**Homework #2 Instructions**

**Casey Bennett, PhD**

**DSC540, Winter 2019**

**DePaul University**

**Overview**

There are two python scripts, implementing Scikit using the same basic template you were shown in class. The first one uses a dataset related to Diabetes from Pima Indians for classification, the target variable being the presence/absence of diabetes in individuals. The second script uses a dataset of quality ratings for various red wines rated 1-10, for regression. We will explore ways of using Random Forests and Bagging to create models, as well as talk about the effects of feature normalization.

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 for Random Forests describing methods and variables available can be found in the included links document.

*\*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 HW2\_Diabetes.py

1. First, let’s run a simple test/train split using a random forest. To do so, we need to do two things, first create a Random Forest Classifier object (clf), then “fit” some data using that object.
   1. Note on line 191, the test/training data is already split for you into separate arrays for both the features (data) and the target
   2. On line 196, we need to create a Random Forest Classifier. We can do this using RandomForestClassifier(). Using the API link in the accompanying document, call that function, and pass in the following parameters:
      1. Set number of trees to 100
      2. Set maximum depth = None
      3. Set minimum # of samples for split to occur = 3
      4. Set splitting criterion = 'entropy'
      5. Set random\_state variable to rand\_st
   3. Now we need to fit data. On line 197, call a fit() function on the clf object. It takes two arrays, the first for the training set feature data (data\_train) and the second for the training set target data (target\_train).
   4. Note that performance is already be calculated here for you on lines 199 and 201, using the test set (data\_test and target\_test)

*\*Question #1a: Run the code 5 times, record the accuracy and AUC score. What do you notice about the scores?*

*\*Question #1b: For the fit() method of a RandomForestClassifier, it lists three possible parameters on the API webpage, what are they? Define what you could pass in to each one?*

1. Let’s repeat step 1 above, this time using cross-validation. To do so, we need to do two things, first create a Random Forest Classifier object (clf), then second pass that object and some data arrays into a “cross-validate” function.
   1. To turn on cross-validation, we need to first on line 32 change the cross\_val flag to equal 1 instead of 0
   2. On line 211, we need to create a Random Forest Classifier. We can do this using RandomForestClassifier(). Using the API link in the accompanying document, call that function, and pass in the following parameters:
      1. Set number of trees = 100
      2. Set maximum depth = None
      3. Set minimum # of samples for split to occur = 3
      4. Set splitting criterion = 'entropy'
      5. Set random\_state variable to rand\_st
   3. Now we need to run cross-validation. On line 212, call a cross\_validate() function and return the value to a scores object (you can do this by setting the scores = cross\_validate() ). For the cross\_validate function, pass in the following parameters:
      1. Set the estimator = clf
      2. Set the data to be fit = data\_np
      3. Set the target variable = target\_np
      4. Set scoring = scorers
      5. Set the # of cv folds = 5
   4. Note we are passing in the WHOLE dataset here, not just training data or training target values
   5. Note that performance is already be calculated here for you with the scorers being setup on line 207, and the results pulled out and printed below.

*\*Question #2: Run the code once, record the accuracy and AUC score.*

1. Let’s explore how the number of trees affects performance of a Random Forest.
   1. On line 211, change the number of trees from 100 to 5
   2. Now set the number of trees to 10
   3. Now set the number of trees to 20
   4. Now set the number of trees to 50
   5. Now set the number of trees to 200
   6. Now set the number of trees to 500
   7. Now set the number of trees to 1000

*\*Question #3: Run the code once for each setting of the number of trees (5,10,20,50,100,200,500, 1000), record the accuracy and AUC scores. What do you notice about the scores? How do they change as the number of trees increases?*

