Applied Statistical Methods

Cross Validation

Xuemao Zhang
East Stroudsburg University

March 27, 2023

Outline

- Model Complexity
- Cross-validation
  - Validation Set Approach
  - Leave-one-out cross-validation (LOOCV).
  - K-fold cross-validation.

Model Complexity

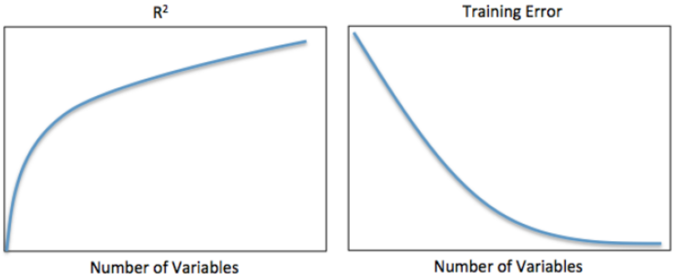
- When we fit a model, we use a **training set** of observations.
- We can evaluate the **training error**, i.e. the extent to which the model fits the observations used to train it. One way to quantify the training error is using the MSE (mean squared error)
- The training error is closely related to the R^2 for a linear model.
  - Big $R^2 \Leftrightarrow$ Small Training Error.
- Training error and R^2 are not good ways to evaluate a model's performance, because they will always improve as more variables are added into the model.

Model Complexity

- The problem? Training error and R^2 evaluate the model's performance on the training observations.
- If I had an unlimited number of features to use in developing a model, then I could surely come up with a regression model that fits the training data perfectly! Unfortunately, this model wouldn't capture the true signal in the data.
- We really care about the model's performance on **test observations** - observations not used to fit the model.

Model Complexity

- As we add more variables into the model. . .



- the training error decreases and the R^2 increases!

Model Complexity

- We really care about the model's performance on observations not used to fit the model!
  - Want to diagnose cancer for a patient not used in model training!
  - Want to predict risk of diabetes for a patient who wasn't used to fit the model!
- What we really care about:

$$(y_{test} - \hat{y}_{test})^2,$$

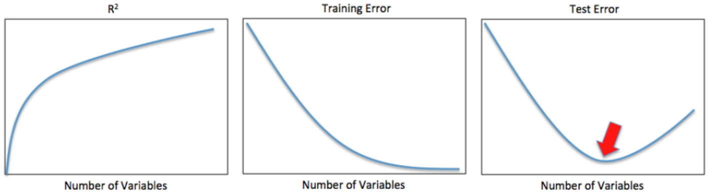
  where

$$\hat{y}_{test} = \hat{\beta}_0 + \hat{\beta}_1 X_{test,1} + \cdots + \hat{\beta}_p X_{test,p},$$

  and (X_{test}, y_{test}) was not used to train the model.
- The test error is the average of $(y_{test} - \hat{y}_{test})^2$ over a bunch of test observations.

Model Complexity

- As we add more variables into the model...



- the training error decreases and the R^2 increases!
- But the test error might not!

Model Complexity

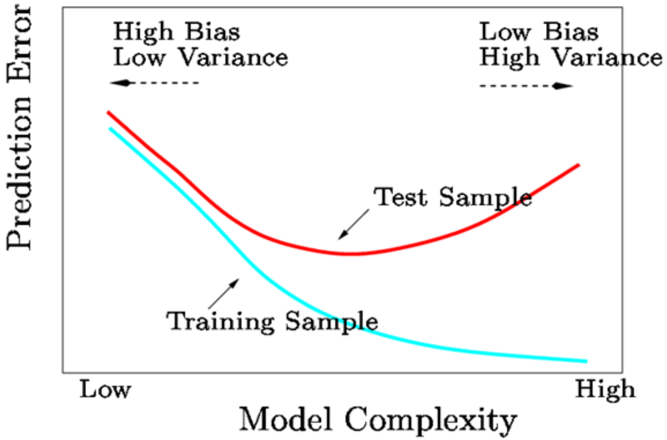
- As we fit more complex models - e.g. models with more variables - the training error will always decrease.
  - But the test error might not.
- Bias and Variance
  - As model complexity increases, if we were to repeat the experiment a huge number of times, the bias of $\hat{\beta}$ will decrease.
  - But as complexity increases, the variance of $\hat{\beta}$ will increase.
  - The test error depends on both the bias and variance:

$$\text{Test Error} = \text{Bias}^2 + \text{Variance}$$

  - There is a **bias-variance trade-off**. We want a model that is sufficiently complex as to have not too much bias, but not so complex that it has too much variance.
  - Fitting an overly complex model - a model that has too much variance - is known as overfitting.
    - ★ Avoid overfitting! Recall the figures about the true model and a overfitted model in last lecture.

