COVID-19 Drug Discovery and Machine Learning

Team 2, Data Science Challenge 2022

By: Shikha Patel, Baixi (Steven) Guo, Carlos Diaz, and Nathaniel Brown

LLNL-PRES-837337

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Team 2: Baixi (Steven) Guo

Degree: Computer Science & Engineering,

Applied Mathematical Sciences: Computational & Data Science Emphasis

Skills: C++, Python, Matlab

Things learned: Supervised Learning Models, Multilayer Perceptron, Google Colab, Pandas, Matplotlib, Scikit-learn, Keras

With more time: MLP by Pytorch in task 1, CNN model in task 2

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Outline

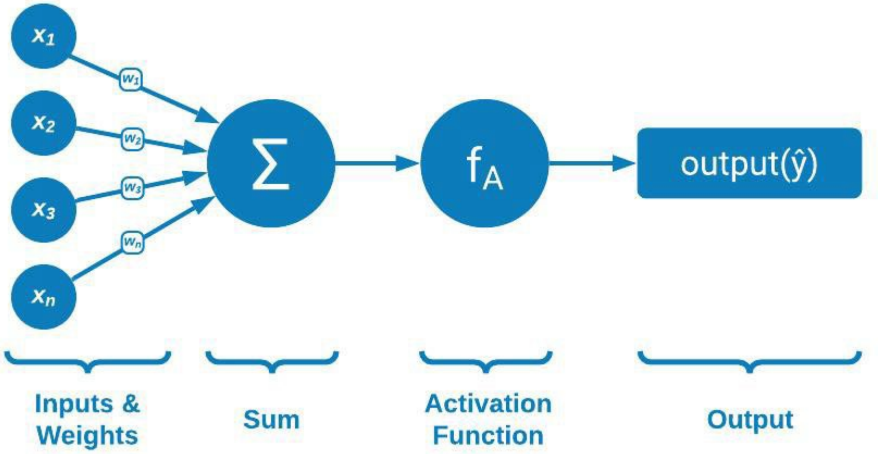
- Introduction/Motivation
- Fingerprints/SVMs (Carlos)
- Random Forests (Shikha)
- SVCs/Gradient Boosting (Nat)
- Multi-layer Perceptrons (Steven)

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Multilayer Perceptron (MLP)

- Introduction to MLP neural network
- MLP Implementation (Scikit-learn & Tensorflow/Keras)
- MLP Performance
- MLP Hyper-parameter Tuning & Observations

What is Multilayer Perceptron (MLP)



Activation Function:

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

Example: Linear, Sigmoid, Tanh, Relu, Softmax

What is Multilayer Perceptron (MLP)

Loss / Cost Function

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} L(f(x^{(i)};W), y^{(i)})$$

Example: Binary Cross Entropy Loss

Multilayer Perceptron Implementation

```
# Artificial Neural Network: Mutilayer Perceptron
from sklearn.neural_network import MLPClassifier
avg_acc = 0
for i in range(0,10):
  mlp = MLPClassifier(hidden_layer_sizes =(200), activation = 'relu', solver = 'adam',\
                      batch_size = 'auto', learning_rate = 'invscaling' , max_iter = 1000,\
                      tol = 1e-4, verbose = False, warm_start= True)
  mlp.fit(x_train_scaled, y_train)
  y_pred = mlp.predict(x_test_scaled)
  avg_acc = avg_acc + accuracy_score(y_test, y_pred)

print(avg_acc/10)
```

Scikit-learn Version

Multilayer Perceptron Implementation

```python
from keras.models import Sequential
from keras.layers import Dense

def MLP(trainX, trainY, testX, testY, n_batchs, n_epochs, n_nodes, n_layers, validation, verb):
  # # Set up model
  model = Sequential()
  # Construct Layers
  model.add(Dense(50 , input_shape = (trainX.shape[1],) , activation = 'relu'))
  for i in range(0,n_layers):
    model.add(Dense(n_nodes , activation = 'relu'))
  model.add(Dense(1 , activation = 'sigmoid'))
  # Compile Function
  model.compile(loss = 'BinaryCrossentropy' , optimizer = 'Adam' , metrics = ["accuracy" , "AUC"])
  # Fit the model
  history = model.fit(trainX , trainY , batch_size=n_batchs , epochs=n_epochs , verbose=verb , validation_split=validation)
  # Model Prediction
  predY = (model.predict(testX) > 0.5).astype("int32").ravel()
  # Output Performance
  print(accuracy_score(testY, predY))
  print(classification_report(testY, predY))
  # Return Values
  return history
```

