Homework #3

Chaonan Shi (1901412)
Casey Bennett, PhD
DSC540, Winter 2019
DePaul University

Pima Diabetes

**Question #1a: Run the code once, record the accuracy and AUC score. What do you notice about the scores?*

Performance	*ACC*	*AUC*	*CV Runtime*
Gradient Boosting	*0.76*	*0.82*	*0.53*
Ada Boosting	*0.76*	*0.83*	*1.10*

Analysis:
By running the code once, I found that the score of Accuracy and AUC are same for both Gradient Boosting and Ada Boosting. However, the CV runtime of Ada Boosting is significant higher than the Gradient Boosting;

From my point of view, since the algorithm of Ada Boosting works by weighting the observations, putting more weight on difficult to classify instance and less on those already handled well. On the other hand, Gradient Boosting which build on the Ada Boosting and learn the loss function from error, also a benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used. Thus the running time of Gradient Boosting be shorter than the Ada Boosting.

**Question #1b: In the Scikit API for Ada Boost Classifier, it tells us that when the base_estimator parameter is set to None, it uses a particular estimator by default. What is this default estimator, and why is it significant?*

Analysis:
In the description, the default is None, which be assumed as `DecisionTreeClassifier`. In this case, weak learners are decision stumps and different weak learners can be specified through the base_estimator parameter. So that this is one of way to obtain a good result.

**Question #2a: Run the code once, record the accuracy and AUC score. What do you notice about the scores? How do they compare to boosting methods? What about run times?*

Performance	*ACC*	*AUC*	*CV Runtime*
MLPClassifier	*0.70*	*0.72*	*0.68*

Analysis:
For MLPClassifier, the result of both ACC and AUC scores are lower than the Gradient Boosting and Ada Boosting. For the running time, it is lower than Ada Boosting but higher than Gradient Boosting.

From my point of view, the running time should depending on the number of layers. I believe if we increase the number of layers, then the accuracy and running time should increase as well.

**Question #2b: In the Scikit API for MLP Classifier, there are different solvers described. When might we use the 'adam' solver?*

Analysis:
From MLPClassifier description, the solver of 'adam' works pretty well on relatively large dataset (with thousands of training samples or even more) in terms of both training time and validation score. Moreover, the 'adam' also refers to a stochastic gradient-based optimizer.

From my point of view, when dataset is relatively large, the more data we collected, the distribution is more close to normal. So that the sample of training and testing is more accuracy. However, the downside of this is that the running time could be also increase significantly.

**Question #3: Run the code once for each setting of the max depth (3,5,7,10), record the accuracy and AUC scores. What do you notice about the scores as the max depth increases? What about run-times?*

Times/ Max Depth	*1(3)*	*2(5)*	*3(7)*	*4(10)*
Accuracy	*0.76*	*0.77*	*0.77*	*0.74*
AUC	*0.82*	*0.83*	*0.81*	*0.80*
Running Time	*0.53*	*1.20*	*2.26*	*3.55*

Analysis:
Throughout this experiment, the results illustrate that there are certain threshold for number of depth of gradient Boosting, increasing number of depth may not improve accuracy and roc as we expect. On the top of that, since number of depth increased, running time also increased significantly from 0.53 to 3.55, but accuract and ROC are remaining at same level.

From my point of view, it is important that determine what the best number of depth. Not only for increasing precision, but also decreasing the cost of running time.

**Question #4a: Run the code once, record the accuracy and AUC scores. What do you notice about the scores? How do they compare to the performance above for Gradient Boosting, Ada Boosting, and Neural Networks with no feature selection? Did you notice any changes in run-times?*

Performance	*ACC*	*AUC*	*CV Runtime*
MLPClassifier	*0.77*	*0.83*	*0.41*

Analysis:

Throughout this experiment, the results illustrate that after feature selection got involved, all of scores increased, which meaning we got better performance comparing with no feature selection.

By comparing the performance above for Gradient Boosting, Ada Boosting, and Neural Networks with no feature selection, the feature selection definitely improve the result performance significantly. Moreover, since number of features reduced during the feature selection, so that that running time also reduced significantly.

