MSDS 422 Assignment 4a Boston Housing

Husein Adenwala

Introduction

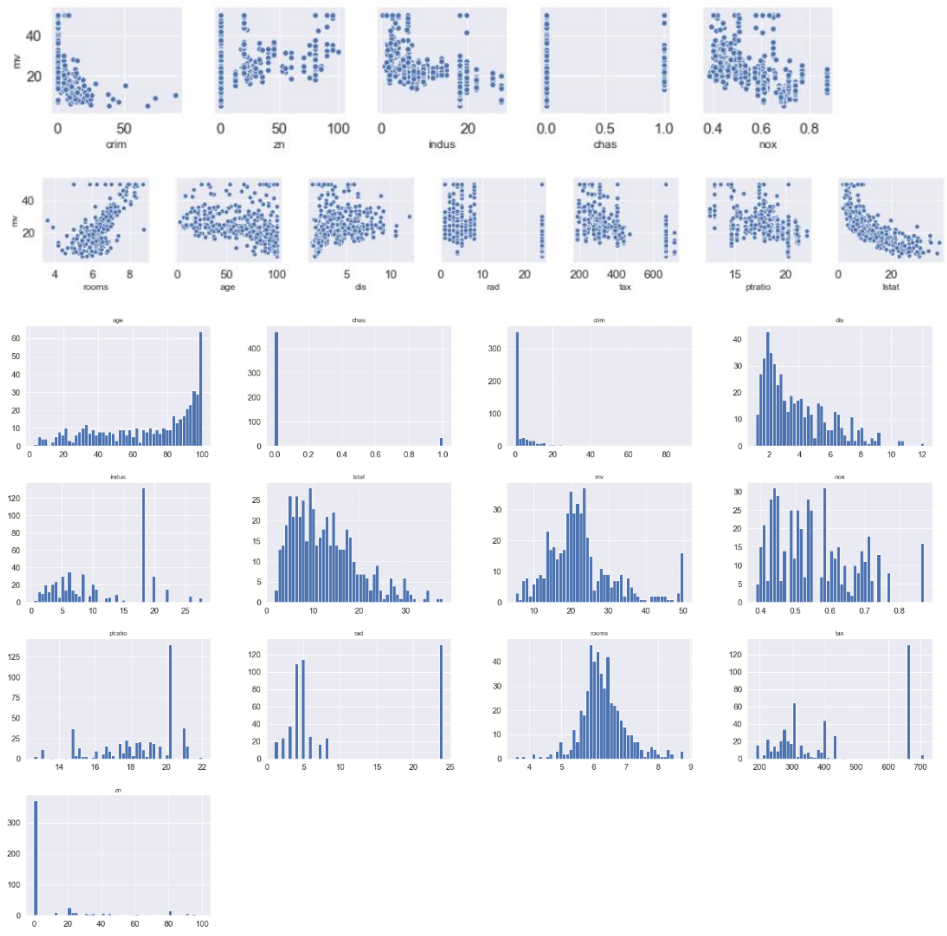
The purpose of this study is to evaluate machine learning regression models for a real estate firm to understand the impact of environmental & other variables and predict the median housing prices in Boston.

Data preparation, exploration, visualization

The Boston Housing Study Data contains 506 observations and 14 features. The data did not have any missing values. For the purpose of this study, neighborhood, a categorical attribute, was removed and all numeric attributes were included.

I created scatter plots and histograms to evaluate all the features from the raw data prior to manually transforming to meet the assumptions of normality and linearity.

From the scatterplot, we can see that some of the distribution, such as lstat, crim, age and indus don't have a linear relationship to the response variable mv. We can also see from the histogram that some of the distributions are skewed, such as age, dis, and lstat.