Model Complexity

- A Really Fundamental Picture

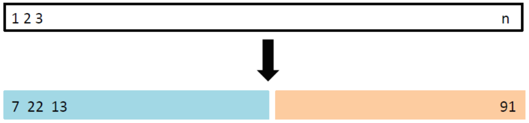


Model Complexity

- We must rely not on training error, but on test error, as a measure of model performance.
- We here consider a class of methods that estimate the test error by holding out a subset of the training observations from the fitting process, and then applying the statistical learning method to those held out observations.
- Three Ways to Estimate Test Error:
    - The validation set approach.
    - Leave-one-out cross-validation.
    - K-fold cross-validation.

Cross-validation - Validation Set Approach

- Split the *n* observations into two sets of approximately equal size. Train on one set, and evaluate performance on the other.



- The resulting validation-set error provides an estimate of the test error. This is typically assessed using MSE in the case of a quantitative response.

Cross-validation - Validation Set Approach

- 1. Split the observations into two sets of approximately equal size, a **training set** and a **validation set**.
    - a. Fit the model using the training observations. Let $\hat{\boldsymbol{\beta}}_{(train)}$ denote the regression coefficient estimates.
    - b. For each observation in the validation set, compute $e_i = (y_i - \boldsymbol{x}_i{}' \hat{\boldsymbol{\beta}}_{(train)})^2$
- 2. Calculate the total validation set error by summing the e_i's over all of the validation set observations.

Cross-validation - Validation Set Approach

- We begin by using the `np.random.choice()` function to split the set of observations into two halves, by selecting a random subset of 196 observations out of the original 392 observations
  - https://numpy.org/doc/stable/reference/random/generated/numpy.random.choice.html
- `numpy.in1d()`: Test whether each element of a 1-D array is also present in a second array.
  - https://numpy.org/doc/stable/reference/generated/numpy.in1d.html

```
import numpy as np
import pandas as pd
Auto=pd.read_csv("../data/Auto.csv")
np.random.seed(1)
train = np.random.choice(Auto.shape[0], 196, replace=False)
select = np.in1d(range(Auto.shape[0]), train)
print(select[0:20])
```

```
## [ True False False False  True  True  True False  True  True Fals
##   True  True  True False  True  True  True  True]
```

Cross-validation - Validation Set Approach

- We then fit a linear regression using only the observations corresponding to the training set.

```
from statsmodels.formula.api import ols
lm_fit=ols('mpg~horsepower',Auto[select]).fit()
```

Cross-validation - Validation Set Approach

```
print(lm_fit.summary())
```

```
##                             OLS Regression Results
## ==============================================================================
## Dep. Variable:                    mpg   R-squared:
## Model:                            OLS   Adj. R-squared:
## Method:                 Least Squares   F-statistic:
## Date:                Tue, 18 Apr 2023   Prob (F-statistic):                1.
## Time:                        14:49:47   Log-Likelihood:                    -
## No. Observations:                 196   AIC:
## Df Residuals:                     194   BIC:
## Df Model:                           1
## Covariance Type:            nonrobust
## ==============================================================================
##                  coef    std err          t      P>|t|      [0.025
## ------------------------------------------------------------------------------
## Intercept     40.3338      1.023     39.416      0.000      38.316
## horsepower    -0.1596      0.009    -17.788      0.000      -0.177
## ==============================================================================
## Omnibus:                        8.393   Durbin-Watson:
## Prob(Omnibus):                  0.015   Jarque-Bera (JB):
## Skew:                           0.516   Prob(JB):
```

