Subset Selection and Shrinkage Methods

By Adrian Chavez-Loya

Background

In car sales, one of the most critical metrics is the number of days a vehicle spends on the lot. Some estimates suggest that every day a vehicle spends on the lot will cost the dealership ~$10/day in depreciation and maintenance. Multiply that by the hundreds (or thousands) of vehicles a dealership may hold in inventory and this quickly becomes one of the largest costs. A dataset provided by DriveTime, contains vehicle information as well as the number of days it spent on the lot, our task is to find any relationships that may explain the increase or decrease in days to sell.

Relevant Datasets

`drive_time_sedans.csv`

Source: https://github.com/Fumanguyen/drivetime-sedans-used-vehicle-market/blob/master/drive_time_sedans.csv

Task 1: Import the dataset and convert the categorical variables to dummy variables.

Important Note: The tasks below can be very computationally intensive. If you don't want to wait a long time for things to run or you don't feel your computer is powerful to complete these tasks in a reasonable time, I suggest dropping the `make.model`, `state`, and/or `makex` variables. Your grade will not be based on the inclusion or exclusion of any variables, I'm more interested in the methods but if you have the resources and are curious to explore more, feel free to use all variables.

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
import numpy as np
```

In [ ]:
```
df = pd.read_csv('drive_time_sedans.csv')
df = df.drop('state', axis = 1)
df = df.drop('vehicle.age.group', axis = 1)
df = df.drop('make.model', axis = 1) #dropped state, make.model,
```

In [ ]:
```
df.head()
```

Out[ ]:

	data.set	total.cost	lot.sale.days	overage	mileage	vehicle.type	domest
0	TRAIN	4037	135	YES	67341	FAMILY.LARGE	
1	TRAIN	4662	18	NO	69384	FAMILY.SMALL	
2	TRAIN	4459	65	NO	58239	ECONOMY	
3	TRAIN	4279	1	NO	58999	ECONOMY	
4	TRAIN	4472	37	NO	47234	FAMILY.MEDIUM	

In [ ]:
```
# Convert categorical variables to dummy variables
df_dummies = pd.get_dummies(df, drop_first=True)

# Convert specific categorical variables to dummy variables
columns_to_convert = ['overage', 'vehicle.type', 'domestic.import
df_dummies = pd.get_dummies(df, columns=columns_to_convert, drop_
# Display the first few rows and structure of the dataset with du
print(df_dummies.head())
print(df_dummies.info())
```

```
  data.set  total.cost  lot.sale.days  mileage  vehicle.age  over
age_YES  \
0    TRAIN        4037            135    67341            8
True
1    TRAIN        4662             18    69384            4
False
2    TRAIN        4459             65    58239            4
False
3    TRAIN        4279              1    58999            3
False
4    TRAIN        4472             37    47234            6
False

   vehicle.type_FAMILY.LARGE  vehicle.type_FAMILY.MEDIUM  \
0                       True                       False
1                      False                       False
2                      False                       False
3                      False                       False
4                      False                        True

   vehicle.type_FAMILY.SMALL  vehicle.type_LUXURY  ...  makex_PLY
MOUTH  \
0                      False                False  ...
False
1                       True                False  ...
False
2                      False                False  ...
False
3                      False                False  ...
False
4                      False                False  ...
False

   makex_PONTIAC  makex_TOYOTA  color.set_BLUE  color.set_GOLD  \
0          False         False           False           False
1          False         False           False           False
2          False         False           False           False
3          False         False           False           False
4          False         False            True           False

   color.set_GREEN  color.set_PURPLE  color.set_RED  color.set_SI
LVER  \
0            False             False          False
True
1            False             False          False
True
2            False             False           True             F
alse
3            False             False           True             F
alse
```

```
4            False               False          False             F
alse

   color.set_WHITE
0            False
1            False
2            False
3            False
4            False

