Brian Dinh, Brian Wei
UCINetIDs: dinhb1, bjwei
IDs: 34546266, 23474935
Kaggle team(s): *bdbd*, *Brians*

CS 178 Final Project

Note: There are *two* Kaggle teams because we did not officially form teams on Kaggle before the deadline.

AUC Scores

Model	Training	Validation	Leaderboard Score (public)	Leaderboard Score (private)
Random Forest	0.92326	0.92546	0.76024	0.75637
Gradient Boost	0.88299	0.74074	0.74875	0.74699
Neural Network	0.87024	0.86935	0.73860	0.73750
Combined	N/A	N/A	0.77084	0.76947

Random Forest

We left this model exactly how it was in Homework 4, Problem 3. The random forest ensemble contained 26 trees. Each tree's parameters were as follows:

- `nBag=26`
- `maxDepth=20`
- `minLeaf=4`
- `nFeatures=np.sqrt(d)`, where d = number of features in the training data

```
for i in range(nBag):
    Xi,Yi = ml.bootstrapData(X,Y,M)
    ensemble[i] = ml.dtree.treeClassify(
           Xi, Yi, maxDepth=20, minLeaf=4, nFeatures=np.sqrt(d))
return ensemble
```

Since bagging was used to control overfitting, the parameters themselves used little complexity control. We chose sqrt(d) for our number of features to select from, because it yielded a more favorable score on Kaggle.

Gradient Boost - https://piazza.com/class/ixmcd5i1cey4?cid=471

Our second learner was a gradient boost model on decision trees. The decision tree learner was from the mltools library. We wanted the basic tree learner to make relatively okay decisions on its own, but still be simple enough for gradient boosting. Thus, we limited the idea of a basic tree to max depth 10 and below. Other parameters were kept default. Through testing we found that a tree with max depth 5 performed the best. Higher depths did not improve

Brian Dinh, Brian Wei
UCINetIDs: dinhb1, bjwei
IDs: 34546266, 23474935
Kaggle team(s): *bdbd*, *Brians*

validation data and they grossly overfit to the training data. Once we had the depth figured out, we ran the gradient boost for 300 iterations. We found that after 300 iterations, the auc score grows very slowly. It likely converges to a single number, so we terminated the iteration there. Some data as shown:

```
Depth 1 – Training: 0.68399; Validation: 0.67798
Depth 2 – Training: 0.74062; Validation: 0.70926
Depth 5 – Training: 0.88299; Validation: 0.74699
Depth 7 – Training: 0.96432; Validation: 0.73903
Depth 10 – Training: 0.99559; Validation: 0.72061
```

Neural Network - http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

We used SciKit Learn's Neural Network models (supervised). We opted to use the Multi-layer Perceptron classifier (MLPClassifier):

```
clf = MLPClassifier(
      hidden_layer_sizes=(200,200,200),
      activation='tanh', solver='adam',
      max_iter=500,shuffle=True,
     tol=1e-6, early_stopping=False)
```

First, we scaled the training data using SciKit Learn's Standard Scaler. We trained the model using 3 hidden layers, with 200 nodes per layer. This combination allowed us to fit the data in a reasonable amount of time. We opted to use 'adam', a stochastic gradient-based optimizer, with 'tanh' as the activation function for the hidden layers. The tanh function minimized bias in the gradients and provided a larger gradient magnitude which helped it learn faster than the other functions. With our large training dataset, backpropagation with tanh greatly accelerated the rate at which stochastic gradient descent converged. We lowered the tolerance for the optimization to 0.000001 so the model would continue to run, as we noticed a slow but downward trend in loss. We also decided not to use early stopping, letting our model continue to learn the data without having to increase validation score every iteration.

Overall Ensemble

For our overall ensemble, we decided to do a straight average on the confidence levels (for class 1) of our three learners' predictions. Averaging reduces the chance of overfitting because the errors in each learner get averaged out. There are no large spikes of irregularity.