Alex Teboul

Homework #3

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.
    a. 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.
    b. On line 278, create a GradientBoostingClassifier(). Using the API link in the accompanying document, call that function, and pass in the following parameters:

    i. Set number of estimators to 100
    ii. Set loss = ‘deviance’
    iii. Set the learning rate = 0.1
    iv. Set maximum depth =3
    v. Set minimum # of samples for split to occur = 3
    vi. Set random_state variable to rand_st

c. Add in a cross_validate function on line 279 (use previous homework as an example) with 5 folds, and pass in the clf object.

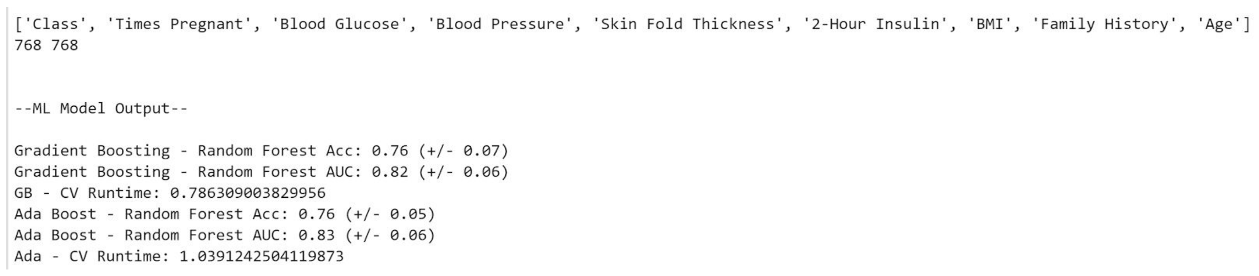
d. You may want to edit the print statements, so they say “Gradient Boosting” when printing scores, to make the output easier to see.

e. 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.

f. Change the clf to AdaBoostClassifier() and pass in the following parameters:

    i. Set number of estimators to 100
    ii. Set base_estimator = None
    iii. Set the learning rate = 0.1
    iv. 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?



```
['Class', 'Times Pregnant', 'Blood Glucose', 'Blood Pressure', 'Skin Fold Thickness', '2-Hour Insulin', 'BMI', 'Family History', 'Age']
768 768


--ML Model Output--

Gradient Boosting - Random Forest Acc: 0.76 (+/- 0.07)
Gradient Boosting - Random Forest AUC: 0.82 (+/- 0.06)
GB - CV Runtime: 0.786309003829956
Ada Boost - Random Forest Acc: 0.76 (+/- 0.05)
Ada Boost - Random Forest AUC: 0.83 (+/- 0.06)
Ada - CV Runtime: 1.0391242504119873
```

- **GB**
    - **Accuracy:** 0.76 (+/- 0.07)
    - **AUC:** 0.82 (+/- 0.06)
- **Ada**
    - **Accuracy:** 0.76 (+/- 0.05)
    - **AUC:** 0.83 (+/- 0.06)
- I notice that the Accuracy and AUC scores are roughly equal for the Gradient Boosting and Ada Boost classifiers.

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 is a Decision Tree classifier with a max depth of 1. This is significant because it means the base estimator is a weak learner that is creating the decision boundaries that allow the ada boost algorithm to work.

2) Now let's try a Neural Network and compare.

    a. 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.

    b. 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"

    c. Change the clf to MLPClassifier() and using theAPI link in the accompanying document, pass in the following parameters:

        i. Set activation = 'logistic'
        ii. Set solver = 'lbfgs'
        iii. Set alpha = 0.0001
        iv. Set the max # of iterations = 1000
        v. Set the hidden layers sizes = (10,)
        vi. Set random_state variable to rand_st

```
--ML Model Output--

Gradient Boosting - Random Forest Acc: 0.76 (+/- 0.07)
Gradient Boosting - Random Forest AUC: 0.82 (+/- 0.06)
GB - CV Runtime: 0.859466552734375
Ada Boost - Random Forest Acc: 0.76 (+/- 0.05)
Ada Boost - Random Forest AUC: 0.83 (+/- 0.06)
Ada - CV Runtime: 1.1826891899108887
Neural Network - Acc: 0.72 (+/- 0.07)
Neural Network - AUC: 0.74 (+/- 0.05)
Ada - CV Runtime: 2.741201877593994
```

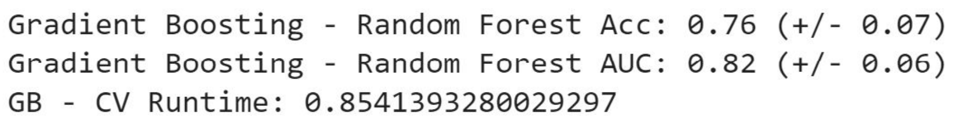
Question #2a: Run the code once, record the accuracy and AUC score. What do you notice about the scores? How do they compare to boosting methods? What about run times?