[5 rows x 37 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17506 entries, 0 to 17505
Data columns (total 37 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   data.set                    17506 non-null  object
 1   total.cost                  17506 non-null  int64 
 2   lot.sale.days               17506 non-null  int64 
 3   mileage                     17506 non-null  int64 
 4   vehicle.age                 17506 non-null  int64 
 5   overage_YES                 17506 non-null  bool  
 6   vehicle.type_FAMILY.LARGE   17506 non-null  bool  
 7   vehicle.type_FAMILY.MEDIUM  17506 non-null  bool  
 8   vehicle.type_FAMILY.SMALL   17506 non-null  bool  
 9   vehicle.type_LUXURY         17506 non-null  bool  
 10  domestic.import_Import      17506 non-null  bool  
 11  makex_CADILLAC              17506 non-null  bool  
 12  makex_CHEVROLET             17506 non-null  bool  
 13  makex_CHRYSLER              17506 non-null  bool  
 14  makex_DAEWOO                17506 non-null  bool  
 15  makex_DODGE                 17506 non-null  bool  
 16  makex_FORD                  17506 non-null  bool  
 17  makex_GEO                   17506 non-null  bool  
 18  makex_HONDA                 17506 non-null  bool  
 19  makex_HYUNDAI               17506 non-null  bool  
 20  makex_KIA                   17506 non-null  bool  
 21  makex_MAZDA                 17506 non-null  bool  
 22  makex_MERCURY               17506 non-null  bool  
 23  makex_MITSUBISHI            17506 non-null  bool  
 24  makex_NISSAN                17506 non-null  bool  
 25  makex_OLDSMOBILE            17506 non-null  bool  
 26  makex_OTHER                 17506 non-null  bool  
 27  makex_PLYMOUTH              17506 non-null  bool  
 28  makex_PONTIAC               17506 non-null  bool  
 29  makex_TOYOTA                17506 non-null  bool  
 30  color.set_BLUE              17506 non-null  bool  
 31  color.set_GOLD              17506 non-null  bool  
 32  color.set_GREEN             17506 non-null  bool  
 33  color.set_PURPLE            17506 non-null  bool  
```

```
 34  color.set_RED                  17506 non-null  bool
 35  color.set_SILVER               17506 non-null  bool
 36  color.set_WHITE                17506 non-null  bool
dtypes: bool(32), int64(4), object(1)
memory usage: 1.2+ MB
None
```

Task 2: This dataset specifies which observations to use as train/test/validate. Split it into three dataframes based on these values.

If you've already converted those to dummy variables, you may have to subset slightly different. Search "*conditional subset pandas dataframe*" for a starting point or reach out to me (before the soft deadline) for guidance.

In [ ]:

```
# Split data according to data.set
X_train = df_dummies[df_dummies['data.set'] == 'TRAIN']
X_validate = df_dummies[df_dummies['data.set'] == 'VALIDATE']
X_test = df_dummies[df_dummies['data.set'] == 'TEST']

# Chose response variable which would be lot.sale.days (days on l
y_train = X_train['lot.sale.days']
y_validate = X_validate['lot.sale.days']
y_test = X_test['lot.sale.days']

# Drop the 'data.set' column from datasets since they are now spl
X_train = X_train.drop(columns=['data.set'])
X_validate = X_validate.drop(columns=['data.set'])
X_test = X_test.drop(columns=['data.set'])

print("Shape of X_train:", X_train.shape)
print("Shape of X_validate:", X_validate.shape)
print("Shape of X_test:", X_test.shape)
```

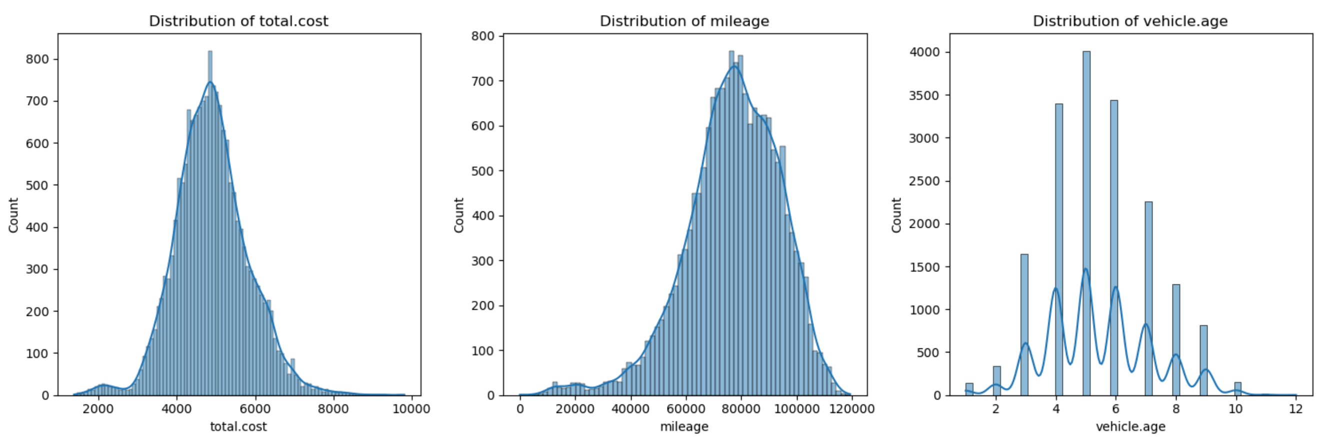
```
Shape of X_train: (8753, 36)
Shape of X_validate: (4377, 36)
Shape of X_test: (4376, 36)
```

Plotted Distributions to Check

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Concatenate X_train, X_validate, and X_test to visualize overal
```

```python
X_all = pd.concat([X_train, X_validate, X_test], axis=0)

# Plot histograms for 'total.cost', 'mileage', and 'vehicle.age'
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.histplot(X_all['total.cost'], kde=True)
plt.title('Distribution of total.cost')

plt.subplot(1, 3, 2)
sns.histplot(X_all['mileage'], kde=True)
plt.title('Distribution of mileage')

plt.subplot(1, 3, 3)
sns.histplot(X_all['vehicle.age'], kde=True)
plt.title('Distribution of vehicle.age')

plt.tight_layout()
plt.show()
```



