**Homework #3 Instructions**

**Casey Bennett, PhD**

**DSC540, Winter 2019**

**DePaul University**

**Overview**

Same two datasets as previous homeworks (Diabetes and Wine Quality), along with the two Python scripts. We will explore ways of using Boosting Methods and Neural Networks to create models, as well as talk about the effects/issues of converting a regression problem into a classification problem via target discretization.

For classification, we will be creating an object we’ll name ‘clf’, and for regression we’ll name ‘rgr’. These are objects we can call methods on (such as fitting a model to some data), and access their internal variables (such as getting predicted class labels). Scikit API links are in the accompanying document. We will only use cross-validation in this homework.

*\*Follow the steps below, record answers to questions in a word document, and turn in both your completed code and the word doc.*

**Pima Diabetes**

Open up HW3\_Diabetes.py

1. First, let’s run a Gradient Boosting Model, then an Ada Boosting Model, and compare.
   1. First we need to import the functions, on line 11, replace the comment with a call to import GradientBoostingClassifier() and AdaBoostClassifier() from the sklearn “ensemble” package.
   2. On line 278, create a GradientBoostingClassifier(). Using the API link in the accompanying document, call that function, and pass in the following parameters:
      1. Set number of estimators to 100
      2. Set loss = ‘deviance’
      3. Set the learning rate = 0.1
      4. Set maximum depth =3
      5. Set minimum # of samples for split to occur = 3
      6. Set random\_state variable to rand\_st
   3. Add in a cross\_validate function on line 279 (use previous homework as an example) with 5 folds, and pass in the clf object.
   4. You may want to edit the print statements, so they say “Gradient Boosting” when printing scores, to make the output easier to see.
   5. Repeat B, C, and D above for Ada Boost. Copy the block of code between lines 277-285. Paste it down *under* the section header “#SciKit Ada Boosting - Cross Val” on line 288.
   6. Change the clf to AdaBoostClassifier() and pass in the following parameters:
      1. Set number of estimators to 100
      2. Set base\_estimator = None
      3. Set the learning rate = 0.1
      4. Set random\_state variable to rand\_st

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

*﻿Gradient Boosting Acc: 0.76 (+/- 0.07)*

*Gradient Boosting AUC: 0.82 (+/- 0.06)*

*CV Runtime: 0.6368677616119385*

*﻿Ada Boosting Acc: 0.76 (+/- 0.05)*

*Ada Boosting AUC: 0.83 (+/- 0.06)*

*CV Runtime: 1.2834949493408203*

*Both Gradient Boosting and Ada Boosting have good score at a very close variation with Ada boost have a very slightly high on AUC score. Both method had the same Acc score at 0.76 with a little bit of different in variation. I notice the CV runtimes on Ada boosting is twice higher than Gradient 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?*

The default estimator for Ada boosting classifier is ‘None’. If the estimator default is none, it means that the base estimator is Decision Tree classifier with max depth equal to 1 which it will try to improve for the misclassification point. Also, you can set the base\_estimator equal to other machine learning algorithm such as Logistic Regression.

1. Now let’s try a Neural Network and compare.
   1. First we need to import the function, on line 12, replace the comment with a call to import MLPClassifier() from the sklearn “neural\_network” package.
   2. Repeat what you did for Ada Boost in Question #1. Copy the block of code for Gradient Boosting (somewhere around line 277, though may have shifted down now that you’ve added lines to the code), paste down under the section header “#SciKit Neural Network - Cross Val”
   3. Change the clf to MLPClassifier() and using theAPI link in the accompanying document, pass in the following parameters:
      1. Set activation = 'logistic'
      2. Set solver = 'lbfgs'
      3. Set alpha = 0.0001
      4. Set the max # of iterations = 1000
      5. Set the hidden layers sizes = (10,)
      6. Set random\_state variable to rand\_st

*\*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?*

*Neural Network Acc: 0.71 (+/- 0.07)*

*Neural Network AUC: 0.74 (+/- 0.06)*

*﻿ CV Runtime: 2.679342031478882*

*It is obvious that boosting method generate better score than Neural Network. The variation between Acc is still in range but the AUC is shown significant drop from 0.83 to 0.74. Moreover, CV run times in Neural Network is higher at least 1 second. So, It could be said that Neural Network not suitable for this dataset.*

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

*For the parameter solver, Adam (Adaptive Moment Estimation) is method that computes adaptive learning rates for each parameter. Because Adaptive algorithms converge very fast and quick to find the right direction in which parameter update should occur. It works well with large dataset.*