- **NN Accuracy:** 0.72 (+/- 0.07)
- **NN AUC:** 0.74 (+/- 0.05)
- I notice accuracy and AUC scores are lower than they were for the boosting methods, but not a great deal lower. That said, the runtime for the neural network is much higher, about 2x that of the Ada Boost and 3x that of the Gradient Boosted example. This initial run would therefore suggest that the boosted methods have some advantages over the neural networks in this example, as we achieve as-good or better performance in terms of Accuracy and AUC, with lower runtimes.

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

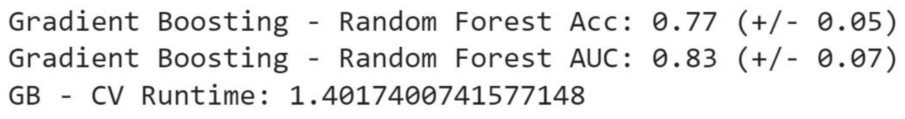
- According to the Scikit MLP Classifier documentation, 'adam' is a stochastic gradient-based optimizer that is most effective when used on large datasets. The 'lbfgs' solver we used in this example is better for smaller datasets like the one we have because it has only 768 data points. If we were using a dataset with thousands of samples, 'adam' would be advised to achieve faster training time and validation score according to the documentation.

3) Let's explore how the depth of each tree affects performance in Gradient Boosting.
    a. Change the max_depth parameter of the GradientBoostingClassifier() from 3 to 5
    b. Now set the max_depth to 7
    c. Now set the max_depth to 10



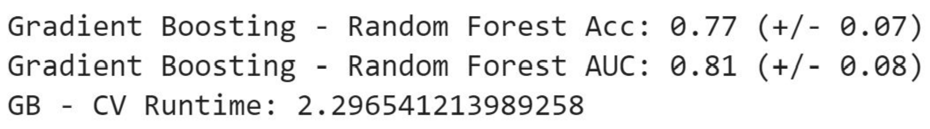
```
Gradient Boosting - Random Forest Acc: 0.76 (+/- 0.07)
Gradient Boosting - Random Forest AUC: 0.82 (+/- 0.06)
GB - CV Runtime: 0.8541393280029297
```

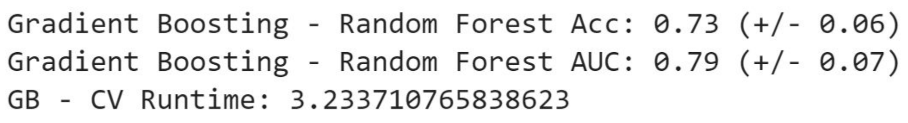
3



```
Gradient Boosting - Random Forest Acc: 0.77 (+/- 0.05)
Gradient Boosting - Random Forest AUC: 0.83 (+/- 0.07)
GB - CV Runtime: 1.4017400741577148
```

5


7
```
Gradient Boosting - Random Forest Acc: 0.77 (+/- 0.07)
Gradient Boosting - Random Forest AUC: 0.81 (+/- 0.08)
GB - CV Runtime: 2.296541213989258
```


10
```
Gradient Boosting - Random Forest Acc: 0.73 (+/- 0.06)
Gradient Boosting - Random Forest AUC: 0.79 (+/- 0.07)
GB - CV Runtime: 3.233710765838623
```

Question #3: Run the code <u>once</u> 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?

Question 3: Results **GB - max_depth: (3,5,7,10)**			
max_depth	**Acc**	**AUC**	**Run-time**
3	0.76 (+/- 0.07)	0.82 (+/- 0.06)	0.85
5	0.77 (+/- 0.05)	0.83 (+/- 0.07)	1.40
7	0.77 (+/- 0.07)	0.81 (+/- 0.08)	2.30
10	0.73 (+/- 0.06)	0.79 (+/- 0.07)	3.23

- As the max depth increases the Accuracy and AUC scores appear to decrease slightly, while the run-times increase. Doubling the max depth more than doubles the runtime on average.