**Question #4b: What features were selected, and which were removed? Were there any differences from when you did feature selection with Random Forests in HW2?*

(From Random Forests in HW2)

Total Features (9)	*Class*	*Blood Glucose*	*BMI*	*'Family History'*	*'Age'*	*Times Pregnant*	*Blood Pressure*	*Skin Fold Thickness*	*2-Hour Insulin*
Selected (4)		√	√	√	√				

Performance	*ACC*	*AUC*	*CV Runtime*
Scores	*0.76*	*0.82*	*1.069*

(From Gradient Boosting in HW3)

Total Features (8)	*Class*	*Blood Glucose*	*BMI*	*'Family History'*	*'Age'*	*Times Pregnant*	*Blood Pressure*	*Skin Fold Thickness*	*2-Hour Insulin*
Selected (3)		√	√		√				

Performance	*ACC*	*AUC*	*CV Runtime*
Scores	*0.77*	*0.83*	*1.069*

Analysis:
Throughout this experiment, the results illustrate that after feature selection got involved, the features: Blood Glucose, BMI, and Age are left. By comparing with result with HW2, the only difference between these two results of feature selection are that 'Family History' been selected at HW2 by RandomForest but Gradient Boosting.

From performance perspective, both of algorithms perform at same level, but Gradient Boosting get one fewer feature than RandomForest. So in this case, the Gradient Boosting perform better than the RandomForest, since less features but same performance.

**Question #5: Run the code once for each setting of the solver, record the accuracy and AUC scores. What do you notice about the scores when we change the solver? What about run-times?*

Performance	*ACC*	*AUC*	*CV Runtime*
MLPClassifier	*0.65*	*0.70*	*1.74*

Analysis:
Throughout this experiment, the results illustrate that the performance of MLPClassifier dropped significantly by setting the solver from 'lbgfs' to 'adam'.

From my point of view, since the 'adam' perform better at relatively large dataset, I assume that setting of 'adam' is sensitive to the data size. In this case, since our data size is not relatively large, so that performance would worse at case of 'adam'. For improve performance of MLPClassifier, I believe increasing dataset size would help better performance for MLPClassifier.

Wine Quality Dataset

**Question #6a: Run the code once, record the RMSE and Explained Variance.*

Algorithm /Performance	*RMSE*	Explained_Variance_Score	*CV Runtime*
GradientBoosting	*0.65*	*0.34*	*0.44*
AdaBoost	*0.66*	*0.31*	*1.39*

Analysis:
Throughout this experiment, the results illustrate that the performance of both GradientBoosting and AdaBoost are stay at same level, but running time of AdaBoost is much more higher than the GradientBoosting, 0.44s compare to 1.39 respectively.

**Question #6b: In the Scikit API for Gradient Boost Regressor, what do you think is the purpose of the learning rate parameter (hint: do some googling)?*

Analysis:

The description of scikit-learn put it on this way: proposed a simple regularization strategy that scales the contribution of each weak learner by a factor ν:

$$Fm(x)=Fm-1(x)+\nu\gamma mhm(x)$$

The parameter ν is also called the **learning rate** because it scales the step length the gradient descent procedure; it can be set via the learning_rate parameter.

After throughout reading material from various paper, I found that the learning rate can be concluded as: Stochastic gradient descent is an optimization algorithm that estimates the error gradient for the current state of the model using examples from the training dataset, then updates the weights of the model using the back-propagation of errors algorithm, referred to as simply backpropagation.
The amount that the weights are updated during training is referred to as the step size or the "learning rate."

("Understand the Impact of Learning Rate on Neural Network Performance", by Jason Brownlee on January 25, 2019)

From my understanding, we can connect with other three terms: error, regularization, and weight decay. Since overfitting problem, so we need to make our data more stationary then we need to apply regularization to penalize the error term. However, only regularization may not help boot our model's performance, in this case we may apply the boosting trying to re-fit the model from error also given weight to them. On the top of that, since learning rate also be referred to step size which is the amount that the weights are updated during training, from this, we can conclude that:

$$\omega_i \leftarrow \omega_i - \eta\frac{\partial E}{\partial \omega_i} \quad \omega_i \leftarrow m \cdot \omega_i - \eta\frac{\partial E}{\partial \omega_i}$$

In this case, the higher learning rate more faster of converge but less accuracy, and vice versa.

**Question #7a: Run the code once, record the RMSE and Explained Variance. What do you notice about the scores? How do they compare to boosting methods? What about run times?*

RMSE	Explained_Variance_Score	Running Time
0.66	0.31	3.20

Analysis:
Throughout this experiment, the results illustrate that the performance of Neural Network is not significantly better than the boosting methods. The score of RMSE, Explained_Variance_Score are 0.66, 0.31 respectively, and running time is 3.20 which is significantly higher than the boosting methods.

