Predictive Modeling

Series 6

Exercise 6.1

In this exercise, we will look at the `College` data set which tracks demographic characteristics of college applications in the USA. The data set has the following structure:

- `Private`: if the applicant is from a public or private high school
- `Apps`: number of applications received
- `Accept`: number of applicants accepted
- `Enroll`: number of new students enrolled
- `Top10perc`: new students from top 10% of high school class
- `Top25perc`: new students from top 25% of high school class
- `F.Undergrad`: number of full-time undergraduates
- `P.Undergrad`: number of part-time undergraduates
- `Outstate`: out-of-state tuition
- `Room.Board`: room and board costs
- `Books`: estimated book costs
- `Personal`: estimated personal spending
- `PhD`: percent of faculty with PhD's
- `Terminal`: percent of faculty with terminal degree
- `S.F.Ratio`: student/faculty ratio
- `perc.alumni`: percent of alumni who donate
- `Expend`: instructional expenditure per student
- `Grad.Rate`: graduation rate

You can read in the data by means of

[1]:
```
import pandas as pd
college = pd.read_csv('college.csv', index_col=0)
college.head()
```

Our goal is to predict the number of applications (`Apps`) received by using the other variables in the dataset.

a) Split the data set into a training and a test set. **Hint:** Use the `sklearn.model_selection.train_test_split()` function.

b) Fit a linear model using least squares and best subset selection on the training set, and report on the test error obtained. **Hint:** Use `statsmodels` to fit an OLS with R like summaries

c) Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report on the test error obtained.

d) Fit a lasso model on the training set, with λ chosen by cross-validation. Report on the test error obtained, along with the number of non-zero coefficient estimates. **Hint:** Use `RidgeCV` and `LassoCV` from `scikit-learn` to have a cross-validated regression function

Exercise 6.2

In this exercise, we will examine the `Boston` data set which contains housing values in 506 suburbs of Boston and a variety of census values. The dataset is constructed as follows:

- `crim`: per capita crime rate by town
- `zn`: proportion of residential land zoned for lots over 25'000 square feet
- `indus`: proportion of non-retail business acres per town
- `chas`: Charles River dummy variable (=1 if tract bounds river, 0 otherwise)
- `nox`: nitrogen oxides concentration (parts per 10 million)
- `rm`: average number of rooms per dwelling
- `age`: proportion of owner-occupied units built prior to 1940
- `dis`: weighted mean of distances to five Boston employment centres
- `rad`: index of accessibility to radial highways
- `tax`: full-value property-tax rate per $10,000
- `ptratio`: pupil-teacher ratio by town
- `lstat`: lower status of the population (percent)
- `medv`: median value of owner-occupied homes in $1000s

You can read in the data by means of

[1]:
```
import pandas as pd
boston = pd.read_csv('boston.csv', index_col=0)
boston.head()
```

We will try to predict the per capita crime rate (`crim`) in the data using the other variables as predictor variables.

a) Split the data set into a training and a test set. **Hint:** Use the `sklearn.model_selection.train_test_split()` function.

b) Fit a linear model using least squares and best subset selection on the training set, and report on the test error obtained.

c) Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report on the test error obtained.

d) Fit a Lasso model on the training set, with λ chosen by cross-validation. Report on the test error obtained, along with the number of non-zero coefficient estimates.

Exercise 6.3

In this exercise, we will take a look at some model agnostic feature importance metrics for the `College` dataset

a) Plot the Permutation Feature Importance for each variable on the training and test set

b) Plot the Partial Dependence Plot for each variable on the training and test set

c) Plot the Accumulated Local Effects for each variable on the training and test set

d) Analyse the various feature importance metrics, if and why they highlight different features as significant

Hint: Partial Dependence and Permutation Feature Importance can be done using `scikit-learn`, and ALE with the `alibi` package

Exercise 6.4

In this exercise, we will have look at a toolbox called `PyCaret` which can be used to test a variety of models (classification, regression, and anomaly detection) on a dataset and get an intuition about which models may be followed up.

Carry out a regression analysis on the `College` data set with the target variable `Apps` on the training set, and compare your results especially with respect to the feature importance on the training and test data set.

Result Checker

Predictive Modeling

Solutions to Series 6

Solution 6.1

a) We first encode the predictor variable **`Private`** as dummy variable.

