Demonstration of Nimbus: Model-based Pricing for Machine Learning in a Data Marketplace

Lingjiao Chen[1] Hongyi Wang[1] Leshang Chen[2] Paraschos Koutris[1] Arun Kumar[3]

[1]University of Wisconsin-Madison [2]University of Pennsylvania [3]University of California, San Diego
{lchen, paris, hongyiwang}@cs.wisc.edu, leshangc@seas.upenn.edu, arunkk@eng.ucsd.edu

ABSTRACT

Various domains such as business intelligence and journalism have made many achievements with help of data analytics based on machine learning (ML). While a lot of work has studied how to reduce the cost of training, storing, and deploying ML models, there is little work on eliminating the data collection and purchase cost. Existing data markets provide only simplistic mechanism allowing the sale of fixed datasets with fixed price, which potentially hurts not only ML model availability to buyers with limited budget, but market expansion and thus sellers' revenue as well. In this work, we demonstrate *Nimbus*, a data market framework for ML model exchange. Instead of pricing data, *Nimbus* prices ML models directly, which we call *model-based pricing* (MBP). Through interactive interfaces, the audience can play the role of sellers to vend their own ML models with different price requirements, as well as the role of buyers to purchase ML model instances with different accuracy/budget constraints. We will further demonstrate how much gain of sellers' revenue and buyers' affordability *Nimbus* can achieve with low runtime cost via both real time and offline results.

CCS CONCEPTS

• **Information systems → Data management systems**; • **Computing methodologies → Machine learning**; • **Theory of computation →** *Algorithmic game theory and mechanism design*;

KEYWORDS

Machine Learning, Pricing, Data Market, Mechanism Design

ACM Reference Format:
Lingjiao Chen, Hongyi Wang, Leshang Chen, Paraschos Koutris, Arun Kumar. 2019. Demonstration of Nimbus: Model-based Pricing for Machine Learning in a Data Marketplace. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3299869.3320231

1 INTRODUCTION

Various domains such as business intelligence and journalism have made many achievements with help of data analytics based on machine learning (ML). Research and industrial efforts have largely focused on performance, scalability and integration of ML with data management systems [5, 7, 8]. However, limited research so far has studied the *cost of acquiring data* for ML-based data analytics. Structured (*relational*) data are often purchased to train ML models, either directly through companies (e.g., Bloomberg, Twitter), or through *data markets* (e.g., BDEX [1], Qlik [3]). Such datasets are typically expensive due to the immense effort to collect, integrate and clean them. Existing pricing schemes either force users to buy the whole dataset or support simplistic pricing mechanisms, without any awareness of the ML task downstream. This means that valuable datasets may not be affordable to potential buyers with limited budgets, and also that data sellers operate in an inefficient market, where they do not maximize their revenue. Simplistic pricing schemes may also create undesirable arbitrage opportunities, where the desired data can be acquired by combining data fragments that together cost a cheaper price.

Example. Consider Alice, a journalist studying the relationship between demographics and economic indicators for an article. She wants to test how predictive some demographic features are of the average annual household income. Datasets with such information exist online, but they are expensive and exceed Alice's budget. In this scenario, if datasets are sold with fixed price, then Alice will not be able to obtain them, and the sellers cannot get revenue as well. If a data marketplace allows Alice to be charged only based on her ML task and desired accuracy, then both Alice and the sellers will be happy. In particular, a linear regression model

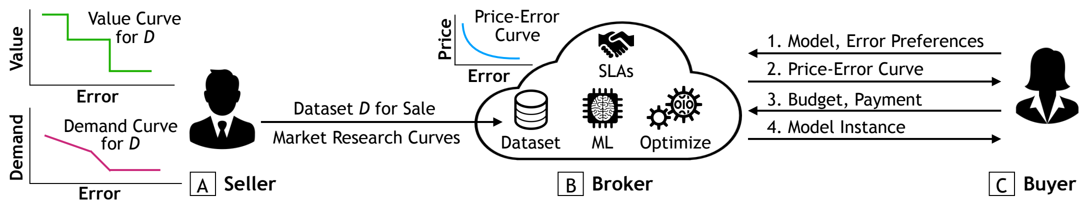


Figure 1: Nimbus market setup. (A) The *seller* is the agent who wants to sell ML model instances trained on their commercially valuable dataset D. (B) The *broker* is the agent that mediates the sale for a set of supported ML models and gets a cut from the seller. (C) The *buyer* is the agent interested in buying a ML model instance.

instance with a square loss might be sufficient for Alice's purposes and cheap enough for Alice to purchase.

