Python/scikit-learn skill testing project: How old is your mollusk?

Objective: Explore the abalone dataset (https://archive.ics.uci.edu/ml/datasets/abalone) and determine the ages of samples based on physical measurements that are provided to you. Note: this is a pretty challenging dataset to draw predictions from, since the correlations between inputs and labels is quite weak – but for that reason, it makes a great testing ground for your new machine learning skills!

Background: Abalones are a type of mollusk that can be found in bodies of cold water the world over. If you want to find out how old an abalone is, you have to bore a hole through the thing, and inspect it carefully with a microscope to count the number of ring-like structures within. The number of rings is a strong indicator of the age of the organism: if you can predict the number of rings based on the physical measurements provided, you'll save a lot of biologists a load of time and effort!

How to begin: We've provided you with three modules containing starter code. They are `data_preprocessing.py`, `data_exploration.py` and `predict.py`. In each module, you'll find function headers, along with descriptions of what each function should do (in the docstrings beneath). Your job will be to fill in the "guts" of the functions until you have production-ready code!

Step 1: load and explore the raw data

Start with the **`load_dataset()`** function. Once you have that working, have a look at your raw data by filling out the **`plot_raw_data()`** function in `data_exploration.py`.

Questions:

1. Look at the raw data plots: do all the features in the raw data look like they're correlated to the `Rings` feature? Remember, the correlation doesn't need to be strong – machine learning is designed to find high-level correlations from a large number of weakly correlated features, after all. But if there are features that really don't seem to be correlated to what you're trying to predict (in this case, the value of `Rings`), you might consider leaving them out.

Step 2: normalization and dimensionality reduction

Next, move on to **`preprocess_dataset()`**, making sure that it can convert the output of **`load_dataset()`** into the desired form. After that, complete the **`dimensionality_reduction()`** function. You're now ready to see what hidden correlations your data may contain!

Questions:

1. Run the **`dimensionality_reduction()`** function using the MDS, Isomap and t-SNE algorithms. You can try a few different sets of hyperparameters for each. Do any of these dimensionality reduction techniques show interesting clumps or groupings of your data?
2. Based on what you've seen so far, would you guess that this dataset will be as easy to work with as the Wines dataset we saw earlier in the Python curriculum?
3. Why do you think we asked you to normalize and preprocess your data before visualizing it with the dimensionality reduction algorithms? Why couldn't we just directly use the raw data provided in **`load_dataset()`** for this step?

Step 3: normalization and dimensionality reduction

Now you can fill out **`split_train_and_test()`**, **`split_inputs_and_labels()`**, and finally **`generate_data()`**. This should provide you with a complete input pipeline for your data. You're ready to do some machine learning!

At this point, you might be wondering: why did we decide to reserve exactly 25% of the dataset for testing? There's no hard answer to this question, and in practice there's a pretty big range of acceptable choices for the train/test data ratio. We chose it because it strikes what we felt was a good balance between having enough data to train the algorithms we'll be using, and having a large enough sample of testing data to allow us to predict reasonably confidently how our models will perform "in the wild".

Once you reach this point, send your code to @yazabi and we'll give you feedback on it!

Step 4: build your models

Complete the **`build_model()`** function. This is probably the most challenging function to complete, but once it's done, you'll have trained algorithms that are ready to make predictions!

Questions:

1. There are actually two ways to look at this problem: because the `Rings` parameter can only take on integer values between 1 and 27, you can think of this as a categorization problem (in which case your learning algorithms are answering the question, "which of these 27 age categories does this abalone fit into?"), or a regression problem (in which case it's asking "what is my best estimate of the *age* of this abalone). Note that there are only 27 possible answers in the case of classification, whereas the regression problem can be answered by

any float between 0.0 and 27.0 (and theoretically, higher). Which formulation (classification or regression) would you expect to give the best results?

2. Do you think we can evaluate model performance the same way if we formulate it as a classification task rather than a regression task?

Step 5: evaluate your models

Code up the **`evaluate_model()`** function. As soon as you've finished, try it out! You'll want to see what combination of model architecture (KNNs/naive Bayes/decision trees/SVMs) and hyperparameters performs best. Now a reminder: this dataset is not trivial to use for prediction (in that the label values aren't particularly strongly correlated to the input values), so don't be disappointed if you find that your classification accuracy scores aren't particularly high – just make sure that you use confusion matrices to see how your algorithm performs!

Questions:

1. Based on your initial overview of the data in Steps 1 and 2, do you have reason to think that any particular algorithm (KNN, Bayes, etc.) would be likely to outperform the others?

Quick comment: You're manually scanning through parameter space to optimize your algorithms right now, but there are ways to automate this process in scikit-learn (see http://bit.ly/gridsearchcv). The reason we're not asking you to use this approach is to encourage you to scrutinize the confusion matrices you obtain from your models, and see how they relate to accuracy scores.

Once you reach this point, send your code to @yazabi and we'll give you feedback on it!

Step 6: choose a model and prepare it for production

Head over to the `predict.py` module, and complete the **`save_model()`**, **`load_model()`**, **`build_and_save_model()`** and **`predict()`** functions.

Step 7: run your optimized model

Finally, complete the contents of the `if __name__ == "__main__":` statement. Congratulations on coding a complete machine learning pipeline!

Once you reach this point, send your code to @yazabi and we'll give you feedback on it!