Model Validation.

W. Wang[1]

[1]Department of Mathematics
University of Houston

MATH 4323

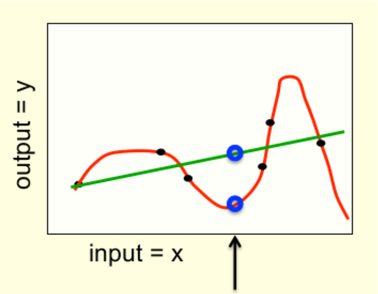
Motivation

Validation techniques are motivated by two fundamental problems in statistical learning: model selection and performance estimation.

- **Model selection.** Almost all learning techniques have one or more free parameters. For instance, the number of neighbors in a *KNN* classification rule. How do we select the "optimal" parameter(s) or model for a given classification problem?
- **Model acessment/Performance estimation.** Once we have chosen a model, how do we estimate its performance? Performance is typically mesaured by the "TRUE" error rate, the classifier's error rate on the "entire" population.

Training error, overfitting.

Training error - not a good metric of model performance. (Why?)



Sometimes good training test performance is more indicative of overfitting ($\implies$ fitting the noise instead of true signal) rather than of a good generalizable model.

Validation Set Approach.

Need an out-of-sample error (meaning out of **training** sample). How to obtain?

One way is **validation** (or **hold-out**) **set** approach:

1. (Randomly) Divide data set into two parts:
   - Training set
   - Validation set

2. Fit the model on training data, and use the fitted model to predict responses for validation data.

3. The validation set error rate $\approx$ test error rate.

Validation Set: S&P 500 example.

Example. *Smarket* data set contains the daily movements in the Standard & Poor's 500 (S&P) stock index over a 5-year period between 2001 and 2005. We will use

- the S&P 500 daily price changes of previous five days ($Lag1, Lag2, Lag3, Lag4, Lag5$) and yesterday's trading volume($Volume$) as predictors,
- today's price direction ("Down"/"Up") as the response.

Given that it is a time series data, we are only interested in **forecasting** the future while using the past:

- Use first t days (obs. # $1, 2, \ldots, t$) as training set,
- remaining $n - t$ days (obs. # $t + 1, t + 2, \ldots, n$) - validation set.

Validation Set: S&P 500 example.

Example (cont'd). Following suit, let's use

- first four years (2001-2004) as training data, and
- year 2005 as validation set.

There's 250 trading days per year $\implies$

- obs. # $1, 2, \ldots, 1000$ - training set
- obs. # $1001, 1002, \ldots, 1250$ - validation set.

```
library(ISLR)
library(class)

n <- nrow(Smarket)

train <- 1:1000
test <- c(1:n)[-train]
```

Validation Set: S&P 500 example.

```
X.train <- Smarket[train,
                   c("Lag1","Lag2","Lag3","Lag4","Lag5","Volume")]
y.train <- Smarket[train, "Direction"]

X.test <- Smarket[test,
                  c("Lag1","Lag2","Lag3","Lag4","Lag5","Volume")]
y.test <- Smarket[test, "Direction"]
```

Running KNN algorithm with $K = 1$

```
knn.pred <- knn(train=X.train,
                test=X.test,
                cl = y.train,
                k=1)
mean(knn.pred != y.test)
[1] 0.488
```

results into a test error of 48.8% (quite a sobering number after the perfect 0% training error).

Validation Set: Choosing Best K.

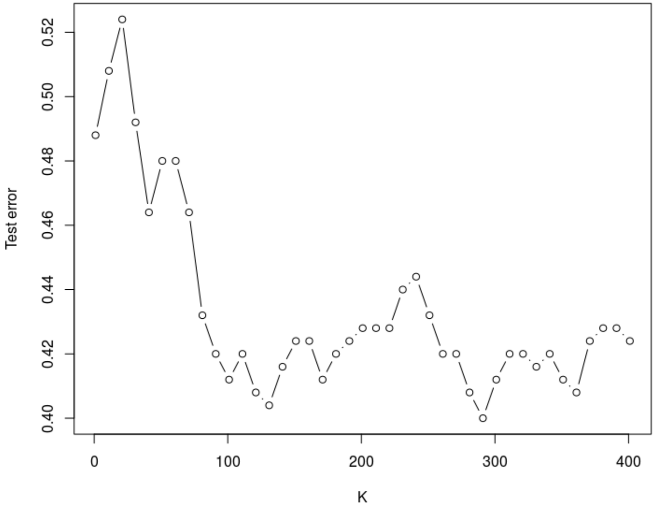
Validation set approach allows us to choose the **best** K.

