

Methods of Cross Validation

Week 5

Dr. Nouman M Durrani

Acknowledgement to all authors whose materials have been used

Basic Concepts: Training and Testing

- **Model:** A **model** is a mathematical formula with **a number of parameters** that need to be **learned from the data**
 - **Fitting** a model to the data is a process known as **model training**
 - **Testing data** is the unseen data for which predictions have to be made
 - **Test data** is used only to assess performance of model
 - **Training data's** output is available to model

Basic Concepts: Training and Testing

- **Success:** instance (record) class is predicted correctly
- **Error:** instance class is predicted incorrectly
- **Error rate:** a percentage of errors made over the whole set of instances (records) used for testing
- **Predictive Accuracy:** a percentage of well classified data in the testing data set.

REMEMBER: We must know the classification (class attribute values) of all instances (records) used in the test procedure.

Training and Testing

Example:

- Testing Rules (testing record #1) = record #1.class –Succ
- Testing Rules (testing record #2) not= record #2.class –Error
- Testing Rules (testing record #3) = record #3.class –Succ
- Testing Rules (testing record #4) = instance #4.class –Succ
- Testing Rules (testing record #5) not= record #5.class –Error

Error rate: 2 errors: #2 and #5

Error rate = $2/5=40\%$

Predictive Accuracy: $3/5 = 60\%$

Confusion Matrix

- A confusion matrix is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known.
- It helps in visualization the performance of an algorithm.

	Predicted Negative	Predicted Positive
Actual Negative	True Negative	False Positive
Actual Positive	False Negative	True Positive

Confusion Matrix

- TN is the number of correct predictions that an instance is negative
- TP is the number of correct predictions that an instance is positive
- FN is the number of incorrect predictions that an instance is negative
- FP is the number of incorrect predictions that an instance is positive

Confusion Matrix

- Confusion Matrix for the following results is:

ID	Actual	Predicted
1	1	1
2	0	0
3	1	1
4	1	0
5	0	1

Actual	Predicted		
		Negative	Positive
	Negative	True Negative 1	False Positive 1
	Positive	False Negative 1	True Positive 2

```
# Example of a confusion matrix in Python
from sklearn.metrics import confusion_matrix

expected = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0]
predicted = [1, 0, 0, 1, 0, 0, 1, 1, 1, 0]
results = confusion_matrix(expected, predicted)
print(results)
```

Output Confusion Matrix: $\begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}$

Confusion Matrix

Several standard terms have been defined for the 2 class matrix

- The *accuracy* (*AC*) is the proportion of the total number of predictions that were correct

$$Accuracy = \frac{TN + TP}{TN + FN + TP + FP}$$

- Accuracy = 3 / 4 = 75%

Confusion Matrix

- The *True positive rate (TPR)* is the proportion of positive cases that were correctly identified

$$TPR = \frac{TP}{TP + FN}$$

- The *false positive rate (FPR)* is the proportion of negatives cases that were incorrectly classified as positive

$$FPR = \frac{FP}{FP + TN}$$

- TPR or **recall** = $2 / 3 = 66.7\%$
- FPR = $0 / 1 = 0 \%$

	Predicted Negative	Predicted Positive
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

Confusion Matrix

- The *true negative rate* (TNR) is defined as the proportion of negatives cases that were classified correctly,

$$TNR = \frac{TN}{FP + TN}$$

- The *false negative rate* (FNR) is the proportion of positives cases that were incorrectly classified as negative

$$FNR = \frac{FN}{FN + TP}$$

- $TNR = 1 / 1 = 100\%$
- $FNR = 1 / 3 = 33.3\%$

	Predicted Negative	Predicted Positive
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

Confusion Matrix

- *Precision* (P) is the proportion of the predicted positive cases that were correct,

$$precision = \frac{tp}{tp + fp}$$

- Precision = $2/2 = 100\%$
- F measure is harmonic mean of precision and recall

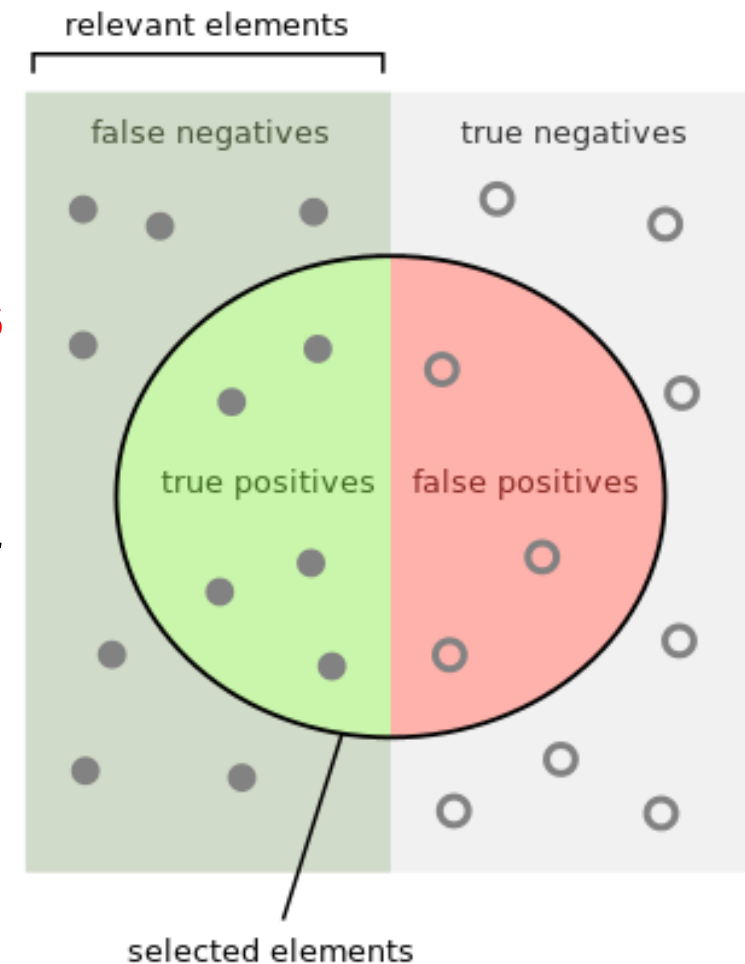
$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- $F1 = (2 * 1 * 0.667)/(1+0.667) = 0.8$

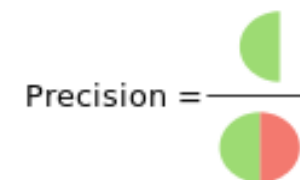
Exercise

		Actual	
		Negative	Positive
	Predicted		
	Negative	9760	40
	Positive	140	60

- **Precision** (also called positive predictive value) is the fraction of **relevant instances** among the **retrieved instances**
- **Recall** (also known as sensitivity) is the fraction of the **total amount of relevant instances** that were **actually retrieved instances**
- Suppose a computer program for recognizing dogs in photographs identifies 8 dogs in a picture containing 12 dogs and some cats. Of the 8 identified as dogs, 5 actually are dogs (true positives), while the rest are cats (false positives)
 - The program's precision is $5/8$ while its recall is $5/12 (=5/(8+4$ for failing to recognize the other 4 cats))
- When a search engine returns 30 pages only 20 of which were relevant while failing to return 40 additional relevant pages, its precision is $20/30 = 2/3$ while its recall is $20/60 = 1/3$. So, in this case, precision is "how useful the search results are", and recall is "how complete the results are"



How many selected items are relevant?



How many relevant items are selected?



Bias and variance of a statistical estimate

- **Consider the problem of estimating a parameter α of an unknown distribution G**
 - To emphasize the fact that α concerns G we will refer to it as $\alpha(G)$
- **We collect N examples $X=\{x_1, x_2, \dots, x_N\}$ from the distribution G**
 - These examples define a discrete distribution G' with mass $1/N$ at each of the examples
 - We compute the statistic $\alpha'=\alpha(G')$ as an estimator of $\alpha(G)$
 - In the context of this lecture, $\alpha(G')$ is the estimate of the true error rate for our classifier
- **How good is this estimator?**
- **The “goodness” of a statistical estimator is measured by**
 - BIAS: How much it deviates from the true value

$$\text{Bias} = E_G[\alpha'(G)] - \alpha(G)$$

$$\text{where } E_G[X] = \int_{-\infty}^{+\infty} x g(x) dx$$

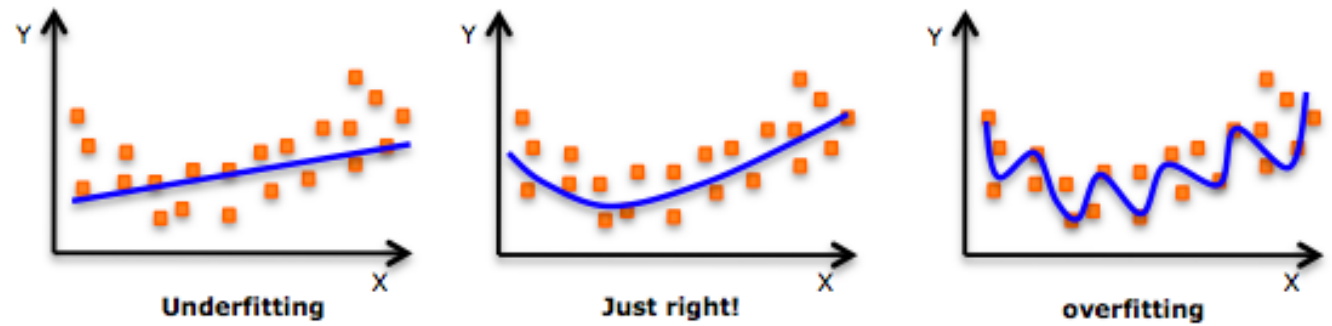
- VARIANCE: How much variability it shows for different samples $X=\{x_1, x_2, \dots, x_N\}$ of the population G