4) 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.

a. First, on line where we call the GradientBoostingClassifier(), change the max_depth back to 3 from where we changed it above in Question #3

b. To turn on feature selection, we need to first on line 38 change the feat_select flag to equal 1 instead of 0

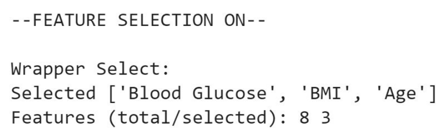
c. Note that there is an option to change the feature selection type is already set to 2 (wrapper-based) on line 39

d. 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.

e. 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).

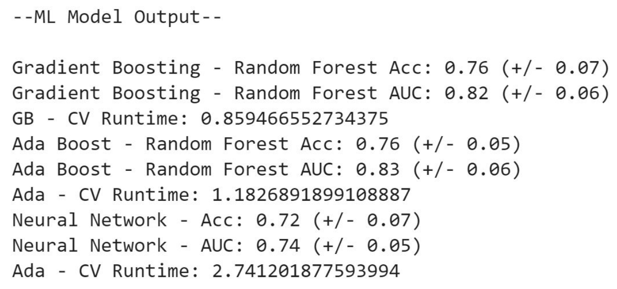
f. Note the SelectFromModel() function being called on line 192, this is where the actual feature selection occurs, with the clf object being passed in

before **after (feat select)**



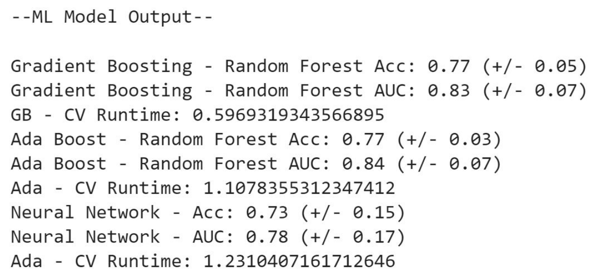
```
--FEATURE SELECTION ON--

Wrapper Select:
Selected ['Blood Glucose', 'BMI', 'Age']
Features (total/selected): 8 3
```



```
--ML Model Output--

Gradient Boosting - Random Forest Acc: 0.76 (+/- 0.07)
Gradient Boosting - Random Forest AUC: 0.82 (+/- 0.06)
GB - CV Runtime: 0.859466552734375
Ada Boost - Random Forest Acc: 0.76 (+/- 0.05)
Ada Boost - Random Forest AUC: 0.83 (+/- 0.06)
Ada - CV Runtime: 1.1826891899108887
Neural Network - Acc: 0.72 (+/- 0.07)
Neural Network - AUC: 0.74 (+/- 0.05)
Ada - CV Runtime: 2.741201877593994
```



```
--ML Model Output--

Gradient Boosting - Random Forest Acc: 0.77 (+/- 0.05)
Gradient Boosting - Random Forest AUC: 0.83 (+/- 0.07)
GB - CV Runtime: 0.5969319343566895
Ada Boost - Random Forest Acc: 0.77 (+/- 0.03)
Ada Boost - Random Forest AUC: 0.84 (+/- 0.07)
Ada - CV Runtime: 1.1078355312347412
Neural Network - Acc: 0.73 (+/- 0.15)
Neural Network - AUC: 0.78 (+/- 0.17)
Ada - CV Runtime: 1.2310407161712646
```

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?

- **GB Acc:** 0.77 (+/- 0.05) | **GB AUC:** 0.83 (+/- 0.07)
- **Ada Acc:** 0.77 (+/- 0.03) | **Ada AUC:** 0.84 (+/- 0.07)
- **NN Acc:** 0.73 (+/- 0.15) | **NN AUC:** 0.78 (+/- 0.17)
- There does not appear to be a decrease in the Accuracy or AUC score performance following feature selection. That said, there is a negative impact on the NNs as model stability goes down. Specifically, the range of Acc and AUC increases as seen above. Feature selection comes with the benefit of decreasing run times in each model. More

specifically, run-times very slightly decrease for GB and Ada models, and are roughly cut in half for the NN model.

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?