**Question #7b: In the Scikit API for MLP Regressor, if you wanted to create a neural network to have two hidden layers of 10 and 10, instead of just a single hidden layer of 20, how would you set the hidden_layers parameter equal to in the function call?*

Analysis:
First, from the description of sklearn.neural_network.MLPClassifier of hidden layers is that: 'The ith element represents the number of neurons in the ith hidden layer'
In this case, if we want to create two hidden layers of 10 and 10, the idea is:

MLPRegressor(activation= 'logistic',solver='lbfgs', alpha=0.0001,
max_iter=1000, hidden_layer_sizes=(10:10,),random_state=rand_st)

means : hidden_layer_sizes is a tuple of size (n_layers -2)

n_layers means no of layers we want as per architecture.

Value 2 is subtracted from n_layers because two layers (input & output) are not part of hidden layers, so not belong to the count.

default(100,) means if no value is provided for hidden_layer_sizes then default architecture will have one input layer, one hidden layer with 100 units and one output layer.

Whit examples:

Example :

1. For architecture 56:25:11:7:5:3:1 with input 56 and 1 output hidden layers will be (25:11:7:5:3). So tuple hidden_layer_sizes = (25,11,7,5,3,)
2. For architecture 3:45:2:11:2 with input 3 and 2 output hidden layers will be (45:2:11). So tuple hidden_layer_sizes = (45,2,11,)

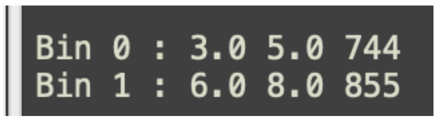
**Question #8a: Run the code once, record the accuracy and AUC score. What do you notice about the scores? How do they compare to the regression scores (or can you compare them)?*

Performance	*ACC*	*AUC*	*CV Runtime*
Gradient Boosting	*0.73*	*0.81*	*0.76*
Ada Boosting	*0.74*	*0.82*	*1.43*
MLPClassifier	*0.72*	*0.81*	*3.77*

Analysis:
From my point of view, we cannot just simply compare between regression and classification. Since both of methods given us different outcome also different measurements. The data type of target is different, for example, the classification's target is categorical, but for regression it is continuous. Moreover, if we discrete data type from continuous to categorical, then our answer would also change. Regression approach given us score number which focus on the explained variance, whereas, the classification given us score number which focus on the confusion matrix.

On the other hand, if we discrete target from continuous to categorical, the answer also become ambiguous. For example, if we set 'class' over 5 as 'high', below as 'low', then the outcome of classification only gives us result about which level that unknow record belongs to. In this case, even the classification approach seems like more optimal, but I do not think we can make comparison between these two approaches.

**Question #8b: Look at the bins that were created (some info should be printed out about the # of samples in each bin, and min and max values). How would you explain what you did to your boss or customer? What are we actually predicting here?*


```
Bin 0 : 3.0 5.0 744
Bin 1 : 6.0 8.0 855
```

Analysis:
From my point of view, this approach discrete the continuous records to categorical. In this case, with two levels. And the number of bin illustrate that number of levels, and first number represent min of that number of one level, and second number represent the max of that number of another level. The number followed by max is that the number of record which belong to that level.
In this case, we could summarize that the first level been discrete as from 3 to 5; and the second level been discrete as from 6 to 8.
Instead of regression, this approach turn out to be a classification problem and try to prediction the record from class which classes belong to.

Question #9: Run the code once, record the accuracy and AUC score. What do you notice about the scores? How do they compare to results in Question #8a?

Performance	*ACC*	*AUC*	*CV Runtime*
Gradient Boosting	*0.73*	*0.81*	*0.77*
Ada Boosting	*0.74*	*0.82*	*1.38*
MLPClassifier	*0.72*	*0.81*	*4.10*

Analysis:
Throughout this experiment, the results illustrate that the performance of these three classifier at this question is not significant different from previous one. From my understanding, it might because of the distribution of original dataset is close to normal distribution. So that the regularization would not help to improve the performance in this case.