$$\text{Var} = E_G[(\alpha' - E_G[\alpha'])^2]$$

Bias and Variance

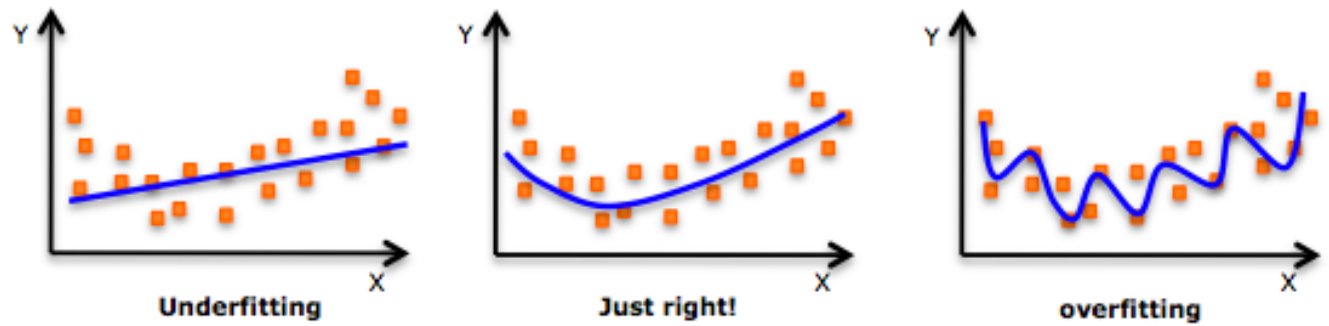
- An **estimator** is a rule for calculating an **estimate** of a given quantity based on observed data: thus the rule (the **estimator**), the quantity of interest (the estimand) and its result (the **estimate**) are distinguished.
- The **bias** error is an error from erroneous assumptions in the learning algorithm.
 - *Bias* is the simplifying assumptions made by the model to make the target function easier to approximate.
 - High **bias** can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
- The **variance** is an error from sensitivity to small fluctuations in the training set.

Overfitting



- **Overfitting** occurs when a statistical model or machine learning algorithm **captures the noise** of the data
- Intuitively, overfitting occurs when the model or the algorithm fits the data too well
- Specifically, overfitting occurs if the model or algorithm shows **low bias** but **high variance**
- **Overfitting** is often a result of an excessively complicated model
- It can be prevented by fitting multiple models and using validation or cross-validation to compare their predictive accuracies on test data

Underfitting

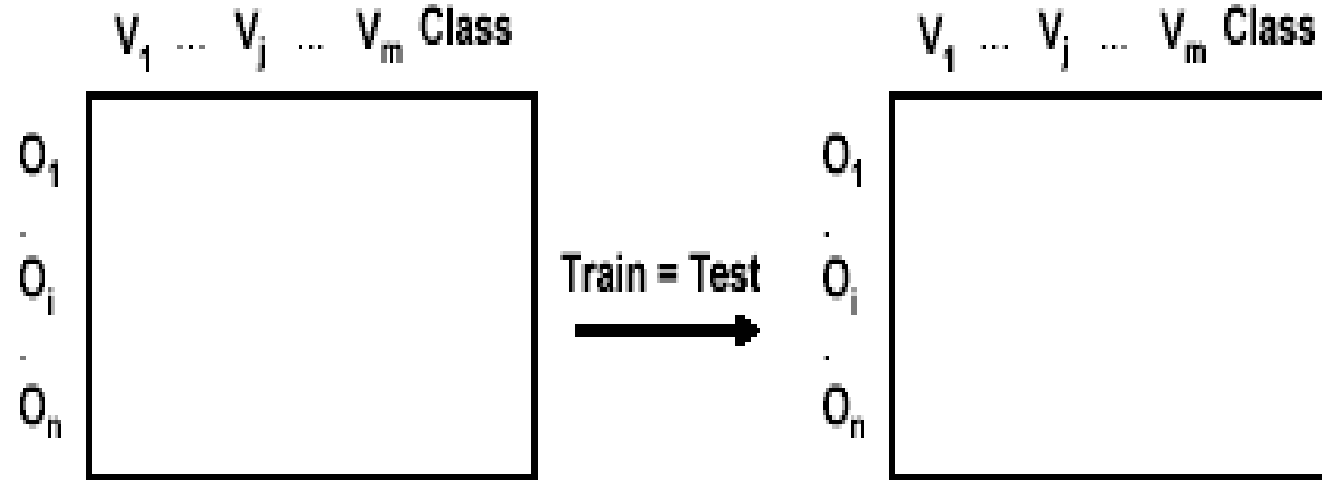


- **Underfitting** occurs when a statistical model or machine learning algorithm **cannot** capture the underlying trend of the data
- Intuitively, underfitting occurs when the model or the algorithm does not fit the data well enough
- Specifically, underfitting occurs if the model or algorithm shows **low variance** but **high bias**
- Underfitting is often a result of an excessively simple model

Both overfitting and underfitting lead to **poor predictions** on new data sets

Predictive Accuracy Evaluation Method: Resubstitution (N,N)

- If all the data is used for training the model and the error rate is evaluated based on outcome vs. actual value from the same training data set, this error is called the **resubstitution error**.
- This technique is called the resubstitution validation technique.



Re-substitution Error Rate

- **Re-substitution error rate** is obtained from training data
- **Training Data Error:** uncertainty of the rules
- The error/error rate on the training data is not always 0% because algorithms involve different (often statistical) parameters and measures that lead to uncertainties
- Re-substitution error rate indicates only how good (bad) are our results (rules, patterns) on the TRAINING data;
 - expresses some knowledge about the algorithm used.

Re-substitution Error Rate

- Re-substitution Error Rate is usually used as the performance measure:
 - The training error rate reflects imprecision of the training results:
the lower, the better
 - Predictive accuracy reflects how good are the training results with respect to the test data:
the higher, the better

(N:N) re-substitution does not compute predictive accuracy

The error on the training data is NOT a good indicator of performance on future data since it does not measure any not yet seen data.

- Solution: Split data into training and testset

Why Cross Validation in Machine Learning?

- In machine learning, we couldn't fit the model on the **training data** and can't say that the model will work accurately for the **real data**.
- For this, we must assure that our **model got the correct patterns from the data**, and it is not getting up too much noise.
- For this purpose, we use the cross-validation technique.

What is Cross-validation?

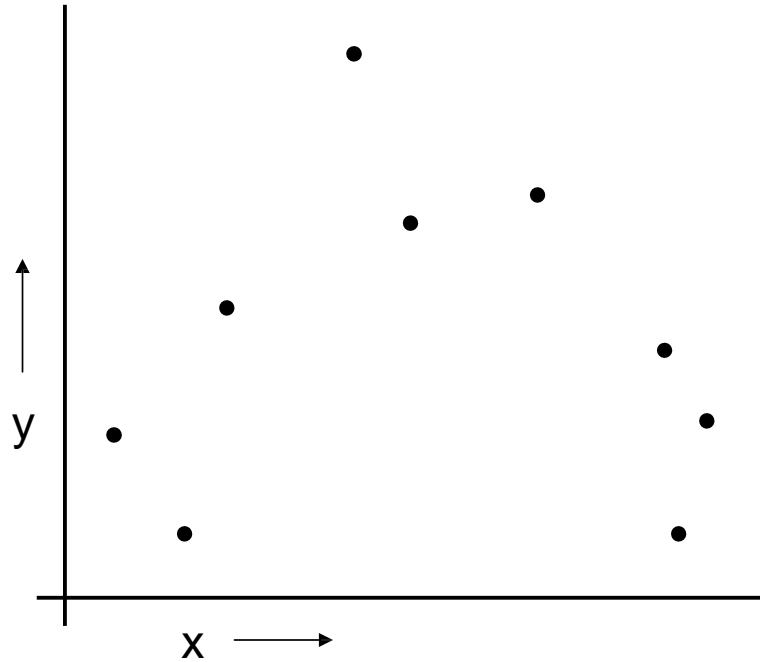
- Cross-validation is a technique in which we **train our model using the subset of the data-set** and then **evaluate using the complementary subset of the data-set**.
- The three steps involved in cross-validation are as follows :
 - Reserve some portion of sample data-set.
 - Using the rest data-set train the model.
 - Test the model using the reserve portion of the data-set.

