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:

outcome

coefficients

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

constant

error (noise)

predictors

The most popular model for making predictions is the *multiple linear regression* model encountered in most introductory statistics courses and textbooks. This model is used to fit a relationship between a numerical outcome variable Y (also called the response, target, or dependent variable) and a set of predictors X_1, X_2, …, X_p (also referred to as independent variables, input variables, regressors, or covariates).

Video

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, must be transformed into binary variables.

- 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

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from dmba import regressionSummary

# 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(y_true(actual values), y_pred (predicted values))

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 intercept = car_lm.intercept_ to get the intercept.

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 <u>validation</u> data

Exhaustive Search = “best subset”

Partial Search Algorithms:

- Forward
- Backward
- Stepwise

PCA vs. Subset Methods

- Principal Component Analysis (PCA) transforms data into a new feature space, while subset selection picks a subset of the original features.
- PCA results in uncorrelated components that may not have straightforward interpretations, whereas subset selection maintains the original features, allowing for easier interpretation.
- PCA focuses on maximizing variance, while subset selection focuses on finding the most relevant features for prediction.
- PCA is useful when dealing with high-dimensional data with multicollinearity, whereas subset selection is ideal when you want to maintain interpretability and understand the relationship between predictors and the outcome.

Exhaustive Search = Best Subset

- Library: from sklearn.metrics import r2_score
- All possible subsets of predictors assessed (single, pairs, triplets, etc.)
- Computationally intensive, not feasible for big data
- Judge by "adjusted R^2"

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

n = Sample size
p = # of independent variables

Penalty for number of predictors

Video-1 , Video-2

- The adjusted R^2 is a modified version of the regular R^2 statistic in linear regression. It is a measure of how well the independent variables in a regression model explain the variability in the dependent variable. The primary difference between the regular R^2 and the adjusted R^2 is that the adjusted R^2 takes into account the number of independent variables used in the model. R^2 is the percentage of variation explained by the relationship between two variables.

- Judged by AIC/BIC (The lower the better)

- A second popular set of criteria for balancing under-fitting and over-fitting are the *Akaike Information Criterion (AIC)* and *Schwartz's Bayesian Information Criterion (BIC)*. AIC and BIC measure the goodness of fit of a model, but also include a penalty that is a function of the number of parameters in the model. As such, they can be used to compare various models for the same data set. AIC and BIC are estimates of prediction error based in information theory.

Video

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

Takes 3 arguments:
variable list (of all features), training model (for a given set of features), 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'])
  predictions = model.predict(train_X[variables])
  AIC = AIC_score(train_y, predictions, model)
  BIC = BIC_score(train_y, predictions, model)
  d = {'n': result['n'], 'r2adj': -result['score'], 'AIC':
  AIC, 'BIC': BIC}
  d.update({var: var in result['variables'] for var in
  allVariables})
  data.append(d)
pd.DataFrame(data, columns=('n', 'r2adj', 'AIC’, 'BIC') +
  tuple(sorted(allVariables)))
```

Exhaustive output shows best model for each number of predictors

The code reports the best model with a single predictor, two predictors, and so on. It can be seen that the R^2adj increases until eight predictors are used and then slowly decreases. The AIC also indicates that a model with eight predictors is good. The dominant predictor in all models is the age of the car, with horsepower, weight and mileage playing important roles as well.

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
```

- You can use different strategies for selecting which variable to remove, but the most common approach is to remove the variable with the highest p-value (least statistically significant). You might also consider domain knowledge to decide which variable to remove.

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, Using AIC, 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 (forward and backward iteratively combined)
- Stepwise selection is more computationally intensive because it considers both adding and removing variables. The reason for considering the stepwise approach is to balance the inclusion of potentially important predictors (forward selection) with the removal of non-significant or redundant predictors (backward elimination). It provides a compromise between model complexity and predictive power.
- No out-of-box support for stepwise in `scikit-learn` or `statsmodels`

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 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
- Two most popular methods: *Ridge Regression* and *Lasso*

Shrinkage - Ridge Regression

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

Ridge Regression in `scikit-learn`

Ordinary Least Squares (OLS) method finds values that minimize the sum of squared deviations between the actual outcome values (Y) and their predicted values based on that model ().

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

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

The alpha parameter can take on values in the range of 0 to positive infinity. The range for alpha is typically defined as follows:

* If alpha = 0: This corresponds to standard linear regression without any regularization. In this case, the model tries to fit the training data with the least squares criterion, and it may lead to overfitting if there are many features or multicollinearity.
* If alpha is a small positive value (close to 0): The model applies a small amount of L2 regularization, which encourages the coefficients to be close to those of standard linear regression. This helps in reducing overfitting, but the effect is relatively mild.
* If alpha is a large positive value: The model applies stronger L2 regularization, which can significantly shrink the coefficients towards zero. This is useful for feature selection and preventing overfitting. The larger the alpha, the stronger the regularization effect.

Shrinkage - Lasso

- OLR minimizes sum of squared errors (residuals) - SSE
- Lasso shrinks some of the coefficients to zero, thereby resulting in a subset of predictors
- 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

```
from sklearn.linear_model import LassoCV
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))
```