Cross-validation - Validation Set Approach

- We now use the predict() method to estimate the response for all 392 observations, and we use the np.mean() function to calculate the MSE of the 196 observations in the validation set.

```
preds=lm_fit.predict(Auto)
square_error = (Auto['mpg'] - preds)**2
print(square_error[0:5])
```

```
## 0    2.495815
## 1    1.015656
## 2    2.602113
## 3    0.149687
## 4    0.966985
## dtype: float64
```

```
train=np.array(train)
print(np.mean(square_error[~select]))
```

```
## 23.361902892587224
```

Cross-validation - Validation Set Approach

- If we choose a different training set instead, then we will obtain somewhat different errors on the validation set.

```
np.random.seed(2)
train = np.random.choice(Auto.shape[0], 196, replace=False)
select = np.in1d(range(Auto.shape[0]), train)
preds=lm_fit.predict(Auto)
square_error = (Auto['mpg'] - preds)**2
train=np.array(train)
print(np.mean(square_error[~select]))
```

```
## 24.70242484684375
```

Cross-validation - Validation Set Approach

- Estimating the test error for the quadratic and cubic regressions.

```
lm2 = ols('mpg~horsepower + I(horsepower**2)', data = Auto[select])
preds = lm2.predict(Auto)
square_error = (Auto['mpg'] - preds)**2
print('--------Test Error for 2nd order--------')
```

```
## --------Test Error for 2nd order--------
```

```
print(square_error[~select].mean())
```

```
## 19.722533470491317
```

Cross-validation - Validation Set Approach

- `smf.glm()` is used to fit **generalized linear models** (used for fitting logistic regressioin later in the course). If we use `glm()` to fit a model without passing in the family argument, then it performs linear regression.
    - https://www.statsmodels.org/stable/generated/statsmodels.formula.api.glm.html
    - https://www.statsmodels.org/stable/generated/statsmodels.genmod.generalized_linear_model.GLM.html

```
import statsmodels.formula.api as smf
# GLM fit. Compare with OLS fit, the coeffs are the same
glm_fit = smf.glm('mpg~horsepower', data = Auto).fit()
print(glm_fit.params)
```

```
## Intercept     39.935861
## horsepower    -0.157845
## dtype: float64
```

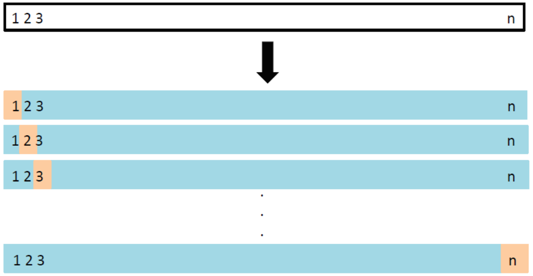
Cross-validation - Validation Set Approach

Drawbacks of validation set approach:

- The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
- In the validation approach, only a subset (training set) of the observations are used to fit the model.
  - This suggests that the validation set error may tend to overestimate the test error for the model fit on the entire data set.

Cross-validation - Leave-One-Out Cross-Validation

- LOOCV: Fit n models, each on $n-1$ of the observations. Evaluate each model on the left-out observation.



Cross-validation - Leave-One-Out Cross-Validation

- 1. For $i = 1, \ldots, n$:
  - a. Fit the model using observations $1, \ldots, i-1, i+1, \ldots, n$. Let $\hat{\boldsymbol{\beta}}_{(i)}$ denote the regression coefficient estimates.
  - b. For each observation in the validation set, compute $e_i = (y_i - \boldsymbol{x}_i{}' \hat{\boldsymbol{\beta}}_{(i)})^2$.
- 2. Calculate $\sum_{i=1}^{n} e_i^2$, the total CV (Cross-validation) error.
- Fortunately, the CV error can be calculated without requiring n separate regression runs.

$$\sum_{i=1}^{n} e_i^2 = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - h_{ii}} \right)^2,$$

where $\hat{y}_i$ is the ith fitted value from the original least squares fit, and h_{ii} is the leverage which is the ith diagonal element of the Hat matrix $\mathbf{H} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$.