Tensorflow/Keras Version

Multilayer Perceptron Performance

Accuracy:

82.03% MLP(Tensorflow/Keras)

81.25% MLP(Scikit-learn)

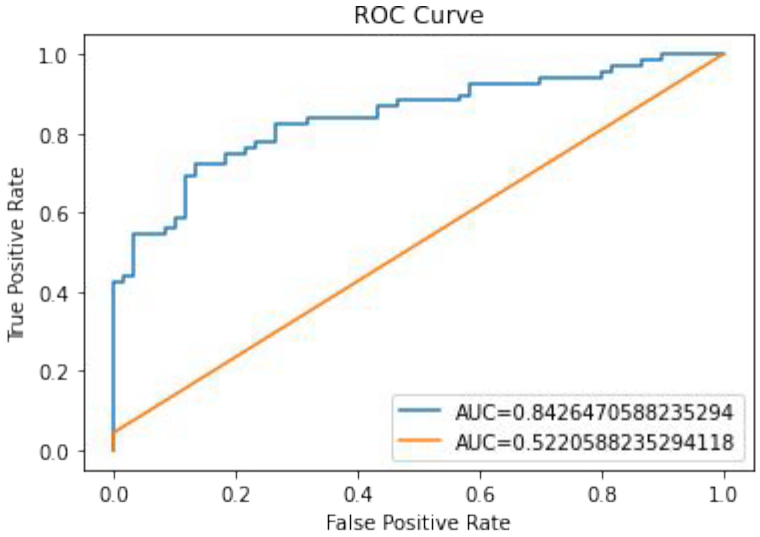
78.91% XGBoost

76.56% Random Forest

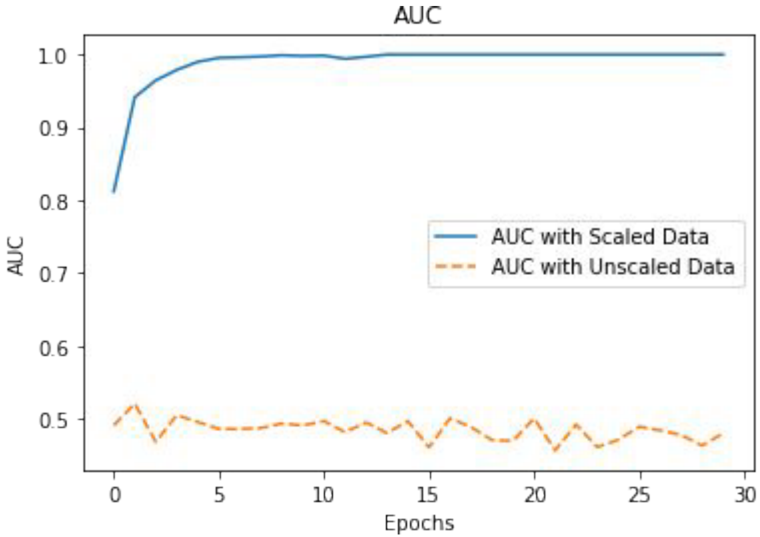
76.56% Logistic Regression

Multilayer Perceptron Performance