The **main methods** are:

- Holdout ($2N/3$; $N/3$)
- Leave-one-out ($N-1$; 1)
- x-fold cross-validation ($N-N/x$; N/x)

where N is the number of records (instances) in the dataset

Example: A Regression Problem



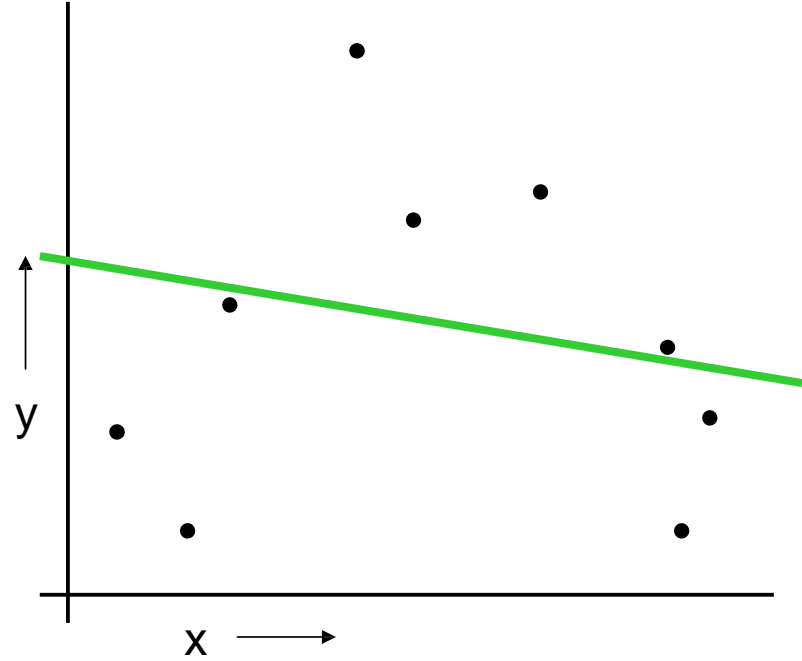
$$y = f(x) + \text{noise}$$

Can we learn f from this data?

The goal is to fit a line to a set of distributed data points.

Let's consider three methods...

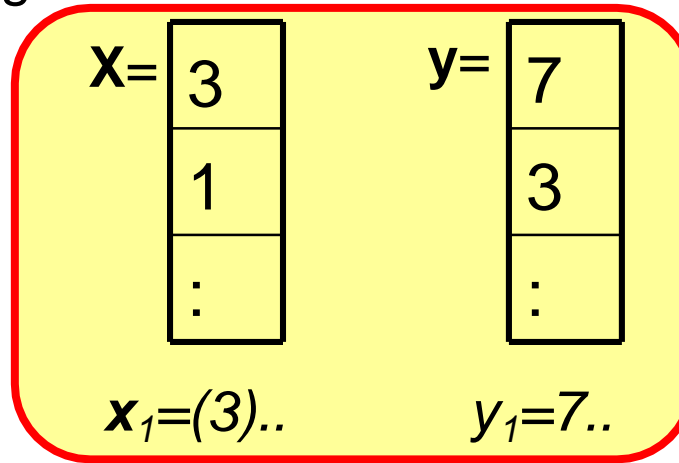
Linear Regression



Linear Regression

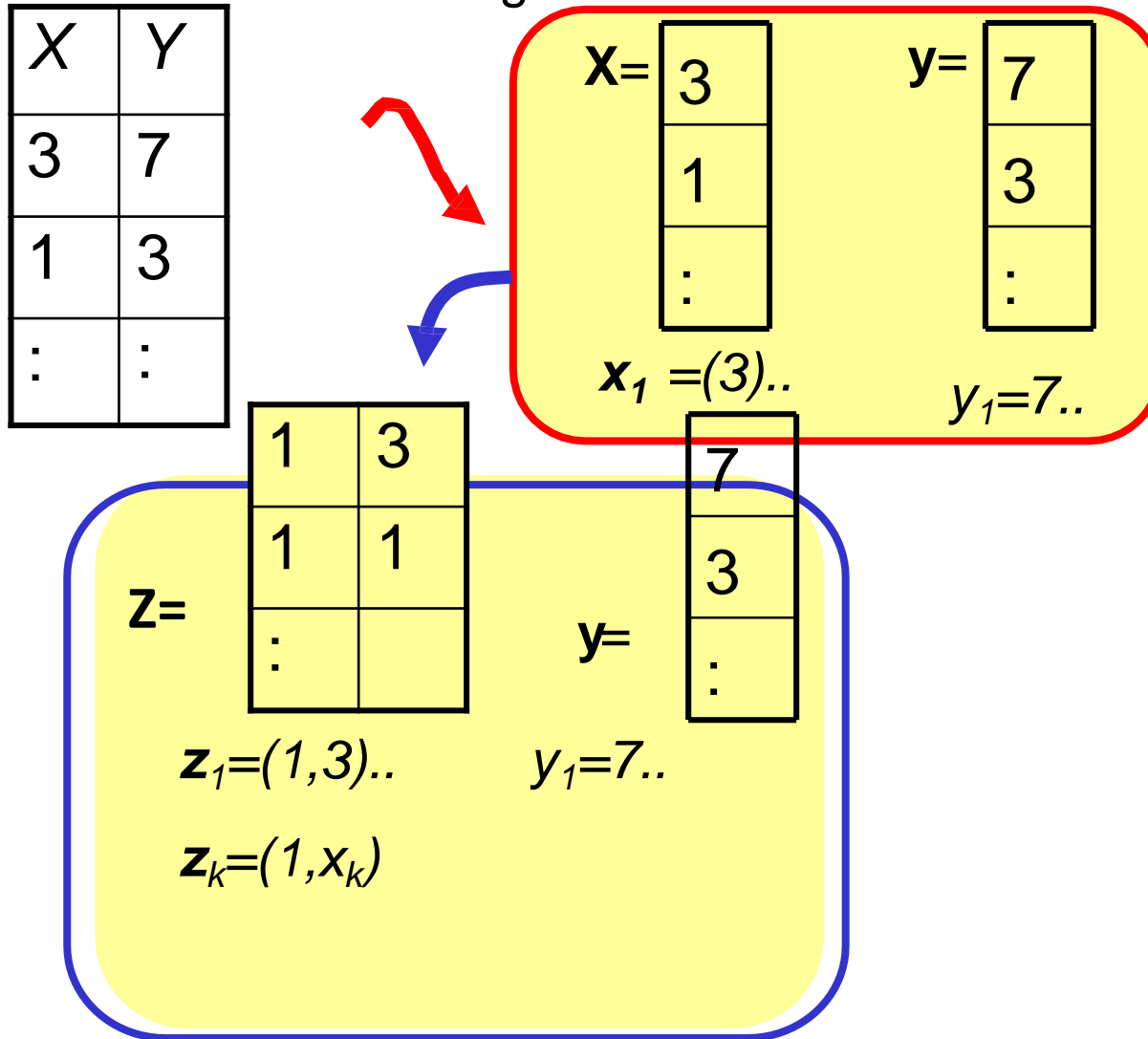
Univariate Linear regression with a constant term:

X	Y
3	7
1	3
\vdots	\vdots



Linear Regression

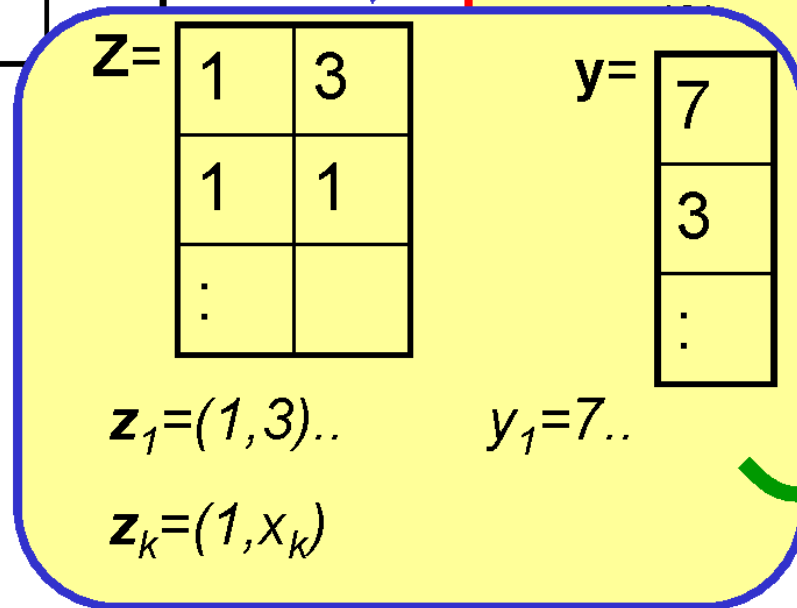
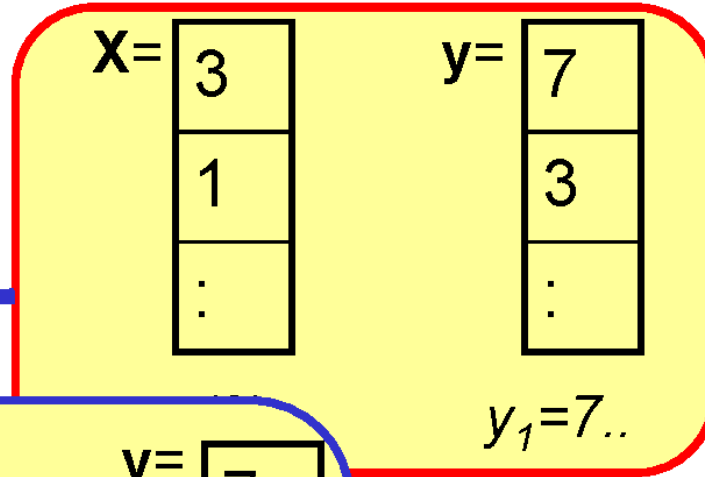
Univariate Linear regression with a constant term:



Linear Regression

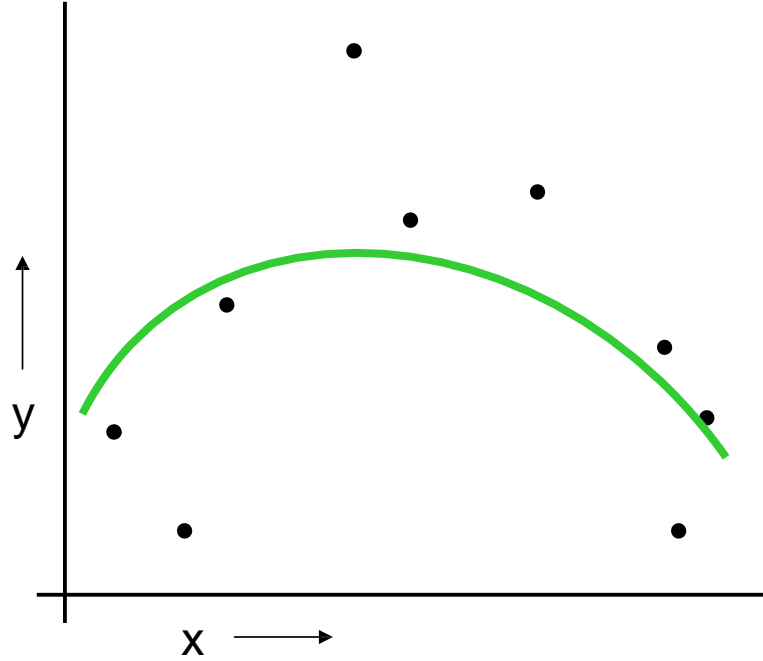
Univariate Linear regression with a constant term:

X	Y
3	7
1	3
⋮	⋮

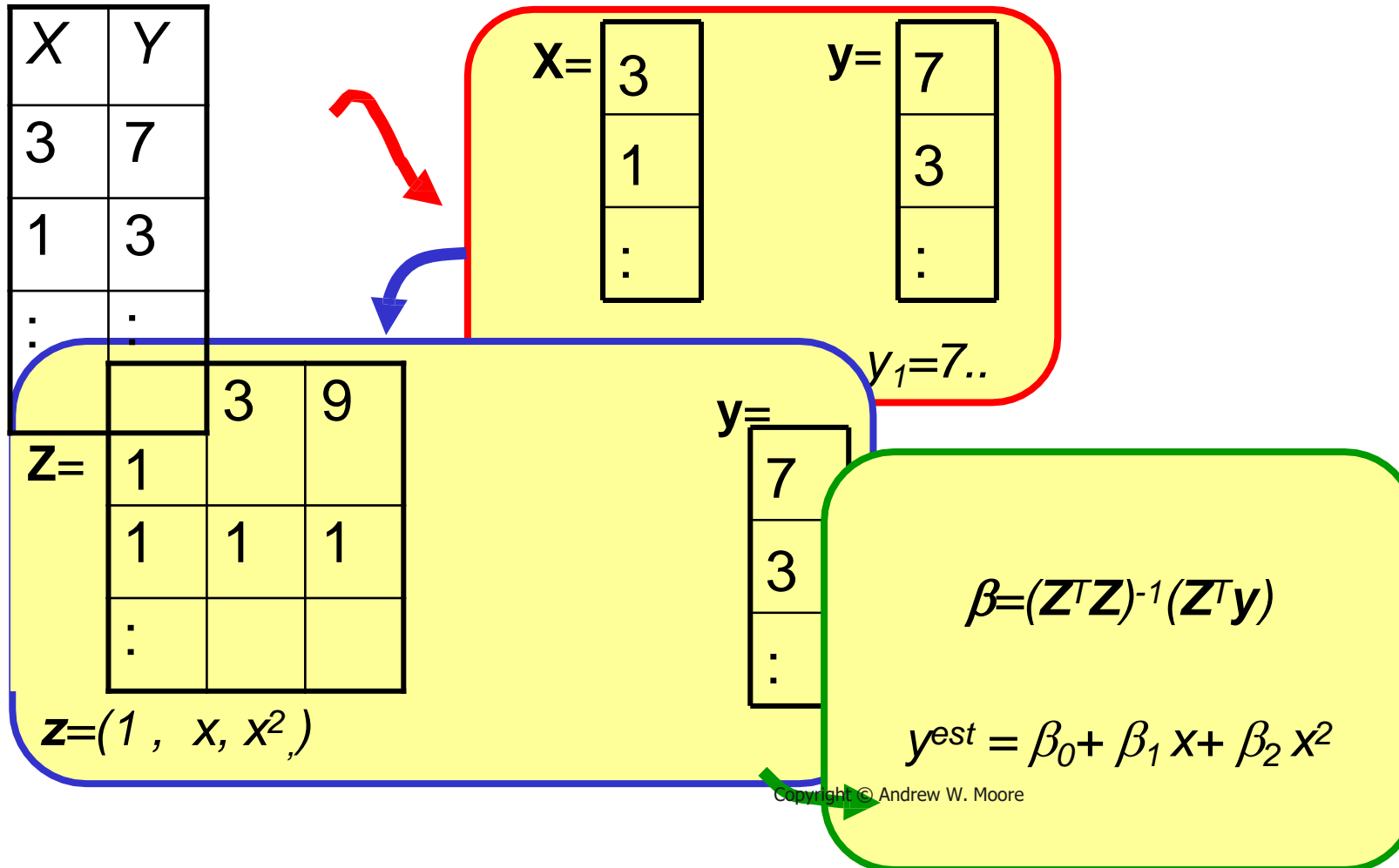


$$\boldsymbol{\beta} = (\mathbf{Z}^T \mathbf{Z})^{-1} (\mathbf{Z}^T \mathbf{y})$$
$$y^{\text{est}} = \beta_0 + \beta_1 x$$