Example (cont'd). Let's train KNN with $K = 1, 11, 21, \ldots, 391, 401$, for first 1000 days, and compare K values via resulting test errors.

```
K.set <- seq(1,401, by=10)
knn.test.err <- numeric(length(K.set))

set.seed(1)
for (j in 1:length(K.set)){
  knn.pred <- knn(train=X.train, test=X.test,
                  cl=y.train,
                  k=K.set[j])
  knn.test.err[j] <- mean(knn.pred != y.test)}

min(knn.test.err)                 # Smallest test error.
[1] 0.4
which.min(knn.test.err)           # The index of best K.
[1] 30
K.set[which.min(knn.test.err)] # The best K value.
[1] 291
```

Validation Set: Choosing Best K.

```
plot(K.set, knn.err, type='b')
```



Validation Set: Variable Subset Selection.

Validation set approach can also be used to compare **predictor subsets**.

Example (cont'd, see source code for details). Let's train KNN with $K = 291$ over first 1000 trading days, but for various subsets of

$$\{"Lag1", "Lag2", "Lag3", "Lag4", "Lag5", "Volume"\}$$

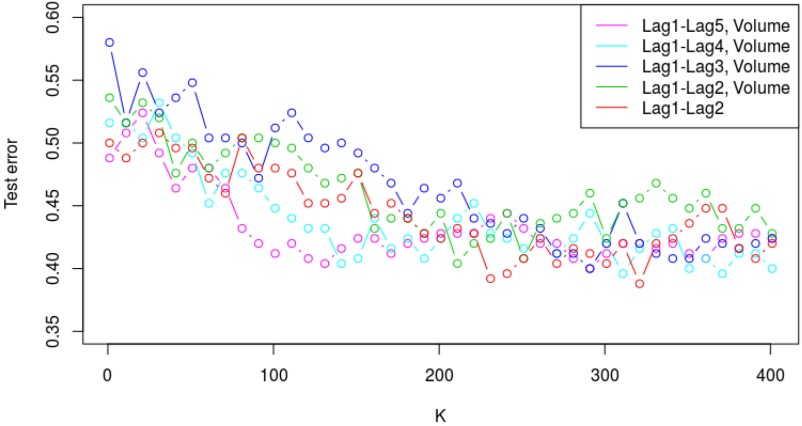
Predictor Subset	Test Error
$\{"Lag1", "Lag2", "Lag3", "Lag4", "Lag5", "Volume"\}$	0.404
$\{"Lag1", "Lag2", "Lag3", "Lag4", "Volume"\}$	0.448
$\{"Lag1", "Lag2", "Lag3", "Volume"\}$	0.400
$\{"Lag1", "Lag2", "Volume"\}$	0.456
$\{"Lag1", "Lag2"\}$	0.416

For $K = 291$, the $\{"Lag1", "Lag2", "Lag3", "Volume"\}$ predictor subset yielded best test performance.

Validation Set: Selecting Variable Subset & K.

Example (cont'd). For a more thorough model search, we could simultaneously calculate test error over:

- all $K = 1, 10, 21, \ldots, 391, 401$, and
- all predictor subsets.



Validation Set: Orange Juice (OJ, from *library*(*ISLR*)) example.

Example. *OJ* contains data on whether

- customers purchased "Citrus Hill" or "Minute Maid" orange juice (binary response, "CH" or "MM"),
- depending on various factors such as price, discounts, store information, etc.

```
> head(OJ, 4)
  Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH ...
1       CH            237       1    1.75    1.99   0.00 ...
2       CH            239       1    1.75    1.99   0.00 ...
3       CH            245       1    1.86    2.09   0.17 ...
4       MM            227       1    1.69    1.69   0.00 ...
```

We'll use KNN with $K = 5$ to predict if a customer buys "CH" or "MM".

