Homework #1

Alex Teboul

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. Both scripts implement a basic decision tree from Scikit to make predictions. We will explore different ways to score the predictions and evaluate results, including setting up different scorers, comparing cross-validation to simple test/training splits, and finally look at the effects of simple feature selection.

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 decision trees 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 HW1_Diabetes.py

1) First, we need to setup scorers for the simple test/train split. Accuracy can be calculated using the score method within the Decision Tree object (clf), but AUC requires us to use a function from the metrics library on predicted labels.

    a. Note on line 111, the test/training data is already split for you into separate arrays for both the features (data) and the target

b. Note the Decision Tree classifier being created on line 116, then fitted using the training data on the line below

c. To calculate accuracy, replace the comment placeholder on line 119 with: clf.score(data_test, target_test)

d. To calculate AUC, replace the comment placeholder on lines 121 with: metrics.roc_auc_score(target_test, clf.predict_proba(data_test)[:,1])

Question #1: Run the code 5 times, record the accuracy and AUC scores of each run. What do you notice about the scores?

Question 1: Results		
Run	**Decision Tree Acc**	**Decision Tree AUC**
Run #1	0.677	0.640
Run #2	0.665	0.637
Run #3	0.710	0.677
Run #4	0.714	0.669
Run #5	0.692	0.652

Observations:

- Accuracy scores for the decision trees range between about 0.66 and 0.71.
- AUC scores range between about 0.63 and 0.67.
- Accuracy score is higher than AUC in all runs.
- These scores suggest that the decision tree classifier is accurate slightly more than ⅔ of the time, which I would not consider as a successful classifier.

2) Next, let's try changing one of the parameters of the Decision Tree.

   a. On line 116, change the criterion option from 'gini' to 'entropy'

Question #2: Run the code again 5 times, record the accuracy and AUC scores of each run. What do you notice about the scores? How do they compare to scores above in question 1?

Question 2: Results **Criterion = ‘entropy’**		
Run	**Decision Tree Acc**	**Decision Tree AUC**
Run #1	0.684	0.646
Run #2	0.665	0.646
Run #3	0.751	0.724
Run #4	0.643	0.620
Run #5	0.684	0.649

Observations:

- Accuracy scores ranged between about 0.64 and 0.75.
- AUC scores ranged between about 0.62 and 0.72.
- Again Accuracy scores are higher than AUC as expected, but not so far off to be cause for major concern.
- **Compared to the scores in Q1:** Scores do not appear to have improved in any significant way by changing the criterion to ‘entropy’.

3) Now, let’s setup scorers for the cross-validation split. This works a bit differently, we have to set up a dictionary of scorers first, then pass that into the cross_validate function call. The function will then return a dictionary of scores, which we can call by name.

a. To turn on cross-validation, we need to first on line 32 change the cross_val flag to equal 1 instead of 0

b. To setup the scorers, replace the comment placeholder on lines 127 with:

{'Accuracy': 'accuracy', 'roc_auc': 'roc_auc'}

c. Note the cross_validate function call on line 132, with clf object passed in, no need to change this yet

d. To calculate accuracy, replace the comment placeholder on lines 134 with: scores['test_Accuracy']

e. To calculate AUC, replace the comment placeholder on lines 136 with:

scores['test_roc_auc']

Question #3: Run the code 5 times, record the accuracy and AUC scores of each run. What do you notice about the scores? How do they compare to the simple test/train split scores in question 1?

Question 3: Results **Scores with Cross-Validation**			
Run	**Decision Tree Acc**	**Decision Tree AUC**	**Runtime**
Run #1	0.71 (+/- 0.08)	0.69 (+/- 0.07)	0.035
Run #2	0.71 (+/- 0.08)	0.69 (+/- 0.07)	0.028
Run #3	0.71 (+/- 0.08)	0.69 (+/- 0.07)	0.027
Run #4	0.71 (+/- 0.08)	0.69 (+/- 0.07)	0.027
Run #5	0.71 (+/- 0.08)	0.69 (+/- 0.07)	0.029

Observations:

- Notice that the Average and Range of Decision Tree Accuracy and AUC remain constant for each run.
- The Decision Tree Accuracy is 0.71 (+/- 0.08) and the AUC is 0.69 (+/- 0.07).
- These are much more stable and important numbers to report out than the results of a single run.
- **Compared to the scores in Q1:** The scores for from Q1 fall within the ranges produced above. In my Q1 results, 0.71 was the max Accuracy I got, but now in Q3 that is the average, with a max that is higher by 0.08. All in all, these numbers are more useful than the ones from Q1.

