Federated Learning Algorithms

Andrew Zhang, Jiayi Tong, Yong Chen

University of Pennsylvania Biostatistics Dept.

Introduction

The objective of this research is to understand and experiment with different approaches to federated learning. Federated Learning algorithms are unique in that they are able to train machine learning models while keeping user-level data secure and private. Normally, Machine learning models must centralize training data to create the model. However with federated learning, the data never needs to be moved, instead only summary statistics are transmitted. The use cases of this range from using patient data in hospitals to customer data in banks.

Approaches

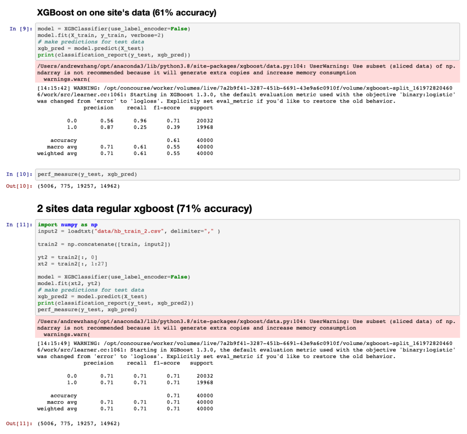
As a part of this project, I had experimented with various approaches to federated machine learning

1. PennCil Group's federated Logistic regression algorithm. This algorithm used taylor approximations and hessians to create a loss function that depends only on summary level statistics.
2. Distributed SVM techniques are primarily for high dimensional datasets. However in practice and in industry, SVM's have fallen out of favor due to inefficiency in training time.
3. XGBoost is a novel decision tree learning algorithm which is both quick and accurate, leading to wide adoption in industry.

Ensemble Learning

Perhaps the simplest federated learning implementation is ensemble learning where a model is trained at each site and then voting occurs between the models. This technique is easy to setup and trains quickly but runs under the assumption that all data sites have a sufficient amount of data to train an efficient model. It also runs under the assumption that each location has data of the same distribution.

Implementation of Ensemble XGBoost

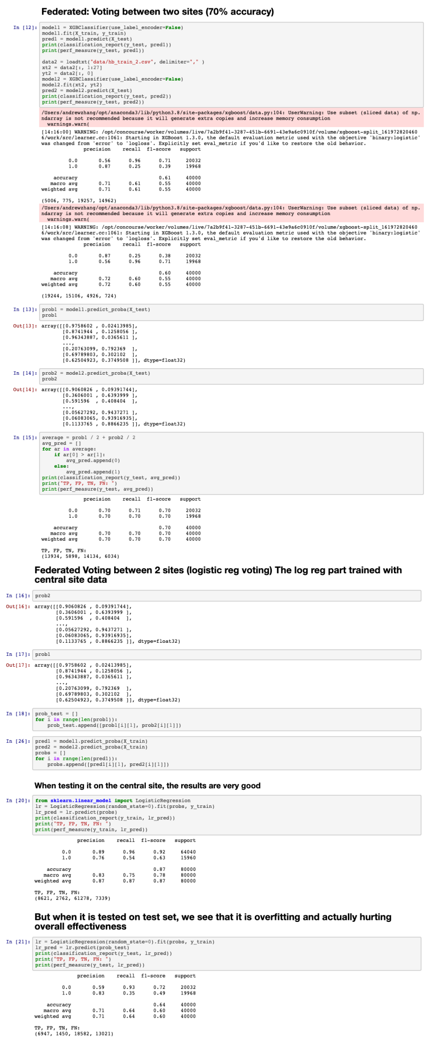


XGBoost on one site's data (61% accuracy)

```
In [9]:  model = XGBClassifier(use_label_encoder=False)
         model.fit(X_train, y_train, verbose=2)
         # make predictions for test data
         xgb_pred = model.predict(X_test)
         print(classification_report(y_test, xgb_pred))

/Users/andrewzhang/opt/anaconda3/lib/python3.8/site-packages/xgboost/data.py:104: UserWarning: Use subset (sliced data) of np.
ndarray is not recommended because it will generate extra copies and increase memory consumption
  warnings.warn(
[14:15:42] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-split_161972820460
6/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
              precision    recall  f1-score   support

         0.0       0.56      0.96      0.71     20032
         1.0       0.87      0.25      0.39     19968

    accuracy                           0.61     40000
   macro avg       0.71      0.61      0.55     40000
weighted avg       0.71      0.61      0.55     40000

In [10]:  perf_measure(y_test, xgb_pred)

Out[10]:  (5006, 775, 19257, 14962)
```

