关于构建机器学习数据交易市场的报告

黄哲 **秦立** **王普**

一、 问题描述

随着机器学习技术在各行各业的深入应用，高质量的训练数据成为了驱动模型性能提升的核心燃料。然而，许多公司，特别是刚刚起步应用人工智能的企业，常常面临着难以获取相关、有效训练数据的困境。这催生了一个自然的需求：建立一个能够高效、实时地匹配数据买卖双方的数据交易市场。

但是，构建这样一个市场并非易事，其根本性的挑战源于数据本身作为一种商品的独特性。首先，数据可以被轻易地以接近零的成本无限复制，这使得传统的基于稀缺性的定价模型失效。其次，数据的价值通常是“组合性”的，即多份数据集结合在一起所产生的预测能力，可能远远超过它们各自独立价值的总和，这使得为单个数据集定价变得极为复杂。再者，数据的有效性与具体的预测任务紧密相关，例如，一份对冲基金认为极具价值的卫星图像数据，对于一个想预测库存的物流公司可能毫无用处。这意味着买家在没有实际使用数据进行模型训练之前，很难预先准确评估其价值。最后，现有的在线广告拍卖或预测市场等机制，都无法很好地应对这些挑战。它们要么假设商品不可复制，要么假设买家对商品价值有清晰的预判，这些前提在数据交易的场景中都不成立。因此，这篇论文的核心目标，就是设计一套全新的算法框架，来解决上述难题，构建一个真正可行的数据市场。

二、 论文整体思路

为了解决上述问题，论文提出了一套完整且精巧的双边市场模型，清晰地定义了买家、卖家和市场平台三方的角色及其互动方式。整个交易流程的设计思想非常新颖，巧妙地规避了让买家直接为“数据”付费的难题，而是转向为“模型性能的提升”付费。

这个流程是这样运作的：首先，数据卖家（例如，拥有顾客匿名客流数据的零售店）将他们的数据流提供给市场平台。数据买家（例如，希望预测未来商品需求的物流公司）则向市场提交一个具体的预测任务（如历史库存数据）以及一个“准确率估值”，即他们愿意为每一个百分点的预测准确率提升支付多少钱。

接下来，市场平台扮演了核心的撮合与计算角色。平台并不会直接将原始数据交给买家，这解决了卖家对于数据泄露和失控的担忧。相反，市场平台会根据买家对准确率的报价以及平台自身动态调整的“信息价格”，来决定“分配”给这个任务的数据质量。这个分配过程可能不是提供全部数据，而是提供带有一定噪声或经过部分采样的数据。然后，平台利用这些分配好的数据，运行一个机器学习模型，为买家生成其任务所需的预测结果。

交易的最后一步是结算和分配。买家根据模型预测结果的实际准确率提升来支付费用，这个费用是基于他们自己先前的报价计算的。这样，买家支付的是可验证的效果，而非难以估值的原始数据。市场平台在收到这笔收入后，并不会将其独占，而是会根据一套公平的分配机制，将其分给那些为这次成功预测贡献了价值的数据卖家们。同时，市场平台还会根据本次交易的结果，利用在线学习算法来更新其内部的“信息价格”，以便在未来的交易中能够最大化整体收入。整个过程形成了一个动态的、不断学习和优化的闭环系统。

三、 关键定理描述

为了让上述市场机制能够稳健运行，论文提出并证明了几个关键的性质，这些性质由相应的算法和机制来保证，可以看作是这套系统的理论基石。

首先是**买家报价的真实性（Truthfulness）**。如何确保买家报出的“准确率估值”是其内心真实的想法，而不是为了投机而随意报出的价格？论文采用了一种源于诺贝尔奖得主迈尔森（Myerson）的拍卖理论的支付函数设计。其精髓在于，买家最终支付的费用与他的出价和最终获得的收益（即准确率提升）巧妙地耦合在一起。在这种机制下，买家说真话成为了最优策略。如果报价过低，他获得的数据质量会很差，预测效果不佳，自身效用不高；如果报价过高，虽然能获得高质量数据和好的预测效果，但支付的成本会超出其真实估值，得不偿失。唯有诚实报价，才能实现自身利益的最大化。这一定理保证了市场能够获取到真实的需求信息。