**Question #10a: Run the code once for both settings of target discretization (binning either 0 or 1). Record the accuracy and AUC scores for binned data, and the RMSE and Explained Variance Scores for un-binned data. What do you notice about the scores? How do they compare to performance above for Gradient Boosting, Ada Boosting, and Neural Networks with no feature selection? Did you notice any changes in run-times?*

Fifure1 with binning:

Performance	ACC	AUC	CV Runtime
Gradient Boosting	0.73	0.81	0.54
Ada Boosting	0.73	0.82	1.13
MLPClassifier	0.74	0.81	2.32

Fifure2 without binning:

Algorithm /Performance	RMSE	Explained_Variance_Score	CV Runtime
GradientBoosting	0.66	0.29	0.22
AdaBoost	0.67	0.30	0.47
MLPClassifier	0.64	0.33	2.89

Analysis:
Throughout this experiment, comparing with previous results with no feature selection. Both classification and regression scores stay at same level. However, since we reduced the number of features so that running time decreased as well.

In the summary, although number if features been reduced, performance of result would not fluctuate significantly. Thus, we should re-construct our model after feature selection, not only running time, but also for simplicity.

Question #10b: What features were selected, and which were removed? How do those features differ between binned vs. un-binned runs?

Feature selection with binning:

Total Features (10)	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
Selected (3)		√					√			√	√

Feature selection without binning:

Total Features (10)	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
Selected (3)		√								√	√

Analysis:
Throughout this experiment, comparing with two results, the *Feature selection with binning* features: volatile acidity, total sulfur dioxide, sulphates, and alcohol been selected; Meanwhile, features: fixed acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, density, and pH been removed from original columns;

For *Feature selection without binning* features: volatile acidity, sulphates, and alcohol been selected; Meanwhile, features: fixed acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, and pH been removed from original columns;

In this case, the feature of 'total sulfur dioxide' is only one difference between binned and un-binned.

**Question #11: Compare the performance of Boosting Methods and Neural Networks here compare to previous methods (decision trees, random forests) from prior Homeworks for both datasets. Did they perform better, worse, or the same in terms of both evaluation scores and run-times? If your boss or customer asked why that might be, how would you explain?*

Analysis:
Throughout this experiment, comparing to previous methods (decision trees, random forests) from prior homeworks for both datasets. The performance does not improve too much and we can summarize that all algorithm stay at same of both evaluation scores, but run-times might differ from one to others.

From my point of view, I believe the reasons which behind this result are: 1.simple dataset; 2. Dataset size is relatively small; 3.algorithm efficiency.

First of all, since the dataset is simple, for example, the most of features are continuous, so we lack of variety in this case, then all the algorithms perform at same level of scores.

Second, the performance may heavily depending on the data size, in general, more data we collect, the higher accuracy we close to. For instance, in term of neural network, the default setting of solver is 'adam' from sklearn package. However, the accuracy of model by using this setting may heavily dependent on the size of data. Thus if we get relatively large dataset, then performance from neural network may improve significantly.

Third, the running time of algorithm differ from one to others since they are all work at different ways. In term of neural network, we do have three layers: input layers, hidden layers, and output layers. Each layer will spend certain time to process, especially for the hidden layers, we have no idea how many layers will be generated and the number of hidden units per layer and not even the kind of connections between layers. In this case, the running time of neural network would be uncertainty and unpredictable.

**Question #12: Can we say anything interesting about diabetes based on the features that were selected, if we were for instance trying to create a diabetes screening program for a local healthcare organization?*

Analysis:
Throughout this experiment, comparing with result of RandomForest at HW2, the Gradient Boosting perform better results in this case, since less features been selected but perform at same level of accuracy and AUC.

In this case, the features of : Blood Glucose, BMI, and AGE been selected. The first two features more relate to the weight, for instance, the calculation of BMI is: weight in kilograms divided by height in meters squared. Then we can conclude that heavy people generally have higher BMI index. On the other hand, the Blood Glucose also highly correlate with the total consumption of sugar. Moreover, more sugar people consume, the more weight they get. Be precisely, we can observe the potential diabetes depending on people's weight, on the top of that, we can also track people's consumption of sugar to determine the probability of potential diabetes.

The last feature we selected during the feature selection is AGE, basically there is nothing we can do with aging. From healthcare point of view, they may need to keep track of diabetes by observing certain group of people.

From insurance point of view, construct a portfolio or healthcare product which charge more money for the overweight people, also make healthcare package for people who be classified as old group.