Federated NBA Learning

John Fattore

Google Colab Link - https://colab.research.google.com/drive/1A337Zb8JivI-HucdIfSA7bRX1ReNx83r?usp=sharing

Abstract: This model is trained through federated learning to predict NBA team win percentages based on season stats.

Data: All the data is obtained from nba.com/stats/teams/traditional, pasted into a txt file and organized by a python script. The data contains 6 parameters including 3-Point %, 3-Point Attempts, Field Goal %, Turnovers, Assists, Offensive Rebounds. The data spans 10 seasons between 2008 and 2019, split up between training and testing data.

Linear Regression Model: The linear model has 6 parameters: 3-Point %, 3-Point Attempts, Field Goal %, Turnovers, Assists, and Offensive Rebounds. The linear regression uses weights assigned to each parameter and an intercept in order to calculate the prediction.

3-Point %	3-Point Att.	Field Goal %	Turnovers	Assists	Off. Rebounds	Intercept
0.388	0.251	0.492	0.148	0.25	0.096	0.0104

Gradient Descent: To optimize the linear regression's MSE cost function, gradient descent is utilized. Partial derivatives with respect to each of the weights are calculated and used to make a "step" closer to an optimized model. Functions that calculate these functions are included in the google colab link.

Federated Learning: The model is trained using federated learning, more precisely FedSGD. Federated learning exists to tackle the problem of training models using sensitive user data, while maintaining privacy. This training algorithm sends a global model to devices that find gradients based on their local data. These gradients are then averaged together and used to make the next step. In this distributed learning algorithm, communication costs dominate because devices are connected through costly wifi and 5G networks, not high speed cable. Training is done on the edge devices in order to promote privacy, but also to reduce communication costs. A set of gradients is much less information to communicate compared to training sets. My FedSGD differs from the FedAvg algorithm (figure 2) because it doesn't let the local devices iterate multiple times through the training process.

Learning Rate: The learning rate was set low for the training (0.0000001), partially because the gradient descent algorithm does not include the small scalar fraction (1/m). This term was omitted for convenience and is easily compensated for by tweaking the learning rate. The model is also simple and lowering the learning rate slightly increases model accuracy.

Results: The model performed well enough, with an MAPE score of about 30%, MAE of 0.11, MSE of 0.0205 and an RMSE of 0.143. The MAPE score is a solid indication of accuracy and the MAE score means each prediction is off on average 11%. A positive observation is that the federated and non federated models are nearly identical in performance. The similarities are aided by the homogeneity of the data, but still supports the practical use of federated learning. Figure 1 shows how the linear model converges at around 600 epoch. After a certain amount of iterations, performance plateaus.

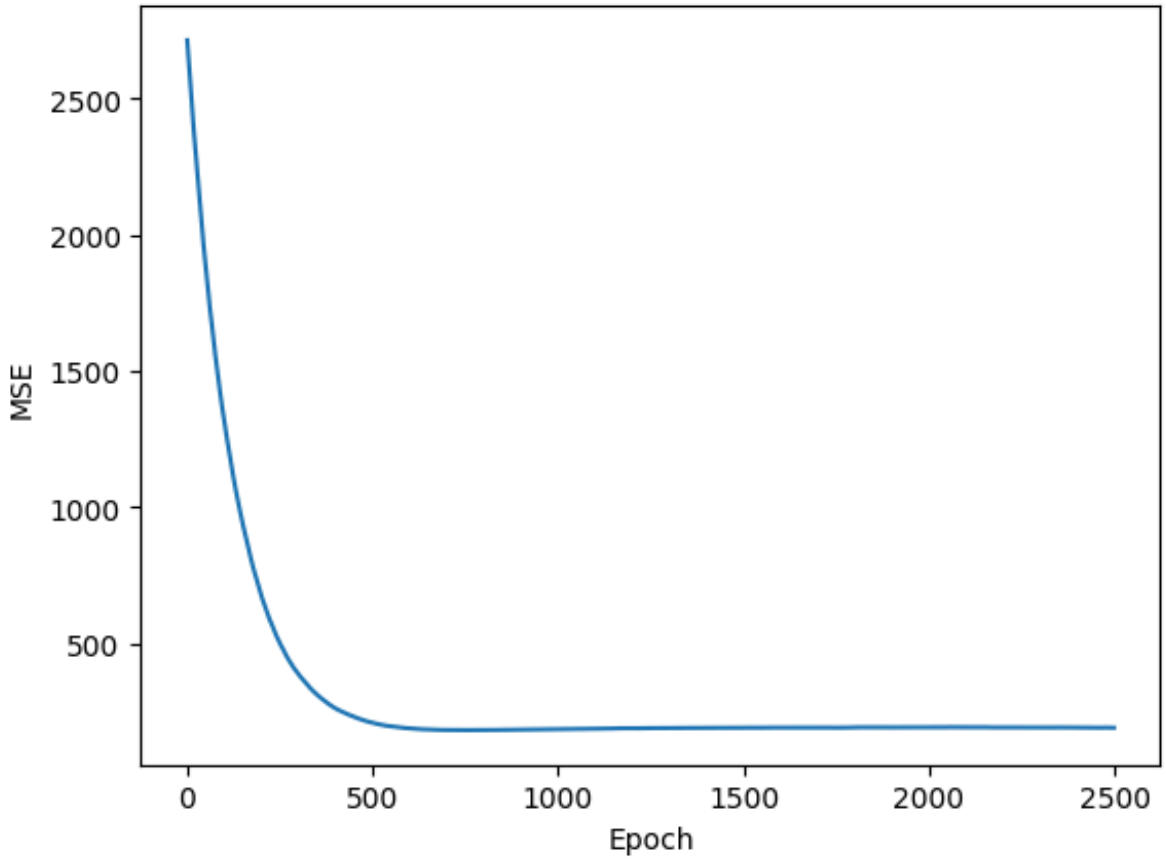

Figure 1: Epoch vs MSE Federated Learning Linear Regression Model

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:

initialize w_0

for each round $t = 1, 2, \dots$ **do**

$m \leftarrow \max(C \cdot K, 1)$

$S_t \leftarrow$ (random set of m clients)

for each client $k \in S_t$ **in parallel do**

$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$

$m_t \leftarrow \sum_{k \in S_t} n_k$

$w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ *// Erratum*[4]

ClientUpdate(k, w): *// Run on client k*

$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)

for each local epoch i from 1 to E **do**

for batch $b \in \mathcal{B}$ **do**

$w \leftarrow w - \eta \nabla \ell(w; b)$

return w to server

Figure 2: The Federated Averaging (FedAvg) Algorithm