1. Let’s explore how the depth of each tree affects performance in Gradient Boosting.
   1. Change the max\_depth parameter of the GradientBoostingClassifier() from 3 to 5
   2. Now set the max\_depth to 7
   3. Now set the max\_depth to 10

*\*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?*

*Max depth = 3*

﻿Gradient Boosting Acc: 0.76 (+/- 0.07)

Gradient Boosting AUC: 0.82 (+/- 0.06)

CV Runtime: 0.7548990249633789

Max depth = 5

﻿Gradient Boosting Acc: 0.77 (+/- 0.05)

Gradient Boosting AUC: 0.83 (+/- 0.07)

CV Runtime: 1.1560428142547607

Max depth = 7

﻿Gradient Boosting Acc: 0.77 (+/- 0.08)

Gradient Boosting AUC: 0.81 (+/- 0.08)

CV Runtime: 2.2064239978790283

Max depth = 10

﻿Gradient Boosting Acc: 0.74 (+/- 0.06)

Gradient Boosting AUC: 0.78 (+/- 0.07)

CV Runtime: 3.7418272495269775

For the variety of max depths in this observation, the scores seem fluctuating with differences number of max depths. ACC and AUC increased from max depth equal 3 to 5 but it drops from max depth equal to 7 and 10. Overall, It can be explained that Acc and AUC reach its peak at some specific max depth in this case is max depth equal 5 , then the scores drop when we increase the max depth after that. Moreover, the runtimes increased due to the number of depth increase.

1. Finally, let’s run feature selection again on the Diabetes dataset, but this time do it using Gradient Boosting. Just like Random Forests, Gradient Boosting is a tree-based method, so we can use it to calculate a measure of “feature importance” natively.
   1. First, on line where we call the GradientBoostingClassifier(), change the max\_depth back to 3 from where we changed it above in Question #3
   2. To turn on feature selection, we need to first on line 38 change the feat\_select flag to equal 1 instead of 0
   3. Note that there is an option to change the feature selection type is already set to 2 (wrapper-based) on line 39
   4. You will need to add a GradientBoostingClassifier(), call to pass to the clf object on line 191, you can use something similar to the calls used elsewhere in the code. Don’t forget to set the parameters, particularly the random\_state and number of estimators to 100.
   5. Note there are two sub-sections under wrapper select feature selection, one for datasets with a binned target (classification) and another for datasets with a continuous target (regression). We are doing classification with diabetes here, so we using the former section (if binning=1).
   6. Note the SelectFromModel() function being called on line 192, this is where the actual feature selection occurs, with the clf object being passed in

*\*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?*

*﻿Gradient Boosting Acc: 0.77 (+/- 0.05)*

*Gradient Boosting AUC: 0.83 (+/- 0.07)*

*CV Runtime: 0.5418369770050049*

*The Acc and AUC with applying feature selection seem strong at .77 and 0.83 respectively which is slightly higher than scores without feature selection. Although, Gradient Boost and Ada Boost have quite the same score with or without feature selection it gives a better score than neural network. We can conclude that we can build model with less feature by doing feature selection to get a better performance and make a simply model. For the run-times, the number in q4a is obviously better than all of the previous question.*

*\*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?*

﻿Wrapper Select:

Selected ['Blood Glucose', 'BMI', 'Age']

Features (total/selected): 8 3

Not Selected Features

﻿['Class', 'Times Pregnant', 'Blood Pressure', 'Skin Fold Thickness', '2-Hour Insulin', 'Family History']

The difference is that Gradient Boosting selects three features but Random Forest selects four features. Three of the features from both algorithms are the same except ‘Family History’ is not selected in Gradient Boosting.

1. Let’s explore changing the solver method for Neural Networks (we first saw in saw in Question #2). Solvers are essentially how the Neural Network goes about searching for optimal weights between its “neurons”, so that choice plays a fundamental role in how our model learns.
   1. Change the solver parameter of the MLPClassifier() from lbgfs to ‘adam’

*\*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?*

(‘Adam’)

Neural Network Acc: 0.72 (+/- 0.07)

Neural Network AUC: 0.80 (+/- 0.08)

CV Runtime: 4.515022039413452

(‘lbfgs’)

﻿Neural Network Acc: 0.72 (+/- 0.12)

Neural Network AUC: 0.76 (+/- 0.18)

CV Runtime: 1.0173029899597168

To begin with the comparison between ‘adam’ and ‘lbfgs’ , the scores in Adam are better at 0.72 and 0.80 for Acc and AUC respectively with the variation at 0.07 and 0.08. The scores from ldfgs have the same Acc but lower AUC at 0.76. The main difference is that the variation from both parameters. It shows the almost twice of the variation from ldfgs at 0.12 and 0.18 which the scores range vary from 0.60 to 0.84 Acc and 0.58 to 0.94 AUC. With this very high in variation, the lbfgs may not be effective enough. Although the lbfgs score has high variation, the parameter generates 4 times lower in CV times.

