Applied Statistical Methods

Linear Model Selection

Xuemao Zhang
East Stroudsburg University

March 29, 2023

Outline

- High-dimensional Data
- Variable Pre-Selection
- Best Subset Selection
- Forward Stepwise Regression
- Backward Stepwise Regression
- Model Selection
    - Best Subset Selection
    - Validation Set
    - k-Fold Cross-Validation

High-dimensional Data

- If a data set $n >> p$, the data set is **low-dimensional**

```
import pandas as pd
Auto=pd.read_csv("../data/Auto.csv")
print(Auto.shape)
```

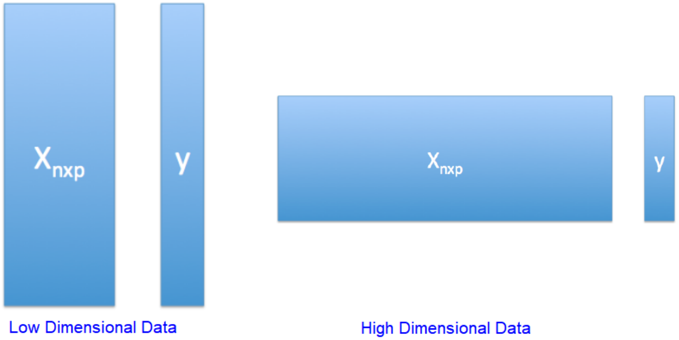
```
## (392, 9)
```

```
print(Auto.columns)
```

```
## Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight
##        'acceleration', 'year', 'origin', 'name'],
##       dtype='object')
```

High-dimensional Data

- Lots of the data sets coming out of modern biological techniques are **high-dimensional**: $n \approx p$ or $n << p$.



- This poses statistical challenges!

High-dimensional Data

- Some predictors are useful, others aren't.
- If we include too many predictors, we will overfit the data.
    - Overfitting: Model looks great on the training data used to develop it, but will perform very poorly on test data.
- When $p \approx n$ or $p > n$, overfitting is guaranteed unless we are very careful.
- Classical statistical techniques, such as linear regression, cannot be applied.
- Even very simple tasks, like identifying variables that are associated with a response, must be done with care.
- High risks of overfitting, false positives, and more.

High-dimensional Data

- We usually cannot perform least squares regression to fit a model with high-dimensional data, because we will get zero training error but a terrible test error.
- Instead, we must fit a less complex model, e.g. a model with fewer variables.
- We will consider five ways to fit less complex models:
  - Variable Pre-Selection
  - Forward Stepwise Regression
  - Ridge Regression
  - Lasso Regression
  - Principal Components Regression

Variable Pre-Selection

- The simplest approach for fitting a model in high dimensions:
- 1. Choose a small set of variables, say the q variables that are most correlated with the response, where $q < n$ and $q < p$.
- 2. Use least squares to fit a model predicting y using only these q variables.
- This approach is simple and straightforward.

Variable Pre-Selection

```python
import numpy as np
from numpy import random
random.seed(1)
xtr=random.normal(0, 1, size=(100, 100))# 100*100 matrix from N(0,1)
beta=np.array([1]*10+[0]*90) # vector of length 100
ytr=np.dot(xtr, beta)+random.normal(0, 1,size=100)
cors=[]
from scipy.stats import pearsonr
for i in range(100):
    cors.append(pearsonr(xtr[:,i],ytr).statistic)

cors=np.array(cors)
sum(abs(cors)>0.2)
#then model fit:
```

```
## 10
```

```python
import statsmodels.api as sm
model = sm.OLS(ytr, xtr[:,abs(cors)>0.2]).fit()
```

Variable Pre-Selection

```
print(model.summary())
```

```
##                                  OLS Regression Results
## ==================================================================================
## Dep. Variable:                      y   R-squared (uncentered):
## Model:                            OLS   Adj. R-squared (uncentered):
## Method:                 Least Squares   F-statistic:
## Date:                Tue, 28 Mar 2023   Prob (F-statistic):                    1
## Time:                        14:32:29   Log-Likelihood:
## No. Observations:                 100   AIC:
## Df Residuals:                      90   BIC:
## Df Model:                          10
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025      0.975]
## ------------------------------------------------------------------------------
## x1             0.7174      0.117      6.142      0.000       0.485       0.949
## x2             1.0578      0.122      8.661      0.000       0.815       1.300
## x3             0.9829      0.110      8.926      0.000       0.764       1.202
## x4             0.9550      0.105      9.059      0.000       0.746       1.164
## x5             0.8421      0.118      7.153      0.000       0.608       1.076
## x6             1.2730      0.115     11.045      0.000       1.044       1.502
## x7             0.7949      0.106      7.464      0.000       0.583       1.006
## x8             0.9710      0.115      8.449      0.000       0.743       1.199
## x9             0.9606      0.103      9.356      0.000       0.757       1.165
```

Variable Pre-Selection

How Many Variable to Use?