- **Selected Features:** Blood Glucose, BMI, and Age.
- **Removed Features:** Family History, Times Pregnant, Blood Pressure, Skin Fold Thickness, and 2-Hour Insulin
  - vs. Feature Selection with RF in HW2:
    - **Selected Features:** Blood Glucose, BMI, Family History, and Age
    - **Removed Features:** Times Pregnant, Blood Pressure, Skin Fold Thickness, 2-Hour Insulin
- This means that there is a difference in the feature selection from HW2. Specifically, Family History was selected as a feature in HW2, but was not with this change in feature selection. Blood Glucose, BMI, and Age were selected as important features for the models to use in both assignments.

5) 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.

a. Change the solver parameter of the MLPClassifier() from lbgfs to 'adam'

```
Neural Network - Acc: 0.72 (+/- 0.07)
Neural Network - AUC: 0.80 (+/- 0.08)
NN - CV Runtime: 4.027127027511597
```

**Question #5: Run the code* <u>once</u> *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?*

Question 3: Results NN - lbgfs vs. adam			
solver	**Acc**	**AUC**	**Run-time**
'lbgfs'	0.73 (+/- 0.15)	0.78 (+/- 0.17)	1.23
'adam'	0.72 (+/- 0.07)	0.80 (+/- 0.08)	4.03

- I notice that the average scores for lbgfs and adam are roughly equal, but the range of scores has sharply decreased with the adam solver. I assume this is due to less variance in our predicted classes with the adam solver. Greater model stability is a good thing. This comes at a cost however, with run-times for adam that are almost 4x what they are for lbgfs. In our example, with such few data points, this trade-off is acceptable. It is interesting then, that performance appears to be improved by using the adam solver despite the documentation recommending lbgfs for smaller datasets like ours.

Wine Quality Dataset

Open up HW3_Wine.py … First, let's repeat the steps we did above for Diabetes.

6) First, let's run a Gradient Boosting Model, then an Ada Boosting Model, and compare.

   a. First we need to import the functions, on line 11, add calls for the GradientBoostingRegressor() and AdaBoostRegressor() from the sklearn "ensemble" package.

   b. On line 278, create a GradientBoostingRegressor(). Using the API link in the accompanying document, call that function, and pass in the following parameters:

      i. Set number of estimators to 100
      ii. Set loss = 'ls'
      iii. Set the learning rate = 0.1
      iv. Set maximum depth =3
      v. Set minimum # of samples for split to occur = 3
      vi. Set random_state variable to rand_st

   c. Add in a cross_validate function on line 279 (use previous homework as an example) with 5 folds, and pass in the clf object.

   d. You may want to edit the print statements, so they say "Gradient Boosting" when printing scores, to make the output easier to see.

   e. 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.

   f. Change the clf to AdaBoostRegressor() and pass in the following parameters:

      i. Set number of estimators to 100
      ii. Set base_estimator = None
      iii. Set loss = 'linear'
      iv. Set the learning rate = 0.5
      v. Set random_state variable to rand_st

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

- **Gradient Boosting**
  - **GB RMSE:** 0.64 (+/- 0.01)
  - **GB Expl Var:** 0.34 (+/- 0.12)
  - **GB - CV Runtime:** 1.50
- **Ada Boosting**
  - **Ada RMSE:** 0.66 (+/- 0.03)
  - **Ada Expl Var:** 0.31 (+/- 0.16)
  - **Ada - CV Runtime:** 1.80

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 learning rate parameter is used to modify or shrink the contribution of each tree to our model. The Scikit documentation notes that there is a trade-off between learning rate and the number of estimators or trees. In general, a high learning rate or if learning rate was just 1 (ie no modification to the contribution of each) then you run the risk of overfitting, so you want to slow down the model tuning with a lower learning rate. In general you want a low enough learning rate to avoid serious over-fitting while also not so low as to under-fit the data.

7) Now let's try a Neural Network and compare.

   a. 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.

   b. 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"

   c. Change the clf to MLPRegressor() and using the API link in the accompanying document, pass in the following parameters:

      i. Set activation = 'logistic'

      ii. Set solver = 'lbfgs'

      iii. Set alpha = 0.0001

      iv. Set the max # of iterations = 1000

      v. Set the hidden layers sizes = (10,)

      vi. Set random_state variable to rand_st

```
--ML Model Output--

Gradient Boosting RMSE:: 0.64 (+/- 0.01)
Gradient Boosting Expl Var: 0.34 (+/- 0.12)
GB - CV Runtime: 1.4196977615356445
Ada Boosting RMSE:: 0.66 (+/- 0.03)
Ada Boosting Expl Var: 0.31 (+/- 0.16)
Ada - CV Runtime: 1.8057646751403809
Neural Network RMSE:: 0.66 (+/- 0.06)
Neural Network Expl Var: 0.30 (+/- 0.10)
NN - CV Runtime: 7.392379283905029
```