Cross-validation - Leave-One-Out Cross-Validation

- Let us fit the regression model in sklearn

```
from sklearn.linear_model import LinearRegression
X = pd.DataFrame(Auto.horsepower)
y = Auto.mpg
model = LinearRegression()
model.fit(X, y)
```

```
## LinearRegression()
```

```
print(model.intercept_)
```

```
## 39.93586102117047
```

```
print(model.coef_)
```

```
## [-0.15784473]
```

Cross-validation - Leave-One-Out Cross-Validation

- We can easily use a for loop to split the data by leaving one point out each time and estimate the test error of the regression model.

```
import numpy as np
X = pd.DataFrame(Auto.horsepower)
y = Auto.mpg
n= Auto.shape[0] #sample size

lr = LinearRegression()
# Initialize lists to store actual and predicted values
actual = []
predicted = []

for i in range(n):
    X_train = X.drop(i)
    y_train = y.drop(i)
    #X_test = X.iloc[i,:].values.reshape(1, -1) # x must be 2-D
    #UserWarning: X does not have valid feature names
    X_test=pd.DataFrame({'horsepower':X.iloc[i]})
    y_test = y.iloc[i]
    #Then fit linear regression model
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)
    actual.append(y_test)
    predicted.append(y_pred[0])
```

```
## LinearRegression()
## LinearRegression()
## LinearRegression()
## LinearRegression()
## LinearRegression()
## LinearRegression()
```

Cross-validation - Leave-One-Out Cross-Validation

```
# Compute mean absolute error (MSE)
mae = np.mean(abs(np.array(actual) - np.array(predicted)) )
print(mae)
```

```
## 3.8487483321548384
```

Cross-validation - Leave-One-Out Cross-Validation

- Leave-one-out cross-validation (LOOCV) can be done using `sklearn.model_selection.LeaveOneOut`
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeaveOneOut.html
- `cross_val_score`: Evaluate a score by cross-validation.
  - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
  - Scoring parameters: https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

```python
import pandas as pd
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
Auto=pd.read_csv("../data/Auto.csv")
X = pd.DataFrame(Auto.horsepower)
y = Auto.mpg
model = LinearRegression()
loocv = LeaveOneOut() #define CV method to use
```

Cross-validation - Leave-One-Out Cross-Validation

```python
import numpy as np
#use LOOCV to evaluate model
scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error',
             cv=loocv, n_jobs=1)
print(np.mean(-scores))
```

```
## 3.8487483321548384
```

Cross-validation - Leave-One-Out Cross-Validation

- Another commonly used metric to evaluate model performance is the root mean squared error (RMSE).

```
X = pd.DataFrame(Auto.horsepower)
y = Auto.mpg
model = LinearRegression()
loocv = LeaveOneOut() #define CV method to use
scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',
                         cv=loocv, n_jobs=1)
print(np.mean(-scores))
```

```
## 24.231513517929226
```

Cross-validation - Leave-One-Out Cross-Validation

- For higher order polynomial fit, we need to generate polynomials in a systematic way
- `sklearn.preprocessing.PolynomialFeatures`: Generate polynomial and interaction features/terms.
    - https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

Cross-validation - Leave-One-Out Cross-Validation

- An `OrderedDict` is a dictionary subclass that **preserves the order** that keys were first inserted.
  - https://docs.python.org/3/library/collections.html
- Below shows how to fit an order 1 to 10 polynomial data and show the loo results; this step may take a few mins.
  - You need two `for` loops if you manually split the data

```
from sklearn.preprocessing import PolynomialFeatures
from collections import OrderedDict
A = OrderedDict()
loocv = LeaveOneOut()
for porder in range(1, 11, 1):
    poly = PolynomialFeatures(degree=porder)
    X_current = poly.fit_transform(X)
    model=LinearRegression()
    test = cross_val_score(model, X_current, y, cv=loocv,
           scoring ='neg_mean_squared_error', n_jobs=1)
    A[str(porder)] = np.mean(-test)

print(A)
```

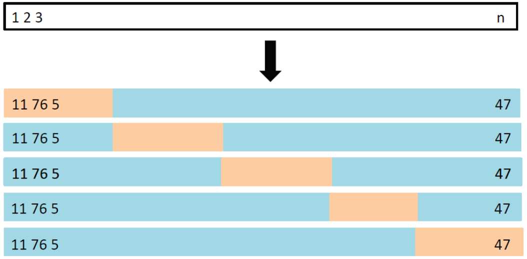
```
## OrderedDict([('1', 24.231513517929226), ('2', 19.248213124489407), ('3', 19.3349
```

