Chapter 6: Multiple Linear Regression

Data Mining for Business Analytics in Python

Shmueli, Bruce, Gedeck & Patel

 rev 10/22/19

We assume a linear relationship between predictors and outcome:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon,$$

outcome

coefficients

constant

predictors

error (noise)

Topics

- Explanatory vs. predictive modeling with regression
- Example: prices of Toyota Corollas
- Fitting a predictive model
- Assessing predictive accuracy
- Selecting a subset of predictors

Explanatory Modeling

Goal: Explain relationship between predictors (explanatory variables) and target

- Familiar use of regression in data analysis
- Model Goal: Fit the data well and understand the contribution of explanatory variables to the model
- “goodness-of-fit”: R^2, residual analysis, p-values

Predictive Modeling

Goal: predict target values in other data where we have predictor values, but not target values

- Classic data mining context
- Model Goal: Optimize predictive accuracy
- Train model on training data
- Assess performance on validation (hold-out) data
- Explaining role of predictors is not primary purpose (but useful)

Example: Prices of Toyota Corolla

ToyotaCorolla.xls

Goal: predict prices of used Toyota Corollas based on their specification

Data: Prices of 1000 used Toyota Corollas, with their specification information

Variables Used

Price in Euros

Age in months as of 8/04

KM (kilometers)

Fuel Type (diesel, petrol, CNG)

HP (horsepower)

Metallic color (1=yes, 0=no)

Automatic transmission (1=yes, 0=no)

CC (cylinder volume)

Doors

Quarterly_Tax (road tax)

Weight (in kg)

Data Sample
(showing only the variables to be used in analysis)

Price	Age	KM	Fuel_Type	HP	Metallic	Automatic	cc	Doors	Quarterly_Tax	Weight
13500	23	46986	Diesel	90	1	0	2000	3	210	1165
13750	23	72937	Diesel	90	1	0	2000	3	210	1165
13950	24	41711	Diesel	90	1	0	2000	3	210	1165
14950	26	48000	Diesel	90	0	0	2000	3	210	1165
13750	30	38500	Diesel	90	0	0	2000	3	210	1170
12950	32	61000	Diesel	90	0	0	2000	3	210	1170
16900	27	94612	Diesel	90	1	0	2000	3	210	1245
18600	30	75889	Diesel	90	1	0	2000	3	210	1245
21500	27	19700	Petrol	192	0	0	1800	3	100	1185
12950	23	71138	Diesel	69	0	0	1900	3	185	1105
20950	25	31461	Petrol	192	0	0	1800	3	100	1185

Preprocessing

Fuel type is categorical (in R - a `factor` variable), must be transformed into binary variables. R's `lm` function does this automatically.

Diesel (1=yes, 0=no)

Petrol (1=yes, 0=no)

None needed* for "CNG" (if diesel and petrol are both 0, the car must be CNG)

*You <u>cannot</u> include all the binary dummies; in regression this will cause a multicollinearity error. Other data mining methods <u>can</u> use all the dummies.

Fitting a Regression Model to the Toyota Data

```
# reduce data frame to the top 1000 rows and select columns for
   regression analysis
car_df = pd.read_csv('ToyotaCorolla.csv')
car_df = car_df.iloc[0:1000]
predictors = ['Age_08_04', 'KM', 'Fuel_Type', 'HP', 'Met_Color',
   'Automatic', 'CC', 'Doors', 'Quarterly_Tax', 'Weight'] outcome = 'Price'

# partition data
X = pd.get_dummies(car_df[predictors], drop_first=True)
y = car_df[outcome]
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4,
  random_state=1)

car_lm = LinearRegression()
car_lm.fit(train_X, train_y)
```

put 40% in validation (test) partition

Output of the Regression Model

```
# print coefficients
print(pd.DataFrame({'Predictor': X.columns, 'coefficient':
  car_lm.coef_}))
```

```
Partial Output

     Predictor         coefficient
0   Age_08_04          -140.748761
1   KM                 -  0.017840
2   HP                   36.103419
3   Met_Color            84.281830
4   Automatic           416.781954
5   CC                    0.017737
6   Doors               -50.657863
7   Quarterly_Tax        13.625325
8   Weight               13.038711
9   Fuel_Type_Diesel 1066.464681
10  Fuel_Type_Petrol 2310.249543
```

Accuracy Metrics for the Regression Model

```
# print performance measures (training data)
regressionSummary(train_y, car_lm.predict(train_X))
```

```
Regression statistics
Mean Error (ME) : 0.0000
Root Mean Squared Error (RMSE) : 1400.5823
Mean Absolute Error (MAE) : 1046.9072
Mean Percentage Error (MPE) : -1.0223
Mean Absolute Percentage Error (MAPE) : 9.2994
```