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 RMSE:** 0.66 (+/- 0.06)
- **Neural Network Expl Var:** 0.30 (+/- 0.10)
- **NN - CV Runtime:** 7.40
- I notice that the RMSE is 0.66 (+/- 0.06) and Explained Variance is 0.30 (+/- 0.10) - neither of which indicated particularly promising/good performance. These are also similar to the scores seen in the boosting methods and other methods performed in previous homeworks.
- Compared to the boosting methods, performance is comparable, except that run-time for the NN is a lot greater than either boosting method. GB has the lower RMSE and greatest stability in that score with +/- 0.01 while NN has +/- 0.06. All suffer from low Expl Var with ranges +/- 0.1.

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?

- Two hidden layers of 10 and 10 you could set hidden_layer_sizes = (10,10), whereas the single hidden layer of 20 would be (20,). Though it also depends what your input layer is and output layer should be - these would change how you set this.

8) 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.

   a. 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.

b. To turn on target discretization, we need to change the binning flag on line 36 to equal 1 instead of 0

c. 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

d. 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.

  i. GradientBoostingClassifier()

  ii. AdaBoostClassifier()

  iii. MLPClassifier()

  iv. Don't forget to copy the scorers line!

e. 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.

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


--ML Model Output--

Gradient Boosting - Random Forest Acc: 0.73 (+/- 0.05)
Gradient Boosting - Random Forest AUC: 0.81 (+/- 0.06)
GB - CV Runtime: 1.6606910228729248
Ada Boost - Random Forest Acc: 0.74 (+/- 0.05)
Ada Boost - Random Forest AUC: 0.82 (+/- 0.06)
Ada - CV Runtime: 1.5485756397247314
Neural Network - Acc: 0.73 (+/- 0.06)
Neural Network - AUC: 0.81 (+/- 0.06)
NN - CV Runtime: 6.011159420013428
```

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)?

- Scores shown above. I notice that these scores are even across the three methods in terms of Accuracy and AUC. The runtimes of GB and Ada are close at about 1.5 seconds, while NN is up at 6 seconds. These scores are also fairly "good" in terms of predicting a good versus bad wine based on the features included. Predicting this 7 out of 10 times based on the chemical composition of the wines suggests that in general people prefer a certain type of wine (or at least the reviewer population did).

- It is a bit strange to compare them, but as the classification task is *easier* in a sense. Here we are trying to classify (discrete), whereas before we were attempting regression (continuous). It is expected that there will be more errors and variance when trying to predict a continuous variable through regression than there would be trying to classify into discrete classes. One isn't necessarily better than the other, it depends on what the aim is. For this particular dataset, as it is about wine quality, I think it is acceptable to use classification instead of regression. The ratings are subjective anyways, so it would be more useful to distinguish between a generally good red wine and a poor one, rather than predict if the wine would be rated a 7 versus a 6.

****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?***

- We are predicting whether or not the wines would be rated *low* versus *high*. The *low* rated wines are scores between 3-5 and the *high* rated wines are rated between 6 and 8. For reference, the wines were rated on a scale from 3-8. There were 744 wines rated as *low* and 855 rated as *high*.
- I would explain that we created these *low* and *high* bins, and then created 3 predictive models to try to predict or classify the wines as *low* or *high* based on their chemical characteristics like 'volatile acidity', 'alcohol', and 'sulphates'.
- The models used were Gradient Boosting, Ada Boosting, and Neural Networks. All had similar performance, showing that we can predict that *low* versus *high* rating about 7 out of 10 times on average based on wine characteristics.

9) 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.

   a. On line 34, change the norm_target flag to equal 1 instead of 0

```
Bin 0 : -3.265164632733176 -0.787822640922809 744
Bin 1 : 0.4508483549823745 2.9281903467927415 855