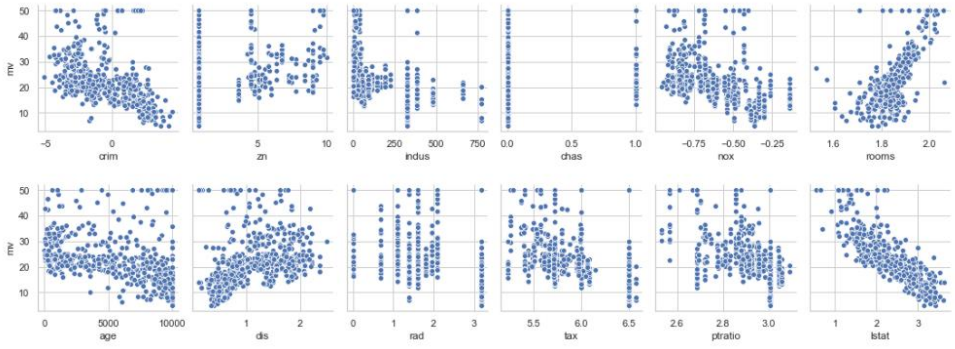
In order to perform linear regression for interpretation, the features have to have more or less normal distribution and a linear relationship to the target variable.

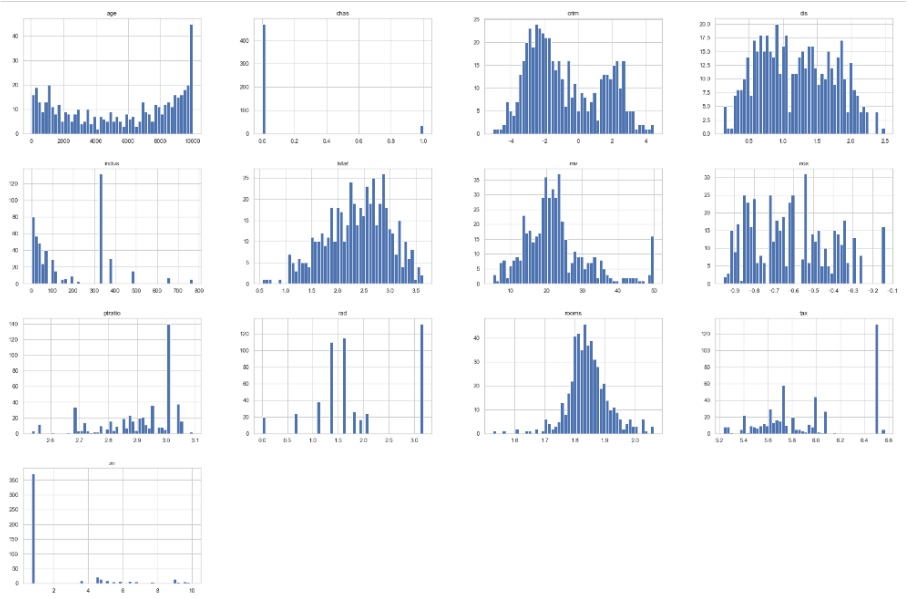
I used natural log, cube root, and square root to transform the features so they would have a linear relationship to target variable MV and a normal distribution if possible:

```
boston_df['crim']= np.log(boston_df['crim'])

boston_df['zn']= (boston_df['zn']+0.5)**(0.5)

boston_df['indus']= (boston_df['indus'])**2

boston_df['nox']= np.log((boston_df['nox']))

boston_df['rooms']= (boston_df['rooms'])**(1/3)

boston_df['age']= (boston_df['age'])**2

boston_df['dis']= np.log(boston_df['dis'])

boston_df['rad']= np.log(boston_df['rad'])#**(0.5)

boston_df['tax']= np.log(boston_df['tax'])

boston_df['ptratio']= np.log(boston_df['ptratio'])

boston_df['lstat']= np.log(boston_df['lstat'])
```

I used trial and error to evaluate if the transformation and scaling were effective. To do this, I created a new set of scatter plots. These showed that the relationship with the response variable mv was closer to linear than before, for example nox, indus, age, dis and lstat but some of the poisson distributions with a high count of 0 could not be transformed to get a normal distribution. It should be noted that the binary features like chas don't need transformation.

The scatterplots shows that all the features are more or less linear and the histograms show that the features (with some exceptions) are more normally distributed than before.