ROC Curve & AUC Score



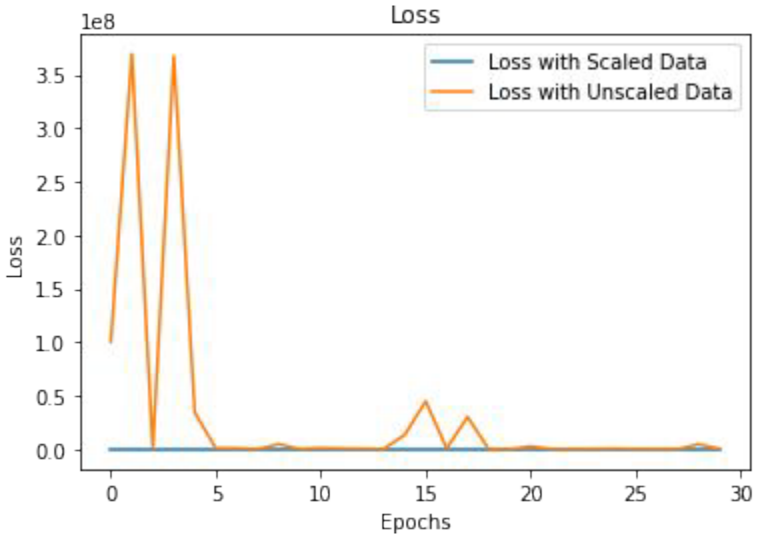
Scikit-Learn Version

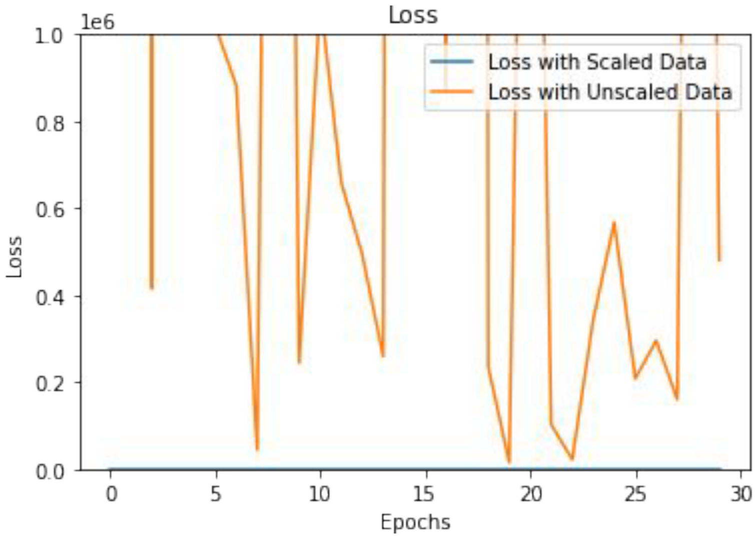


Tensorflow / Keras

Multilayer Perceptron Performance (Keras)

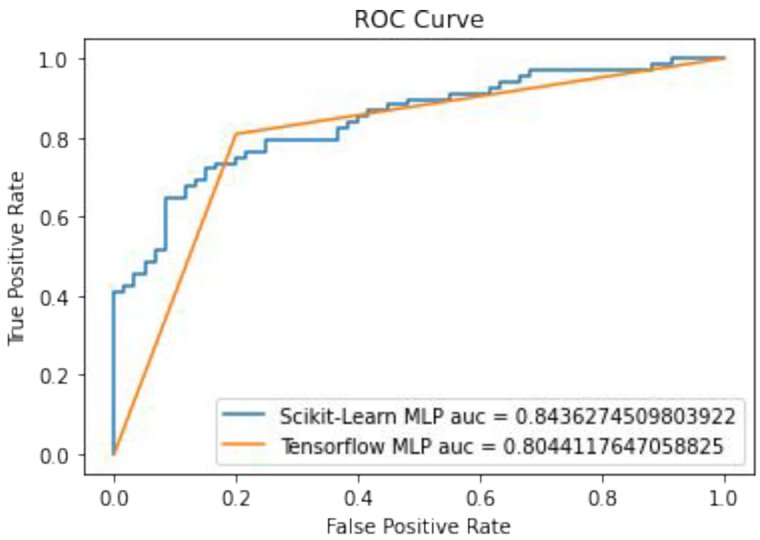
Loss Curve





Multilayer Perceptron Performance (Both)