[1]:
```
from sklearn.model_selection import train_test_split
college['Private'] = college['Private'].map({'Yes':1,'No':0})

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    college[['Accept', 'Enroll', 'Top10perc', 'Top25perc', 'F.Undergrad',
       'P.Undergrad', 'Outstate', 'Room.Board', 'Books', 'Personal',
       'PhD', 'Terminal', 'S.F.Ratio', 'perc.alumni', 'Expend',
       'Grad.Rate', 'Private'
    ]],
    college[['Apps']]
)
```

b) As there is no built-in algorithm for Best Subset Selection, we first need to program the Best Subset Selection algorithm. We choose the best subset of predictor variables on the basis of the (lowest) AIC value.

[1]:
```
import operator
import time
import itertools
import statsmodels.api as sm

def processSubset(feature_set, X, y):
    # Fit model on feature_set and calculate RSS
    model = sm.OLS(y,X[list(feature_set)])
    regr = model.fit()
    rss = regr.ssr
    aic = regr.aic
    return {'model':regr, 'RSS':rss, 'AIC':aic}

def getBest(X, y, k):
    tic = time.time()

    results = []

    for combo in itertools.combinations(X.columns, k):
        results.append(processSubset(combo, X, y))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)
```

```
    # Choose the model with the lowest AIC
    best_model = models.loc[models['AIC'].argmin()]

    toc = time.time()
    print(
        "Processed", models.shape[0], "models on", k,
        "predictors in", (toc-tic), "seconds."
        )

    # Return the best model, along with some other
    # useful information about the model
    return best_model

models_best = pd.DataFrame(columns=['RSS', 'AIC', 'model'])

start = time.time()
for i in range(1,18):
    models_best.loc[i] = getBest(X_train, y_train, i)

end = time.time()
print("Total elapsed time:", (end-start), "seconds.")
print(models_best)
```

The model with the lowest AIC value contains the following 11 features:

$$\begin{aligned}\texttt{Apps} = {} & 1.6991 \cdot \texttt{Accept} - 0.9403 \cdot \texttt{Enroll} \\ & + 33.788 \cdot \texttt{Top10perc} + 0.0570 \cdot \texttt{P.Undergrad} \\ & - 0.0720 \cdot \texttt{Outstate} + 0.1196 \cdot \texttt{Room.Board} \\ & + 0.4477 \cdot \texttt{Books} - 14.8977 \cdot \texttt{Terminal} \\ & + 0.0576 \cdot \texttt{Expend} + 5.7741 \cdot \texttt{Grad.Rate} - 623.6925 \cdot \texttt{Private}\end{aligned}$$

To compute the performance on the test set, we predict **Apps** on the test set and and subsequently determine the mean squared error.

[1]:
```
best_model_res = models_best.loc[11, 'model']
y_pred = best_model_res.predict(
    X_test[[
        'Accept', 'Enroll', 'Top10perc', 'P.Undergrad', 'Outstate',
        'Room.Board', 'Books', 'Terminal', 'Expend',  'Grad.Rate', 'Private'
    ]]
)
print((sum((y_test['Apps'] - y_pred) ** 2)))

## 275996505.4282418
```

c) The fitting of the Ridge regression can be carried out as follows

[1]:
```
alphas = 10**(np.linspace(-4,2,100))
ridge = RidgeCV(alphas=alphas, cv=5).fit(X_train, y_train)

# Calculate MSE
y_pred = ridge.predict(X_test)
print('{:.6E}'.format(sum((y_test['Apps'] - y_pred[:,0]) ** 2)))
```

Which gives us a chosen lambda of 14.175 and an mean squared error of $2.656107E+08$ on the test set.

d) Similarly, a Lasso model can be fitted

[1]:
```
lasso = LassoCV(cv=5).fit(X_train, y_train['Apps'])
print(lasso.alpha_)
print(lasso.coef_)

y_pred = lasso.predict(X_test)
print('{:.6E}'.format(sum((y_test['Apps'] - y_pred) ** 2)))
```

which results in a lambda of 25138.772 and the following non-zero coefficients

- **Accept**: 1.5103
- **F.Undergrad**: -0.0308
- **Outstate**: -0.0319
- **Room.Board**: 0.03990
- **Expend**: 0.09498

with a test mean squared error of $3.13304450E+08$

Solution 6.2

a) Split the data in training and test set

[1]:
```
X_train, X_test, y_train, y_test = train_test_split(
    boston[[
        'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad',
        'tax', 'ptratio', 'lstat', 'medv'
    ]],
    boston[['crim']]
)
```

b) To find the best subset, we proceed as follows

[1]:
```
models_best = pd.DataFrame(columns=['RSS', 'AIC', 'model'])

start = time.time()
for i in range(1,13):
    models_best.loc[i] = getBest(X_train, y_train, i)

end = time.time()
print("Total elapsed time:", (end-start), "seconds.")
```