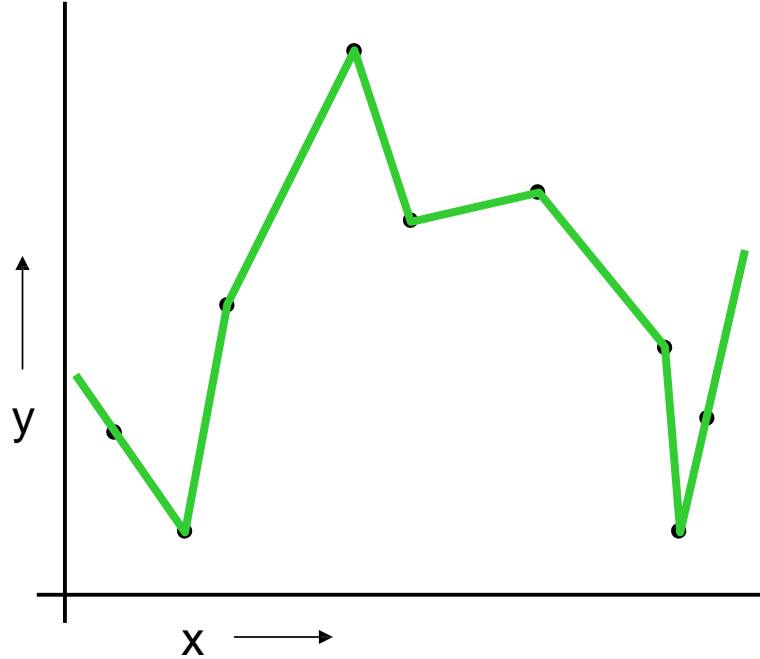
Quadratic Regression



Quadratic Regression

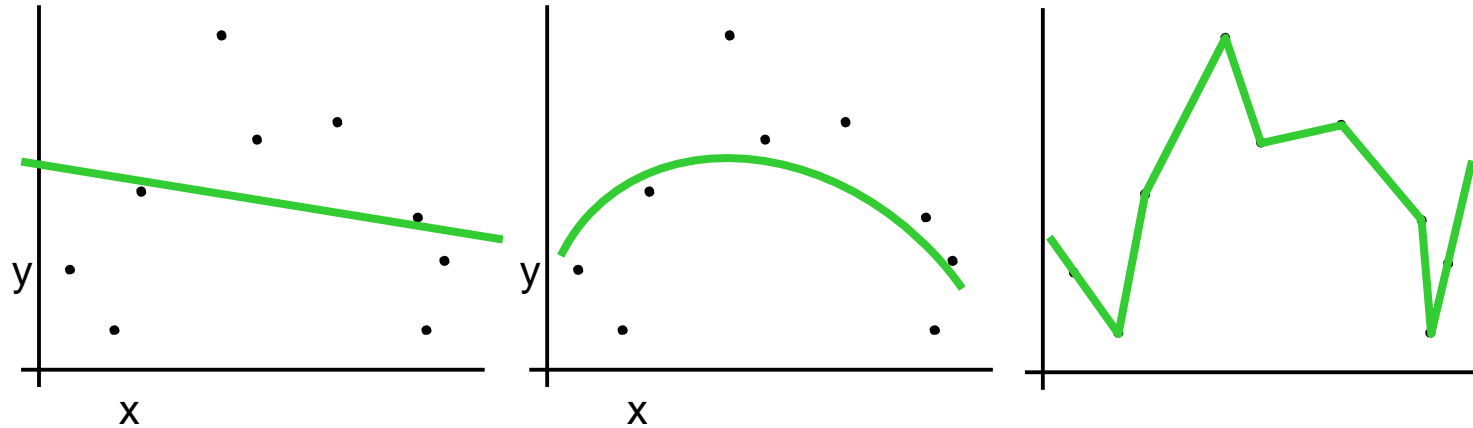


Join-the-dots



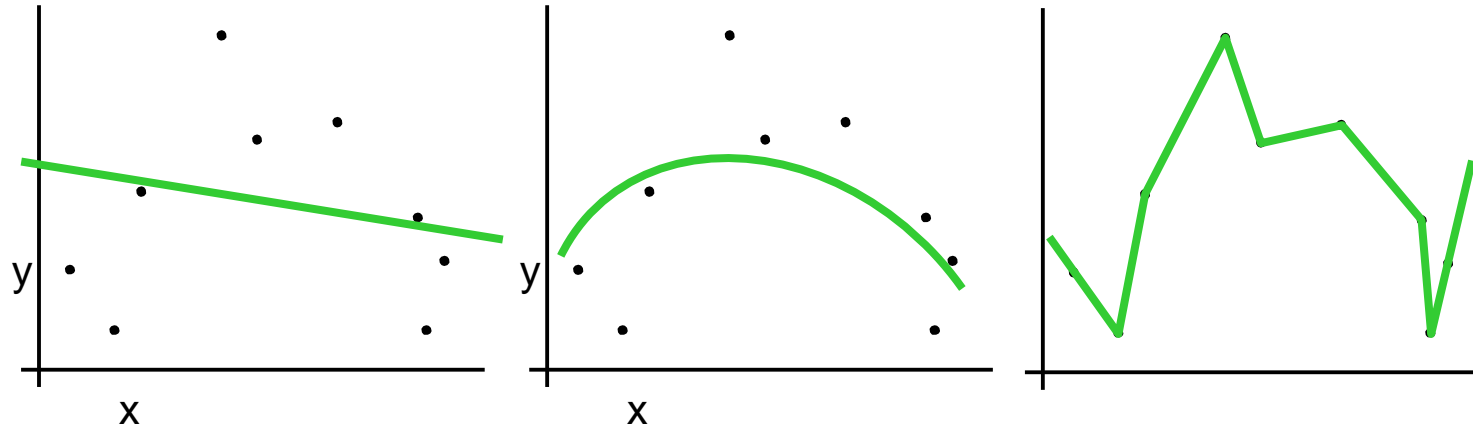
Also known as **piecewise linear nonparametric regression** if that makes you feel better

Which is best?



Why not choose the method with the best fit to the data?

What do we really want?



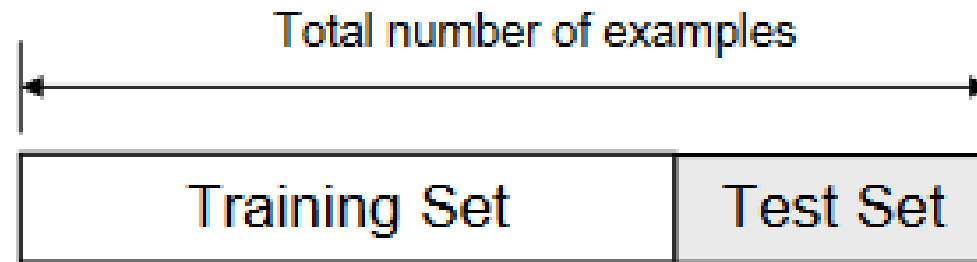
Why not choose the method with the best fit to the data?

“How well are you going to predict future data drawn from the same distribution?”

The holdout method

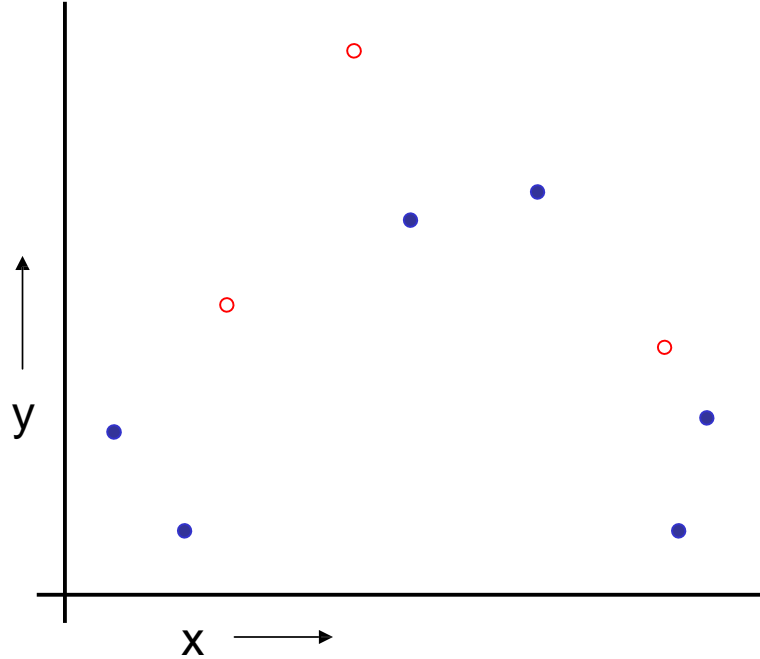
■ Split dataset into two groups

- **Training set:** used to train the classifier
- **Test set:** used to estimate the error rate of the trained classifier



- In this approach, we reserve 30% of the dataset for validation and the remaining 70% for model training.

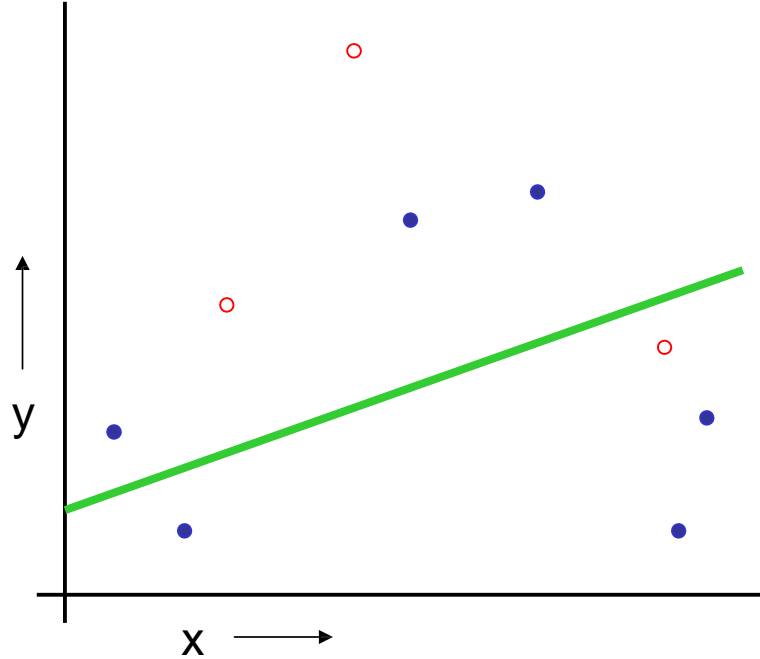
The test set/hold-out method



1. Randomly choose 30% of the data to be in a **test set**

2. The remainder is a **training set**

The test set/hold-out method



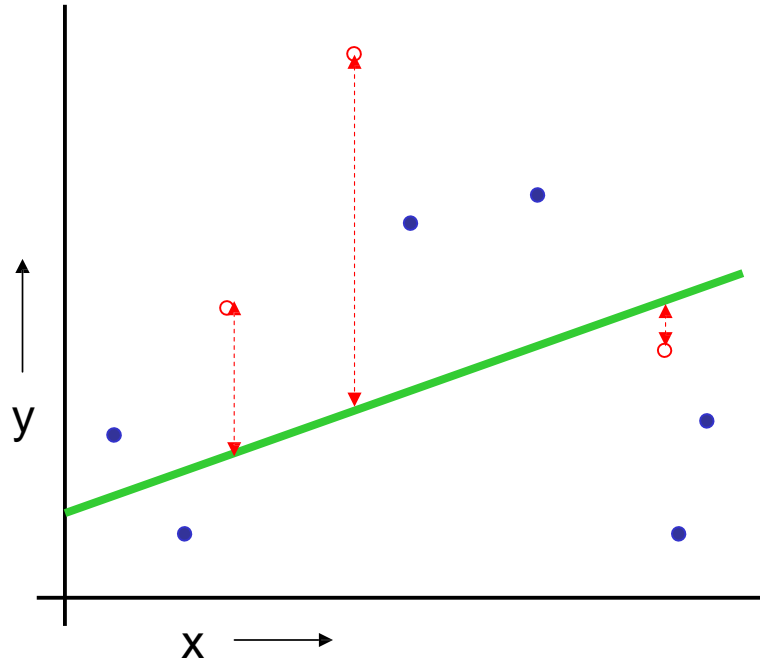
(Linear regression example)