其次是**市场收入的最大化（Revenue Maximization）**。市场平台如何为“信息”设定一个最优的价格，从而在长期运营中获得最大收益？由于买家的任务和估值是动态变化的，平台不可能预知这个最优价格。论文引入了“乘法权重更新算法”（Multiplicative Weights Algorithm），这是一种强大的在线学习方法。简单来说，市场平台会维护一系列候选价格，并将它们看作一群“专家”。每完成一笔交易，平台就会评估在这次交易中每个“专家”（即每个候选价格）本可以带来多少收入。然后，它会给表现好的“专家”增加权重，给表现差的“专家”降低权重。通过持续的交易和迭代，平台能够动态地、自适应地学习出在当前环境下接近最优的价格策略，从而实现长期收入的最大化。

再次是**卖家收入分配的公平性（Fairness）**，这一点尤为关键。当多个卖家的数据共同促成了一次成功的预测时，如何公平地论功行赏？特别是在不同卖家的数据可能高度相关（例如，相邻两家商场的客流数据）的情况下，简单地评估单个数据的边际贡献是行不通的。论文采用了博弈论中著名的“夏普利值”（Shapley Value）思想。夏普利值的核心理念是，一个卖家（或其数据）的贡献，是通过计算在所有可能的数据组合中，该数据的加入平均能带来多大的价值提升来衡量的。这完美地解决了数据相关性的问题。然而，精确计算夏普利值的计算量是天文数字。因此，论文提出的算法通过随机抽样的方式来近似计算夏普利值，在保证结果足够精确的同时，将计算时间控制在可接受的范围内。

最后，也是本文一个非常重要的创新点，是**对数据复制行为的稳健性（Robustness to Replication）**。如果一个卖家通过简单地复制自己的数据，并将其作为多个“新数据”提交给市场，他是否能利用夏普 LEY 值的机制漏洞来骗取更多的收入分成？答案是肯定的，标准的夏普利

值对此无能为力。为了解决这个致命问题，论文设计了一种“抗复制”的增强版分配机制。该机制在分配收入前，会先计算市场上所有数据两两之间的“相似度”。一个卖家的最终收入，不仅取决于其数据的贡献大小，还会根据其数据与市场上其他数据的相似程度进行惩罚性地“衰减”。如果一个卖家提交了大量重复或高度相似的数据，那么这些数据在分配收入时权重会被大幅降低。这从根本上打压了投机性的复制行为，激励卖家提供真正独特、有价值的信息，保证了市场的长期健康发展。

四、 核心算法介绍

4.1 特征分配函数 $\mathcal{AF}^*$

特征分配函数根据市场价格 p_n 和买家出价 b_n 决定数据质量。当出价低于市场价格时，特征会加入噪声：

$$\tilde{X}_j(t) = X_j(t) + \max(0, p_n - b_n) \cdot \mathcal{N}(0, \sigma^2)$$

其中噪声大小由价格差决定，标准差由 σ 控制。这部分代码实现如下：

```python
def allocate_features(
    X: NDArray[np.float64],
    p: float,
    bid: float,
    sigma: float = 1.0,
    rng: np.random.Generator | None = None,
) -> NDArray[np.float64]:
    noise_scale = max(0.0, p - bid)
    if noise_scale == 0.0:
        return X.copy()
    noise = rng.normal(loc=0.0, scale=sigma * noise_scale, size=X.shape)
    return X + noise
```

4.2 收入函数 $\mathcal{RF}^*$

收入函数的核心是根据买家的出价和市场设定的价格来计算收入。通过 Myerson 的支付规则，我们可以得到如下收入公式：

$$\mathcal{RF}^*(p_n, b_n, Y_n) = b_n \cdot G\left(Y_n, \hat{Y}_n\right) - \int_0^{b_n} G\left(Y_n, \hat{Y}_z\right) \mathrm{d}z$$

其中，$G\left(Y_n, \hat{Y}_n\right)$ 表示增益函数，衡量预测结果和目标值之间的差距，$\hat{Y}_n$ 是模型在买家数据上预测的结果，而积分部分表示了买家在不同出价下的增益。