The best model we obtain has 6 features, an AIC of 2374 and a formula of

$$\begin{aligned}\texttt{crim} = 0.0407 \cdot \texttt{zn} + -0.0889 \cdot \texttt{indus} \\ + -0.5059 \cdot \texttt{dis} + 0.4783 \cdot \texttt{rad} \\ + 0.2091 \cdot \texttt{lstat} + -0.0645 \cdot \texttt{medv}\end{aligned}$$

To calculate the test mean squared error we predict the test set and calculate the sum of the squared residuals:

[1]:
```
best_model_res = models_best.loc[6, 'model']
y_pred = best_model_res.predict(
    X_test[['zn', 'indus', 'dis', 'rad', 'lstat', 'medv']]
    )
print('{:.6E}'.format(sum((y_test['crim'] - y_pred) ** 2)))
```

Which results in a SSR of $8.4733E+03$

c) The fitting of the Ridge regression is carried out as follows

[1]:
```
alphas = 10**(np.linspace(-4,2,100))
ridge = RidgeCV(alphas=alphas, cv=5).fit(X_train, y_train)

# Calculate MSE
y_pred = ridge.predict(X_test)
print('{:.6E}'.format(sum((y_test['crim'] - y_pred[:,0]) ** 2)))
```

Which gives us a chosen lambda of 100.0 and an mean squared error of $2.279576E+03$ on the test set.

d) We fit a Lasso model to the training data set:

[1]:
```
from sklearn.linear_model import LassoCV
lasso = LassoCV(cv=5).fit(X_train, y_train)
print(lasso.alpha_)
print(list(zip(lasso.coef_, X_train)))

y_pred = lasso.predict(X_test)
print('{:.6E}'.format(sum((y_test['crim'] - y_pred) ** 2)))
```

which results in a lambda of 0.902854504341 and the following non-zero coefficients

- **zn**: 0.03500
- **age**: 0.02116
- **dis**: -0.05604
- **rad**: 0.56388
- **lstat**: 0.11457
- **medv**: -0.12797

with a mean squared test error of $2.248840E+03$

Solution 6.3

a) Permutation Feature Importance. First, we read in the data set **College**

[1]:
```
import pandas as pd
college = pd.read_csv('./data/college.csv', index_col=0)
college.head()
from sklearn.model_selection import train_test_split
college['Private'] = college['Private'].map({'Yes':1,'No':0})

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    college[['Accept', 'Enroll', 'Top10perc', 'Top25perc', 'F.Undergrad',
       'P.Undergrad', 'Outstate', 'Room.Board', 'Books', 'Personal',
       'PhD', 'Terminal', 'S.F.Ratio', 'perc.alumni', 'Expend',
       'Grad.Rate', 'Private'
    ]],
    college[['Apps']]
)
```

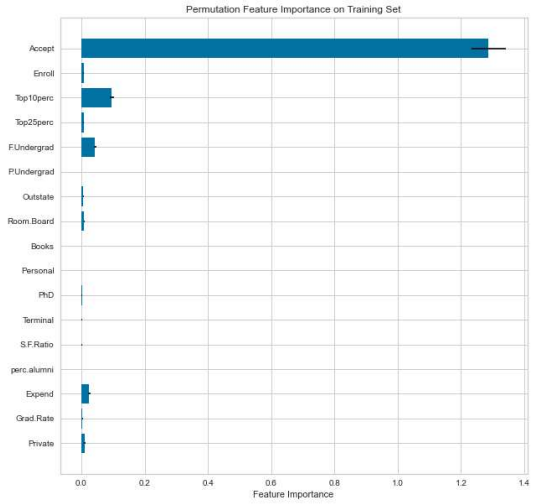
[1]:
```
from sklearn.inspection import permutation_importance
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import numpy as np

r = permutation_importance(
    lr, X_train, y_train, n_repeats=30, random_state=0
)
for i in r.importances_mean.argsort()[::-1]:
    if r.importances_mean[i] - 2 * r.importances_std[i] > 0:
        print(f"{X_train.columns[i]:<8}"
            f"{r.importances_mean[i]:.3f}"
            f" +/- {r.importances_std[i]:.3f}")
```