Nimbus: A Model-based Pricing Framework. In this work, we demonstrate *Nimbus*, a model-based pricing framework which establishes a marketplace for selling and buying ML models over relational data. The key observation is that, instead of selling raw data to the buyers, the market can directly sell *ML model instances* with different accuracy options. Since the price is based on the model, we call this *model-based pricing* (MBP)[4, 6].

A high level view of Nimbus is demonstrated in Figure 1. The data market involves three agents, namely, the *seller* who provides the datasets, the *buyer* who is interested in buying ML model instances, and the *broker* (market) who interacts between them. First, the seller and/or the broker perform market research to ascertain curves representing demand and value for the ML model instances among potential buyers. These curves plot demand and value as a function of the error/accuracy of the ML model trained. The broker uses the market research information to build price-error curves that are presented to the buyer. The buyer specifies a desired price or error budget and pays the broker, who computes an appropriate ML model instance, and returns it to the buyer.

Summary of Contributions. In summary, this work makes the following contributions.

- We demonstrate how *Nimbus* enables sellers to set different price requirements and buyers to put various ML model accuracy and budget constraints.
- We design and implement a concrete version of *Nimbus*, including a back-end focusing on model generation and price calculation, and front-end for sellers and buyers to interact with the data market.
- We empirically demonstrate the benefits of *Nimbus* compared to naive pricing approaches. In particular, running on both offline and real time data, *Nimbus* can maintain high seller revenue, large buyer affordability as well as low runtime cost.

Outline. Section 2 introduces the MBP framework and relevant desiderata. Section 3 presents *Nimbus*'s functionality and implementation details. In Section 4, we study the applicable scenarios of *Nimbus*. We conclude in Section 5.

2 MODEL-BASED PRICING

In this section, we introduce the framework of model-based pricing (MBP), outline properties it satisfies and provide an instance of our framework that satisfies all the properties.

2.1 Market Setup and Agents

Our framework involves three types of agents that interact in a data marketplace: the *seller*, the *broker* and the *buyer*.

Agents. The seller provides the dataset D for sale. W.L.O.G, we let $z = (\mathbf{x}, y)$ be a labeled example in D, where $\mathbf{x}$ and y are the feature vector and label, respectively. The broker supplies a menu of ML models, such as linear regression, logistic regression, and support vector machines. Let λ, ϵ denote the error function used for training and testing datasets, respectively. Fixing an ML model $\mathcal{H}$ from the model set, the broker releases model instances $h \in \mathcal{H}$ by a randomized mechanism $\mathcal{K}$. More specifically, let $h^*_\lambda(D)$ be the optimal model instance in the selected ML model/hypothesis space $\mathcal{H}$ and $\{W_\delta | \delta \in R_+\}$ be a set of probability distributions. Given a dataset D, an error function λ, and a noise control parameter δ, the broker samples a random variable $w \in \mathcal{W}_\delta$, and releases a model instance $h^\delta_\lambda(D) = \mathcal{K}(h^*_\lambda(D), w)$. The buyer specifies a model instance he/she is interested in learning over the dataset, with a particular accuracy and/or budget requirement, among the model instances provided by the broker.

Interactions. There are two types of interactions, namely, seller-broker and buyer-broker interactions. In seller-broker interaction, the seller helps the broker determine the ML models to sell, and specify requirements for the price associated with each model instance. In buyer-broker interaction, the buyer first provides the broker which models he/she is interested in buying. The broker then provides information about that particular model, such as a price accuracy curve. After obtaining the buyer's choice of model instance, $h^\delta_\lambda(D)$, the broker computes its associated price $p_{\epsilon,\delta}(\delta, D)$. Finally, the buyer pays for the price and acquire the model instance.

2.2 Pricing Function Desiderata

For the market to work, the pricing function $p_{\epsilon,\lambda}$ needs to satisfy a set of desiderata that provide some guarantees to both the seller and the buyer. In a sense, these guarantees act as the service-level agreement (SLA) for MBP. In particular, we want them to satisfy the following requirements.