4) Let's see how the number of cross-validation folds affects performance.
    a. On line 132, change the cv option from 5 to cv=10

Decision Tree Acc: 0.71 (+/- 0.14)

Decision Tree AUC: 0.67 (+/- 0.16)

CV Runtime: 0.054543256759643555

b. Now set the cv option to cv=3

Decision Tree Acc: 0.69 (+/- 0.06)

Decision Tree AUC: 0.67 (+/- 0.05)

CV Runtime: 0.016736745834350586

c. Now set the cv option to cv=8

Decision Tree Acc: 0.69 (+/- 0.06)

Decision Tree AUC: 0.66 (+/- 0.07)

CV Runtime: 0.04172873497009277

Question #4: Run the code once for each cv setting (3,8,10), record the accuracy and AUC scores. What do you notice about the scores? How do they compare to the CV performance above in question 3?

Question 4: Results **Scores with Different CV Settings (3, 5, 8, 10)**			
CV Setting	**Decision Tree Acc**	**Decision Tree AUC**	**Runtime**
cv = 3	0.69 (+/- 0.06)	0.67 (+/- 0.05)	0.017
cv = 5	0.71 (+/- 0.08)	0.69 (+/- 0.07)	0.029
cv = 8	0.69 (+/- 0.06)	0.66 (+/- 0.07)	0.042
cv = 10	0.71 (+/- 0.14)	0.67 (+/- 0.16)	0.055

Observations:

- By increasing the cv setting parameter there is a general increase in the variance or range of prediction accuracy and AUC for the decision trees.
- Including the cv = 5 setting from the previous question, we don't see a big difference in the average accuracy or AUC for the different parameter settings.
- The main concern seems to be with a larger range of scores for high CV values. I tested cv = 30 and it confirms this trend of a much higher range, but also at that point lower accuracy.
- As expected, the runtime increases fairly linearly with an increase in the cv or number of folds parameter.

- **Compared to CV performance in Q3:** The parameter setting of cv=5 seems to be sufficient in this case. Increasing CV does not improve accuracy, AUC, or the range of the two. Decreasing CV doesn't provide a benefit either, with the exception of a lower runtime, which for our purposes here does not matter..

Wine Quality Dataset

Open up HW1_Wine.py … First, let's repeat the steps we did above for Diabetes, with some tweaks.

5) First, we need to setup scorers for the simple test/train split. For regression problems, both RMSE and Explained Variance requires us to use a function from the metrics library on predicted labels. Note that we have to take the square root of the mean_squared_error metric (MSE>>RMSE).

   a. Note on line 186, the test/training data is already split for you into separate arrays for both the features (data) and the target

   b. Note the Decision Tree classifier being created on line 191, then fitted using the training data on the line below

   c. To calculate RMSE, replace the comment placeholder on line 194 with:

   math.sqrt(metrics.mean_squared_error(target_test, rgr.predict(data_test)))

   d. To calculate Explained Variance, replace the comment placeholder on lines 196 with: metrics.explained_variance_score(target_test, rgr.predict(data_test))

Question #5: Run the code 5 times, record the RMSE and Expl Variance scores of each run. What do you notice about the scores?

Question 5:		
Run	**RMSE**	**Explained Variance**
Run #1	0.79	0.07
Run #2	0.85	-0.05
Run #3	0.80	-0.02
Run #4	0.77	0.00
Run #5	0.84	-0.11

Observations:

- The RMSE is between 0.77 and 0.85. High RMSE indicates poor fit.
- The Explained Variance is between -0.05 and 0.07. Such low Explained Variance indicates that the dependent variable variance is not explained by the independent variables included in the decision tree model.
- The RMSE and Explained Variance scores reflect that the model is very poor at fitting the data.

6) Next, let's try changing one of the parameters of the Decision Tree.
    a. On line 191, change the criterion option from 'mse' to 'friedman_mse'

Question #6: Run the code again 5 times, record the RMSE and Expl Variance of each run. What do you notice about the scores? How do they compare to scores above in question 5?

Question 6: Criterion = 'friedman_mse'		
Run	**RMSE**	**Explained Variance**
Run #1	0.81	0.07
Run #2	0.82	0.04
Run #3	0.82	0.04
Run #4	0.80	0.06
Run #5	0.79	0.06

Observations:

- RMSE between 0.79 and 0.82 | Explained Variance between 0.04 and 0.07.
- The RMSE and Explained Variance scores seem more stable in terms of showing a smaller range of values.
- **Compare with scores in Q5:** There doesn't seem to be a big difference. RMSE is still high and Explained variance is still very low. Also, only 5 runs were made so these observations probably won't show the full range of values possible.