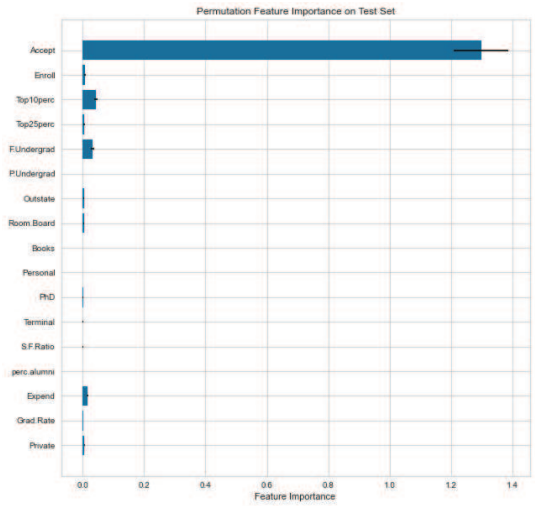
```
fig, ax = plt.subplots()
y_pos = np.arange(len(r.importances_mean))
ax.barh(y_pos, r.importances_mean, xerr=r.importances_std, align='center')
plt.yticks(y_pos, labels=X_train.columns)
ax.invert_yaxis()  # labels read top-to-bottom
ax.set_xlabel('Feature Importance')
ax.set_title('Permutation Feature Importance')
plt.show()

rt = permutation_importance(
    lr, X_test, y_test, n_repeats=30, random_state=0
)
for i in rt.importances_mean.argsort()[::-1]:
    if rt.importances_mean[i] - 2 * rt.importances_std[i] > 0:
        print(f"{X_train.columns[i]:<8}"
            f"{rt.importances_mean[i]:.3f}"
            f" +/- {rt.importances_std[i]:.3f}")

fig, ax = plt.subplots()
y_pos = np.arange(len(rt.importances_mean))
ax.barh(y_pos, rt.importances_mean, xerr=rt.importances_std, align='center')
plt.yticks(y_pos, labels=X_train.columns)
ax.invert_yaxis()  # labels read top-to-bottom
ax.set_xlabel('Feature Importance')
ax.set_title('Permutation Feature Importance')
plt.show()
```



(a) Permutation Feature Importance on the training set



(b) Permutation Feature Importance on the test set

b) Partial Dependence Plots

[1]:
```
from sklearn.linear_model import LinearRegression
from sklearn.inspection import PartialDependenceDisplay

feature_names = [
    'Accept', 'Enroll', 'Top10perc', 'Top25perc', 'F.Undergrad',
    'P.Undergrad', 'Outstate', 'Room.Board', 'Books', 'Personal',
    'PhD', 'Terminal', 'S.F.Ratio', 'perc.alumni', 'Expend',
    'Grad.Rate', 'Private'
]
lr = LinearRegression().fit(X_train, y_train)
# change X to get PDP for test set
disp = PartialDependenceDisplay.from_estimator(lr,
```

```
plt.tight_layout()
# Adjust the spacing between subplots, so that feature name can be seen
plt.subplots_adjust(
    left  = 0.125,
    right = 0.9,
    bottom = 0.1,
    top = 0.9,
    wspace = 0.2,
    hspace = 0.8
)
plt.show()
```