Feature Selection

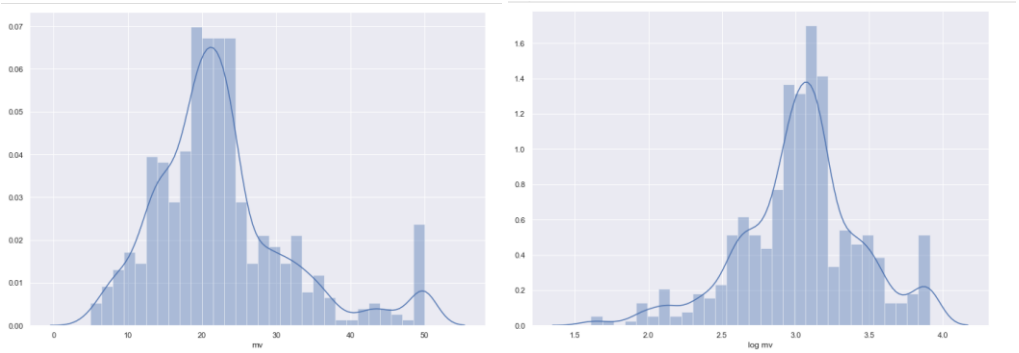
I used all the features in all the models.

crim	zn	indus	chas	nox	rooms	age	dis	rad	tax	ptratio	lstat

It is important to note that some of the features have high collinearity, for example chas and nox.

Target Variable Transformation

I used the transformed and non-transformed target variable mv in all the models for comparison. I used natural log to make mv more normally distributed. The two plots show before and after log transformation, and show that the log transformed variable is slightly more normally distributed and the R squared and RMSE is better for the transformed variable.



Review research design and modeling methods

Standard linear Regression, Random Forest and Extra Tree Regression

For this analysis, I used a regression model because of the continuous nature of our response variable MV and its approximate linear relationships with all predictor variables and Tree based models as they perform well on regression and there are no model assumptions to validate (some of the data is not linear or normal). Specifically, I used standard

Linear, Random Forest and Extra Trees. Each of these models have various characteristics that make them suitable for specific datasets.

Standard linear regression is simple to implement and was the benchmark model. Random Forest and Extra Trees are ensemble methods of Decision Tree that can be more accurate and reduce variance compared to linear regression. Also, they require less effort for data preparation during pre-processing and do not require normalization of data. On the other hand, a small change in the data can lead to a large change in the structure of the optimal decision tree and make it unstable.

The difference between Random Forest and Extra Tress is that Random Forest uses bootstrap replicas, whereas Extra Trees uses the whole original sample by default (which can be changed). Another difference is the selection of cut points in order to split nodes. Random Forest chooses the optimum split while Extra Trees chooses it randomly. However, once the split points are selected, the two algorithms choose the best one between all the subset of features. Therefore, Extra Trees adds randomization but still has optimization.

On one hand, using the whole original sample instead of a bootstrap replica reduces bias. On the other hand, randomly choosing the split point of each node reduces variance.

The Extra Trees algorithm is faster. This algorithm saves time because the whole procedure is the same, but it randomly chooses the split point and does not calculate the optimal one.

To evaluate these models, I used all 12 features and did a 70/30 split on the dataset for Training and Testing respectively. I used the same test and training data on all of the models to do a fair comparison.

I also optimized all the models using hyperparameter tuning.

I also used the k-fold cross validation (k=10) technique to calculate and compare the root mean square error for each model. Additionally, I ran a comparison of R squared and root mean squared error between train and test data as an evaluation metric.

I did a comparison with the log and non-transformed target variable for each of the models and found no difference in the tree-based models; the log transformed target variable performed better in the linear regression on the test data. Therefore, in the evaluation of the models, I will include only the log transformed target variable.

I also calculated the OOB score to compare the Random Forest and Extra Trees models.