1. Now let’s try applying feature selection method we used for the wine dataset in Homework #1 to the diabetes dataset. We will turn on the Wrapper-Based Feature Selection, which essentially builds lots of models with different subsets of features, and picks the subset that performs the best. For simplicity here though, we will just build a single subset and select the top variables. We will use the same Random Forest model for this.
   1. First, on line 211, change the number of trees back to 100
   2. To turn on feature selection, we need to first on line 37 change the feat\_select flag to equal 1 instead of 0
   3. Note that there is an option to change the feature selection type on line 38, but the homework code is hard-coded to only use wrapper-based, so this doesn’t matter for now
   4. You will need to add a RandomForestClassifier() call to pass to the clf object on line 148, you can use something similar to the calls used elsewhere in the code (e.g. line 196). Don’t forget to set the parameters, particularly the random\_state and number of trees to 100.
   5. Note the SelectFromModel() function being called on line 149, 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 in question #2?*

*\*Question #4b: What features were selected, and which were removed?*

1. Random Forests and similar tree methods also produce a “feature importance” score that can also be used for feature selection. Let’s try manually setting up a feature selection section for that.
   1. Feature selection should already be turned on, with line 37 having feat\_select flag set to equal 1
   2. On line 38, change fs\_type from 2 to 4
   3. You will need to add a RandomForestClassifier() call to pass to the clf object on line 156 (under where it says “if fs\_type=4”), 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 trees to 100.
   4. You will then need a line to fit the data. On line 157, call a fit() function on the clf object.
   5. On the line below that, add a line creating an empty array, and call it sel\_idx.
   6. Now we need to create a For loop, taking each value out of the feature\_importances\_ in the clf object in turn. Call this value ‘x’.
   7. Within the For loop, create an If-Else statement. Set “If” so that if x >= mean of the feature\_importances\_ array, then append the value of 1 to sel\_idx. For “Else”, append a 0.
   8. Hint, you can use numpy to calculate means and other statistics of arrays, e.g. np.mean (numpy is given the alias of ‘np’ in the code).
   9. Try to run the code.

*\* Question #5: Run the code once, record the accuracy and AUC scores. What features were selected, and which were removed? How do the selected features compare to what you saw in Question #4 above? Was the performance (accuracy, AUC) different than in Question #4?*

**Wine Quality Dataset**

Open up HW2\_Wine.py … First, let’s repeat the steps we did above for Diabetes, with some tweaks. We will skip the Train/Test version, and jump right to the Cross-Val version.

1. Let’s repeat what we did for the Diabetes dataset in Question #2 above here for the Wine dataset. To do so, we need to do two things, first create a Random Forest Classifier object (clf), then second pass that object and some data arrays into a “cross-validate” function.
   1. On line 32, cross-validation is already turned on for you (set to 1)
   2. On line 211, we need to create a Random Forest Regressor. We can do this using RandomForestRegressor(). Using the API link in the accompanying document, call that function, and pass in the following parameters:
      1. Set number of trees = 100
      2. Set max features (for each split) = .33
      3. Set maximum depth = None
      4. Set minimum # of samples for split to occur = 3
      5. Set splitting criterion = 'mse'
      6. Set random\_state variable to rand\_st
   3. Now we need to run cross-validation. On line 212, call a cross\_validate() function and return the value to a scores object (you can do this by setting the scores = cross\_validate() ). For the cross\_validate function, pass in the following parameters:
      1. Set the estimator = rgr
      2. Set the data to be fit = data\_np
      3. Set the target variable = target\_np
      4. Set scoring = scorers
      5. Set the # of cv folds = 5
   4. Note we are passing in the WHOLE dataset here, not just training data or training target values
   5. Note that performance is already be calculated here for you with the scorers being setup on line 207, and the results pulled out and printed below.

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

1. Let’s explore how the number of trees affects performance of a Random Forest.
   1. On line 211, change the number of trees from 100 to 5
   2. Now set the number of trees to 10
   3. Now set the number of trees to 20
   4. Now set the number of trees to 50
   5. Now set the number of trees to 200
   6. Now set the number of trees to 500
   7. Now set the number of trees to 1000

*\*Question #7a: Run the code once for each setting of the number of trees (5,10,20,50,100,200,500, 1000), record the RMSE and Expl Variance. What do you notice about the scores? How do they change as the number of trees increases? Is this the same as you for the Diabetes dataset in Question #3?*

*\*Question #7b: What about run-times, how do those change as you change the number of trees? What do the changes in scores and run-times tell us about choosing the right number of trees?*