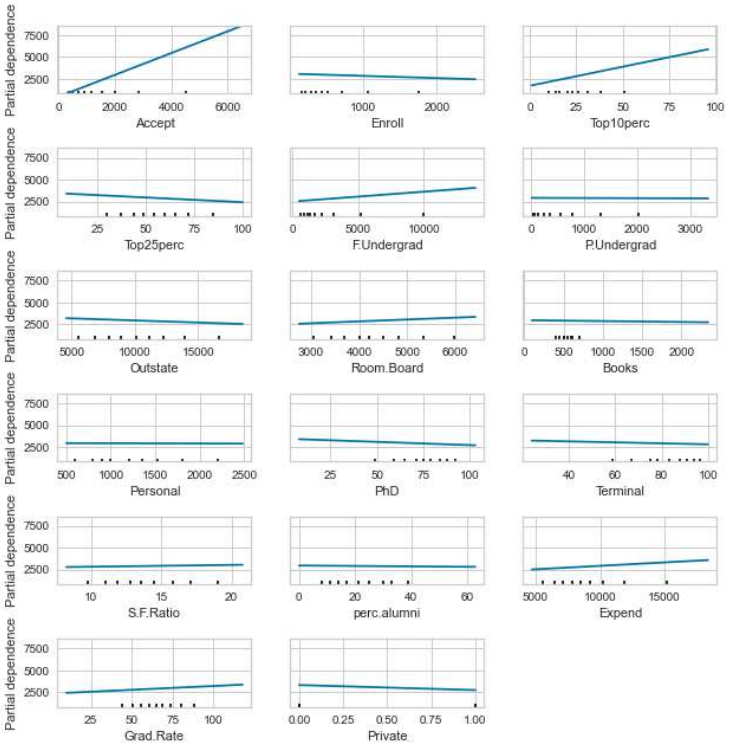


Figure 2: Partial Dependence Plot on the training set

[1]c)
```
from alibi.explainers import ALE, plot_ale

# ALE can't handle DataFrames so we pass raw numpy arrays
lr = LinearRegression().fit(np.array(X_train), np.array(y_train))
ale = ALE(lr.predict, feature_names=feature_names, target_names=['Apps'])
# change X_train to X_test to get ALE for test set
exp = ale.explain(np.array(X_train))
plot_ale(exp)
plt.show()
```

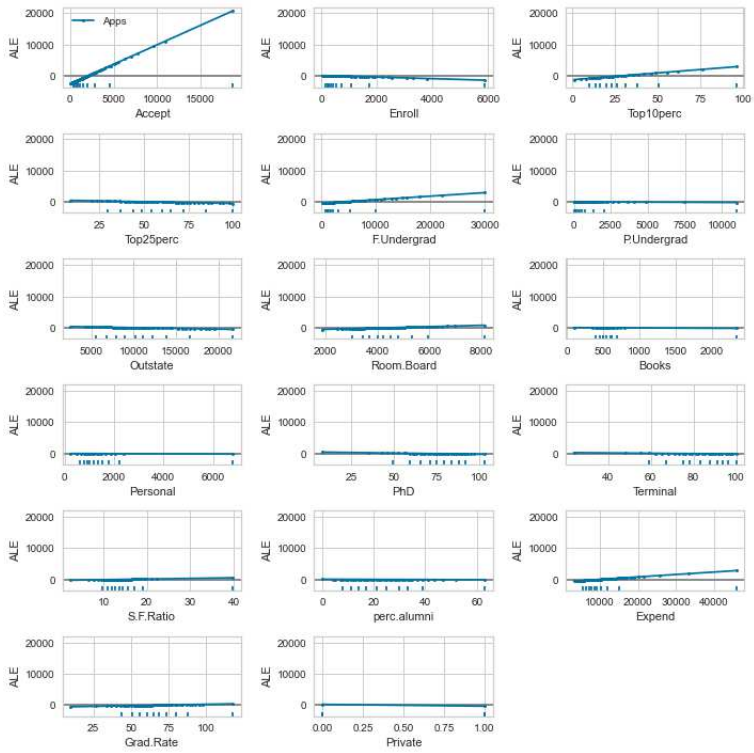


Figure 3: Accumulated Local Effects on the training set

As we can see, all agnostic metrics correctly identify the importance of **`Accept`**, **`Top10perc`**, **`F.Undergrad`** and **`Expend`**

Solution 6.4 Initialisiation of PyCaret is straightforward

[1]:
```
from pycaret.regression import *
import pandas as pd
college = pd.read_csv('./data/college.csv', index_col=0)
college.head()
from sklearn.model_selection import train_test_split
college['Private'] = college['Private'].map({'Yes':1,'No':0})
pcr = setup(data = college,  target = 'Apps')
```

By calling **`compare_models`** we train all default regressors on the dataset and save them in **`best_model`**

