**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?*

*\*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?*

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?*

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

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?*

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?*

*\*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?*

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?*

**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.*

*\*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)?*

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?*

*\*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?*

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)?*

*\*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?*

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?*

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?*

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

**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?*

*\*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?*