Test data with Log mv

```
Linear Regression R_squared =  0.7981115408585668
Linear Regression RMSE =  0.1856107371549965
```

Test data with mv

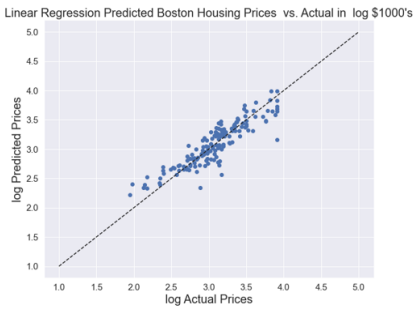
```
Linear Regression R_squared =  0.7895963159795983
Linear Regression RMSE =  4.517479237778285
```

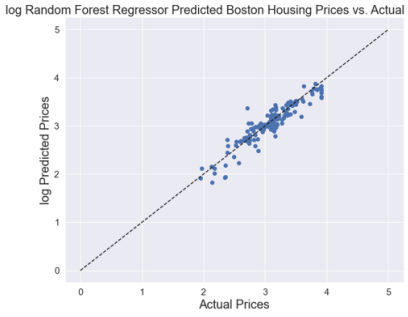
RMSE for log mv is **18%** variation in price of mv

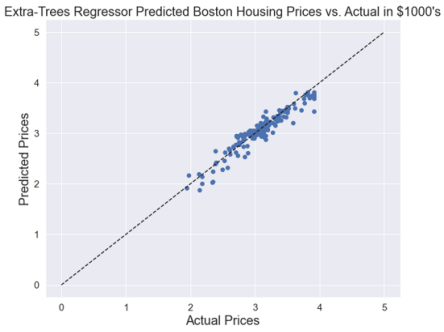
RMSE for mv is 4.51/ 22 (mean price) = **20%**

Review results, evaluate models

The scatter plot for actual vs predicted value on the test data shows that the Random Forest and Extra Trees performed better than the linear regression model.







The R squared and residual mean square error for test and train data is similar for RF and ET. The difference in RMSE for test and train data for RF and ET models suggests there is some over-fitting. Linear regression is slightly better in performance for Test than in Train, but has the highest RMSE value.

The Linear Regression model can explain between 75-80 % of the variation in the mean home price whereas Random Forest and Extra Trees can explain almost 99% variation on train data and 85% and 89% on test data respectively. This indicates overfitting where linear regression is more constant high bias with low variance. The linear regression has an RMSE of approx. 20% on mean house price and Random Forest and Extra Trees have RMSE of 15-13% on mean price on the test data.

	Linear R2	Linear RMSE	Random Fores R2	Random Forest RMSE	Extra Tree R2	Extra Tree RMSE
Train	0.748134	0.202560	0.978143	0.059672	0.981292	0.055205
Test	0.798112	0.185611	0.858960	0.155138	0.894938	0.133897

Mean RMSE using K-fold cross validation on the dataset shows that Extra Trees has the lowest standard error followed by Random Forest and then Linear regression. This is in line with our test and train split comparison and shows that the Extra Tree outperforms the rest of the models:

```
Average results from 10-fold cross-validation
in standardized units (mean 0, standard deviation 1)

Method               Root mean-squared error
Linear Regression          0.216386
Random Forest Regressor    0.182236
Extra Trees Regressor      0.174748
```

Impact of Features on Price

The equation from the linear regression below shows the estimated impact of each feature on the mean housing prices:

Linear Regression: *Y= 5.68 + 0.46*crime -0.12*zn +0.000..*indus +0.1096*chas –0.2686*nox +0.47*rooms + 0.000..*age -0.139*dis + 0.098*rad -0.21*tax -0.5038*ptratio -0.4421*lstat*

(log price = log(mv))

Y Intercept = 5.68 is the log median price of house when other coefficients are zero. This is just a placeholder as it is out of bounds for mean housing price, where all other coefficients cannot be zero.

Crime = 0.46 estimates that if log crime increases by one unit, then log price increases by 0.46 units.

Zn = -0.12 estimates that if sqrt zn increases by one unit, log price decreases by 0.12 units.

Indus < 0.000 (square) has negligible effect on log price.

Chas = 0.1096 estimates that if chas increases by one unit, log price increases by 0.1096 units.

Nox = -0.2686 estimates that if log nox increases by one unit log price decreases by 0.286 units.

Rooms = 0.47 estimates that if cube root room increases by 1 unit, then log price increases by 0.47 units.