--ML Model Output--

Gradient Boosting - Random Forest Acc: 0.73 (+/- 0.05)
Gradient Boosting - Random Forest AUC: 0.81 (+/- 0.06)
GB - CV Runtime: 1.6827313899993896
Ada Boost - Random Forest Acc: 0.74 (+/- 0.05)
Ada Boost - Random Forest AUC: 0.82 (+/- 0.06)
Ada - CV Runtime: 1.5600368976593018
Neural Network - Acc: 0.73 (+/- 0.06)
Neural Network - AUC: 0.81 (+/- 0.06)
NN - CV Runtime: 6.046165943145752
```

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?

- Scores shown above for the 3 models. I notice that it becomes a little harder to interpret the bins that were created as now it is based on the distribution of the scores. That said Accuracy, AUC and Runtime do not appear to have been altered from this change.
- The scores are comparable (roughly equivalent) to the scores from Question #8a.

10) 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.

a. 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

b. To turn on feature selection, we need to first on line 38 change the feat_select flag to equal 1 instead of 0

c. Note that there is an option to change the feature selection type is already set to 2 (wrapper-based) on line 39

d. 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.

e. 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).

f. 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

g. 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?*

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


--FEATURE SELECTION ON--

Wrapper Select:
Selected ['volatile acidity', 'total sulfur dioxide', 'sulphates', 'alcohol']
Features (total/selected): 11 4


--ML Model Output--

Gradient Boosting - Random Forest Acc: 0.73 (+/- 0.05)
Gradient Boosting - Random Forest AUC: 0.81 (+/- 0.05)
GB - CV Runtime: 0.894233226776123
Ada Boost - Random Forest Acc: 0.73 (+/- 0.06)
Ada Boost - Random Forest AUC: 0.82 (+/- 0.06)
Ada - CV Runtime: 1.3653311729431152
Neural Network - Acc: 0.73 (+/- 0.06)
Neural Network - AUC: 0.81 (+/- 0.06)
NN - CV Runtime: 5.967337608337402
```

```
Bin 0 : -3.265164632733176 -0.787822640922809 744
Bin 1 : 0.4508483549823745 2.9281903467927415 855


--ML Model Output--

Gradient Boosting - Random Forest Acc: 0.73 (+/- 0.05)
Gradient Boosting - Random Forest AUC: 0.81 (+/- 0.06)
GB - CV Runtime: 1.6827313899993896
Ada Boost - Random Forest Acc: 0.74 (+/- 0.05)
Ada Boost - Random Forest AUC: 0.82 (+/- 0.06)
Ada - CV Runtime: 1.5600368976593018
Neural Network - Acc: 0.73 (+/- 0.06)
Neural Network - AUC: 0.81 (+/- 0.06)
NN - CV Runtime: 6.046165943145752
```

binning=1

```
--FEATURE SELECTION ON--

Wrapper Select:
Selected ['volatile acidity', 'sulphates', 'alcohol']
Features (total/selected): 11 3


--ML Model Output--

Gradient Boosting RMSE: 0.66 (+/- 0.02)
Gradient Boosting Expl Var: 0.29 (+/- 0.14)
GB - CV Runtime: 0.5177464485168457
Ada Boosting RMSE: 0.67 (+/- 0.04)
Ada Boosting Expl Var: 0.30 (+/- 0.13)
Ada - CV Runtime: 0.45784449577331543
Neural Network RMSE: 0.64 (+/- 0.04)
Neural Network Expl Var: 0.34 (+/- 0.11)
NN - CV Runtime: 5.176642894744873
```

```
--ML Model Output--

Gradient Boosting RMSE:: 0.64 (+/- 0.01)
Gradient Boosting Expl Var: 0.34 (+/- 0.12)
GB - CV Runtime: 1.4196977615356445
Ada Boosting RMSE:: 0.66 (+/- 0.03)
Ada Boosting Expl Var: 0.31 (+/- 0.16)
Ada - CV Runtime: 1.8057646751403809
Neural Network RMSE:: 0.66 (+/- 0.06)
Neural Network Expl Var: 0.30 (+/- 0.10)
NN - CV Runtime: 7.392379283905029
```

binning=0

- Scores recorded/shown above.
- ***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?***
    - I notice that for the binned models, there is no significant difference in performance (Accuracy & AUC) between the feature selected and original models. This is good because we are getting comparable scores with fewer features. There is also a benefit - though slight, in terms of run-time. GB had the biggest decrease in run-time from the feature selection.
    - I notice that for the un-binned models, there isn't a large change in the RMSE or Expl Var between the models with and without feature selection turned on. This is good as we haven't lost any performance by these metrics, while decreasing the number of features. That said, run-time is cut significantly in the feature selected models, especially for GB and Ada.
- ***Did you notice any changes in run-times?***
    - Yes, run-times are lower for the feature selected models. The change isn't that large proportionally for the Neural Network models, but it is so for the GB and Ada models. Here run-times drop to as much as ⅓ their original run-times.