**Wine Quality Dataset**

Open up HW3\_Wine.py … First, let’s repeat the steps we did above for Diabetes.

1. First, let’s run a Gradient Boosting Model, then an Ada Boosting Model, and compare.
   1. First we need to import the functions, on line 11, add calls for the GradientBoostingRegressor() and AdaBoostRegressor() from the sklearn “ensemble” package.
   2. On line 278, create a GradientBoostingRegressor(). Using the API link in the accompanying document, call that function, and pass in the following parameters:
      1. Set number of estimators to 100
      2. Set loss = ‘ls’
      3. Set the learning rate = 0.1
      4. Set maximum depth =3
      5. Set minimum # of samples for split to occur = 3
      6. Set random\_state variable to rand\_st
   3. Add in a cross\_validate function on line 279 (use previous homework as an example) with 5 folds, and pass in the clf object.
   4. You may want to edit the print statements, so they say “Gradient Boosting” when printing scores, to make the output easier to see.
   5. Repeat B, C, and D above for Ada Boost. Copy the block of code between lines 277-285. Paste it down *under* the section header “#SciKit Ada Boosting - Cross Val” on line 288.
   6. Change the clf to AdaBoostRegressor() and pass in the following parameters:
      1. Set number of estimators to 100
      2. Set base\_estimator = None
      3. Set loss = ‘linear’
      4. Set the learning rate = 0.5
      5. Set random\_state variable to rand\_st

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

*﻿ Gradient Boosting Regressor RMSE:: 0.64 (+/- 0.01)*

*Gradient Boosting Regressor Expl Var: 0.34 (+/- 0.12)*

*CV Runtime: 0.5388569831848145*

*Ada Boosting Regressor RMSE:: 0.66 (+/- 0.03)*

*Ada Boosting Regressor Expl Var: 0.31 (+/- 0.16)*

*CV Runtime: 1.9699971675872803*

*\*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)?*

*The main purpose of learning rate parameter is to reduce the overfit which cause by the nature of the gradient boost that create decision tree and added to the model quickly. The learning rate technique is used to slow down the learning in the gradient boosting by applied the weighting (shrinkage or learning rate) factor for the correction by new trees by added to the model.*

1. Now let’s try a Neural Network and compare.
   1. First we need to import the function, on line 12, replace the comment with a call to import MLPRegressor()from the sklearn “neural\_network” package.
   2. Repeat what you did for Ada Boost in Question #6. Copy the block of code for Gradient Boosting (somewhere around line 278, though may have shifted down now that you’ve added lines to the code), paste down under the section header “#SciKit Neural Network - Cross Val”
   3. Change the clf to MLPRegressor() and using the API link in the accompanying document, pass in the following parameters:
      1. Set activation = 'logistic'
      2. Set solver = 'lbfgs'
      3. Set alpha = 0.0001
      4. Set the max # of iterations = 1000
      5. Set the hidden layers sizes = (10,)
      6. Set random\_state variable to rand\_st

*\*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?*

*﻿Neural Network Regressor RMSE:: 0.66 (+/- 0.05)*

*Neural Network Regressor Expl Var: 0.30 (+/- 0.13)*

*CV Runtime: 3.982743978500366*

*The score of RMSE and Expl Var are both moderate at 0.66 RMSE and Expl var .3 with high variation at 0.13. The performance from Neural Network Regressor is not well compared to Boosting with a lower Expl Var and lower variation. The Cv-runtimes from neural network is also higher than boosting method at 4 second.*