There seems to be skewness in the different distributions:

`total.cost` - positively skewed

`mileage` - negatively skewed

`vehicle.age` - multimodal, positively skewed

Therefore , I will attempt to normalize these models

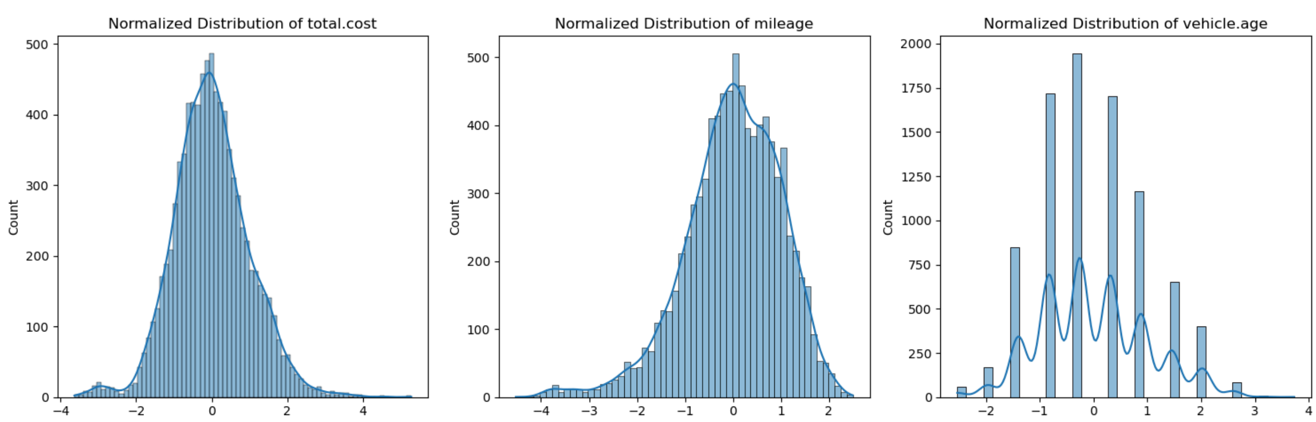
In [ ]:
```python
# Checked skewness for fun (indeed, there seems to be some skewne
skew_total_cost = X_train['total.cost'].skew()
skew_mileage = X_train['mileage'].skew()
skew_vehicle_age = X_train['vehicle.age'].skew()

print(f'Skewness of total.cost: {skew_total_cost}')
print(f'Skewness of mileage: {skew_mileage}')
print(f'Skewness of vehicle.age: {skew_vehicle_age}')
```

```
Skewness of total.cost: 0.19591485721467414
Skewness of mileage: -0.6652483557868974
Skewness of vehicle.age: 0.2844999446508882
```

Task 3: Normalize `total.cost`, `mileage`, and `vehicle.age`

In [ ]:
```python
from sklearn.preprocessing import StandardScaler

# utilized standard scaler for normalization (thanks Youtube)
scaler = StandardScaler()

# Normalize 'total.cost', 'mileage', and 'vehicle.age'
X_train[['total.cost', 'mileage', 'vehicle.age']] = scaler.fit_tr
X_validate[['total.cost', 'mileage', 'vehicle.age']] = scaler.tra
X_test[['total.cost', 'mileage', 'vehicle.age']] = scaler.transfo
```

In [ ]:
```python
# I  will plot to see if the distributions look more normalized n
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.histplot(X_train['total.cost'], kde=True)
plt.title('Normalized Distribution of total.cost')

plt.subplot(1, 3, 2)
sns.histplot(X_train['mileage'], kde=True)
plt.title('Normalized Distribution of mileage')

plt.subplot(1, 3, 3)
sns.histplot(X_train['vehicle.age'], kde=True)
plt.title('Normalized Distribution of vehicle.age')

plt.tight_layout()
plt.show()
```



Looks Better!!! The distributions center more around zero! Wow! (please give me feedback on if there is a better way to normalize**

Task 4: Use the code from the applied lecture to perform forward stepwise selection, with the single validation set from before (as opposed to cross-validation). Return not only the AIC, BIC, and Adjusted R^2, as was shown in the lecture, but also the MSE on the validation set.