7) Now, let's setup scorers for the cross-validation split. This works a bit differently, we have to set up a dictionary of scorers first, then pass that into the cross_validate function call. The function will then return a dictionary of scores, which we can call by name. For RMSE, we have to again take the square root of MSE (in this case flipping the negative sign first).

a. To turn on cross-validation, we need to first on line 32 change the cross_val flag to equal 1 instead of 0

b. To setup the scorers, replace the comment placeholder on lines 202 with:

{'Neg_MSE': 'neg_mean_squared_error', 'expl_var': 'explained_variance'}

c. Note the cross_validate function call on line 207, with rgr object passed in, no need to change this yet

d. To calculate RMSE, replace the comment placeholder on lines 209 with: np.asarray([math.sqrt(-x) for x in scores['test_Neg_MSE']])

e. To calculate Explained Variance, replace the comment placeholder on lines 210 with:

scores['test_expl_var']

Question #7: Run the code 5 times, record the RMSE and Expl Variance scores of each run. What do you notice about the scores? How do they compare to the simple test/train split scores in question 5?

Question 7: Results

RMSE and Expl Variance with Cross-Validation

Run	**RMSE**	**Explained Variance**	**Runtime**
Run #1	0.90 (+/- 0.10)	-0.31 (+/- 0.17)	0.05
Run #2	0.90 (+/- 0.10)	-0.31 (+/- 0.17)	0.05
Run #3	0.90 (+/- 0.10)	-0.31 (+/- 0.17)	0.05
Run #4	0.90 (+/- 0.10)	-0.31 (+/- 0.17)	0.05
Run #5	0.90 (+/- 0.10)	-0.31 (+/- 0.17)	0.05

Observations:

- The RMSE, Explained Variance, and Runtime are the same in each run.
- RMSE is 0.90 (+/- 0.10); Explained Variance is -0.31 (+/- 0.17); and Runtime is 0.05.
- **Compare with scores in Q5:** The RMSE in Q5 tended to be a bit lower, with some values falling out of the range shown now in Q7, but it was also using ‘mse’ instead of ‘friedman_mse’. Explained variance is a lot lower in Q7 than it was in Q5. I’m not even sure how to explain a negative Explained Variance.

8) Let’s see how the number of cross-validation folds affects performance.
    a. On line 207, change the cv option from 5 to cv=10
    b. Now set the cv option to cv=3
    c. Now set the cv option to cv=8

Question #8: Run the code <u>once</u> for each cv setting (3,8,10), record the RMSE and Expl Variance. What do you notice about the scores? How do they compare to the CV performance above in question 7?

Question 8: Results **Scores with Different CV Settings (3, 5, 8, 10)**			
CV Setting	**RMSE**	**Explained Variance**	**Runtime**
cv = 3	0.98 (+/- 0.10)	-0.46 (+/- 0.28)	0.03
cv = 5	0.90 (+/- 0.10)	-0.31 (+/- 0.17)	0.05
cv = 8	0.93 (+/- 0.11)	-0.50 (+/- 0.64)	0.09
cv = 10	0.91 (+/- 0.15)	-0.44 (+/- 0.80)	0.11

Observations:

- Increasing CV seems to decrease RMSE but the range of RMSEs also increases.
- Explained Variance changes but without a discernable pattern, except that the explained variance has a huge range and is also negative.
- Runtime goes up as expected so at least the cross validation should be taking place appropriately.

- **Compared to CV performance in Q7:** It's close to what happened in Q7, which is listed in the table above for cv=5. With values so poor for RMSE and Explained variance maybe the model continues to fit noise or something strange in the dataset.

9) Finally let's see how feature selection affects performance. First, let's turn on the LV Filter, which will filter out variables with low variance, e.g. if there are 100 samples but 95 of them have the exact same value for a variable, it's probably not that useful a predictor feature.

   a. Set the cv option on line 207 back to cv=5

   b. To turn on the LV Filter, we need to first on line 39 change the lv_filter flag to equal 1 instead of 0

Question #9: Run the code once, record the RMSE and Expl Variance. What do you notice about the scores? How do they compare to the CV performance above in question 7? What features were selected, and which were removed?

