal, A. Chaturvedi, C. P. Gupta, "Amazon EC2 spot price prediction using regression random forests," *IEEE Transactions on Cloud Computing*, vol. 8, no. 1, pp. 59–72, 2020.

[12] J. Lancon, Y. Kunwar, D. Stroud, M. McGee, R. Slater, "AWS EC2 instance spot price forecasting using LSTM networks," *SMU Data Science Review*, vol. 2, no. 2, p. 8, 2019.

[13] M. Malik, N. Bagmar, "Forecasting price of amazon spot instances using machine learning," *International Journal of Artificial Intelligence and Machine Learning*, vol. 11, pp. 71–82, 07 2021.

[14] V. Chittora, C. P. Gupta, "Dynamic spot price forecasting using stacked lstm networks," in *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*, 2020, pp. 1080–1085.

[15] W. Liu, P. Wang, Y. Meng, C. Zhao, Z. Zhang, "Cloud spot instance price prediction using knn regression," *Human-centric Computing and Information Sciences*, vol. 10, no. 1, p. 34, 2020.

[16] Kaggle, "AWS Spot Pricing Market Dataset." Accessed: Oct. 15, 2021, [Online]. Available: https://www. kaggle.com/noqcks/aws-spot-pricing-market.

[17] Western Sydney University, "Spot Price Archive." [Online]. Available: http://spot.scem.uws.edu.au/ ec2si.

[18] B. Javadi, R. Thulasiram, R. Buyya, "Statistical Modeling of Spot Instance Prices in Public Cloud Environments," in *4th IEEE/ACM International Conference on Utility and Cloud Computing*, Melbourne, Australia, 2011, 2011, IEEE.

[19] Amazon Web Services, "EC2-Boto 3 Docs." Accessed: Oct. 15, 2021, [Online]. Available: http://boto3.readthedocs.io/en/latest/ reference/ services/ec2.html#EC2.Client. describe_spot_price_history.

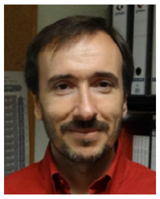
[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Alejandro Baldominos

Alejandro Baldominos holds a Ph.D. in Computer Science and Technology by Universidad Carlos III de Madrid, where he is currently working as a researcher in the Evolutionary Algorithms, Neural Networks and Artificial Intelligence group. His current research line involves the application of evolutionary computation to the evolution of the topology of deep neural networks. Additionally, he has also published several papers in journals and international conferences regarding the application of machine learning to diverse real-world fields.

Yago Saez

Yago Saez received the degree in computer engineering in 1999. He got his Ph.D. in Computer Science from the Universidad Politécnica de Madrid, Spain, in 2005. Since 2007 till 2015 he was vice-head of the Computer Science Department from the Carlos III University of Madrid, where he got a tenure and is nowadays associate professor. He belongs to the Evolutionary Computation, Neural Networks and Artificial Intelligence research group (EVANNAI) and member of the IEEE Computational Finance and Economics Technical committee.



David Quintana

David Quintana holds Bachelor degrees in Business Administration and Computer Science. He has an M.S. in Intelligent Systems from Universidad Carlos III de Madrid and a Ph.D. in Finance from Universidad Pontificia Comillas (ICADE). He is currently Associate Professor at the Computer Science Department at Universidad Carlos III de Madrid. There, he is part the bio-inspired algorithms group EVANNAI. His current research interests are mainly focused on applications of Computational Intelligence in finance and economics.

Pedro Isasi

Pedro Isasi is Graduate and Doctor in Computer science by the Polytechnic University of Madrid since 1994. Currently, he is University professor and head of the Evolutionary Computation and Neural Networks Laboratory in the Carlos III University of Madrid. His research is centered in the field of the artificial intelligence, focusing on problems of Classification, Optimization and Machine Learning, fundamentally in Evolutionary Systems, Metaheuristics and artificial neural networks.