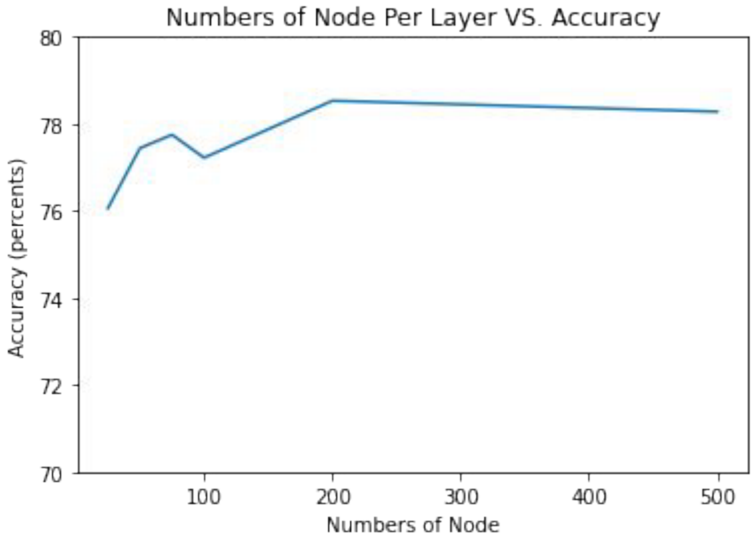
ROC Curve & AUC Score

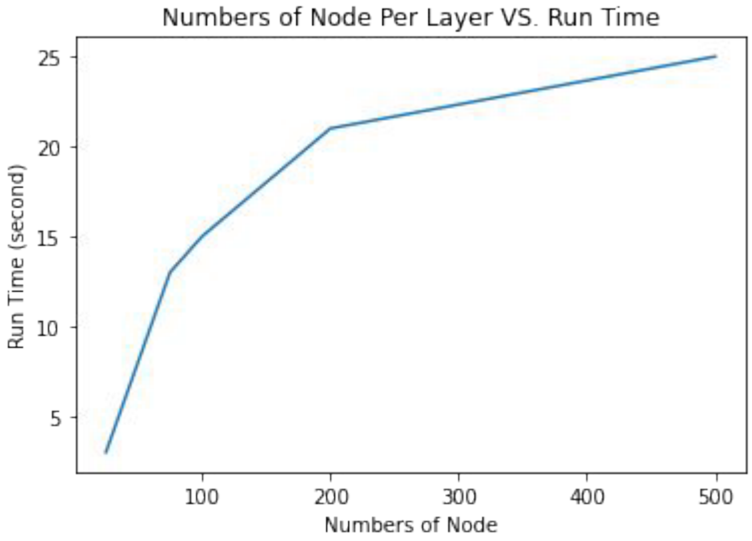


AUC1 Scikit-learn: 0.844
AUC2 Keras: 0.804

AUC1 is better than AUC2 by 0.05%

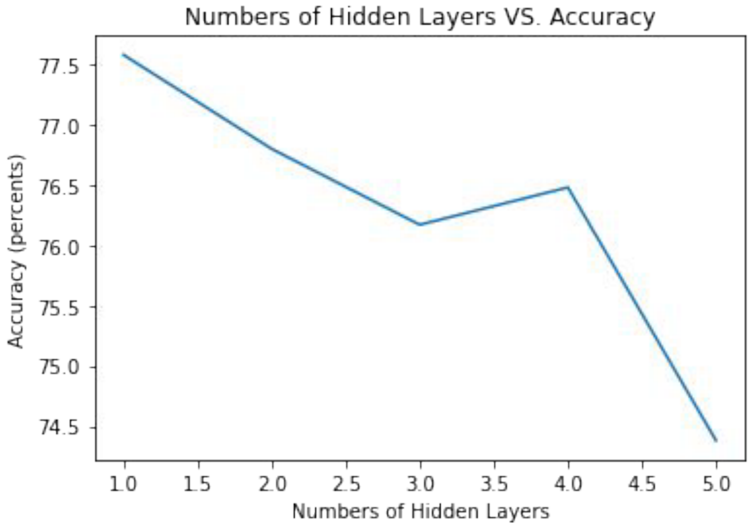
MLP Parameter Observation(Scikit-learn)