1. Randomly choose 30% of the data to be in a **test set**

2. The remainder is a **training set**

3. Perform your regression on the training set

The test set/hold-out method



(Linear regression example)
Mean Squared Error = 2.4

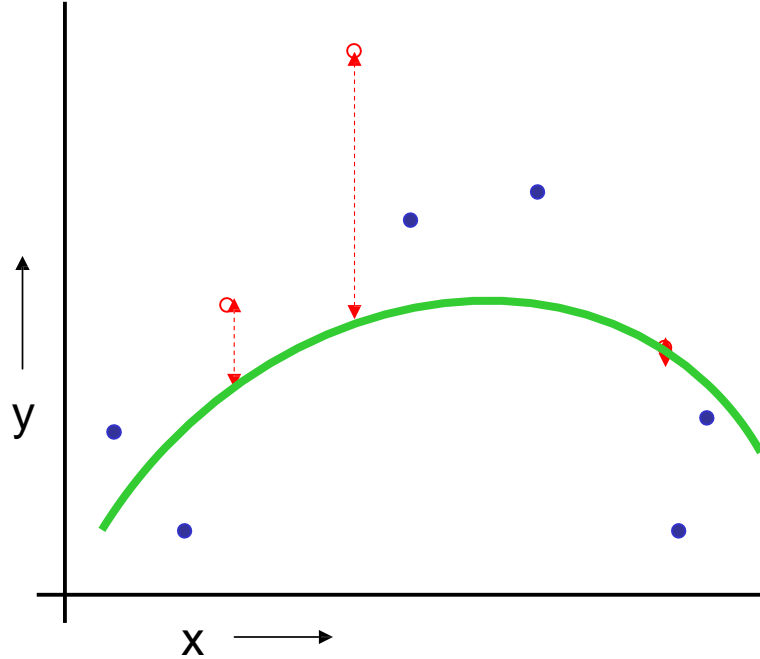
1. Randomly choose 30% of the data to be in a **test set**

2. The remainder is a **training set**

3. Perform your regression on the training set

4. Estimate your future performance with the test set

The test set/hold-out method



(Quadratic regression example)

Mean Squared Error = 0.9

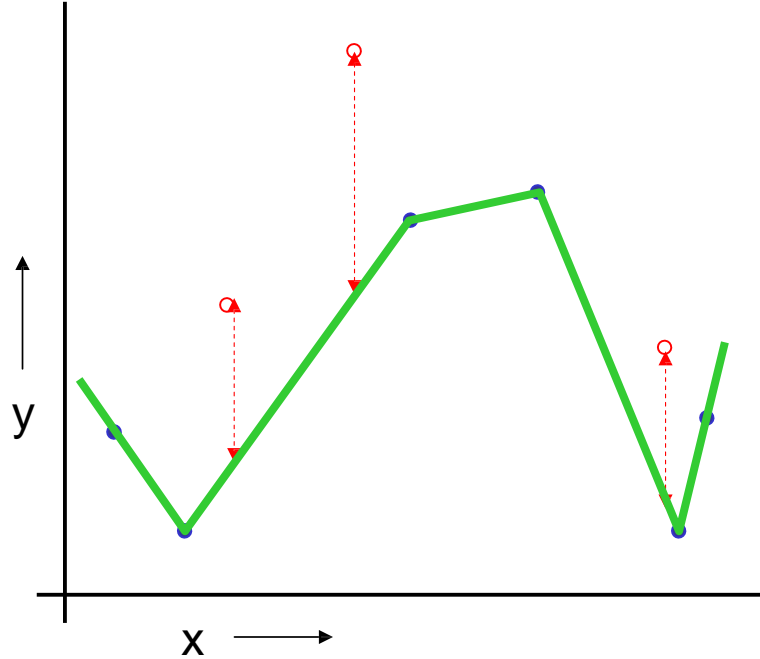
1. Randomly choose 30% of the data to be in a **test set**

2. The remainder is a **training set**

3. Perform your regression on the training set

4. Estimate your future performance with the **test set**

The test set/hold-out method



(Join the dots example)

Mean Squared Error = 2.2

1. Randomly choose 30% of the data to be in a **test set**

2. The remainder is a **training set**

3. Perform your regression on the training set

4. Estimate your future performance with the test set

The test set/hold-out method

Good news:

- Very very simple
- Can then simply choose the method with the best test-set score

Bad news:

- What's the downside?

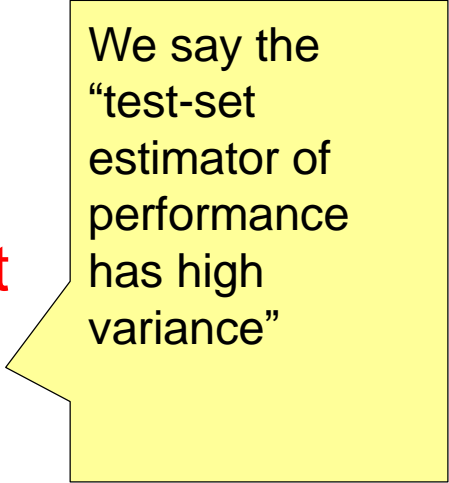
The test set/hold-out method

Good news:

- Very very simple
- Can then simply choose the method with the best test-set score

Bad news:

- Wastes data: we get an estimate of the best method to apply to 30% less data
- If we don't have much data, our test-set might just be lucky or unlucky



We say the “test-set estimator of performance has high variance”

The test set or hold-out method

- However, a major disadvantage of this approach is that since we are training a model on only 70% of the dataset, there is a huge possibility that we might miss out on some interesting information about the data which will lead to a higher bias
- Python Code:

```
train, validation = train_test_split(data, test_size=0.30, random_state = 5)
```

The test set or hold-out method

■ The holdout method has two basic drawbacks

- In problems where we have a sparse dataset we may not be able to afford the “luxury” of setting aside a portion of the dataset for testing
- Since it is a single train-and-test experiment, the holdout estimate of error rate will be misleading if we happen to get an “unfortunate” split

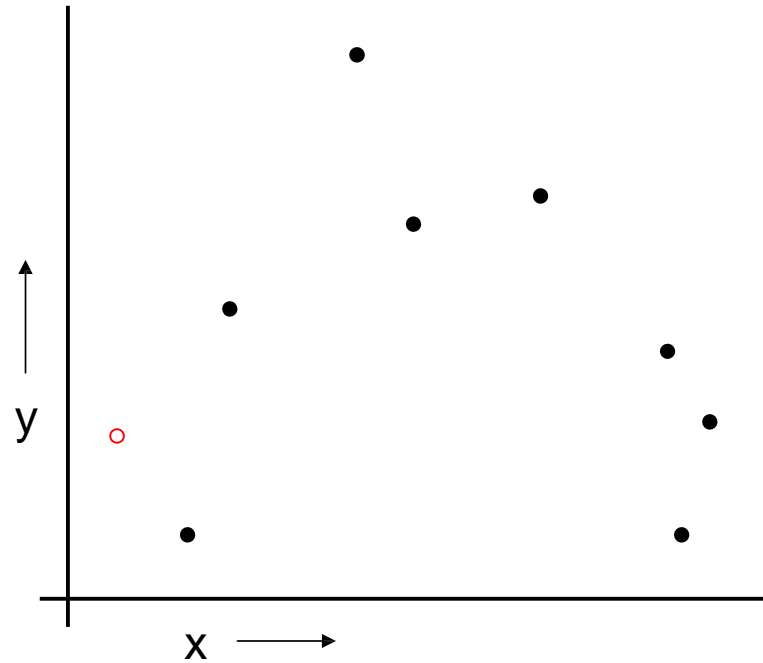
■ The limitations of the holdout can be overcome with a family of re-sampling methods at the expense of higher computational cost

- **Cross Validation**
 - Random Subsampling
 - K-Fold Cross-Validation
 - Leave-one-out Cross-Validation
- **Bootstrap**

Leave one out cross validation (LOOCV)

- In this approach, we reserve only one data point from the available dataset, and train the model on the rest of the data.
- This process iterates for each data point.