Age < 0.000 (log) and has negligible effect on log price.

Dis = -0.139 estimates that if the log dis increases by 1 unit, then the log price decreases by 0.139 units.

Rad = 0.098 estimates that if the log rad increases by 1 unit, then log price increases by 0.098 units.

Tax = -0.21 estimates that if the log tax increases by 1 unit, then log price decreases by 0.21 units.

Ptratio = -0.5038 estimates that if the log ptratio increases by 1 unit, then log price decreases by 0.538 units.

Lstat = -0.4421 estimates that if the log lstat increases by 1 unit, then log price decreases by 0.4421 units.

Random Forest gives us a Feature importance score which indicates the relative score/importance of each feature in the decision tree/model:

```
'crim - 0.101',
'zn - 0.0',
'indus - 0.007',
'chas - 0.002',
'nox - 0.031',
'rooms - 0.197',
'age - 0.02',
'dis - 0.05',
'rad - 0.003',
'tax - 0.019',
'ptratio - 0.007'
'lstat - 0.563']
```

Extra Trees gives us a Feature importance score which indicates the relative score/importance of each feature in the decision tree/model:

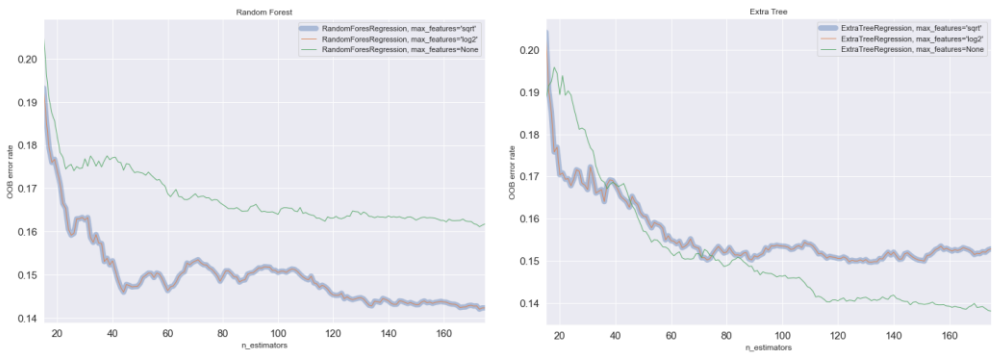
```
'crim - 0.083',
'zn - 0.007',
'indus - 0.039',
'chas - 0.005',
'nox - 0.072',
'rooms - 0.167',
'age - 0.04',
'dis - 0.046',
'rad - 0.021',
'tax - 0.064',
'ptratio - 0.053'
'lstat - 0.402']
```

OOB Score and Error

The OOB score is equivalent to the R squared, which shows that the ET performs slightly better than the RF.

Random_fores oob scoret	Extra Tree oob score
0.836477	0.858719

The graph shows the relationship between the number of estimators and the OOB error:



K-fold Validation

The K-fold Validation shows that ET has the lowest RMSE.

```
Average results from 10-fold cross-validation
in standardized units (mean 0, standard deviation 1)

Method               Root mean-squared error
Linear Regression          0.216386
Random Forest Regressor    0.182236
Extra Trees Regressor      0.174748
```

Implementation and programming

Using Python Pandas package, I loaded the Boston housing market data csv file to the Data Frame object, then obtained a new subset Data Frame by dropping the Neighborhood features.

Exploratory data analysis was done utilizing the function in the Pandas Numpy, seaborn and matplotlib packages.

I then created two numpy arrays, (y) and natural log (y), for response variable mv and removed them from the Boston Data Frame in order to compare model performance with log (y) and (y).

I then used python built-in math and NumPy methods to do the manual variable transformation on all the remaining features in the Data Frame.

I selected all the features which were then put into a numpy array (X).

At this point, I created a 70/30 train and test split of the Numpy array (X: response variable) and Numpy array (Y: features), then used the linear regression function in scikit learn package to create a linear regression model and then evaluated RMSE and R squared for train and test data.

I used the Linear Regression function from scikit learn to create the linear regression model.