[1]:
```
best_model = compare_models()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
et	Extra Trees Regressor	425.5770	710864.5822	823.7219	0.9378	0.1850	0.1475	0.0780
gbr	Gradient Boosting Regressor	453.0638	827757.6899	888.1948	0.9253	0.2333	0.1817	0.0370
rf	Random Forest Regressor	466.2524	929253.1082	944.1030	0.9166	0.1900	0.1482	0.1120
lightgbm	Light Gradient Boosting Machine	475.7932	972581.9330	951.8362	0.9111	0.2402	0.1799	0.0210
llar	Lasso Least Angle Regression	620.4028	1136655.9483	1029.6645	0.8957	0.6177	0.4095	0.0070
lr	Linear Regression	645.9120	1144768.9720	1040.7789	0.8940	0.6413	0.4413	0.5990
lasso	Lasso Regression	645.4906	1144776.5274	1040.7037	0.8940	0.6608	0.4403	0.2350
ridge	Ridge Regression	645.4975	1144739.2762	1040.6871	0.8940	0.6630	0.4404	0.0080
en	Elastic Net	639.2949	1160666.7103	1046.1484	0.8922	0.6264	0.4195	0.0100
ada	AdaBoost Regressor	848.2179	1352570.8183	1154.2613	0.8743	0.6459	0.8476	0.0290
br	Bayesian Ridge	614.9258	1358211.8993	1120.8323	0.8707	0.4769	0.3785	0.0070
huber	Huber Regressor	504.3058	1654480.9778	1159.0349	0.8617	0.2587	0.1543	0.0150
omp	Orthogonal Matching Pursuit	597.5328	1656559.1207	1171.4062	0.8607	0.2928	0.2580	0.0060
dt	Decision Tree Regressor	618.0649	1834101.3288	1305.5342	0.8391	0.2611	0.1958	0.0090
par	Passive Aggressive Regressor	862.7446	2068920.3022	1360.1362	0.8126	0.6556	0.5857	0.0080
knn	K Neighbors Regressor	715.3817	1846506.2942	1315.3372	0.8088	0.4526	0.4249	0.0110
lar	Least Angle Regression	977.1363	3220541.7408	1567.2629	0.6175	0.7115	0.6573	0.0070
dummy	Dummy Regressor	2342.4602	12008440.0179	3414.5232	-0.0262	1.1818	2.1425	0.0060

Figure 4: Output of compare_models from PyCaret

Once we trained the best regressor (the extra tree regressor) separately, then we may evaluate the model by using

[1]:
```
et = create_model('et')
evaluate_model(et)
```

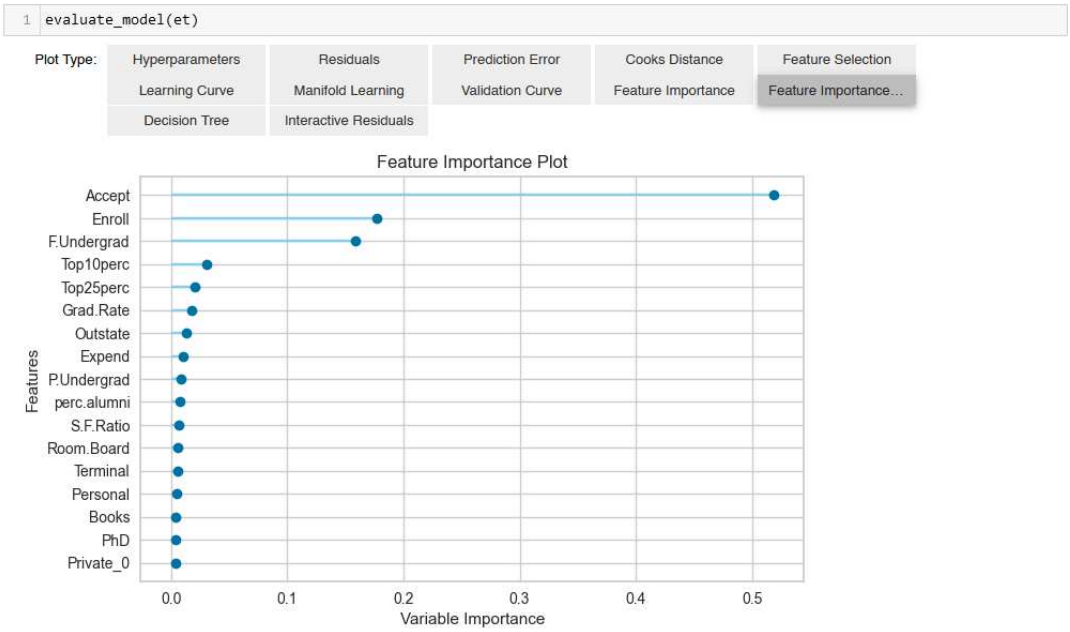


Figure 5: PyCaret evaluation on the Extra Tree Regressor.