Non-negativity. Clearly, the pricing function has to be *non-negative*, since the buyer should not be able to make money from the broker by obtaining an ML model instance.

Error Monotonicity. Next, if for a parameter δ_1, we obtain an expected error smaller than for a parameter δ_2, then the price should be larger for the former model instance. Otherwise, a buyer may have no incentive to buy the former instance. The formal definition is as follows.

Definition 1. *A pricing function $p_{\epsilon,\lambda}$ is* error-monotone *in dataset D if for every parameters δ_1, δ_2, $\mathbb{E}\left[\epsilon(\hat{h}_{\lambda}^{\delta_1}(D), D)\right] \leq \mathbb{E}\left[\epsilon(\hat{h}_{\lambda}^{\delta_2}(D), D)\right]$ implies that $p_{\epsilon,\lambda}(\delta_1, D) \geq p_{\epsilon,\lambda}(\delta_2, D)$.*

The error monotonicity property implies that the price does not depend on the actual parameter δ of the mechanism, but on the error that this parameter induces.

Arbitrage-freeness. The final property is *arbitrage-freeness.* Suppose a buyer wants to buy one model instance with a small error but large price. Suppose further she also buys more of such model instances at different prices, the sum of all of which is lower than that of the desired single model instance. Meanwhile, suppose she is able to "combine" the latter set of model instances to construct a new model instance with an error smaller than the originally desired single model instance. In this case, she would rather just buy the latter set of model instances instead of the original model instance to get an error lower than what the market is set up for. Such a situation is called *arbitrage.* For the market to work well, we need to ensure that it is *arbitrage-free*, i.e., situations such as these do not happen (or are extremely unlikely).

Definition 2 (k-Arbitrage). *We say that a pricing function $p_{\epsilon,\lambda}$ exhibits k-arbitrage in dataset D if there exist δ_0, δ_1, $\delta_2, \cdots, \delta_k$, and a function $g : \mathcal{H}^k \to \mathcal{H}$ such that*

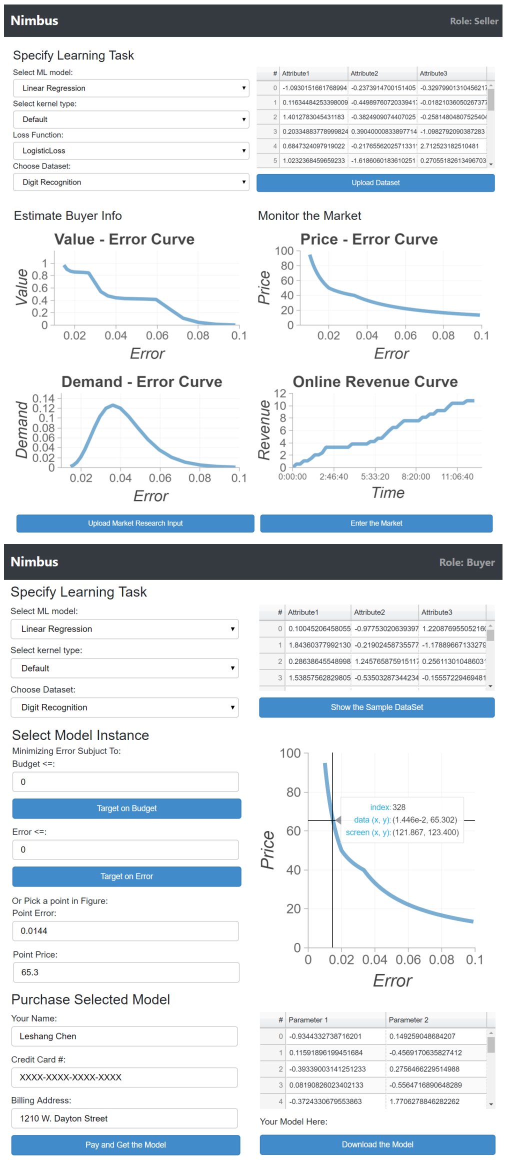
(1) $\sum_{i=1}^{k} p_{\epsilon,\lambda}(\delta_i, D) < p_{\epsilon,\lambda}(\delta_0, D)$*, and*

(2) $\mathbb{E}\left[\epsilon(\tilde{h}, D)\right] \leq \mathbb{E}\left[\epsilon(\hat{h}_{\lambda}^{\delta_0}(D), D)\right]$*, where $\tilde{h}$ is the model* $\tilde{h} = g(\hat{h}_{\lambda}^{\delta_1}(D), \hat{h}_{\lambda}^{\delta_2}(D), \ldots, \hat{h}_{\lambda}^{\delta_k}(D))$ *s.t.* $\mathbb{E}\left[\tilde{h}\right] = h_{\lambda}^{*}(D)$.