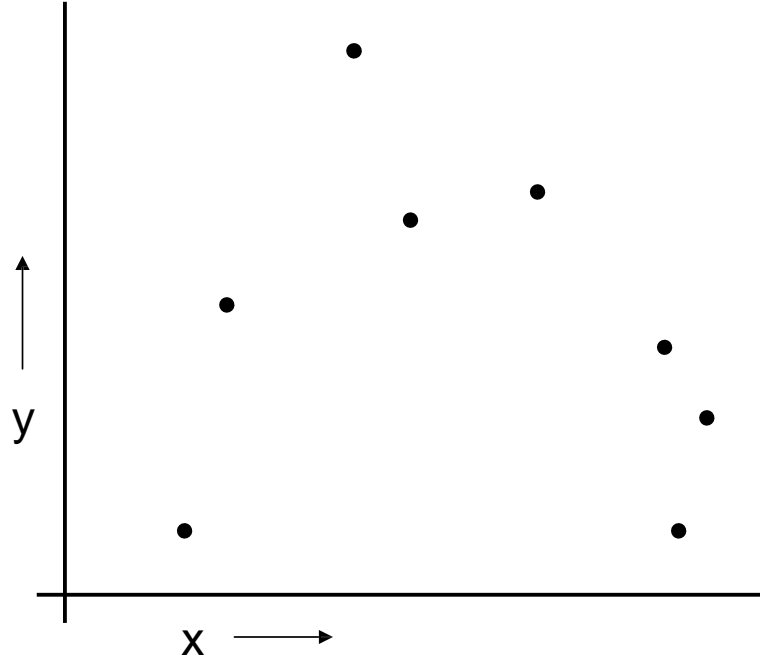
LOOCV (Leave-one-out Cross Validation)



For $k=1$ to R

1. Let (x_k, y_k) be the k^{th} record

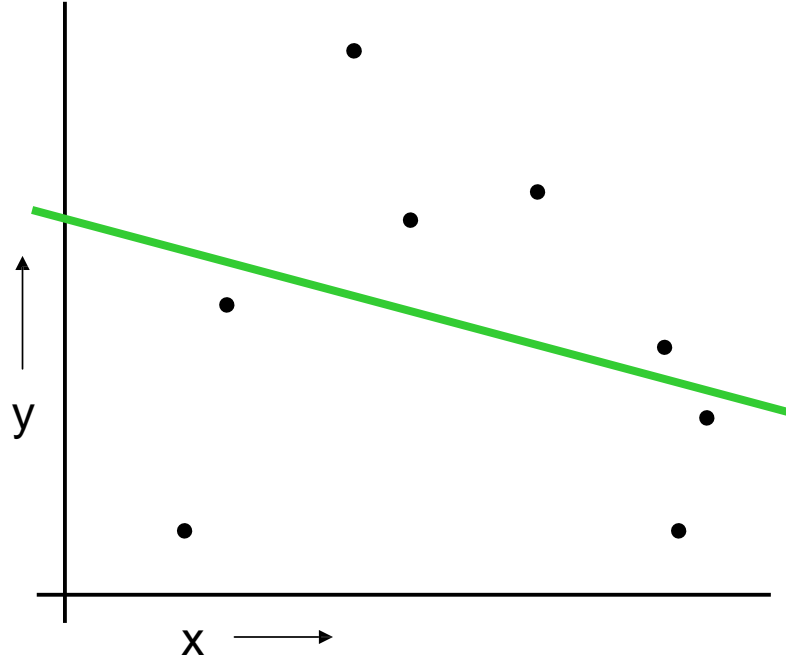
LOOCV (Leave-one-out Cross Validation)



For $k=1$ to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset

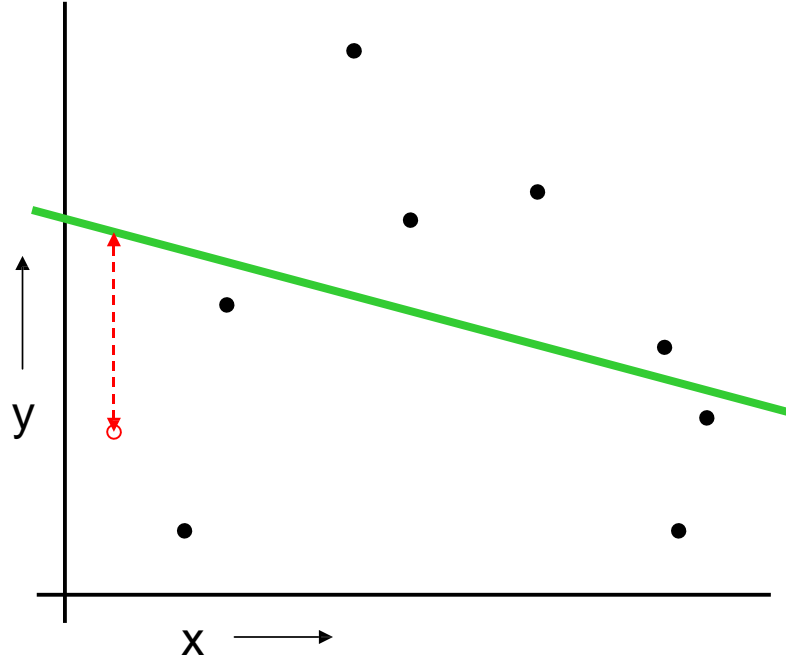
LOOCV (Leave-one-out Cross Validation)



For $k=1$ to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints

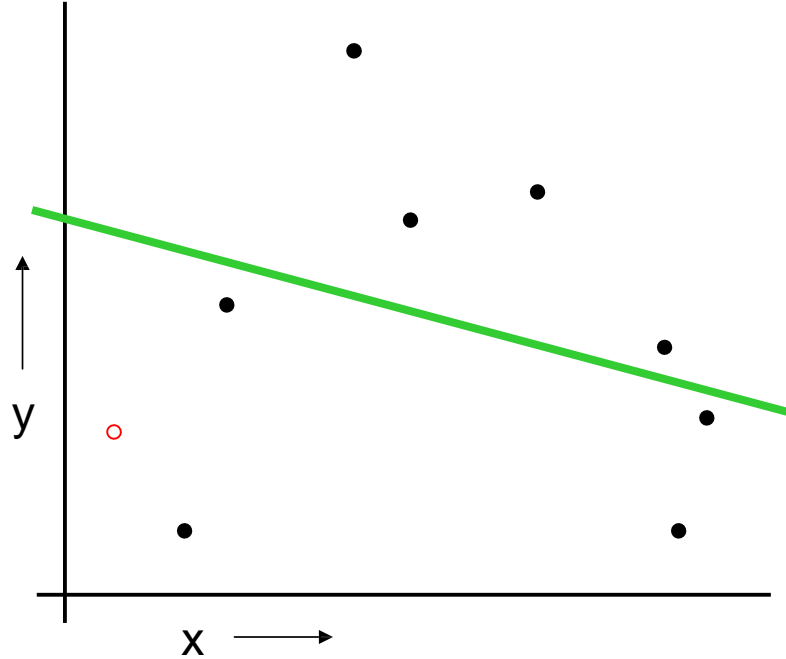
LOOCV (Leave-one-out Cross Validation)



For $k=1$ to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

LOOCV (Leave-one-out Cross Validation)



For $k=1$ to R

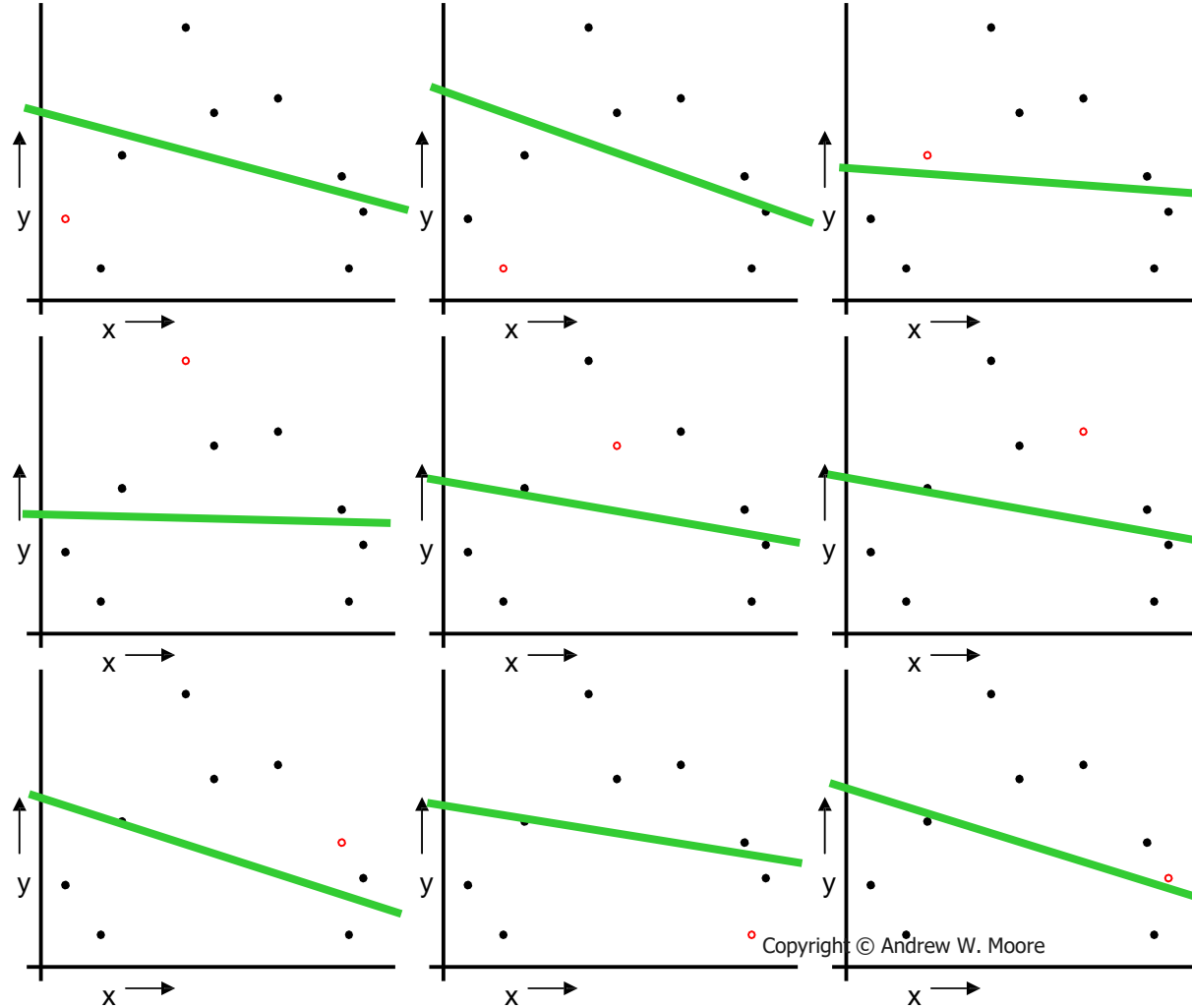
1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