*\*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?*

﻿hidden\_layer\_sizes = (10,10) ---- 2 hidden layers with 20 neurons

1. A fundamental question you will deal with as data scientists when it comes to regression problems, is whether you should try to predict the target “as-is” as a continuous variable, or discretize the target into bins and then treat it as a classification problem. Let’s try this here with the Wine Quality dataset and see what happens.
   1. We’re gonna do this using a function called KBinsDiscretizer, notice that it is already imported for you on line 19, and down in the Preprocessing section it is setup to run. You have to do a little maneuvering with numpy arrays to make sure everything stays in the proper shape, so I’ve done that for you.
   2. To turn on target discretization, we need to change the binning flag on line 36 to equal 1 instead of 0
   3. Note on line 37 below that, the bin\_cnt is already set to equal 2, so we will be creating 2 bins for the wine ratings
   4. Now we need to add classifier versions of Gradient Boosting, Ada Boosting, and Neural Networks underneath the Section labeled ####Cross-Val Classifiers#### … in the original code this was on line 296, but it’s probably shifted down since you added things. Easiest thing to do is just copy and paste your classifier code from the Diabetes Python script, should run as is.
      1. GradientBoostingClassifier()
      2. AdaBoostClassifier()
      3. MLPClassifier()
      4. Don’t forget to copy the scorers line!
   5. You will also need to add those functions to your import statements back up around Line 11, before you can call them. Note that you can import multiple functions from the same Scikit module on each line, by making a comma-separated list.

*\*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)?*

*Bin 0 : 3.0 5.0 744*

*Bin 1 : 6.0 8.0 855*

--ML Model Output--

Gradient Boosting Acc: 0.73 (+/- 0.05)

Gradient Boosting AUC: 0.81 (+/- 0.06)

CV Runtime: 0.9002690315246582

Ada Boosting Acc: 0.74 (+/- 0.05)

Ada Boosting AUC: 0.82 (+/- 0.06)

CV Runtime: 1.7767088413238525

Neural Network Acc: 0.72 (+/- 0.07)

Neural Network AUC: 0.81 (+/- 0.06)

CV Runtime: 4.6300342082977295

The score from all algorithms are moderate to high with all Acc scores are .73, .74, .72 and AUC is .81, .82, .81. The variation from all scores is relative is the same which It does not make any notable differences. The cv-runtime shows that Gradient boosting is the fastest following with Ada boosting and Neural Network.

As we treat this question with bin (discretizing) and performs it with classification algorithm. I think we cannot compare the result to regression since the expecting output and the purpose are different.

*\*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?*

*I would tell my boss or customer that what I’m doing is to try to predict sample dataset(training set) by converted distribution of this dataset to 2 categorical(binning) and try to label to see the accuracy. However, this approach may not be suitable for labeling data because all the continuous are put in 2 bins (0 and 1) by setting range. And, the classification may create overfitting.*

1. Another question might be if the target variable has a weird distribution, or say a bunch of outliers. That could affect your discretization of it. So let’s see what happens when we normalize the target variable *before* we discretize it.
   1. On line 34, change the norm\_target flag to equal 1 instead of 0

*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?*

*﻿*

*﻿Bin 0 : -3.265164632733176 -0.787822640922809 744*

*Bin 1 : 0.4508483549823745 2.9281903467927415 855*

*--ML Model Output--*

*Gradient Boosting Acc: 0.73 (+/- 0.05)*

*Gradient Boosting AUC: 0.81 (+/- 0.06)*

*CV Runtime: 0.8752157688140869*

*Ada Boosting Acc: 0.74 (+/- 0.05)*

*Ada Boosting AUC: 0.82 (+/- 0.06)*

*CV Runtime: 1.695634126663208*

*Neural Network Acc: 0.72 (+/- 0.07)*

*Neural Network AUC: 0.81 (+/- 0.06)*

*CV Runtime: 4.289659738540649*

1. Finally, let’s run feature selection again on the Wine dataset, just like we did for Diabetes in Question#5 above (using Gradient Boosting). This time though, we’ll do it for both the binned target and the un-binned target, and look at the effects.
   1. First turn target normalization off. On line 34, change the norm\_target flag to back equal to 0 as it was originally, instead of 1
   2. To turn on feature selection, we need to first on line 38 change the feat\_select flag to equal 1 instead of 0
   3. Note that there is an option to change the feature selection type is already set to 2 (wrapper-based) on line 39
   4. You will need to add a GradientBoostClassifier(), call to pass to the clf object on line 191, you can use something similar to the calls used elsewhere in the code. Don’t forget to set the parameters, particularly the random\_state and number of estimators to 100.
   5. Since we are going to compare feature selection for both binned and non-binned targets, we also need to add a GradientBoostRegressor() on line 195. So now you should have a version under both sub-sections (binning=1 and binning=0).
   6. Note the SelectFromModel() function being called on line 192/196, this is where the actual feature selection occurs, with the clf/rgr object being passed in
   7. To run the code with the target binned and unbinned, we will toggle the binning flag on line 36 to either 0 (unbinned) or 1 (binned)