Thus, we say a pricing function $p_{\epsilon,\lambda}$ is *arbitrage-free* in dataset D iff it does not exhibit k-arbitrage for any $k \in \mathbb{N}^+$.

2.3 Gaussian Mechanism: A Concrete Instance

For the rest of this paper, we will focus on the Gaussian mechanism, a concrete instance. Given a hypothesis space $\mathcal{H}$ such that each $h \in \mathcal{H}$ can be represented by a d-dimensional vector, the Gaussian mechanism, denoted $\mathcal{K}_G$, generates a model instance by adding independent Gaussian noise to the optimal model. Formally, $\mathcal{K}_G(h_{\lambda}^{*}(D), w) = h_{\lambda}^{*}(D) + w, \quad w \sim \mathcal{N}(\mathbf{0}, (\delta/d) \cdot \mathbf{I}_d)$. Gaussian Mechanism simplifies all the desired properties to a simple condition. Roughly speaking, a pricing function $p_{\epsilon,\lambda}$ satisfies all pricing function desiderata for the Gaussian mechanism $\mathcal{K}_G$ if and only if the function $p_{\epsilon,\lambda}(1/\phi(x), D)$ is nonnegative, monotone and subadditive, where $\phi(\cdot)$ is a mapping function between the noise control parameter and the desired ML model error. More details can be found in [6].



Specify Learning Task

Select ML model: Linear Regression

Select kernel type: Default

Loss Function: LogisticLoss

Choose Dataset: Digit Recognition

#	Attribute1	Attribute2	Attribute3
0	-1.0930151661768994	-0.2373914700151405	-0.3297990131045621
1	0.11634484253398009	-0.449897607203394	-0.0182103605026737
2	1.4012783045431183	-0.3824909074407025	-0.258148048075254
3	0.20334883778999824	0.39040000833897714	-1.0982792090387283
4	0.6847324097919022	-0.217655620257133	2.71252318251048
5	1.0232368459659233	-1.6186060183610251	0.27055182613496703

Estimate Buyer Info

Monitor the Market

Specify Learning Task

Select ML model: Linear Regression

Select kernel type: Default

Choose Dataset: Digit Recognition

#	Attribute1	Attribute2	Attribute3
0	0.10045206458055	-0.97753020639397	1.2208769550521
1	1.84360377992130	-0.21902458735577	-1.1788966713327
2	0.28638645548998	1.24576587591511	0.2561130104860
3	1.53857562829805	-0.53503287344234	-0.15557229469481

Select Model Instance

Minimizing Error Subjuct To:

Budget <=: 0

Error <=: 0

Or Pick a point in Figure:

Point Error: 0.0144

Point Price: 65.3

index: 328
data (x, y): (1.446e-2, 65.302)
screen (x, y): (121.867, 123.400)

Purchase Selected Model

Your Name: Leshang Chen

Credit Card #: XXXX-XXXX-XXXX-XXXX

Billing Address: 1210 W. Dayton Street

#	Parameter 1	Parameter 2
0	-0.9344332738716201	0.149259048684207
1	0.11591896199451684	-0.4569170635827412
2	-0.39339003141251233	0.2756466229514988
3	0.08190826023402133	-0.5564716890648289
4	-0.3724330679553863	1.7706278846282262

Your Model Here:

Figure 2: Nimbus GUIs. (a) Seller; (b) Buyer.

3 SYSTEM OVERVIEW

We discuss Nimbus's architecture in this section.