Question 9: Results Scores with LV filter and CV=5			
CV Setting	**RMSE**	**Explained Variance**	**Runtime**
cv = 5	0.95 (+/- 0.03)	-0.47 (+/- 0.24)	0.03
Q7	0.90 (+/- 0.10)	-0.31 (+/- 0.17)	0.05

Observations:

- This didn't seem to help improve the scores at all. RMSE still high and Explained Variance very low with a big range.
- **Compare CV performance with Q7:** It's actually even worse than it was in Q7, with the exception of a faster runtime because of the fewer features included.
- **5 Selected Features:** ['fixed acidity', 'residual sugar', 'free sulfur dioxide', 'total sulfur dioxide', 'alcohol']
- **7 Removed Features:** ['Class', 'volatile acidity', 'citric acid', 'chlorides', 'density', 'pH', 'sulphates']

10) Now let's see how a more involved feature selection method affects performance. 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 Decision Tree regressor model for this.

a. Set the lv_filter on line 39 back to lv_filter=0

b. To turn on feature selection, we need to first on line 37 change the feat_select flag to equal 1 instead of 0

c. 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

d. You will need to add a DecisionTreeRegressor() call to pass to the rgr object on line 147, you can use something similar to the calls used elsewhere in the code (e.g. line 191 or 206). Don't forget to set the parameters, particularly the random_state.

e. Note the SelectFromModel() function being called on line 148, this is where the actual feature selection occurs, with the rgr object being passed in

Question #10: Run the code once, record the RMSE and Expl Variance. What do you notice about the scores? How do they compare to the CV performance above in question 7? What features were selected, and which were removed?

Question 10: Results Using Wrapper-Based Feature Selection			
CV Setting	**RMSE**	**Explained Variance**	**Runtime**
Q10	0.91 (+/- 0.08)	-0.35 (+/- 0.26)	0.02
Q7	0.90 (+/- 0.10)	-0.31 (+/- 0.17)	0.05

Observations:

- **I notice and Compare to Q7:** I notice that the scores are not too much different, except that the explained variance has a wider range. Also the runtime is shorter in Q10 because fewer features remain.
- **3 Selected Features:** ['volatile acidity', 'sulphates', 'alcohol']
- **9 Removed Features:** ['Class', 'fixed acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH']

- Interestingly, ‘alcohol’ is the only feature selected by the wrapper method that was also selected by the low variance filter.

Summary Questions

Question #11: Were there any notable differences in performance between the two datasets? Given that one was a classification problem and the other a regression problem, can we say that the Decision Tree model performed better on one of them? Why or why not?

- **Notable differences in performance between the two datasets:**
    - Yes, the Diabetes Dataset was better fit by Decision Trees than the Wine Quality Dataset was.
    - The application of decision trees to the first dataset, ‘Pima_diabetes.csv’, produced Accuracy and AUC scores indicated decent fit of the data. Accuracy Scores were close to 0.71 (+/- 0.08) and AUC of 0.69 (+/- 0.07) was about the same for the best models. For the Wine Quality Dataset on the other hand, RMSE values ended up close to 0.91 (+/- 0.08) and Explained Variance somehow ended up negative at around -0.35 (+/- 0.26) with a big range.
- **Did DT perform better on one of them?**
    - Yes, even though they are different types of problems, when we applied Decision Trees to both, the appropriate metrics indicated poor performance in the Wine Quality dataset and reasonable performance in the Diabetes Dataset.
    - I say poor because Wine Quality dataset had RMSE close to 0.9 and negative explained variance with a with range around -0.35. This means the error was high on the predictions and that the dependent variables variance wasn’t well explained by the independent variables for wine.
    - I say reasonable performance because while 0.71 accuracy isn’t anything to write home about, it at least serves a function and is better than simple guessing at random. It is a good baseline to move on to other methods and compare to see if other methods offer better fit.
    - While Accuracy/AUC and RMSE/Explained Variance are two different sets of metrics for two different types of problems, it is still clear what scores constitute good performance for each respective metric.

Question #12: Based on the results you obtained, would you say that Decision Tree is a “good” model for these two datasets? If your boss or a customer asked you to build a decision tree for one of them, what would you tell him/her?

- I don’t think it is a “good” model for either dataset based on the results that were obtained. It seems to be an “okay” model for the diabetes dataset and a “pretty bad” model for the wine quality dataset. It is possible that I made a mistake somewhere in changing values in the code template, but I went through it a second time in a fresh file and got the same results.

- I would not use a decision tree for either of these, though certainly not for assessing Wine Quality.
- If my boss or a customer asked me to build a decision tree for one of them, I would do so and provide the appropriate metrics for how well the decision tree performed on whatever data I could pull into it. Based on the results, I would make my recommendations on whether it would meet the expectations of my boss. I would also provide alternative models and if one of these models was discovered to be significantly better than a decision tree for a particular task, I would recommend it. As part of my recommendation I would explain clearly why I would recommend it over a decision tree.