在代码实现中，`revenue_function` 计算了 Myerson 支付规则下的收入。首先计算了实际增益 $G\left(Y_n, \hat{Y}_n\right)$，然后通过数值积分计算总收入：

```python
def revenue_function(
    p: float,
    bid: float,
    y_true: NDArray[np.float64],
    model: MLModel,
    X_alloc_func: Callable[[float], NDArray[np.float64]],
```

```
    integral_grid: int = 15,
) -> tuple[float, float]:
    Xb = X_alloc_func(bid)
    y_test, y_test_hat = model.fit_predict(Xb, y_true)
    g_bid = gain(y_test, y_test_hat)

    zs = np.linspace(0.0, bid, integral_grid)
    gs = [
        gain(
            model.fit_predict(X_alloc_func(z))[0],
            model.fit_predict(X_alloc_func(z))[1],
        ) for z in zs
    ]
    integral = float(np.trapezoid(gs, zs))

    rev = bid * g_bid - integral
    rev = max(0.0, rev)  # Ensure non-negative revenue
    return rev, g_bid
```

该函数首先计算实际增益 `g_bid`，然后通过数值积分计算在不同出价下的增益。最后，使用 Myerson 支付规则计算并返回实际收入。收入计算的准确性和效率是实现该市场机制的关键。

4.3 价格更新函数 $\mathcal{PF}^*$（Algorithm 1）

根据每个候选价格的收益，市场动态地调整价格以最大化总收入。具体来说，使用 MWU 来实现这一目标：

$$w_i^{(n+1)} = w_i^{(n)} \cdot \left(1 + \delta \cdot g_i^{(n)}\right)$$

其中，$g_i^{(n)}$ 是每个候选价格的收益，δ 是更新步长。

`MWUPricer` 类实现了这一更新规则。通过计算每个候选价格的收益，并根据这些收益更新价格权重：

```
class MWUPricer:
    def __init__(self, b_min, b_max, N, L_lipschitz=1.0, rng=None):
        self.rng = rng or np.random.default_rng()
        # epsilon-net granularity
        self.eps = 1.0 / (L_lipschitz * np.sqrt(max(N, 1)))
        self.grid = np.arange(b_min, b_max + 1e-9, self.eps, dtype=np.float64)
        self.num_exp = len(self.grid)
        self.delta = np.sqrt(np.log(self.num_exp) / max(N, 1))
        # Initialise weights
        self.weights = np.ones(self.num_exp, dtype=np.float64)

    def update(self, gains: NDArray[np.float64]):
        self.weights *= 1.0 + self.delta * gains
```

每次更新时，权重通过 `update` 方法进行调整。通过这种方式，市场能够根据收益动态调整价格，以达到最佳的市场均衡。

4.4 Shapley 分配 $\mathcal{PD}_A^*$（Algorithm 2）

Shapley 值是用来公平分配收入的一种方法，衡量每个卖家在市场中所做的贡献。我们使用蒙特卡罗方法来近似计算 Shapley 值：

$$\psi^{\text{shapley}}(m) = \sum_{T \subseteq [M] \setminus \{m\}} \frac{|T|!(M - |T| - 1)!}{M!} \left(G\left(Y_n, \mathcal{M}\left(X_{T \cup \{m\}}\right)\right) - G(Y_n, \mathcal{M}(X_T)) \right)$$

代码中的 `shapley_approx` 函数通过计算每个卖家的边际贡献来近似 Shapley 值：

```python
def shapley_approx(
    y_true: NDArray[np.float64],
    X: NDArray[np.float64],
    model: MLModel,
    num_samples: int = 200,
    rng: np.random.Generator | None = None,
) -> NDArray[np.float64]:
    contrib = np.zeros(X.shape[0])
    for _ in range(num_samples):
        perm = rng.permutation(X.shape[0])
        prefix_indices = tuple()
        for j in perm:
            contrib[j] += marginal_gain(prefix_indices, j)
            prefix_indices += (j,)
    contrib /= num_samples
    total = contrib.sum()
    if total > 0:
        contrib /= total  # Normalize to sum to 1
    return contrib
```