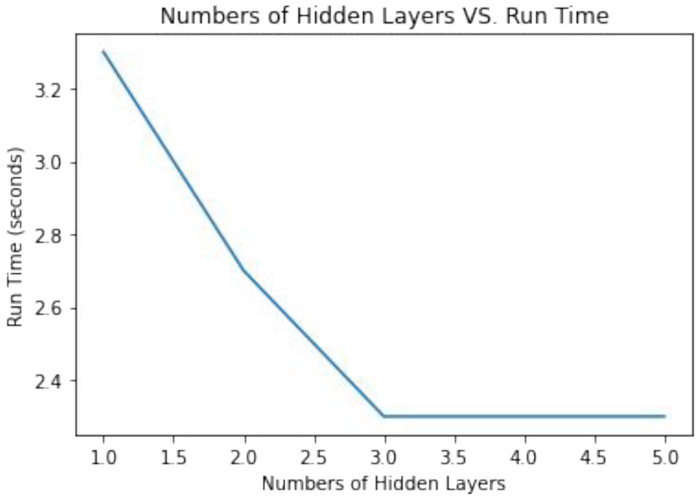




- Control Variable: Hidden Layer = 1
- Accuracy reaches maximum when neurons = 200
- More neurons added required more runtime

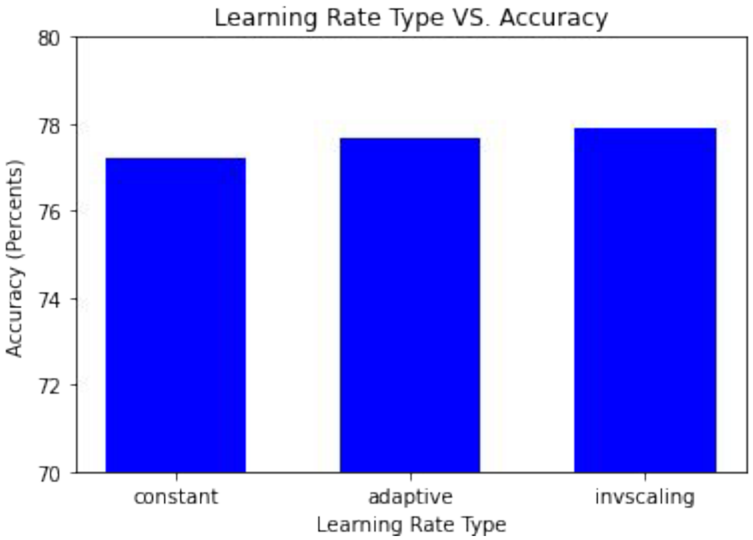
MLP Parameter Observation(Scikit-learn)





- Control Variable: Nodes per layers = 50
- Surprisingly, the more hidden layers added , accuracy didn’t improve worsen a bit and required slightly less runtime

MLP Parameter Observation(Scikit-learn)



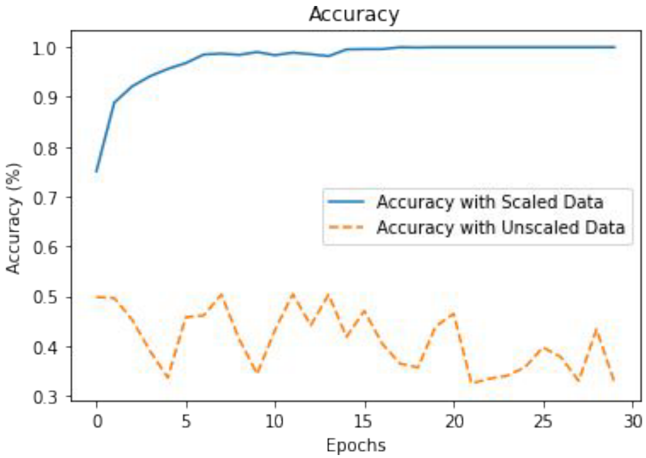
- Types of learning rate didn’t have a large effect on the accuracy, but the invscaling did slightly better

