# FDML: A Collaborative Machine Learning Framework for Distributed Features

**Yaochen Hu[1], Di Niu[1], Jianming Yang[2], Shengping Zhou[2]**

*[1] ECE, University of Alberta; [2] PCG, Tencent*

## Introduction

The effectiveness of a machine learning model depends on the availability of high-quality features. An important scenario is to collaboratively learn from distributed features, where multiple parties may possess different features about a same sample, but do not wish to share these features (e.g. sensitive user features) with each other.

We aim to improve the predictive power at one party by leveraging additional features from another domain or party, without requiring any party to share its features. We design, implement and extensively evaluate a practical Feature Distributed Machine Learning (FDML) system based on real-world datasets.

## Design Goals

To make the solution *practical* with the most conservative assumption on information sharing, we bear the following goals:

- To minimize information leakage, no party should share its feature set. Neither should any of its local model parameters be communicated to other parties.
- The prediction made by the joint model should outperform the prediction made by each isolated model trained only with a single party's feature set, provided that such improvement from joint features also exists in centralized training.
- The joint model produced should approach the model trained in a centralized manner if all the features were collected centrally.
- The system should be efficient in the presence of both large numbers of features and samples.

## FDML Model

Consider a system of $m$ different parties, each party holding different aspects about the same training samples. Let $\{(\xi^1_i, \xi^2_i, ..., \xi^m_i), y_i\}, \forall i,$ represents the set of $n$ training samples, $\xi^j_i$ being the features of the $i^{th}$ sample located on $j^{th}$ party, and $y_i$ being the label. We adopt a specific class of **composite** model that has the form

$$p(x,\xi) = \sigma\Big(\sum_{j=1}^{m} \alpha^j(x^j, \xi^j)\Big),$$

Where $\sigma(\cdot)$ is some continuous differentiable function, $\alpha^j(\cdot)$ is local prediction model with parameter $x^j$, as illustrated in Fig. 1.

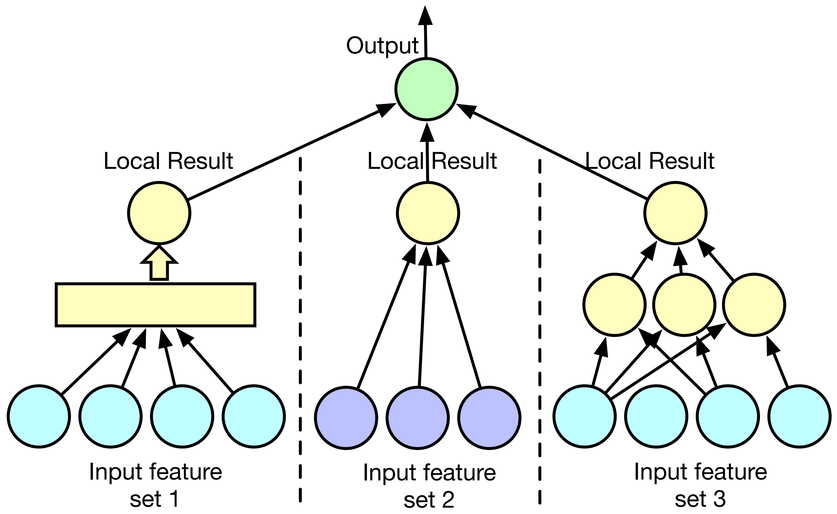


Fig. 1 Illustrating FDML model, where the local predictions, each depending on the local model on a party, are aggregated into a final output

## Algorithm and System

Train the model by solving the optimization problem with loss function $L(\cdot)$

$$\underset{x}{\text{minimize}} \frac{1}{n}\sum_{i=1}^{n} L(x; \xi_i, y_i)$$

For each party $j$, the gradient w.r.t the local parameter $x^j$ is

$$H(\text{global predictions})\frac{\partial \alpha^j(x^j, \xi^j_{i(t)})}{\partial x^j}$$

where $i(t)$ is the index of sample drawn at iteration $t$. Only the global prediction and other local data is needed for local parameter update.

**Gradient Decent Algorithm**

**Local party:** (1) evaluate the local prediction and push it to central server; (2) get the global prediction from central server; (3) evaluate the gradient and update the local parameters.

**Central server:** (1) upon push request, update the corresponding global prediction; (2) upon pull request, return the global prediction.

- It is shown that this algorithm converges in $O(1/\sqrt{T})$, $T$ being the number of iterations.

**System Implementation Highlights**

- Sample indexing, alignment and shuffling.
- Support stale synchronous update.
- Privacy is further protected by perturbing the local predictions.

## Experiment

**Datasets:** (1) T*encent MyApp dataset* contains 5*M* labeled samples indicating whether a user will download an app or not. Each sample contains around 8.7*K* (sparse) features, among which around 7*K* features come from Tencent MyApp itself, while the remaining 1.7*K* features are from the other two apps. (2) *a9a*, a classical census dataset, where the prediction task is to determine whether a person makes over 50*K* a year. There are 48,842 samples, each with 124 features. 32,661 samples are training data and 16,281 samples are testing data. We split the 124 features into two sets of 67 and 57.

**Scenarios:** We run both a **logistic regression (LR)** and a two layered fully connected **neural network (NN)** under three different training schemes for both data sets:

- **Local:** only use the 7*K* local features from MyApp or the 67 features of *a9a* to train a model.
- **Centralized:** collect all the 8.7*K* features to a central server or using all the 124 features in *a9a* and train the model using the standard mini-batched SGD.
- **FDML:** use FDML system to train a joint model for app recommendation based on all 8.7*K* features distributed in three apps or train the *a9a* classification model on all 124 features from two different parties, without centrally collecting data.

**Results**: As shown in Table 1 and Table 2, FDML can closely approximate centralized training (the latter collecting all data centrally), while significantly outperforming the models trained only based on the local features.

Table 1: The performance on Tencent MyApp data.

| Algorithm | Train loss | Test loss | Test AUC | Time(s) |
|---|---|---|---|---|
| LR local | 0.1183 | 0.1220 | 0.6573 | 546 |
| LR centralized | 0.1159 | **0.1187** | **0.7037** | 1063 |
| LR FDML | **0.1143** | 0.1191 | 0.6971 | 3530 |
| NN local | 0.1130 | 0.1193 | 0.6830 | 784 |
| NN centralized | **0.1083** | 0.1170 | **0.7284** | 8051 |
| NN FDML | 0.1101 | **0.1167** | 0.7203 | 4369 |

Table 2: The performance on a9a data.

| Algorithm | Train loss | Test loss | Test AUC | Time(s) |
|---|---|---|---|---|
| LR local | 0.3625 | 0.3509 | 0.8850 | 41 |
| LR centralized | 0.3359 | 0.3247 | 0.9025 | 45 |
| LR FDML | **0.3352** | **0.3246** | **0.9026** | 99 |
| NN local | **0.3652** | 0.3484 | 0.8864 | 53 |
| NN centralized | 0.4008 | **0.3235** | **0.9042** | 57 |
| NN FDML | 0.4170 | 0.3272 | 0.9035 | 110 |