Issue #1: *StoreID* appears to be an expendable variable (Why?)

```
OJ$StoreID <- NULL
```

Validation Set: OJ example.

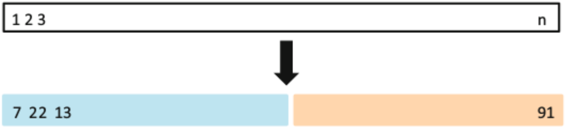
Unlike stock market data (which was time series), *OJ* is a cross-sectional data set $\Longrightarrow$ there're no natural "past"/"future" values.

For cross-sectional data, training/validation set subdivision has to be picked **randomly**.

Example (cont'd). If you wished to randomly split n observations into

- training set, 50% of observations;
- test set, remaining 50% of observations;

this subdivision may look as follows



First - randomly reshuffle your data, **second** - split it.

Validation Set: OJ example.

Example (OJ, cont'd). We randomly subdivide *OJ* data set into

- training set, 80% of observations;
- test set, remaining 20% of observations.

```
set.seed(1) # That's for reproducible train/test subdivisions.

n <- nrow(OJ)
train <- sample(1:n, 0.8*n)
X.train <- OJ[train, -1]; y.train <- OJ[train, "Purchase"]
X.test <- OJ[-train, -1]; y.test <- OJ[-train, "Purchase"]

set.seed(1)  # That's for reproducible KNN results.
knn.pred <- knn(train=X.train, test=X.test,
                cl = y.train,
                k=5)
```

Validation Set: Orange Juice (OJ) example.

Issue #2: Running *knn*() will give you an error due to one predictor being non-numeric: $Store7 = "Yes" / "No"$

```
X.train <- OJ[,-1]
str(X.train)
...
$ PriceDiff      : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 ...
$ Store7         : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 ...
$ PctDiscMM      : num  0 0.151 0 0 0 ...
 ...
```

Solution: Convert it into a numerical dummy variable, also known as one-hot encoding.

$$Store7 = \begin{cases} 1, & \text{Store7 = "Yes"}, \\ 0, & \text{Store7 = "No"} \end{cases}$$

```
X.train$Store7 <- ifelse(X.train$Store7 == "Yes", 1, 0)
```

Validation Set: Unstable results.

Example (cont'd). Running KNN with $K = 5$ (code from slides 13-14):

```
mean(knn.pred != y.test)
[1] 0.2663551
```

26.6% test error rate for this particular train/test subdivision.

Issue #3: Resulting test error heavily depends on the random train/test subdivision $\Longrightarrow$ can't rely on results based on just one validation set.

```
 set.seed(1)
 for (j in 1:5){
  ... Code from slide 12 (bar the set.seed(1) command) ...
  print(mean(knn.pred != y.test))}
[1] 0.2663551
[1] 0.2757009
[1] 0.2523364
[1] 0.2149533
[1] 0.2616822
```

Leave-One-Out Cross-Validation (LOOCV).

Leave-one-out cross-validation (**LOOCV**).

In LOOCV, for **each data point** i, $i = 1, \ldots, n$, we

1. Split data into two subsets:
   - Training set: $(x_1, y_1), \ldots, (x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), \ldots, (x_n, y_n)$
   - Test "set" of just one observation: (x_i, y_i).
2. Use training set to fit the model and produce prediction $\hat{y}_i$.
3. Calculate the misclassification error: $Err_i = \mathbb{I}(y_i \neq \hat{y}_i)$

The **LOOCV estimate** for test (squared) error is

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} Err_i$$

Leave-One-Out Cross-Validation (LOOCV).

Illustration of data subdivision for LOOCV as opposed to validation set approach (where training and test set were of comparable sizes).

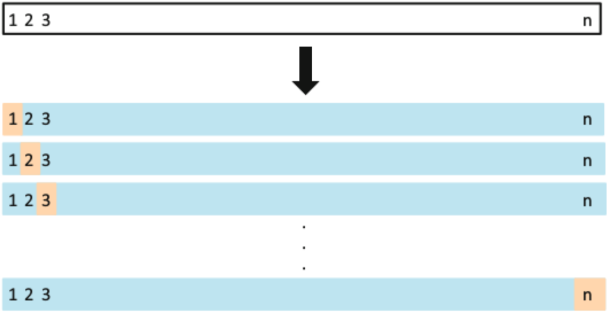


FIGURE 5.3. *A schematic display of LOOCV. A set of n data points is repeatedly split into a training set (shown in blue) containing all but one observation, and a validation set that contains only that observation (shown in beige). The test error is then estimated by averaging the n resulting MSE's. The first training set contains all but observation 1, the second training set contains all but observation 2, and so forth.*

Leave-One-Out Cross-Validation: Pros & Cons.