- We need a way to choose q, the number of variables used in the regression model.
- We want q that **minimizes the test error**.
- For a range of values of q, we can perform the validation set approach, leave-one-out cross-validation, or K-fold cross-validation in order to estimate the test error.
- Then choose the value of q for which the estimated test error is smallest.

Variable Pre-Selection

Estimating the Test Error For a Given q:

The right way to estimate the test error using the *validation set approach*:

- 1. Split the observations into a training set and a validation set.
- 2. Using the training set only:
  - a. Identify the q variables most associated with the response.
  - b. Use least squares to fit a model predicting y using those q variables.
  - c. Let $\hat{\beta}_1, \ldots, \hat{\beta}_q$ denote the resulting coefficient estimates.
- 3. Use $\hat{\beta}_1, \ldots, \hat{\beta}_q$ obtained on training set to predict response on validation set, and compute the validation set MSE.

Variable Pre-Selection

Estimating the Test Error For a Given q:

This is the wrong way to estimate the test error using the validation set approach:

- 1. Identify the q variables most associated with the response on the full data set.
- 2. Split the observations into a training set and a validation set.
- 3. Using the training set only:
  - a. Use least squares to fit a model predicting y using those q variables.
  - b. Let $\hat{\beta}_1, \dots, \hat{\beta}_q$ denote the resulting coefficient estimates.
- 4. Use $\hat{\beta}_1, \dots, \hat{\beta}_q$ obtained on training set to predict response on validation set, and compute the validation set MSE.

Variable Pre-Selection

- The variable pre-selection approach is simple and easy to implement - all we need is a way to calculate correlations, and software to fit a linear model using least squares.
- But it might not work well: just because a bunch of variables are correlated with the response doesn't mean that when used together in a linear model, they will predict the response well.
- What we really want to do: pick the q variables that best predict the response.

Best Subset Selection

- We would like to consider all possible models using a subset of the p **predictors**.
- In other words, we'd like to consider all 2^p possible models.
- Then we select a single best model using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.
- Unfortunately, this is computationally intractable. Best subset selection cannot be applied with very large p ($>$10).
- There could be overfitting and it is not straightforward to choose a best model among models with different number of predictors.

Forward Stepwise Regression

- Forward stepwise selection begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all of the predictors are in the model.
- In particular, at each step the variable that gives the greatest *additional* improvement to the fit is added to the model.
- Forward Stepwise Regression cab be in two directions. We now consider 1-direction only.

Forward Stepwise Regression

- 1. Use least squares to fit p *univariate* regression models, and select the predictor corresponding to the best model (according to e.g. training set MSE).
- 2. Use least squares to fit $p-1$ models containing that one predictor in step 1, and each of the $p-1$ other predictors. Select the predictors in the best two-variable model.
- 3. Now use least squares to fit $p-2$ models containing those two predictors, and each of the $p-2$ other predictors. Select the predictors in the best three-variable model.
- 4. And so on...

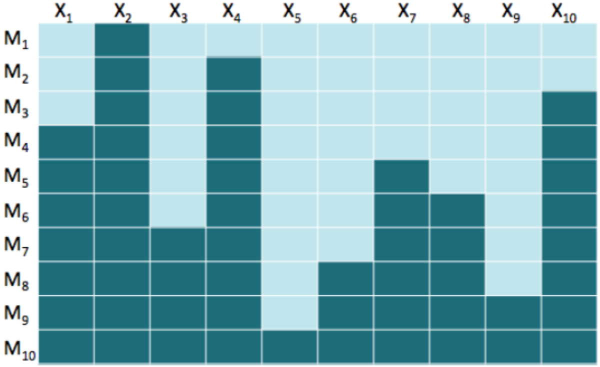
This gives us a nested set of models, containing the predictors

$$\mathcal{M}_1 \subseteq \mathcal{M}_2 \subseteq \mathcal{M}_3 \ldots$$

Forward Stepwise Regression

- We just need to fit $p + (p-1) + (p-2) + \cdots + 1 = \frac{(1+p)p}{2}$ models. Then we Select a single best model from among $\mathcal{M}_1, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p, AIC, BIC, or adjusted R^2.