These are traditional metrics, i.e. measured on the training data

Make the Predictions for the Validation Data
(and show some residuals)

```
# Use predict() to make predictions on a new set
car_lm_pred = car_lm.predict(valid_X)
result = pd.DataFrame({'Predicted': car_lm_pred,
    'Actual': valid_y, 'Residual': valid_y - car_lm_pred})
print(result.head(20))
```

```
       Predicted      Actual       Residual
507  10607.333940    11500        892.666060
818   9272.705792     8950       -322.705792
452  10617.947808    11450        832.052192
368  13600.396275    11450      -2150.396275
242  12396.694660    11950       -446.694660
929   9496.498212     9995        498.501788
262  12480.063217    13500       1019.936783
```

How Well did the Model Do With the Validation Data?

```
# print performance measures (validation data)
regressionSummary(valid_y, car_lm_pred)
```

```
Regression statistics
Mean Error (ME) : 103.6803
Root Mean Squared Error (RMSE) : 1312.8523
Mean Absolute Error (MAE) : 1017.5972
Mean Percentage Error (MPE) : -0.2633
Mean Absolute Percentage Error (MAPE) : 9.0111
```

Selecting Subsets of Predictors

Goal: Find parsimonious model (the simplest model that performs sufficiently well)

- More robust
- Higher predictive accuracy

We will assess predictive accuracy on validation data

Exhaustive Search = "best subset"

Partial Search Algorithms

- Forward
- Backward
- Stepwise

Exhaustive Search = Best Subset

- All possible subsets of predictors assessed (single, pairs, triplets, etc.)
- Computationally intensive, not feasible for big data
- Judge by "adjusted R^2"
- Use `regsubsets()` in package `leaps`

$$R^2_{adj} = 1 - \frac{n-1}{n-p-1}(1-R^2)$$

Penalty for number of predictors

`scikit-learn` and `statsmodels` Lack Out-of-Box Support for Exhaustive Search

Use Exhaustive Search Function (see appendix)

Takes 3 arguments - variable list, training model, scoring model

```
def train_model(variables):
model = LinearRegression()
model.fit(train_X[list(variables)], train_y)
return model

def score_model(model, variables):
pred_y = model.predict(train_X[list(variables)])
# we negate as score is optimized to be as low as possible
return -adjusted_r2_score(train_y, pred_y, model)

allVariables = train_X.columns
results = exhaustive_search(allVariables, train_model,
   score_model)
```

Exhaustive Search Code, cont.

```
data = []
for result in results:
model = result['model']
variables = list(result['variables'])
AIC = AIC_score(train_y, model.predict(train_X[variables]),
  model)
d = {'n': result['n'], 'r2adj': -result['score'], 'AIC':
  AIC}
d.update({var: var in result['variables'] for var in
  allVariables})
data.append(d)
pd.DataFrame(data, columns=('n', 'r2adj', 'AIC') +
  tuple(sorted(allVariables)))
```

Exhaustive output shows best model for each number of predictors

Output

	n	r2adj	AIC	Age_08_04	Automatic	CC	Doors	Fuel_Type_Diesel
0	1	0.767901	10689.712094	True	False	False	False	False
1	2	0.801160	10597.910645	True	False	False	False	False
2	3	0.829659	10506.084235	True	False	False	False	False
3	4	0.846357	10445.174820	True	False	False	False	False
4	5	0.849044	10435.578836	True	False	False	False	False
5	6	0.853172	10419.932278	True	False	False	False	False
6	7	0.853860	10418.104025	True	False	False	False	True
7	8	0.854297	10417.290103	True	True	False	False	True
8	9	0.854172	10418.789079	True	True	False	True	True
9	10	0.854036	10420.330800	True	True	False	True	True
10	11	0.853796	10422.298278	True	True	True	True	True

	Fuel_Type_Petrol	HP	KM	Met_Color	Quarterly_Tax	Weight
0	False	False	False	False	False	False
1	False	True	False	False	False	False
2	False	True	False	False	False	True
3	False	True	True	False	False	True
4	False	True	True	False	True	True
5	True	True	True	False	True	True
6	True	True	True	False	True	True
7	True	True	True	False	True	True
8	True	True	True	False	True	True
9	True	True	True	True	True	True
10	True	True	True	True	True	True

Performance metrics improve as you add predictors, up to approx. 8

Backward Elimination

- Start with all predictors
- Successively eliminate least useful predictors one by one
- Stop when all remaining predictors have statistically significant contribution

Backward Elimination, Using AIC

```
def train_model(variables):
  model = LinearRegression()
  model.fit(train_X[variables], train_y)
  return model

def score_model(model, variables):
  return AIC_score(train_y, model.predict(train_X[variables]), model)

allVariables = train_X.columns
best_model, best_variables = backward_elimination(allVariables,
   train_model,
  score_model, verbose=True)

print(best_variables)

regressionSummary(valid_y, best_model.predict(valid_X[best_variables]))
```