In [ ]:

```
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm

# I converted boolean values since statsmodels needs to work with
def convert_bool_to_int(df):
    bool_columns = df.select_dtypes(include='bool').columns
    df[bool_columns] = df[bool_columns].astype('int64')
    return df

# Forward stepwise selection
def forward_stepwise_selection(X_train, y_train, X_validate, y_va
    X_train = convert_bool_to_int(X_train)
    X_validate = convert_bool_to_int(X_validate)

    remaining_predictors = list(X_train.columns)
    selected_predictors = []
    best_criteria = {'AIC': float('inf'), 'BIC': float('inf'), 'A
    best_model = None

    while remaining_predictors:
        results = []
        for predictor in remaining_predictors:
            predictors = selected_predictors + [predictor]
            X_train_model = sm.add_constant(X_train[predictors])
            X_validate_model = sm.add_constant(X_validate[predict

            model = sm.OLS(y_train, X_train_model).fit()
            predictions = model.predict(X_validate_model)
            mse = mean_squared_error(y_validate, predictions)

            results.append((model, mse))

        results.sort(key=lambda x: x[1])  # Sort by MSE
```

```
        best_new_model, best_new_mse = results[0]

        if best_new_mse < best_criteria['MSE']:
            best_model = best_new_model
            best_criteria['MSE'] = best_new_mse
            best_criteria['AIC'] = best_model.aic
            best_criteria['BIC'] = best_model.bic
            best_criteria['Adj_R2'] = best_model.rsquared_adj
            best_new_predictor = best_model.model.exog_names[-1]
            selected_predictors.append(best_new_predictor)
            remaining_predictors.remove(best_new_predictor)
        else:
            break

    return best_model, best_criteria, selected_predictors

best_model, best_criteria, selected_predictors = forward_stepwise

# Results of algorithm
print("Selected Predictors from Algorithm:", selected_predictors)
print("AIC:", best_criteria['AIC'])
print("BIC:", best_criteria['BIC'])
print("Adjusted R^2:", best_criteria['Adj_R2'])
print("MSE on Validation Set:", best_criteria['MSE'])
```

```
Selected Predictors from Algorithm: ['lot.sale.days', 'mileage',
'makex_MAZDA']
AIC: -548148.9391221282
BIC: -548120.6305150172
Adjusted R^2: 1.0
MSE on Validation Set: 3.687022241911182e-29
```

Selected Predictors from Algorithm: ['lot.sale.days', 'mileage', 'makex_MAZDA']

AIC: -548148.9391221282

BIC: -548120.6305150172

Adjusted R^2: 1.0

MSE on Validation Set: 3.687022241911182e-29

** Wow... Perfect fit!?!?! It seems like have the perfect model, or it more likely means we overfitted the model, perhaps too many predictors. Lets try another method

Task 5: Using the code from the shrinkage methods lecture, find the optimal α and λ for an Elastic Net regression using Cross-Validation.

Note: Remember that λ is the argument `alpha` in scikit-learn and α is the `l1_ratio` argument. Sorry that nobody can settle on terminology.

In [ ]:

```
from sklearn.linear_model import ElasticNetCV
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# Initialize ElasticNetCV
elastic_net = ElasticNetCV(
    l1_ratio=[0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1],  # l1_ratio cor
    alphas=[0.1, 1, 10, 100],  # alphas corresponds to lambda in 
    cv=5,  # Number of cross-validation folds
    random_state=42,
    n_jobs=-1  # Use all available CPUs
)

# Fit the model
elastic_net.fit(X_train, y_train)

# Optimal parameter values I found for l1 ratio and lamba (aka al
print("Optimal l1_ratio:", elastic_net.l1_ratio_)
print("Optimal alpha (lambda):", elastic_net.alpha_)
```

```
Optimal l1_ratio: 0.1
Optimal alpha (lambda): 0.1
```

- The low l1 ratio of 0.1 indicates Ridge regularization (aka l2 penalty, I call it the sqaure penalty) may before better
- Low lambda value of 0.1 shows regularization penalty is fairly mild

Question: Given all of the results you've found, which model would you choose and why? Hint: There is no right answer but you will need to justify any answer you give.

Based on my results for Elastic Net and Forward Subset Regression, I would choose FSR as my model.

Although I do have concerns for overfitting, the predictors were statistically significant. I might try LOOCV or other methods to rule out which variables have the most significance and reduce the chances of multi-collinearity. The MSE was nearly zero, which indicates that the model performs well on unseen data.