Lab 4: Regression Challenge

This assignment is due to Gradescope by the beginning of lab next week (2:45p on 2/24). You may work with a partner on this lab – if you do, submit only one solution as a "group" on Gradescope.

Introduction

The purpose of this lab is for you to gain additional practice implementing shallow ML pipelines. Unlike Lab 3, this lab will involve a regression problem and will focus on model training and optimization rather than preprocessing.

Provided Files

- `Lab4.pdf`: This file
- `Lab4.py`: Code scaffold
- `movies_X_train.csv`: Training examples
- `movies_y_train.csv`: Training labels
- `movies_X_test.csv`: Test examples (note that test labels are not provided)
- `Lab4_Questions.txt`: Open-ended questions

Instructions

In this lab, you will be attempting to predict the IMDb rating of movies from a number of features. IMDb ratings are float values on a scale of 1-10, so this is a *regression* problem. All essential preprocessing has already been performed on the data set, so you should be able to go directly to model fitting, hyperparameter optimization, and performance evaluation. Specifically, no examples have NaN values, all features are numeric, and all features have been standardized.

Part 1: Baseline

Your first task is to implement the `fit_predict()` function in the provided `Lab4.py` file to have the following behavior:

1. *(Provided)* Load the training examples, training labels, and test examples into local variables `X_train`, `y_train`, and `X_test`, respectively.

2. Train a `LinearRegression` model using the training data, training labels, and default model parameters.

3. Predict and **return** labels for the test data as a Pandas series or NumPy array. There should be as many predicted labels as there are rows in the test data.

Once you have `fit_predict()` working, upload your `Lab4.py` to Gradescope to verify that it works with the leaderboard. **Record** your mean absolute error from the leaderboard into `Lab4_Questions.txt`.

Part 2: Optimization

Now that you have a linear regression baseline, the next step is to improve your regression performance through a more powerful model and automated hyperparameter optimization. Implement the `optimize()` function in `Lab4.py` to have the following behavior:

1. *(Provided)* Load the training examples and training labels into local variables `X_train` and `y_train`.

2. Use `GridSearchCV` to find good hyperparameters for your choice of support vector regression (`SVR`) or decision tree regression (`DecisionTreeRegressor`) models. You will need to tell `GridSearchCV` to use a regression-appropriate scoring metric, such as `"neg_mean_absolute_error"`. **Record** the optimal parameters found by the grid search to `Lab4_Questions.txt`.

3. Print the best hyperparameters found by the grid search.

Once you have run `optimize()` and are satisfied with the hyperparameters it finds, replace the linear regression model in `fit_predict()` with the support vector regressor or decision tree regressor with your optimized hyperparameters. Then upload your `Lab4.py` to Gradescope and **record** your mean absolute error from the leaderboard into `Lab4_Questions.txt`.

Part 3: Random Forests

The next step is to see whether an ensemble method will perform better than a single model. Implement the `forest()` method to perform a hyperparameter optimization grid search for a `RandomForestRegressor` model. **Record** the optimal parameters found by the grid search to `Lab4_Questions.txt`.

Once you have run `forest()` and are satisfied with the hyperparameters it finds, replace the model in `fit_predict()` with a random forest with your optimized hyperparameters. Then upload your `Lab4.py` to Gradescope and **record** your mean absolute error from the leaderboard into `Lab4_Questions.txt`.

Part 4: Evaluation

The last steps are to plot a learning curve to see whether the random forest model is overfitting and to use the random forest to compute feature importances. Implement the `evaluate()` function in `Lab4.py` to have the following behavior:

1. *(Provided)* Load the training examples and training labels into local variables `X_train` and `y_train`.

2. Use the Scikit-Learn `learning_curve` function to compute training and validation scores (mean absolute errors) for your optimized random forest model for increasing numbers of training examples.

3. Plot the training error *and* validation error versus the number of training examples in the same figure. **Save** your plot as an image with name `learning_curve.pdf`.

4. Use the optimized random forest model to calculate and print a ranked list of feature importances. **Record** the five most important features in `Lab4_Questions.txt`.

(Optional) Part 5: Further Iteration

If you would like to keep experimenting (or keep improving your position on the leaderboard...) you can continue to try different models, ensemble methods, and/or hyperparameter searches. Do not modify `optimize()`, `forest()`, or `evaluate()`, but you may modify `fit_predict()` to test new models on the leaderboard.

Deliverables

Submit your final version of `Lab4.py`, `learning_curve.pdf`, and `Lab4_Questions.txt` to Gradescope.

Grading

Your grade will be based on the following:

- 10 pts: `fit_predict()` function
- 10 pts: `optimize()` function
- 10 pts: `forest()` function
- 10 pts: `evaluate()` function
- 5 pts: Learning curve plot
- 5 pts: Code style & clarity
- 10 pts: `Lab4_Questions.txt`

Extra Credit: If your model is in place $p \leq 10$ on the leaderboard when the lab is due next week, you will receive $\frac{11-p}{2}$ extra credit points. For example, if your model has the best performance, you will receive 5 points of extra credit.

Appendix: Data Preprocessing Details

The following preprocessing steps have already been applied to the movies data:

1. Non-U.S. movies, natural language movie descriptions, and movie titles were dropped
2. Missing values in the "budget," "usa_gross_income," and "worldwide_gross_income" columns were replaced with 0.
3. The "year," "month," and "duration" columns were cast to integers
4. The "genre" column was one-hot encoded
5. The data were divided into 80% train and 20% test sets using `train_test_split()`.
6. The "production_company," "director," and "writer" columns were target encoded using a `TargetEncoder` object from the `category_encoders` package that was fit to the train set.
7. All features were standardized using a `StandardScaler` object fit to the train set.

Extra Credit Opportunity

If you find a bug anywhere in this lab, please inform Prof. Apthorpe. The first student(s) to find any particular bug will be given a small amount of extra credit. This will help make the course better for students in future years.