*\*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?*

With Binning

Bin 0 : 3.0 5.0 744

Bin 1 : 6.0 8.0 855

﻿Gradient Boosting Acc: 0.73 (+/- 0.05)

Gradient Boosting AUC: 0.81 (+/- 0.05)

CV Runtime: 0.5579788684844971

Ada Boosting Acc: 0.73 (+/- 0.06)

Ada Boosting AUC: 0.82 (+/- 0.06)

CV Runtime: 1.3486478328704834

Neural Network Acc: 0.75 (+/- 0.06)

Neural Network AUC: 0.81 (+/- 0.07)

CV Runtime: 2.3697099685668945

Without Binning

﻿Gradient Boosting Regressor RMSE:: 0.66 (+/- 0.02)

Gradient Boosting Regressor Expl Var: 0.29 (+/- 0.14)

CV Runtime: 0.23413300514221191

Ada Boosting Regressor RMSE:: 0.67 (+/- 0.04)

Ada Boosting Regressor Expl Var: 0.30 (+/- 0.13)

CV Runtime: 0.4859650135040283

Neural Network Regressor RMSE:: 0.65 (+/- 0.03)

Neural Network Regressor Expl Var: 0.33 (+/- 0.11)

CV Runtime: 4.137418031692505

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

*With Binning Selected 4 features*

Not Selected ﻿['Class', 'fixed acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', , 'density', 'pH']

Selected : ['volatile acidity', 'total sulfur dioxide', 'sulphates', 'alcohol']

*Without Binning Selected 3 features*

Not selected : ﻿['Class', 'fixed acidity', , 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH']

Selected : ['volatile acidity', 'sulphates', 'alcohol']

**Summary Questions**

*\*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?*

*First, comparing the result from Classification method with feature selection in Pima Diabetes, the scores from boosting is slightly better than random forest and decision tree. The cv-run times for boosting are also lower which mean it run faster in boosting. Moreover, the neural network with ‘adam’ as a parameter has a slightly lower in score and run slower than all of the method.*

*Second, in Wine dataset (Regressor), both boosting method and neural network perform better than Random forest and Decision tree. The scores show that boosting and neural network produce less error (RMSE) and gained more on explained variance (Expl Var). The tradeoff is that boosting and neural network take more times to run than decision tree and random forest. In the huge dataset it could take days to run for once.*

*I would tell my boss that boosting method and Neural Network perform better in term of score than random forest and decision tree with less error (misclassification) which it’s more reliable. However, the complexity of the algorithm is very slow to run in a huge dataset in which could cost more. Moreover, it is hard to explain to customer because of the structure and the complication of algorithms especially Neural Network.*

*\*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?*

'Blood Glucose', 'BMI', 'Age'

*What we learn from the Pima diabetes is that the main feature that are selected in to model are Blood Glucose , BMI (Body Mass Index) and Age of the patient. These features are selected in all algorithm including Random Forest, Boosting and Neural Network which it performs very well. For example, with some specific range of Blood Glucose level, body mass index and age, we could make an assumption which patient could potentially have risk in diabetes. Moreover, Family History is selected in random forest which we may consider this feature to include in to model if any member in family have diabetes before.*

*With these features, we can used it to pre-screen diabetes from patient more effective combining with doctor decision(domain expert).*

-1.5 Missing Python code

Q1b: -0.5: The significance is that we are using weak learners as an ensemble to create a performing classifier by aggregating their votes.

Q3: -0.5: There is minimal, if any, improvement in prediction performance

Q5: -0.5: Scores are essentially the same as Q2 for the lbfgs solver, not outside the range of expected variation.

Q8b: -0.5: Using these bins we are predicting whether a wine will be above average or below average.

Q9: -0.5: No explaination

Q10b: -0.5: No explaination

Q10b: -0.5: total sulfur dioxide shows up as an important feature when we bin the target, but not when we don’t.

Q11: -0.5: Performance was very similar for Boosting methods and Neural Networks to performance of Random Forests in HW2, though Neural Networks did perform a little worse on the Diabetes dataset. Neural Networks take longer to run than other modeling methods we’ve used up to this point.

Also remember boosting uses an ensemble of weak learners to create a classifier by aggregating their votes.

NN uses a model inspired by natural intelligence.