Cross-validation - K-fold cross-validation

- Widely used approach for estimating test error.
- Estimates can be used to select best model, and to give an idea of the test error of the final chosen model.
- Idea is to randomly divide the data into K equal-sized parts. We leave out part k, fit the model to the other $K - 1$ parts (combined), and then obtain predictions for the left-out kth part.
- This is done in turn for each part $k = 1, 2, \ldots, K$, and then the results are combined.

Cross-validation - K-fold cross-validation

- 5-Fold Cross-Validation: Split the observations into 5 sets. Repeatedly train the model on 4 sets and evaluate its performance on the 5th.



Cross-validation - K-fold cross-validation

A generalization of K-fold cross-validation:

- 1. Split the n observations into K equally-sized folds.
- 2. For $k = 1, \ldots, K$:
  - a. Fit the model using the observations not in the kth fold.
  - b. Let e_k denote the test error for the observations in the kth fold.
- 3. Calculate $\sum_{k=1}^{K} e_k$, the total CV error.
- Note. Setting $K = n$ yields n-fold or leave-one out cross-validation (LOOCV).
- Since each training set is only $(K-1)/K$ as big as the original training set, the estimates of prediction error will typically be biased upward. And LOOCV estimate has high variance, as noted earlier.
  - $K = 5$ or $K = 10$ provides a good compromise for this bias-variance tradeoff.

Cross-validation - K-fold cross-validation

- Again, let's manually split the data into *k* folds and evaluate the test error of a regression model

```
X = pd.DataFrame(Auto.horsepower)
y = Auto.mpg
n= Auto.shape[0] #sample size
lr = LinearRegression()
# Initialize lists to store actual and predicted values
actual = []
predicted = []

k = 5 # the number of folds
fold_size = n//k #the number of samples in each fold
indices = np.random.permutation(n) # Shuffle the data

for i in range(k):
    # the indices of the samples in the i-th fold
    fold_indices = indices[i * fold_size: (i + 1) * fold_size]
    X_test = X.iloc[fold_indices]
    y_test= y.iloc[fold_indices]
    X_train = X.drop(fold_indices)
    y_train = y.drop(fold_indices)
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)
    actual.extend(y_test)
    predicted.extend(y_pred)
```

```
## LinearRegression()
## LinearRegression()
## LinearRegression()
## LinearRegression()
## LinearRegression()
```

Cross-validation - K-fold cross-validation

```
# Compute mean absolute error (MSE)
mae = np.mean(abs(np.array(actual) - np.array(predicted)) )
print(mae)
```

```
## 3.8638548865290328
```

Cross-validation - K-fold cross-validation

- K-fold validation is exactly same as LOO with CV defined by `KFold`.
- `sklearn.model_selection.KFold`: K-Folds cross-validator. Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).
  - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

```
import numpy as np
from sklearn.model_selection import KFold
from sklearn.preprocessing import PolynomialFeatures
from collections import OrderedDict
np.random.seed(2)
A = OrderedDict()
n_split = 10
k_fold = KFold(n_splits=10, random_state=1, shuffle=True)
for porder in range(1, 11, 1):
    poly = PolynomialFeatures(degree=porder)
    X_current = poly.fit_transform(X)
    model=LinearRegression()
    test = cross_val_score(model, X_current, y, cv=k_fold,
           scoring ='neg_mean_squared_error', n_jobs=1)
    A[str(porder)] = np.mean(-test)

print(A)
```

```
## OrderedDict([('1', 24.097675731883065), ('2', 19.178889864889648), ('3', 19.21385952367195), ('4', 19.212807
```

Cross-validation - K-fold cross-validation

```
A2=pd.DataFrame.from_dict(A,orient='index')
A2
```

```
##              0
## 1   24.097676
## 2   19.178890
## 3   19.213860
## 4   19.212807
## 5   18.757985
## 6   18.632038
## 7   18.820835
## 8   18.975736
## 9   18.937633
## 10  18.792513
```

License