2 sites data regular xgboost (71% accuracy)

```
In [11]:  import numpy as np
          input2 = loadtxt("data/hb_train_2.csv", delimiter="," )

          train2 = np.concatenate([train, input2])

          yt2 = train2[:, 0]
          xt2 = train2[:, 1:27]

          model = XGBClassifier(use_label_encoder=False)
          model.fit(xt2, yt2)
          # make predictions for test data
          xgb_pred2 = model.predict(X_test)
          print(classification_report(y_test, xgb_pred2))
          perf_measure(y_test, xgb_pred)

/Users/andrewzhang/opt/anaconda3/lib/python3.8/site-packages/xgboost/data.py:104: UserWarning: Use subset (sliced data) of np.
ndarray is not recommended because it will generate extra copies and increase memory consumption
  warnings.warn(
[14:15:49] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-split_161972820460
6/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
              precision    recall  f1-score   support

         0.0       0.71      0.71      0.71     20032
         1.0       0.71      0.71      0.71     19968

    accuracy                           0.71     40000
   macro avg       0.71      0.71      0.71     40000
weighted avg       0.71      0.71      0.71     40000

Out[11]:  (5006, 775, 19257, 14962)
```



Federated: Voting between two sites (70% accuracy)

```
In [12]:  model1 = XGBClassifier(use_label_encoder=False)
          model1.fit(X_train, y_train)
          pred1 = model1.predict(X_test)
          print(classification_report(y_test, pred1))
          print(perf_measure(y_test, pred1))

          data2 = loadtxt("data/hb_train_2.csv", delimiter="," )
          xt2 = data2[:, 1:27]
          yt2 = data2[:, 0]
          model2 = XGBClassifier(use_label_encoder=False)
          model2.fit(xt2, yt2)
          pred2 = model2.predict(X_test)
          print(classification_report(y_test, pred2))
          print(perf_measure(y_test, pred2))

/Users/andrewzhang/opt/anaconda3/lib/python3.8/site-packages/xgboost/data.py:104: UserWarning: Use subset (sliced data) of np.
ndarray is not recommended because it will generate extra copies and increase memory consumption
  warnings.warn(
[14:16:00] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-split_161972820460
6/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
              precision    recall  f1-score   support

         0.0       0.56      0.96      0.71     20032
         1.0       0.87      0.25      0.39     19968

    accuracy                           0.61     40000
   macro avg       0.71      0.61      0.55     40000
weighted avg       0.71      0.61      0.55     40000

(5006, 775, 19257, 14962)
/Users/andrewzhang/opt/anaconda3/lib/python3.8/site-packages/xgboost/data.py:104: UserWarning: Use subset (sliced data) of np.
ndarray is not recommended because it will generate extra copies and increase memory consumption
  warnings.warn(
[14:16:08] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-split_161972820460
6/work/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
              precision    recall  f1-score   support

         0.0       0.87      0.25      0.38     20032
         1.0       0.56      0.96      0.71     19968

    accuracy                           0.60     40000
   macro avg       0.72      0.60      0.55     40000
weighted avg       0.72      0.60      0.55     40000

(19244, 15106, 4926, 724)

In [13]:  prob1 = model1.predict_proba(X_test)
          prob1

Out[13]:  array([[0.9758602 , 0.02413985],
                 [0.8741944 , 0.1258056 ],
                 [0.96343887, 0.0365611 ],
                 ...,
                 [0.20763099, 0.792369  ],
                 [0.69789803, 0.302102  ],
                 [0.62504923, 0.3749508 ]], dtype=float32)

In [14]:  prob2 = model2.predict_proba(X_test)
          prob2

Out[14]:  array([[0.9060826 , 0.09391744],
                 [0.3606001 , 0.6393999 ],
                 [0.591596  , 0.408404  ],
                 ...,
                 [0.05627292, 0.9437271 ],
                 [0.06083065, 0.93916935],
                 [0.1133765 , 0.8866235 ]], dtype=float32)

In [15]:  average = prob1 / 2 + prob2 / 2
          avg_pred = []
          for ar in average:
              if ar[0] > ar[1]:
                  avg_pred.append(0)
              else:
                  avg_pred.append(1)
          print(classification_report(y_test, avg_pred))
          print("TP, FP, TN, FN: ")
          print(perf_measure(y_test, avg_pred))

              precision    recall  f1-score   support

         0.0       0.70      0.71      0.70     20032
         1.0       0.70      0.70      0.70     19968

    accuracy                           0.70     40000
   macro avg       0.70      0.70      0.70     40000
weighted avg       0.70      0.70      0.70     40000

TP, FP, TN, FN:
(13934, 5898, 14134, 6034)
```