Front-end. There are two graphical user interfaces (GUIs) for the seller and the buyer, respectively, both implemented in Python using the Bokeh [2] library, as shown in Figure 2. Through the GUI, the seller first uploads the datasets, ML models for sale, and the estimated market information. He/She also has the option to specify additional constraints on the price. For example, he/she can set prices for ML model instances with certain amount of error. Once the broker obtains the information and publish the ML models associated with their prices, the real time revenue will be presented to the seller. The buyer's GUI allows the buyer to choose a particular model instance by either picking a point in the price-accuracy curve or writing a simple query, based on his/her budget/accuracy requirements. After the broker computes the desired model instance and the price, the buyer needs to pay for the price via online payment methods such as credit cards or debit cards. Finally, the purchased model instance will become downloadable through the buyer's GUI.

Back-end. The back-end is also implemented in Python. There are two parts, i) generating the model instance, and ii) computing the price. Given the data and ML model information provided by the seller, we first generate the optimal model instance for each fixed hypothesis. Next, taking into market estimation and price requirements, the price computation engine computes the pricing function (on the model instance space) that maximizes the seller's revenue. Noting that the revenue optimization problem is in general $co\mathcal{NP}$-hard [6], we use a dynamic programming algorithm to approximately solve it. We stress that the above process is a one time cost, i.e., this only needs to be done once before any buyer joins the market. Now suppose a buyer comes. Since one property of the pricing function is monotonicity w.r.t. model accuracy, the pricing function is first transformed from the model instance domain to the model accuracy domain and then shown to the buyer via the front-end. The buyer then specifies accuracy or budget constraints by i) issuing a simple optimization query, or ii) simply picking a point on the pricing function (of model accuracy). Either way, the back-end can then easily compute the desired noise control parameter, which is then used to generate the desired noisy model instance. The price is then simply the value of the pricing function on the desired noisy model instance.

4 DEMONSTRATION SCENARIOS

We divide the demonstration into 3 phases: i) a brief introduction to model based pricing with a simple example, ii) a "hands-on" phase in which attendees can play the role of the seller and the buyer , and iii) a performance comparison of the revenue optimization and other naive approaches. We now explain them in details.

A brief introduction: In this phase, we present the market setup, explain the model generation mechanism and pricing process, and highlight the desiderata as well as the revenue optimization. One running example may be used throughout this phase, where a seller is selling a logistic regression model on a private dataset and a buyer with budget constraint is interested in buying the model. We will show how Nimbus enables a seller to obtain more revenue and a buyer to glean useful model with limited budget.

Hands-on Trial: During this phase, attendees are able to play the role of both seller and buyer. As a seller, an attendee may specify different ML models/hypothesizes, different number of fixed price values (as pricing constraints), different buyer distribution and demand curves (as estimated market information). Simulated buyers will be generated and the real time revenue will be shown to the attendee. As a buyer, an attendee may specify the model instances by either choosing a point on the price-accuracy curve or writing a simple query, pay the price using a simulated credit card and download the model instances. A model combiner is also provided to demonstrate why a buyer cannot obtain arbitrage by combining model instances.

Performance Comparison: In this phase, we compare the performance of the proposed revenue optimization part with naive approaches. We vary the number of model instances to sell, the buyer distribution, the demand curve, and the constraints set by the seller. Then we show the price curve, revenue curve, and the runtime curve of our proposed approach along with other methods. A set of "offline" results will be presented. This will give the attendees a better understanding of the benefits of the revenue optimization engine.

5 CONCLUSION

We demonstrate Nimbus, a generic framework towards model based pricing for ML in a data market. Nimbus allows the seller to sell various model instances with different price constraints while enables the buyer to buy customized model instances with appropriate price.

REFERENCES

[1] [n. d.]. Big Data Exchange. www.bigdataexchange.com.
[2] [n. d.]. Bokeh. https://github.com/bokeh/bokeh.
[3] [n. d.]. QLik Data Market. qlik.com/us/products/qlik-data-market.
[4] L. Chen et al. 2017. Model-based Pricing: Do not Pay for More than What You Learn!. In *SIGMOD DEEM Workshop*.
[5] L. Chen et al. 2017. Towards Linear Algebra over Normalized Data. In *PVLDB*.
[6] L. Chen et al. 2019. Towards Model-based Pricing for ML in a Data Marketplace. In *SIGMOD*.
[7] Joseph M. Hellerstein et al. 2012. The MADlib Analytics Library or MAD Skills, the SQL. In *VLDB*.
[8] Ce Zhang et al. 2014. Materialization Optimizations for Feature Selection Workloads. In *SIGMOD*.