Backward Elimination, Using AIC, Output

```
Variables: Age_08_04, KM, HP, Met_Color, Automatic, CC, Doors,
   Quarterly_Tax, Weight, Fuel_Type_Diesel, Fuel_Type_Petrol
Start: score=10422.30
Step: score=10420.33, remove CC
Step: score=10418.79, remove Met_Color
Step: score=10417.29, remove Doors
Step: score=10417.29, remove None

['Age_08_04', 'KM', 'HP', 'Automatic', 'Quarterly_Tax', 'Weight',
'Fuel_Type_Diesel', 'Fuel_Type_Petrol']
```

Regression statistics

```
Mean Error (ME) : 103.3045
Root Mean Squared Error (RMSE) : 1314.4844
Mean Absolute Error (MAE) : 1016.8875
Mean Percentage Error (MPE) : -0.2700
Mean Absolute Percentage Error (MAPE) : 8.9984
```

Forward Selection

- Start with no predictors
- Add them one by one (add the one with largest contribution)
- Stop when the addition is not statistically significant

Forward Selection, Using AIC

```
# The initial model is the constant model - this requires special handling
# in train_model and score_model
def train_model(variables):
if len(variables) == 0:
return None
model = LinearRegression()
model.fit(train_X[variables], train_y)
return model

def score_model(model, variables):
if len(variables) == 0:
return AIC_score(train_y, [train_y.mean()] * len(train_y), model, df=1)
return AIC_score(train_y, model.predict(train_X[variables]), model)
best_model, best_variables = forward_selection(train_X.columns,
   train_model, score_model,
verbose=True)
print(best_variables)
```

Forward Selection, Output

```
print(best_variables)

Output
Start: score=11565.07, constant
Step: score=10689.71, add Age_08_04
Step: score=10597.91, add HP
Step: score=10506.08, add Weight
Step: score=10445.17, add KM
Step: score=10435.58, add Quarterly_Tax
Step: score=10419.93, add Fuel_Type_Petrol
Step: score=10418.10, add Fuel_Type_Diesel
Step: score=10417.29, add Automatic
Step: score=10417.29, add None
['Age_08_04', 'HP', 'Weight', 'KM', 'Quarterly_Tax', 'Fuel_Type_Petrol',
'Fuel_Type_Diesel', 'Automatic']
```

Stepwise

- Like Forward Selection
- Except at each step, also consider dropping non-significant predictors

(No out-of-box support for stepwise in `scikit-learn` or `statsmodels`; see appendix for function `stepwise_selection`)

Comparing Methods
(in this particular dataset, same results)

Variable	Forward	Backward	Both	Exhaustive
Age_08_04	✔	✔	✔	✔
KM	✔	✔	✔	✔
HP	✔	✔	✔	✔
Met_Color				
Automatic	✔	✔	✔	✔
CC				
Doors				
Quarterly_Tax	✔	✔	✔	✔
Weight	✔	✔	✔	✔
Fuel_TypeDiesel	✔	✔	✔	✔
Fuel_TypePetrol	✔	✔	✔	✔

Regularization (shrinkage)

- Alternative to subset selection
- Rather than binary decisions on including variables, penalize coefficent magnitudes
- This has the effect of “shrinking” coefficients, and also reducing variance
- Predictors with coefficients that shrink to zero are effectively dropped
- Variance reduction improves prediction performance

Shrinkage - Ridge Regression

- OLR minimizes sum of squared errors (residuals) - SSE
- Ridge regression minimizes SSE subject to penalty being below specified threshold
- Penalty, called **L2, is *sum of squared coefficients***
- Predictors are typically standardized

Ridge Regression in `scikit-learn`

alpha is penalty threshold, “0” would be no penalty, i.e. same as OLS

```
ridge = Ridge(normalize=True, alpha=1)
ridge.fit(train_X, train_y)
regressionSummary(valid_y, ridge.predict(valid_X))
```

Shrinkage - Lasso

- OLR minimizes sum of squared errors (residuals) - SSE
- Ridge regression minimizes SSE + penalty
- Penalty, called **L1, is *sum of absolute values for coefficients***
- Predictors are typically standardized

lasso - in `scikit-learn`

alpha is penalty threshold, “0” would
be no penalty, i.e. same as OLS

```
lasso = Lasso(normalize=True, alpha=1)
lasso.fit(train_X, train_y)
regressionSummary(valid_y, lasso.predict(valid_X))
```

or choose penalty threshold
automatically thru cross-validation

```
lasso_cv = LassoCV(normalize=True, cv=5)
lasso_cv.fit(train_X, train_y)
regressionSummary(valid_y, lasso_cv.predict(valid_X))
```

Summary

- Linear regression models are very popular tools, not only for explanatory modeling, but also for prediction
- A good predictive model has high predictive accuracy (to a useful practical level)
- Predictive models are fit to training data, and predictive accuracy is evaluated on a separate validation data set
- Removing redundant predictors is key to achieving predictive accuracy and robustness
- Subset selection methods help find “good” candidate models. These should then be run and assessed.