- Forward Stepwise Regression With $p = 10$



	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
M_1	□	■	□	□	□	□	□	□	□	□
M_2	□	■	□	■	□	□	□	□	□	□
M_3	□	■	□	■	□	□	□	□	□	■
M_4	■	■	□	■	□	□	□	□	□	■
M_5	■	■	□	■	□	□	■	□	□	■
M_6	■	■	□	■	□	□	■	■	□	■
M_7	■	■	■	■	□	□	■	■	□	■
M_8	■	■	■	■	□	■	■	■	□	■
M_9	■	■	■	■	□	■	■	■	■	■
M_{10}	■	■	■	■	■	■	■	■	■	■

Forward Stepwise Regression

- Computational advantage over best subset selection is clear.
- Forward stepwise selection isn't guaranteed to give you the best model containing q variables.
- To get the best model with q variables, you'd need to consider every possible one; computationally intractable.

Python: Forward Stepwise Regression

- There are some missing values here, so before we proceed we will remove them. The `dropna()` function removes all of the rows that have missing values in any variable
- Some of our predictors are categorical, so we'll want to clean those up as well. We'll ask pandas to generate dummy variables for them, separate out the response variable, and stick everything back together again:

```
import pandas as pd
Hitters=pd.read_csv("../data/Hitters.csv", header=0, na_values='NA')
Hitters = Hitters.dropna()
dummies = pd.get_dummies(Hitters[['League', 'Division', 'NewLeague']])
y = Hitters['Salary']
X_ = Hitters.drop(['Salary', 'League', 'Division', 'NewLeague'],
     axis=1).astype('float64')
X = pd.concat([X_, dummies[['League_N', 'Division_W', 'NewLeague_N']]], axis=1)
```

Python: Forward Stepwise Regression

- `sklearn.feature_selection.SequentialFeatureSelector`: Sequential Feature Selection can add features one at a time (forward selection) or remove features from the list of the available features (backward selection), based on a cross-validated score maximization
    - https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SequentialFeatureSelector.html

```
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LinearRegression
model = LinearRegression()
sfs = SequentialFeatureSelector(model,n_features_to_select='auto',
       tol=None, direction='forward') #try 'backward'
sfs.fit(X, y)
```

```
## SequentialFeatureSelector(estimator=LinearRegression(),
##                           n_features_to_select='auto')
```

```
sfs.n_features_to_select_
```

```
## 9
```

```
sfs.support_
```

```
## array([False, False, False,  True,  True,  True, False,  True, False,
##         True,  True, False,  True, False, False, False,  True,  True,
##        False])
```

```
X.columns[sfs.support_]
```

Backward Stepwise Regression

- Like forward stepwise selection, backward stepwise selection provides an efficient alternative to best subset selection.
- However, unlike forward stepwise selection, it begins with the full least squares model containing all p predictors, and then iteratively removes the least useful predictor, one-at-a-time.
- Like forward stepwise selection, the backward selection approach searches through only $\frac{p(p+1)}{2}$ models, and so can be applied in settings where p is too large to apply best subset selection.
- Like forward stepwise selection, backward stepwise selection is not guaranteed to yield the best model containing a subset of the p predictors.
- Backward selection requires that the number of samples n is larger than the number of variables p (so that the full model can be fit). In contrast, forward stepwise can be used even when $n < p$, and so is the only viable subset method when p is very large.

Python: Best Subset Selection

- Helper function for fitting linear regression (Sklearn)
  - https://xavierbourretsicotte.github.io/subset_selection.html and http://www.science.smith.edu/~jcrouser/SDS293/labs/lab8-py.html
- Mean squared error regression loss: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

```
#from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

def fit_linear_reg(feature_set):
    #Fit linear regression model and return RSS and R squared values
    model_k = LinearRegression(fit_intercept = True)
    regr=model_k.fit(X[list(feature_set)],y)
    RSS = mean_squared_error(y, model_k.predict(X[list(feature_set)]))*len(y)
    #R_squared = model_k.score(X, y)
    return {"model":regr, "RSS":RSS}
```