Hyperparameter Tuning

Based on the observation made on batch size, hidden layers and neurons, etc. We set:

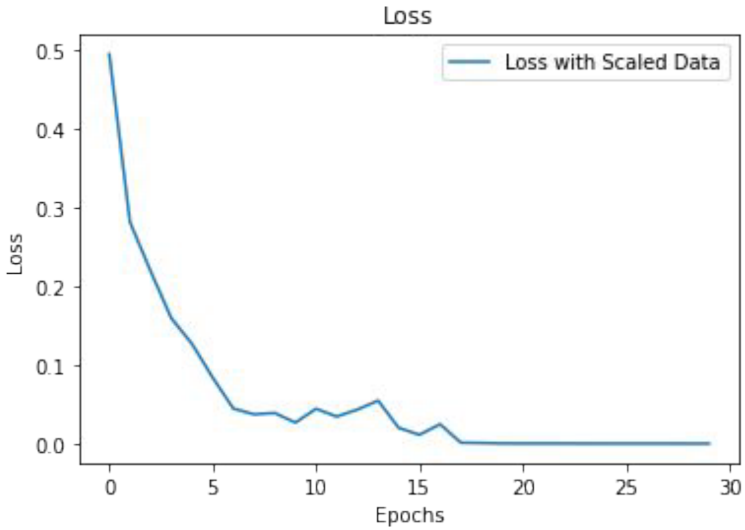
- Hidden Layer = 1
- Neuron Per Layer = 200
- Batch Size = Auto (Range from 200 ~ Sample Size)
- Learning Rate = Inverse Scaling / “invscaling”
- Activation = RELU
- Solver/Minimization Method = Adam Function

MLP Parameter Observation(Tensorflow/Keras)

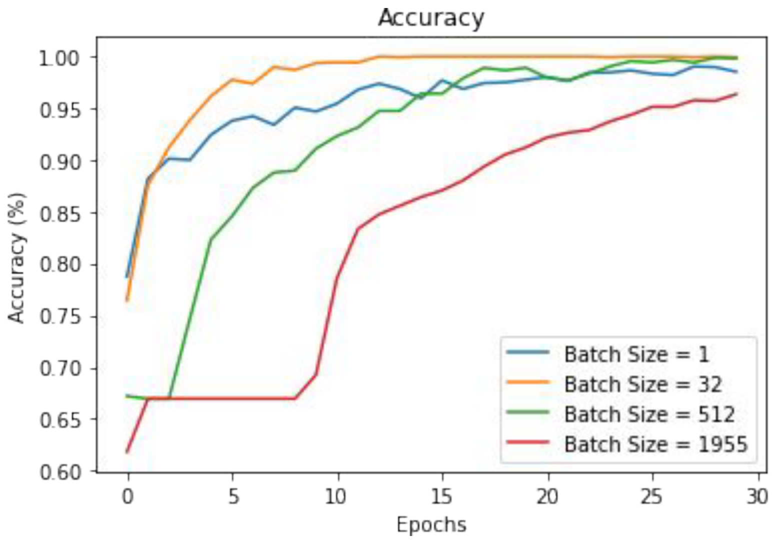


← As epoch increases, accuracy increases until epoch = 20

As epoch increases, loss decreases until epoch = 20 →



MLP Parameter Observation(Tensorflow/Keras)



- Smaller batch size leads to faster learning rate
- Smaller batch size tends to cause more noise and tend to be less stable in the later iterations
- Batch Size = 32 seems has very good learning rate while it’s quite stable. (Standard Batch Size)

In Conclusion

- Studied and analyzed the performance of many different machine learning methods
- Gradient Boosted Decision Trees on the original 208 features provided some of our best results
- Needed more time to explore 3D-CNNs and 3D voxelization
- Utilized visualization as a way for hyper-parameter tuning of Multilayer Perceptron

Acknowledgements

- Many thanks to:
  - Brian Gallagher
  - Professor Sindi
  - Hyojin Kim
  - Jen Bellig
  - And all the LLNL mentors that came to Merced and or gave a talk
  - Everyone who supported/helped out

Lawrence Livermore National Laboratory