A few advantages for LOOCV over validation set approach:

- Yields test error estimates that are much more stable.
- Uses nearly whole data set ($n - 1$ out of n total observations) at each step (more data $\implies$ less bias in test error estimate).

Issues with LOOCV:

- computationally demanding
- bias-variance trade-off (more later...)

Both validation set and LOOCV are very general methods, and can be used with any predictive model (KNN, linear/logistic regression, SVM, random forests, neural networks, etc).

LOOCV for KNN in *R*: *knn.cv*() function.

Function to conduct LOOCV for KNN methods in *R* is *knn.cv*(). It does not require a test set (*test*) to be supplied as an argument.

Example (cont'd). LOOCV on KNN with $K = 1$ for *OJ* data:

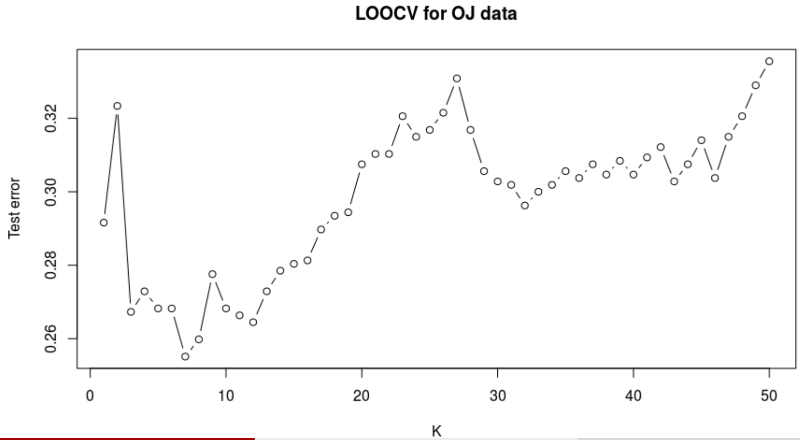
```
X.train <- OJ[,-1]
y.train <- OJ[,"Purchase"]

set.seed(1)

knn.pred <- knn.cv(train=X.train,
                   cl = y.train,
                   k=5)
print(mean(knn.pred != y.train))
[1] 0.2682243
```

A (much more stable) test error estimate is 26.8%.

LOOCV: Choosing K.

Given that the test error estimate resulting from LOOCV is always stable, we can safely use it to compare KNN models for different K:

Example (cont'd). *OJ* data LOOCV test error estimates for KNN with $K = 1, 2, \ldots, 50$ (see the source code for details). $K = 7$ **wins**.



K-fold Cross-Validation.

***K*-fold Cross-Validation** is an alternative to LOOCV where

1. Data is randomly divided into K subsets of $\approx$ same size n_K.
2. For **each subset** j, $j = 1, \ldots, K$, we
   - use it as a validation set, while
   - using other $K - 1$ subsets to train the model.
   - Calculate $Err_j = \frac{1}{n_k} \sum_{i \in \{validation\ set\}} \mathbb{I}(y_i \neq \hat{y}_i)$.
3. The K-fold CV (squared) test error estimate is

$$CV_{(k)} = \frac{1}{K} \sum_{j=1}^{K} Err_j$$

Question: For what K does K-fold CV become a leave-one-out CV?

$K = n$, k-fold CV $\Leftrightarrow$ LOOCV

K-fold Cross-Validation.