Python: Best Subset Selection

- Implementing Best subset selection (using itertools.combinations)
  - https://docs.python.org/3/library/itertools.html#itertools.combinations
- The following function returns a DataFrame containing the best model that we generated, along with some extra information about the model.

```
import time
import itertools
def getBest(k):
    tic = time.time()
    results = [] #store RSS
    for combo in itertools.combinations(X.columns, k):
      results.append(fit_linear_reg(combo))
    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)
    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].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 mode
    return best_model
```

Python: Best Subset Selection

- Now we want to call the function for each number of predictors `k`:

```
models_best = pd.DataFrame(columns=["RSS", "model"])
tic = time.time()

for i in range(1,8):
    models_best.loc[i] = getBest(i)
```

```
## Processed 19 models on 1 predictors in 0.06248974800109863 secon
## Processed 171 models on 2 predictors in 0.6135194301605225 secon
## Processed 969 models on 3 predictors in 4.970108270645142 second
## Processed 3876 models on 4 predictors in 27.6130211353302 second
## Processed 11628 models on 5 predictors in 62.891472578048706 sec
## Processed 27132 models on 6 predictors in 128.36549139022827 sec
## Processed 50388 models on 7 predictors in 259.42592573165894 sec
```

```
toc = time.time()
print("Total elapsed time:", (toc-tic), "seconds.")
```

```
## Total elapsed time: 484.044052362442 seconds.
```

Python: Best Subset Selection

- Now we have one big DataFrame that contains the best models we've generated along with their RSS:

```
print(models_best)
```

```
##                RSS              model
## 1  36179679.255042  LinearRegression()
## 2  30646559.890373  LinearRegression()
## 3  29249296.855867  LinearRegression()
## 4  27970851.815816  LinearRegression()
## 5  27149899.432012  LinearRegression()
## 6  26194903.927595  LinearRegression()
## 7  25906547.500624  LinearRegression()
```

- We can access the details of each model:

```
print(models_best.loc[2, "model"].n_features_in_)
```

```
## 2
```

```
print(models_best.loc[2, "model"].feature_names_in_)
```

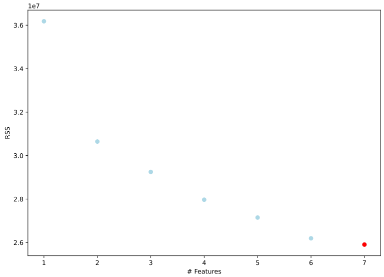
```
## ['Hits' 'CRBI']
```

```
print(models_best.loc[2, "model"].coef_)
```

```
## [3.30084457 0.6898994 ]
```

Python: Best Subset Selection

```
import matplotlib.pyplot as plt
import pandas as pd
min_Rss=models_best['RSS'].min()
min_index=pd.DataFrame(models_best['RSS']).astype(float).idxmin()
#min_index=models_best[models_best['RSS']==min_Rss].index.values
numb_features=models_best.index #models_best.shape[0]
Rss=models_best['RSS']
plt.scatter(numb_features, y=Rss, color='lightblue')
plt.plot(min_index, min_Rss, color='red', marker='o')
plt.xlabel('# Features')
plt.ylabel('RSS')
plt.show()
```



Best Subset Selection with library abess

- abess (Adaptive BEst Subset Selection) library aims to solve general best subset selection
  - https://abess.readthedocs.io/en/latest/Python-package/linear/Linear.html

```
import pandas as pd
import numpy as np
from abess.linear import LinearRegression
model = LinearRegression()
#X.values convert X to a matrix
model.fit(X.values, y)
```

```
## LinearRegression()
```

```
ind = np.nonzero(model.coef_)
print("estimated non-zero: ", ind)
```

```
## estimated non-zero:  (array([ 1, 11, 13, 17], dtype=int64),)
```

```
print("features to keep: ", X.columns[ind])
```

```
## features to keep:  Index(['Hits', 'CRBI', 'PutOuts', 'Division_W'], dtype='objec
```

```
print("estimated intercept: ", model.intercept_)
```

```
## estimated intercept:  13.923104428774991
```

```
print("estimated coef: ", model.coef_[ind])
```

Model Selection by Cross-Validation

- In the above, we can use Validation Set or Cross Validation to compare the best 19 regression models manually
  - For example, the following split the data into a training data set and test data set, you may continue the programming. But it will be very slow

```
import numpy as np
np.random.seed(1)
train = np.random.choice(Hitters.shape[0],round(Hitters.shape[0]/2),
replace=False)
select = np.in1d(range(Hitters.shape[0]), train)
X_train=X.iloc[train,:]
y_train=y.iloc[train]
```