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

- **binned**
    - **Selected Features:** total sulphur dioxide, volatile acidity, sulphates, alcohol
    - **Removed Features:** fixed acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, density, pH
- **un-binned**
    - **Selected Features:** volatile acidity, sulphates, alcohol
    - **Removed Features:** fixed acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, density, pH, total sulphur dioxide
- **Difference:**
    - In the binned version, total sulphur dioxide is also selected as a feature.

Summary Questions

****Question #11: Compare the performance of Boosting Methods and Neural Networks here compared 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 11: Compare RF, DT, GB, Ada, and NN UN-BINNED & Feature Selected				
	Diabetes		**Wine**	
Model	**Accuracy**	**AUC**	**RMSE**	**Expl Var**
H2: RF	0.76 (+/- 0.05)	0.82 (+/- 0.06)	0.65 (+/- 0.02)	0.33 (+/- 0.11)
H1: DT	0.71 (+/- 0.14)	0.67 (+/- 0.16)	0.91 (+/- 0.08)	-0.35 (+/- 0.26)
H3: GB	0.77 (+/- 0.05)	0.83 (+/- 0.07)	0.66 (+/- 0.02)	0.29 (+/- 0.14)
H3: Ada	0.77 (+/- 0.03)	0.84 (+/- 0.07)	0.67 (+/- 0.04)	0.30 (+/- 0.13)
H3: NN	0.73 (+/- 0.15)	0.78 (+/- 0.17)	0.64 (+/- 0.04)	0.34 (+/- 0.11)

- In terms of Run-times, Neural Networks were the slowest models, with Decision Trees being the fastest models. I could suggest to boss or customer that NN are the most complex model and Decision Trees are the simplest and this is why one takes longer to run than the other. Random Forests, Gradient Boosted, and Ada Boosted models were about the same in terms of run-time. I could explain these run-times being slower than Decision Trees because these models work by creating ensembles of weaker models or tuning themselves to build stronger models - taking time.
- In the chart above I included scores for feature selected models as they were comparable to the un-feature selected models.
- In terms of the best models for Wine:
    - Feature Selected GB or Ada seem to be the best in terms of run-times and scores, though regression was not ideal for this dataset.
    - A better approach is to classify the wines as *high* or *low* and try to predict that, which is also well modeled by GB and Ada. Accuracy of 0.73 (+/- 0.05) and AUC of 0.81 (+/- 0.06) was found for GB on this binary classification task.
- In terms of the best models for Diabetes:
    - Blood Glucose, BMI, and Age appear to be important features regardless of the model used, so these are important data points to track.
    - Model performance is pretty comparable for RF, GB, and Ada. With DT and NN having some worse stability in terms of accuracy and AUC scores that vary more. For the NN in particular, turning feature selection on actual makes the model less stable.

- In the end I could tell my boss that Decision Trees are too simple for the tasks at hand and don't have ideal performance. Neural Networks are too complex for the tasks at hand and do not give us any increased performance, even though they take more computational time. Diabetes is impacted strongly by Blood Glucose, BMI, and Age and I would use either RF, GB, or Ada model to predict this. For the Wine dataset, it appears that volatile acidity, sulphates, and alcohol are highly predictive of whether or not people will consider a red wine good or bad. In this sense, it would be good to test these features to give our wines the best chance of appealing to the most customers.

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?

- According to the models we have created, Blood Glucose, BMI, and Age are sufficient to provide a reasonable prediction of diabetes or risk. That said, more domain expert input would be needed here, as there may be different types and severities of diabetes that are more or less influenced or precipitated by different markers/features.
- If we were setting up a diabetes screening program for a local healthcare organization, based on our findings here, it would be useful to record individuals Blood Glucose, BMI, and Age. If the findings in this Pima dataset extrapolated/generalized well to the greater public then the screening could help identify those at risk or displaying diabetes over 70% of the time on average. One of the most common tests currently used involves measuring fasting blood glucose levels, so this screening would be in-line with conventional practices, while also taking BMI and Age into account.