通过反复随机排列特征并计算每个特征的边际贡献，最终得到每个卖家贡献的 Shapley 值。

4.5 鲁棒 Shapley 分配 $\mathcal{PD}_B^*$（Algorithm 3）

为了避免特征冗余问题，论文提出了一种鲁棒化的 Shapley 分配方法，通过计算特征之间的余弦相似度，对冗余特征进行惩罚。具体公式如下：

$$\psi_n(m) = \hat{\psi}_n(m) \cdot \exp\left(-\lambda \sum_{j \neq m} SM(X_m, X_j) \right)$$

在代码中，`robustify_shapley` 函数实现了这一过程。它计算特征之间的余弦相似度，并对特征的 Shapley 值进行惩罚：

```python
def robustify_shapley(psi: NDArray[np.float64], X: NDArray[np.float64], lam:
float = np.log(2.0)) -> NDArray[np.float64]:
    sim = np.abs(X @ X.T)  # Cosine similarity
    penalty = np.exp(-lam * (sim.sum(axis=1) - 1.0))  # Exclude self-similarity
    return psi * penalty / penalty.sum()
```

这样，市场可以对相似特征进行适当的惩罚，避免冗余特征导致的收入不公平。

五、 实验结果与分析

运行结果如 Figure 1 所示：

在本次实验中，我们生成了一个包含 20 个卖家和 120 个买家的市场。卖家的特征是通过与潜在因子相结合生成的，每个买家的目标向量则是基于部分卖家的特征，且买家有自己的私人估值 mu。买家在市场中根据其私人估值和市场价格 p_n 选择合适的出价，卖家根据市场价格提供相应的特征。

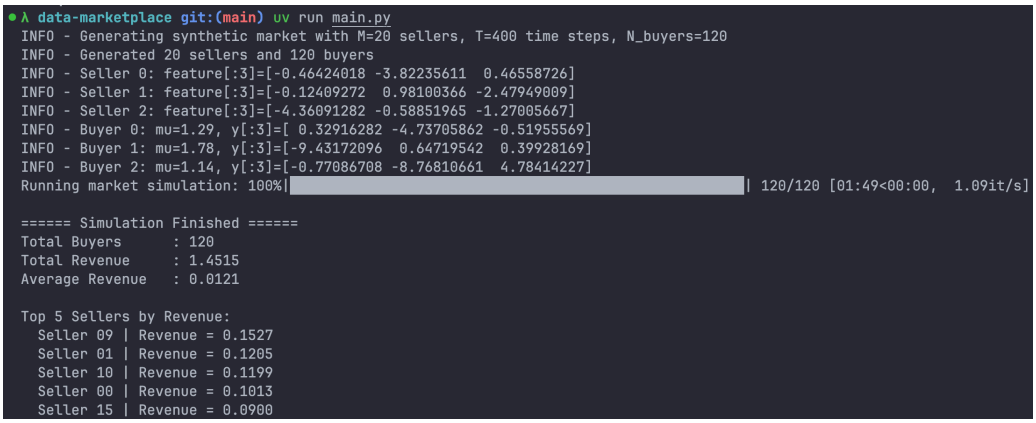

图 1：实验结果

5.1 收入分析

```
Total Buyers      : 120
Total Revenue     : 1.4515
Average Revenue   : 0.0121
```

模拟运行的结果显示市场总收入为 1.4515，平均每个买家的收入为 0.0121。这个结果表明，在模拟中，市场成功产生了正向收益，即买卖双方都能从交易中受益。

5.2 卖家收入分布

卖家的收入呈现出不均匀分布，部分卖家因提供更有价值的特征而获得更高的收入。前五名卖家的收入分别为：

```
Top 5 Sellers by Revenue:
  Seller 09 | Revenue = 0.1527
  Seller 01 | Revenue = 0.1205
  Seller 10 | Revenue = 0.1199
  Seller 00 | Revenue = 0.1013
  Seller 15 | Revenue = 0.0900
```

这表明市场机制通过公平分配机制保证了大部分卖家根据贡献获得了相应的收入。