1. Now let’s turn on Wrapper-Based Feature Selection, which essentially builds lots of models with different subsets of features, and picks the subset that performs the best. For simplicity here though, we will just build a single subset and select the top variables. We will use the same Random Forest model for this.
   1. First, on line 211, change the number of trees back to 100
   2. To turn on feature selection, we need to first on line 37 change the feat\_select flag to equal 1 instead of 0
   3. Note that there is an option to change the feature selection type on line 38, but the homework code is hard-coded to only use wrapper-based, so this doesn’t matter for now
   4. You will need to add a RandomForestRegressor() call to pass to the clf object on line 148, you can use something similar to the calls used elsewhere in the code (e.g. line 196). Don’t forget to set the parameters, particularly the random\_state and number of trees to 100.
   5. Note the SelectFromModel() function being called on line 149, this is where the actual feature selection occurs, with the clf object being passed in

*\*Question #8a: Run the code once, record the RMSE and Expl Variance. What do you notice about the scores? How do they compare to the performance above in question #6?*

*\*Question #8b: What features were selected, and which were removed?*

1. Let’s see if normalizing the features changes performance. To do this we are going to use the simple scaling function, but there are many different types of normalization available in Scikit (see the preprocessing link in the accompanying API document).
   1. First, turn off feature selection, on line 37 change the feat\_select flag back to 0
   2. To turn on feature normalization, on line 34 change the norm\_features flag to equal 1
   3. In the Preprocessing Section, on line 96, we need to call the scale() function, and pass into it the WHOLE dataset of features, which is data\_np. See the API link in the accompanying document.
   4. Note that we are simply returning the output of the scale function (i.e. normalized features) back to the same object.

*\*Question #9: Run the code once and record the RMSE and Expl Variance. What do you notice about the scores? How do they differ from question #6?*

1. Finally, let’s try to setup Bagging, using Decision Tree Regressors. Remember though, Bagging can use any sort of classifier or regressor as an ensemble, but generally simple fast methods (decision trees, K-nearest neighbors, naïve bayes, etc) are best because you are going to build a whole bunch of them. A neural network, for instance, would probably take too long on any decent sized dataset.
   1. First, turn off feature normalization, on line 34 change the norm\_features flag back to 0
   2. We’ll do this in the single test/train split section, so turn cross-validation off on line 32 by setting the cross\_val flag to 0
   3. We need to import the function into our script before we can call it. On line 11, add BaggingRegressor to the comma-separated list of packages imported from the sklearn.ensemble module.
   4. Let you do the rest yourself. You can see between lines 195-202 I’ve left space for you code You’ll need to add a DecisionTreeRegressor() on line 195. You can use your code from HW1 if you want, with the same parameters. Make sure you set random\_state= rand\_st.
   5. Create a BaggingRegressor() on the line below, pass it to the bag object, then fit the bag to data. Make sure you set parameters:
      1. Set max\_samples =0.6
      2. Set random\_state = rand\_st
   6. You will need to create some scoring on the lines down below replacing the commented lines. Refer back to Homework 1, but protip: we need to score/predict using the bag object, not the rgr object.

*\*Question #10a: Run the code 5 times and record the RMSE and Expl Variance. What do you notice about the scores? How stable are they? How do they differ from question #6?*

*\*Question #10b: Based on the API webpage for the BaggingRegressor() in the accompanying API links document, what two parameters do we need to change to create a Random Subspaces model?*

*\*Question #10c: Based on the API webpage for the BaggingRegressor(), notice that the function when called also calculates the out-of-bag error (oob\_score). Should we be using that metric then rather than a test/train split or cross-validation with bagging, or can we use them together (hint: do some googling)?*

**Summary Questions**

*\*Question #11: Compare the performance of Random Forests here to the Decision Tree models for both datasets in Homework #1. Did Random Forest perform better, worse, or the same? If your boss or customer asked why that might be, how would you explain?*

*\*Question #12: We’ve now seen several different sampling and evaluation techniques. When it comes to evaluating model performance, what is the “gold standard” approach?*