Federated Voting between 2 sites (logistic reg voting) The log reg part trained with central site data

```
In [16]:  prob2

Out[16]:  array([[0.9060826 , 0.09391744],
                 [0.3606001 , 0.6393999 ],
                 [0.591596  , 0.408404  ],
                 ...,
                 [0.05627292, 0.9437271 ],
                 [0.06083065, 0.93916935],
                 [0.1133765 , 0.8866235 ]], dtype=float32)

In [17]:  prob1

Out[17]:  array([[0.9758602 , 0.02413985],
                 [0.8741944 , 0.1258056 ],
                 [0.96343887, 0.0365611 ],
                 ...,
                 [0.20763099, 0.792369  ],
                 [0.69789803, 0.302102  ],
                 [0.62504923, 0.3749508 ]], dtype=float32)

In [18]:  prob_test = []
          for i in range(len(prob1)):
              prob_test.append([prob1[i][1], prob2[i][1]])

In [26]:  pred1 = model1.predict_proba(X_train)
          pred2 = model2.predict_proba(X_train)
          probs = []
          for i in range(len(pred1)):
              probs.append([pred1[i][1], pred2[i][1]])
```

When testing it on the central site, the results are very good

```
In [20]:  from sklearn.linear_model import LogisticRegression
          lr = LogisticRegression(random_state=0).fit(probs, y_train)
          lr_pred = lr.predict(probs)
          print(classification_report(y_train, lr_pred))
          print("TP, FP, TN, FN: ")
          print(perf_measure(y_train, lr_pred))

              precision    recall  f1-score   support

         0.0       0.89      0.96      0.92     64040
         1.0       0.76      0.54      0.63     15960

    accuracy                           0.87     80000
   macro avg       0.83      0.75      0.78     80000
weighted avg       0.87      0.87      0.87     80000

TP, FP, TN, FN:
(8621, 2762, 61278, 7339)
```

But when it is tested on test set, we see that it is overfitting and actually hurting overall effectiveness

```
In [21]:  lr = LogisticRegression(random_state=0).fit(probs, y_train)
          lr_pred = lr.predict(prob_test)
          print(classification_report(y_test, lr_pred))
          print("TP, FP, TN, FN: ")
          print(perf_measure(y_test, lr_pred))

              precision    recall  f1-score   support

         0.0       0.59      0.93      0.72     20032
         1.0       0.83      0.35      0.49     19968

    accuracy                           0.64     40000
   macro avg       0.71      0.64      0.60     40000
weighted avg       0.71      0.64      0.60     40000

TP, FP, TN, FN:
(6947, 1450, 18582, 13021)
```

Summary

This Summer yielded many fruitful projects and learning experiences.

1. I completed the Stanford Online Machine Learning certificate.
2. I became familiar with popular tools such as Tensorflow, Sklearn, XGBoost, LightBGM, and others.
3. I conducted literature review of distributed SVM methods and investigated how it stacks up with alternative federated learning approaches.
4. I pivoted to XGBoost and initiated research into both the open-source code and Berkeley's Rise Lab's research into creating histogram based, Federated XGBoost.

Web Application Project

Alongside Machine Learning projects, I helped develop a data sharing web application with Flask and MongoDB. This website aims to facilitate data transfer between sites for users of PDA (privacy preserving distributed algorithms) by allowing for summary statistics to be shared in json format.

I worked on the REST API, database, and parts of the frontend.