Illustration of random data subdivision into training and testing subsets for 5-fold CV (as opposed to LOOCV and validation set approaches):

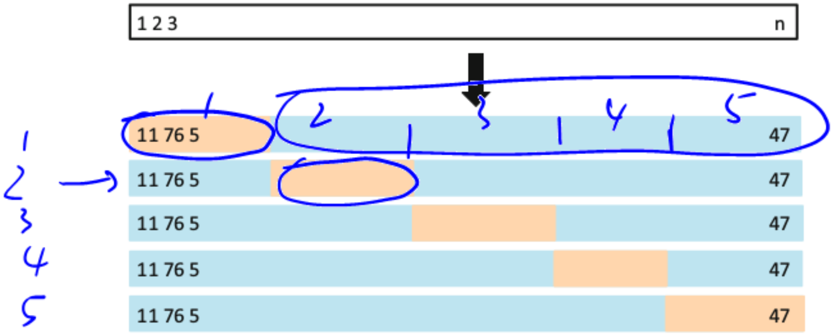


FIGURE 5.5. *A schematic display of 5-fold CV. A set of n observations is randomly split into five non-overlapping groups. Each of these fifths acts as a validation set (shown in beige), and the remainder as a training set (shown in blue). The test error is estimated by averaging the five resulting MSE estimates.*

K- fold Cross-Validation for KNN in *R*: library(caret)

To conduct *K*- fold cross validation for KNN methods in *R*, we can use *library*(*caret*).

Example (cont'd). *K*-fold cross validation on KNN for *OJ* data:

10-fold CV

```
library(caret)
train.control <- trainControl(method  = "CV", 10)
train(Purchase ~.,
              method     = 'knn',
              tuneGrid   = expand.grid(k = 1:20),
              trControl  = train.control,
              metric     = "Accuracy",
              data       = OJ)
```

glm / svm

c(1, 5, 7, 13, ...)

K-fold CV test error rate estimate is 25.99%.

K-fold Cross-Validation: Pros & Cons.

Several advantages to K-fold CV over LOOCV:

- Computational: Doing *LOOCV* ($\equiv K$-fold CV for $K = n$) is tough for computationally intensive models, as opposed to 5- or 10-fold CV, especially for large n. Here, model is fit only $K << n$ times.
- K-fold CV doesn't lose in estimation quality to *LOOCV*.
- The variability in K-fold error estimates is negligible.

Bias-Variance Tradeoff.

Computational issues aside, K-fold CV also often gives more accurate estimates of the test error rate than does LOOCV. This has to do with a **bias-variance trade-off**:

- *LOOCV* estimates model's test error with less bias, as it uses $\approx$ all observations ($n-1$ out of n) to obtain the estimate at each run.
- Nonetheless, the sample-to-sample variation of *LOOCV* is higher than that for K-fold CV $\implies$ if we were to use a different sample from the population, then LOOCV test error estimate would (on average) change more drastically compared to K-fold CV.

  Why? In *LOOCV* we train n models on an almost identical set of observations $\implies$ error estimates are highly correlated and are very sample-dependent.

  In contrast, for K-fold CV we average the outputs of K models trained on less correlated subsets (overlap is smaller).

How many folds are needed

- **With a large number of folds:** The bias of the true arror rate estimator will be small (Estimation will be more accurate); Computational time will be very large as well.
- **With a small number of folds:** Computation time are reduced; variance of estimator will be smaill; Bias will be large.
- In practice, the choice of the number of folds depends to the size of the dataset.
- $K = 5$ or 10 were **shown empirically** to yield optimal test error estimates.

Three-way data splits

If model selection and true error estimates are to be computed simultaneously, the data needs to be divided into three disjoint sets.

- Training set: a set of observations used for learning: to fit the parameters of the classifier
- Validation set: a set of examples used for tuning the parameters of a classifier
- Test set: a set of examples used **only** to assess the performance of a fully-trained classifier. (**Note**: After assessing the final model with the test set, we can not further tune the model!)

Data scaling during model validation

See lab 3 material

Confusion Matrix

$$\text{error rate} = \frac{122}{250} = 0.488$$

$63/113$

A **confusion matrix** is a table that is often used to describe the performance of a classification model (or “classifier”) on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm.

	A	B	C
A			
B			
C			

```
knn.pred <- knn(train=X.train,
                test=X.test,
                cl = y.train,
                k=1)
mean(knn.pred != y.test)
[1] 0.488
table(knn.pred, y.test)
                y.test
knn.pred Down   Down Up
         UP       50 63
                  59 78
```

→ true label

predicted

total correct predictions: $50+78=128$

$59+63=122$