Model Selection by Cross-Validation

- The library abess can do model selection with Cross-Validation for us

```
import pandas as pd
import numpy as np
from abess.linear import LinearRegression
model = LinearRegression(cv=10) #10-fold CV
#X.values convert X to a matrix
model.fit(X.values, y)
```

```
## LinearRegression(cv=10)
```

```
ind = np.nonzero(model.coef_)
print("estimated non-zero: ", ind)
```

```
## estimated non-zero:  (array([ 0,  1,  5,  6,  8, 10, 11, 12, 13, 17], dtype=int6
```

```
print("features to keep: ", X.columns[ind])
```

```
## features to keep:  Index(['AtBat', 'Hits', 'Walks', 'Years', 'CHits', 'CRuns', '
##         'PutOuts', 'Division_W'],
##        dtype='object')
```

```
print("estimated intercept: ", model.intercept_)
```

```
## estimated intercept:  190.64538732291555
```

```
print("estimated coef: ", model.coef_[ind])
```

Model Selection by Cross-Validation

- We create an array that allocates each observation to one of k = 10 folds

```
import pandas as pd
import numpy as np
from abess.linear import LinearRegression
model = LinearRegression(cv=10) #10-fold CV
np.random.seed(2)
foldid=np.random.choice(np.arange(1,11), Hitters.shape[0], replace=True)
#X.values convert X to a matrix
model.fit(X.values, y, cv_fold_id=foldid)
```

```
## LinearRegression(cv=10)
```

```
ind = np.nonzero(model.coef_)
print("estimated non-zero: ", ind)
```

```
## estimated non-zero:  (array([ 0,  1,  5,  6, 10, 11, 12, 13, 14, 17], dtype=int6
```

Model Selection by Cross-Validation

```
print("features to keep: ", X.columns[ind])
```

```
## features to keep:  Index(['AtBat', 'Hits', 'Walks', 'Years', 'CRuns', 'CRBI', 'C
##        'Assists', 'Division_W'],
##       dtype='object')
```

```
print("estimated intercept: ", model.intercept_)
```

```
## estimated intercept:  206.5672284811826
```

```
print("estimated coef: ", model.coef_[ind])
```

```
## estimated coef:  [  -2.25568583    7.03787663    6.27932462  -16.74148577    0.8
##     0.65085148   -0.78829896    0.28248194    0.18722919 -123.22618929]
```

```
print(model.eval_loss_)
```

```
## 59238.14822937847
```

```
print(np.array(ind).size)
```

```
## 10
```

Model Selection by Cross-Validation

- Let's see the procedure of choosing a best model from the best 19 models
  - https://abess.readthedocs.io/en/latest/auto_gallery/1-glm/plot_1_LinearRegression.html#sphx-glr-auto-gallery-1-glm-plot-1-linearregression-py

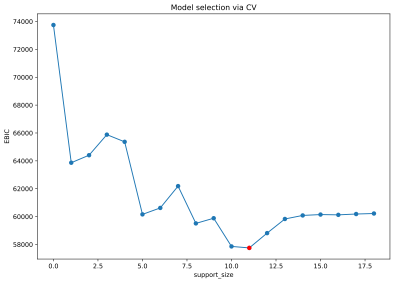
```
import numpy as np
from abess import LinearRegression
coef = np.zeros((19, 19))
ic = np.zeros(19)
loss = np.zeros(19)
for s in range(19):
    model = LinearRegression(cv=10,support_size=s+1)
    model.fit(X.values, y)
    coef[s, :] = model.coef_
    #ic[s] = model.ic_coef # a constant?
    loss[s]=model.eval_loss_
```

```
## LinearRegression(cv=10, support_size=1)
## LinearRegression(cv=10, support_size=2)
## LinearRegression(cv=10, support_size=3)
## LinearRegression(cv=10, support_size=4)
## LinearRegression(cv=10, support_size=5)
## LinearRegression(cv=10, support_size=6)
## LinearRegression(cv=10, support_size=7)
```

Model Selection by Cross-Validation

- We can compare the CV errors to see which model is the best.

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot(loss, 'o-')
min_loss=loss.min()
min_index=np.where(loss==min_loss)
plt.plot(min_index, min_loss, color='red', marker='o')
plt.xlabel('support_size')
plt.ylabel('EBIC')
plt.title('Model selection via CV')
plt.show()
```



License