Then I used the RandomForest Regressor and ExtraTree Regressor function from scikit learn to create the Random forest and Extra Trees models and evaluated RMSE and R squared for train and test data.

I used the GridSearchCV package from scikit-learn to do hyper-parameter optimization for all of the models and applied to all the models.

I used matplot lib from scikit learn make_classification and OrderDirect package to create OOB errors and accuracy plot for Random Forest and Extra Trees.

Finally, I did K-Fold cross-validation (k=10) using the K-Fold package in scikit-learn. To do this, I created a numpy array of all the data for all 12 response variables and calculated the mean RMSE of the 10 folds for comparison.

Exposition, problem description and management recommendations

After evaluating all three models, my recommendation to the management is to utilize the Extra Trees regression model. Both Extra Trees and Random Forest outperform linear regression, but Extra Trees has the overall best performance and is a better fit for this dataset.

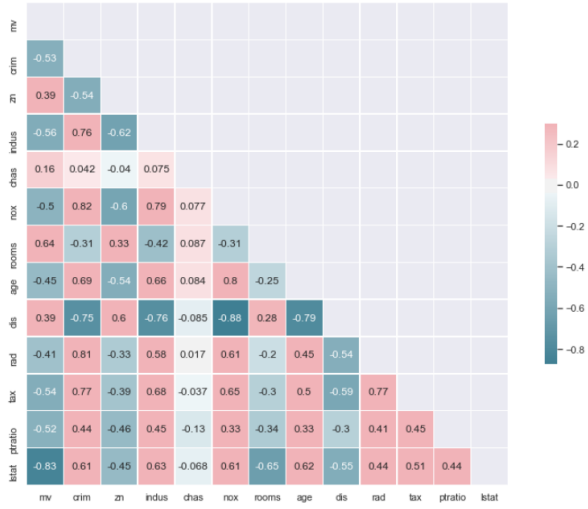
The R squared and RMSE for test vs train shows that there is overfitting in the Random Forest and Extra Tree models, but they perform well compared to linear regression, even though linear regression has low variance. However, with only approximately 500 records, K-fold validation (k=10) is a better technique to evaluate the model fit. Based on the root mean square error calculated by cross validation, I found that the RMSE was lowest for Extra Tree, followed by Random Forest and then standard linear regression. Extra Trees seems to keep a higher performance compared to Random Forest in presence of noisy features and based on our EDA, we have included features that have

multicollinearity and we have not done any feature selection which makes the data noisy (see Appendix for corrplot). Also from the ET model we can get a feature importance score that shows the relative impact of each feature in the model.

All the reasons mentioned above make a strong case for using Extra Tree regression for predicting the median housing mean price and understanding how important a feature is in affecting the house price.

