MLDL Assignment #2

2019-11563
자유전공학부 김지원

Problem 1

(a). The total X data is shuffled and split with a ratio of 9:1 using 'train_test_split' function of scikit-learn. The test error of the linear model with least squares was the following (rounded) :

MSE: 1198657.88275
R^2: 0.89139

(b). First, before performing the model fit, the data was scaled to have mean zero and standard deviation of 1 using 'StandardScaler'. Also, 10 fold cross validation was done using 'RidgeCV'. It found that the best λ value is 10. It seems the ridge model performs better on the data. The test error for ridge regression was the following (rounded) :

MSE: 1120786.72628
R^2: 0.89845

(c). Data was scaled before being fit, same as in problem 1(b). After fitting through cross validation, the optimal alpha (λ) chosen by 10-fold cross-validation was 3.70939 (rounded). Also, the test error for lasso regression was the following (rounded) :

MSE: 1172532.98902
R^2: 0.89376

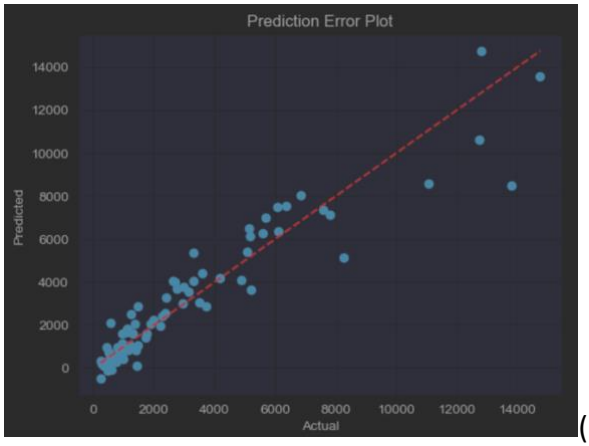
Also, the number of Non-Zero Coefficient Estimates was 14 out of 17 features. The features that were shrinked to zero were the three following: [P.Undergrad, Books, perc.alumni].

(d). Overall, the three models (least squares linear regression, ridge, lasso) performed similarly in terms of MSE and R-squared scores. MSE can be big according to the scale of the Y values so it might be better to determine the model's performance and how accurately it predicted the number of college applications received based on R-squared.

The best performing model was ridge in this data case and it had an R-squared score of 0.89845 which was fairly high in that it was close number to 1 (perfect score). We can also visually see if the three models accurately predicted the Y values by plotting the true and predicted Y values on a scatter plot(below). We can see that the three models performed similarly in terms of prediction and they were pretty accurate as we can see the dots are pretty much aligned along the diagonal red line (which indicates the perfect model).

Speaking about the model itself, however, the three models look different. The coefficients for the three models can be examined to compare models. The linear model is very different from ridge and lasso. This maybe because for ridge and lasso, the values are scaled before fitting the model but linear model doesn't do scaling. Hence the coefficients for linear is very different but ridge and lasso's coefficients doesn't seem to vary too much in comparison. But Lasso used 14 out of 17 features so some of the coefficients were shrinked to zero resulting in a more 'sparse'

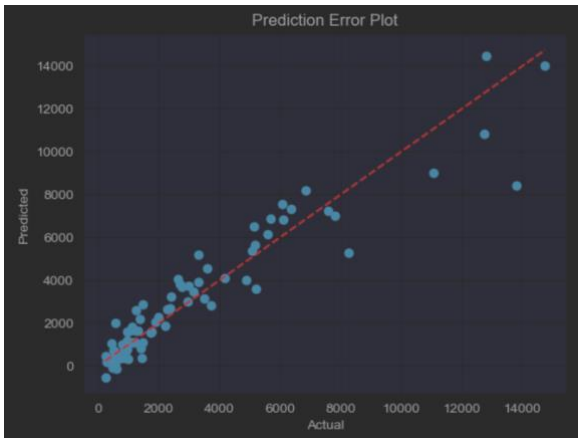
model than linear and ridge. The models are different in that there are differences among coefficients and among the predicted results but the performance of the three models were overall similar.

(1) 

(Linear)

	Coefficient
Accept	1.612270
Enroll	-0.988152
Top10perc	50.702717
Top25perc	-14.944999
F.Undergrad	0.067535
P.Undergrad	-0.007131
Outstate	-0.079386
Room.Board	0.156488
Books	-0.023707
Personal	0.065593
PhD	-11.616929
Terminal	-0.834469
S.F.Ratio	20.313881
perc.alumni	0.293683
Expend	0.072673
Grad.Rate	9.054810
Private_Yes	-580.787882
Intercept	-469.509819

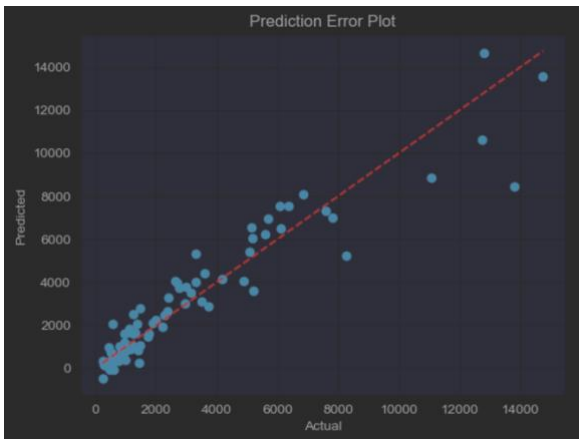
(Linear Coefficients)

(2) 

(Ridge)

