Chapter 6: Multiple Linear Regression

Data Mining for Business Analytics in Python

Shmueli, Bruce, Gedeck & Patel

© Galit Shmueli, Peter Bruce and Peter Gedeck  2019       rev 10/22/19
We assume a linear relationship between predictors and outcome: 

\[ Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon, \]
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)

<table>
<thead>
<tr>
<th>Price</th>
<th>Age</th>
<th>KM</th>
<th>Fuel_Type</th>
<th>HP</th>
<th>Metallic</th>
<th>Automatic</th>
<th>cc</th>
<th>Doors</th>
<th>Quarterly_Tax</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>13500</td>
<td>23</td>
<td>46986</td>
<td>Diesel</td>
<td>90</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>3</td>
<td>210</td>
<td>1165</td>
</tr>
<tr>
<td>13750</td>
<td>23</td>
<td>72937</td>
<td>Diesel</td>
<td>90</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>3</td>
<td>210</td>
<td>1165</td>
</tr>
<tr>
<td>13950</td>
<td>24</td>
<td>41711</td>
<td>Diesel</td>
<td>90</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>3</td>
<td>210</td>
<td>1165</td>
</tr>
<tr>
<td>14950</td>
<td>26</td>
<td>48000</td>
<td>Diesel</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>2000</td>
<td>3</td>
<td>210</td>
<td>1165</td>
</tr>
<tr>
<td>13750</td>
<td>30</td>
<td>38500</td>
<td>Diesel</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>2000</td>
<td>3</td>
<td>210</td>
<td>1170</td>
</tr>
<tr>
<td>12950</td>
<td>32</td>
<td>61000</td>
<td>Diesel</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>2000</td>
<td>3</td>
<td>210</td>
<td>1170</td>
</tr>
<tr>
<td>16900</td>
<td>27</td>
<td>94612</td>
<td>Diesel</td>
<td>90</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>3</td>
<td>210</td>
<td>1245</td>
</tr>
<tr>
<td>18600</td>
<td>30</td>
<td>75889</td>
<td>Diesel</td>
<td>90</td>
<td>1</td>
<td>0</td>
<td>2000</td>
<td>3</td>
<td>210</td>
<td>1245</td>
</tr>
<tr>
<td>21500</td>
<td>27</td>
<td>19700</td>
<td>Petrol</td>
<td>192</td>
<td>0</td>
<td>0</td>
<td>1800</td>
<td>3</td>
<td>100</td>
<td>1185</td>
</tr>
<tr>
<td>12950</td>
<td>23</td>
<td>71138</td>
<td>Diesel</td>
<td>69</td>
<td>0</td>
<td>0</td>
<td>1900</td>
<td>3</td>
<td>185</td>
<td>1105</td>
</tr>
<tr>
<td>20950</td>
<td>25</td>
<td>31461</td>
<td>Petrol</td>
<td>192</td>
<td>0</td>
<td>0</td>
<td>1800</td>
<td>3</td>
<td>100</td>
<td>1185</td>
</tr>
</tbody>
</table>
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 cannot include all the binary dummies; in regression this will cause a multicollinearity error. Other data mining methods can 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)
Output of the Regression Model

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

Partial Output

<table>
<thead>
<tr>
<th>Predictor</th>
<th>coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age_08_04</td>
<td>-140.748761</td>
</tr>
<tr>
<td>KM</td>
<td>-0.017840</td>
</tr>
<tr>
<td>HP</td>
<td>36.103419</td>
</tr>
<tr>
<td>Met_Color</td>
<td>84.281830</td>
</tr>
<tr>
<td>Automatic</td>
<td>416.781954</td>
</tr>
<tr>
<td>CC</td>
<td>0.017737</td>
</tr>
<tr>
<td>Doors</td>
<td>-50.657863</td>
</tr>
<tr>
<td>Quarterly_Tax</td>
<td>13.625325</td>
</tr>
<tr>
<td>Weight</td>
<td>13.038711</td>
</tr>
<tr>
<td>Fuel_Type_Diesel</td>
<td>1066.464681</td>
</tr>
<tr>
<td>Fuel_Type_Petrol</td>
<td>2310.249543</td>
</tr>
</tbody>
</table>
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
# 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))

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Actual</th>
<th>Residual</th>
</tr>
</thead>
<tbody>
<tr>
<td>507</td>
<td>10607.333940</td>
<td>11500</td>
</tr>
<tr>
<td>818</td>
<td>9272.705792</td>
<td>8950</td>
</tr>
<tr>
<td>452</td>
<td>10617.947808</td>
<td>11450</td>
</tr>
<tr>
<td>368</td>
<td>13600.396275</td>
<td>11450</td>
</tr>
<tr>
<td>242</td>
<td>12396.694660</td>
<td>11950</td>
</tr>
<tr>
<td>929</td>
<td>9496.498212</td>
<td>9995</td>
</tr>
<tr>
<td>262</td>
<td>12480.063217</td>
<td>13500</td>
</tr>
</tbody>
</table>
# 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

© Galit Shmueli and Peter Bruce 2017
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`

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

Penalty for number of predictors
Python code:

```python
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)
```

**Note:**

- `scikit-learn` and `statsmodels` lack out-of-box support for exhaustive search.
- Use the exhaustive search function (see appendix).
- Takes 3 arguments - variable list, training model, scoring model.
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)
data.append(d)

pd.DataFrame(data, columns=('n', 'r2adj', 'AIC') + tuple(sorted(allVariables)))
Exhaustive output shows best model for each number of predictors.

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

<table>
<thead>
<tr>
<th>Fuel_Type_Petrol</th>
<th>HP</th>
<th>KM</th>
<th>Met_Color</th>
<th>Quarterly_Tax</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

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
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)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Forward</th>
<th>Backward</th>
<th>Both</th>
<th>Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age_08_04</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>KM</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>HP</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Met_Color Automatic</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>CC</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Doors</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Quarterly_Tax</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Weight</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Fuel_TypeDiesel</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Fuel_TypePetrol</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Regularization (shrinkage)

- Alternative to subset selection
- Rather than binary decisions on including variables, penalize coefficient 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

```python
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

\[ \text{lasso} = \text{Lasso}(\text{normalize}=\text{True}, \alpha=1) \]
\[ \text{lasso}.\text{fit}(\text{train}_X, \text{train}_y) \]
\[ \text{regressionSummary}(\text{valid}_y, \text{lasso}.\text{predict}(\text{valid}_X)) \]

or choose penalty threshold automatically thru cross-validation

\[ \text{lasso_cv} = \text{LassoCV}(\text{normalize}=\text{True}, \text{cv}=5) \]
\[ \text{lasso_cv}.\text{fit}(\text{train}_X, \text{train}_y) \]
\[ \text{regressionSummary}(\text{valid}_y, \text{lasso_cv}.\text{predict}(\text{valid}_X)) \]
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.