Appendix

Correlation plot



K-FOLD Cross-Validation.

```
# Establish number of cross folds employed for cross-validation
N_FOLDS = 10

# Setup numpy array for storing results
cv_results = np.zeros((N_FOLDS, len(names)))
cv_r2 =np.zeros((N_FOLDS, len(names)))
# Initiate splitting process
kf = KFold(n_splits = N_FOLDS, shuffle=False, random_state = RANDOM_SEED)

# Check the splitting process by looking at fold observation counts
index_for_fold = 0  # Fold count initialized
for train_index, test_index in kf.split(model_data):
    print('\nFold index:', index_for_fold, '----------------------------------------------------------------------------')

# The structure of modeling data for this study has the response variable coming first and explanat
# so 1:model_data.shape[1] slices for explanatory variables and 0 is the index for the response var
    x_train = model_data[train_index, 1:model_data.shape[1]]
    x_test = model_data[test_index, 1:model_data.shape[1]]
    y_train = model_data[train_index, 0]
    y_test = model_data[test_index, 0]

    index_for_method = 0  # Method count initialized
    #index_r = 0
    for name, reg_model in zip(names, regressors):
        reg_model.fit(x_train, y_train)  # Fit on the train set for this fold

        # Evaluate on the test set for this fold
        y_test_predict = reg_model.predict(x_test)
        fold_method_result = accuracy_score(y_test, y_test_predict)
        #R_squared = reg_model.score(X_test,y_test)
        cv_results[index_for_fold, index_for_method] = fold_method_result

        #cv_r2[index_r, index_for_method]= R_squared
        index_for_method += 1
        #index_r += 1

    index_for_fold += 1

cv_results_df = pd.DataFrame(cv_results)
cv_results_df.columns = names
cv_r2df= pd.DataFrame(cv_r2)
cv_r2df.columns = names
print('\n----------------------------------------------------------------------------------------')
print('Average results from ', N_FOLDS, '-fold cross-validation\n',      'in standardized units (me
print(cv_results_df.mean())
print(cv_results_df)
```

```
Fold index: 0 ----------------------------------------------------------------------------

Fold index: 1 ----------------------------------------------------------------------------

Fold index: 2 ----------------------------------------------------------------------------

Fold index: 3 ----------------------------------------------------------------------------

Fold index: 4 ----------------------------------------------------------------------------

Fold index: 5 ----------------------------------------------------------------------------

Fold index: 6 ----------------------------------------------------------------------------

Fold index: 7 ----------------------------------------------------------------------------

Fold index: 8 ----------------------------------------------------------------------------

Fold index: 9 ----------------------------------------------------------------------------

----------------------------------------------------------------------------------------
Average results from 10-fold cross-validation
in standardized units (mean 0, standard deviation 1)

Method    accuracy score
Logistic          0.823845
RandomForest      0.821573
ExtraTree         0.829438
```

OOB error score plot

```
# Map a classifier name to a list of (<n_estimators>, <error rate>) pairs.

from collections import OrderedDict
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

# Author: Kian Ho <hui.kian.ho@gmail.com>
#         Gilles Louppe <g.louppe@gmail.com>
#         Andreas Mueller <amueller@ais.uni-bonn.de>
#
# License: BSD 3 Clause

print(__doc__)

RANDOM_STATE = 123

#fig size
fig_dims = (15, 10)
fig, ax = plt.subplots(figsize=fig_dims)


# NOTE: Setting the `warm_start` construction parameter to `True` disables
# support for parallelized ensembles but is necessary for tracking the OOB
# error trajectory during training.
ensemble_clfs = [
    ("ExtraTreeClassification, max_features='sqrt'",
       ExtraTreesClassifier(warm_start=True, oob_score=True,
                            max_features="sqrt",
                            random_state=RANDOM_STATE,bootstrap=True)),
    ("ExtraTreeClassification, max_features='log2'",
         ExtraTreesClassifier(warm_start=True, max_features='log2',
                              oob_score=True,
                              random_state=RANDOM_STATE,bootstrap=True)),
    ("ExtraTreeClassification, max_features=None",
         ExtraTreesClassifier(warm_start=True, max_features= None,
                              oob_score=True,
                              random_state=RANDOM_STATE,bootstrap=True))
]

# Map a classifier name to a list of (<n_estimators>, <error rate>) pairs.
error_rate = OrderedDict((label, []) for label, _ in ensemble_clfs)

# Range of `n_estimators` values to explore.
min_estimators = 15
max_estimators = 175

for label, clf in ensemble_clfs:
    for i in range(min_estimators, max_estimators + 1):
        clf.set_params(n_estimators=i)
        clf.fit(x, y)

        # Record the OOB error for each `n_estimators=i` setting.
        oob_error = 1 - clf.oob_score_
        error_rate[label].append((i, oob_error))

# Generate the "OOB error rate" vs. "n_estimators" plot.
for label, clf_err in error_rate.items():
    al= 1
    ln = 1
    if label == "ExtraTreeClassification, max_features='sqrt'":
        ln = 8.0
        al = 0.4

    xs, ys = zip(*clf_err)
    plt.plot(xs, ys, label=label, alpha= al, linewidth = ln)




plt.xlim(min_estimators, max_estimators)
plt.xlabel("n_estimators")
plt.ylabel("OOB error rate")
plt.legend(loc="upper right")
plt.title("Extra Tress")
plt.show()
```

```
Automatically created module for IPython interactive environment
```