	Coefficient
Accept	3595.368450
Enroll	-403.227156
Top10perc	753.566709
Top25perc	-194.044477
F.Undergrad	191.475639
P.Undergrad	-6.519395
Outstate	-258.998096
Room.Board	190.841461
Books	1.117189
Personal	31.979661
PhD	-164.065108
Terminal	-27.169077
S.F.Ratio	76.441710
perc.alumni	-32.386353
Expend	393.656248
Grad.Rate	168.546180
Private_Yes	-260.130314
Intercept	2994.343348

(Ridge Coefficients)

(3) 

(Lasso)

	Coefficient
Accept	3952.397102
Enroll	-738.707705
Top10perc	845.143423
Top25perc	-253.716131
F.Undergrad	195.946965
P.Undergrad	-0.000000
Outstate	-302.010807
Room.Board	165.875757
Books	-0.000000
Personal	38.404074
PhD	-179.099131
Terminal	-12.238365
S.F.Ratio	73.454311
perc.alumni	-0.000000
Expend	377.811847
Grad.Rate	146.369113
Private_Yes	-254.447525
Intercept	2994.343348

(Lasso Coefficients)

Problem 2

(a). First for forward and backward selection, 'SequentialFeatureSelector' was used. This function can perform CV internally. Also, if the 'n_features_to_select' are not mentioned, it is set to 'auto' which stops the iteration at each step if the score isn't incremented by at least a threshold value the function initially holds. The forward and backward selection yielded same results. Both selected ['zn', 'nox', 'dis', 'rad', 'ptratio', 'medv'] features probably because they were highly related to predicting the Y value significantly than the other features.
'n_features_to_select'=6 was optimal when performing cross validation on training data, however fitting on test data, changing the hyper parameter 'n_features_to_select' seemed to work best at 7 after trying out different values. The forward and backward selection selected same 7 features again which were : ['zn', 'nox', 'dis', 'rad', 'tax', 'ptratio', 'medv']. After feature selection, the linear model was fitted using only the selected features. Below are the performance metric for the linear model (rounded) :

'6 features (Auto)' MSE: 29.87285 '7 features' MSE: 29.50254
'6 features (Auto)' R^2: 0.42992 '7 features' R^2: 0.43698

Ridge and lasso models were fitted after scaling the data. Overall, lasso performed slightly better than all the other models in terms of MSE and R^2 scores. The below are the result for ridge and lasso regression (rounded) :

Ridge MSE: 28.95456 Lasso MSE: 28.68358
Ridge R^2: 0.44744 **Lasso R^2: 0.45261**

Examining deeper into the lasso model, based on cross validation on training dataset, the model chose 0.022 for the alpha value, and it yielded the below coefficients where non of the coefficients were shrinked to zero. However, on actual test data, a bit higher value of λ was optimal (0.1). Here, two of the coefficients were shrinked to zero yielding a more sparse model. This seemed to work better on test data due to bias-variance trade-off, it seemed to have lower variance for new data input (MSE: 28.64301, R^2: 0.45339 rounded). As the value of λ got bigger than 0.1 however, it shrinked more features' coefficients to zero, but they seemed to work poorly on test data. Therefore, finding the optimal spot between bias and variance was λ=0.1.

(λ = 0.022)

Number of Non-Zero Coefficient Estimates: 12
Summary of Coefficients:

	Coefficient	Non-Zero
zn	0.969133	True
indus	-0.333405	True
chas	-0.222805	True
nox	-1.147424	True
rm	0.392754	True
age	0.178753	True
dis	-1.988616	True
rad	5.140104	True
tax	-0.275724	True
ptratio	-0.622054	True
lstat	0.611997	True
medv	-2.034418	True
Intercept	3.676738	True

(λ = 0.1)

Number of Non-Zero Coefficient Estimates: 10
Summary of Coefficients:

	Coefficient	Non-Zero
zn	0.726646	True
indus	-0.190490	True
chas	-0.186434	True
nox	-0.508075	True
rm	0.206636	True
age	0.000000	False
dis	-1.394603	True
rad	4.652784	True
tax	-0.000000	False
ptratio	-0.342101	True
lstat	0.603621	True
medv	-1.548871	True
Intercept	3.676738	True

(b). Tried Random Forest Regression and Gradient Boosting method for the dataset. Tuned the hyperparameter using 'GridSearchCV' to find the best estimator. After finding the best model using cross validation, the models were applied on testing sets which yielded results below :

Random Forest - Best Params: {'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 100}
Random Forest - Mean Squared Error: 13.57023, **R^2 Score: 0.74103**
Gradient Boosting - Best Params: {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 50}
Gradient Boosting - Mean Squared Error: 16.61184, **R^2 Score: 0.68299**

Like the shown results, random forest seemed to work best on the test dataset. For random forest, the best model used sqrt(12) features. Therefore as we predicted in 2-(a), sparse models seemed to work best for boston dataset. For gradient boosting, out of [1, 2, 3, 4, 5] depth, max_depth 1 worked the best. We can imply that such slow learning works best for boosting models.

(c). The chosen model 'random forest regressor' doesn't use all the features in the dataset. As mentioned above, the best estimator resulted from cross validation uses 'max_features' = sqrt. It means that out of 12 features, it uses only sqrt(12) features and this results in the best cross validation score. It is because using less features can lead to less variance when applying to new dataset. Too much features can lead to overfitting which might make the model perform worse when little variation within datasets occur. That's why for 2-(a) problem, with the same reason, the lasso model performed best since it doesn't use all the features by shrinking some to zero. So compared with linear and ridge models which fully uses all the features, lasso performed better. And as the model gets sparser (on random forest regressor), it seems to perform better due to variance reduction.