5. The true error is estimated as the average error rate on test examples

$$E = \frac{1}{K} \sum_{i=1}^K E_i$$

LOOCV (Leave-one-out Cross Validation)



For $k=1$ to R

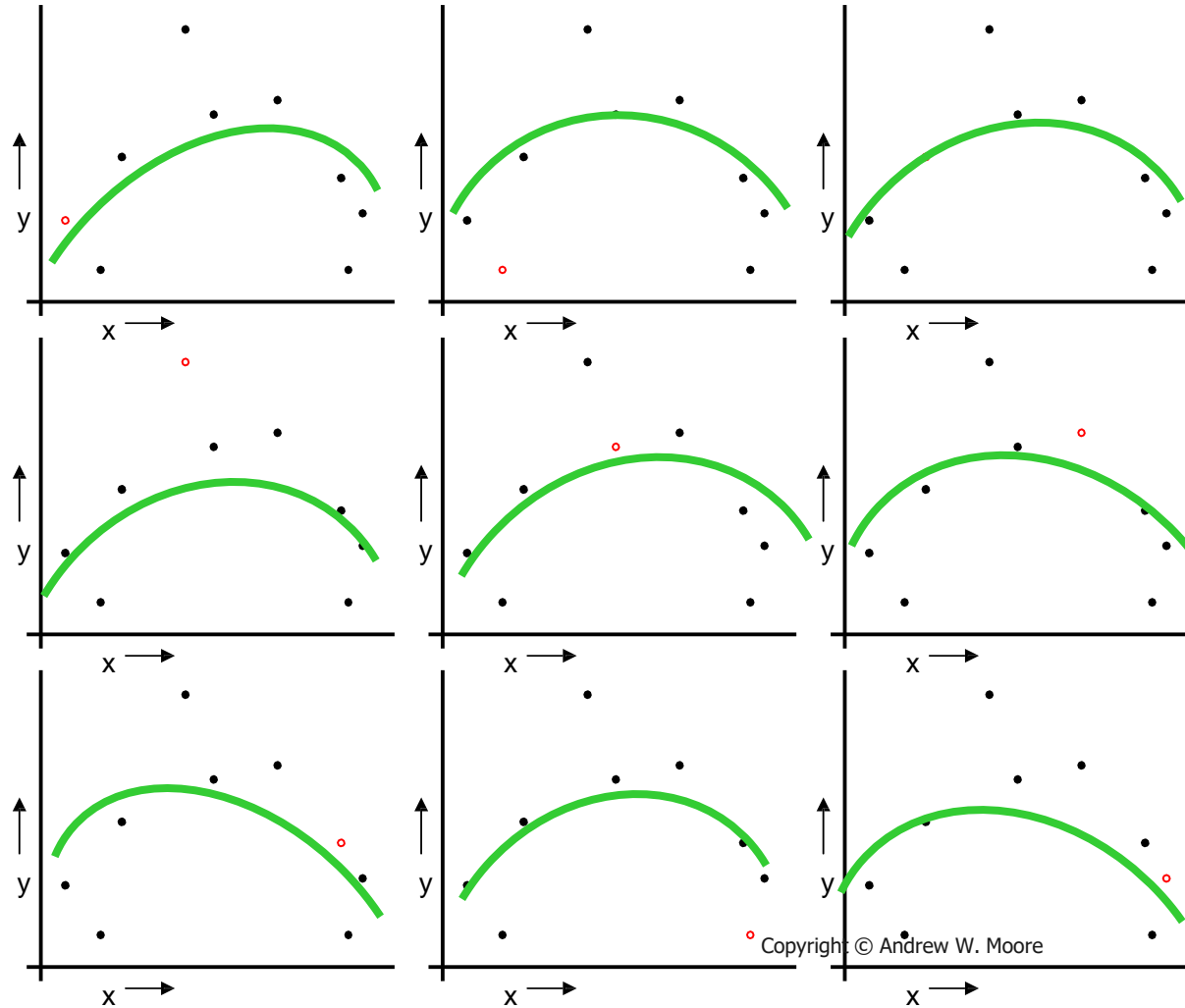
1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 2.12$$

Copyright © Andrew W. Moore

LOOCV for Quadratic Regression



For $k=1$ to R

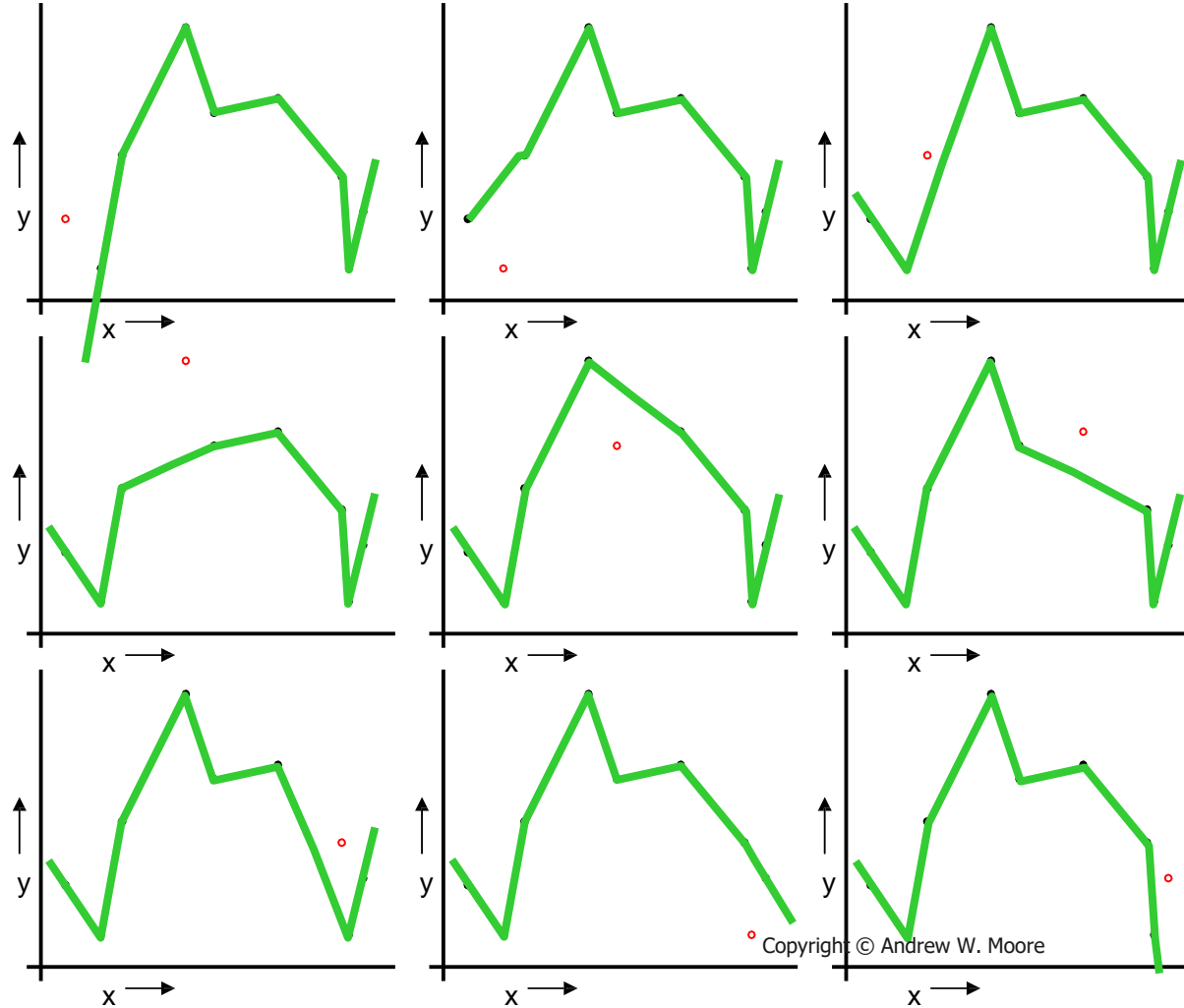
1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{\text{LOOCV}} = 0.962$$

Copyright © Andrew W. Moore

LOOCV for Join The Dots



For $k=1$ to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 3.33$$

Copyright © Andrew W. Moore

LOOCV Advantages and Disadvantages:

- We make use of all data points, hence the bias will be low
- We repeat the cross validation process n times (where n is number of data points) which results in a higher execution time
- This approach leads to higher variation in testing model effectiveness because we test against one data point. So, our estimation gets highly influenced by the data point
- If the data point turns out to be an outlier, it can lead to a higher variation

Which kind of Cross Validation?

	Downside	Upside
Test-set	Variance: unreliable estimate of future performance	Cheap
Leave-one-out	Expensive. Has some weird behavior	Doesn't waste data

..can we get the best of both worlds?

k-fold cross validation

From the above two validation methods, we've learnt:

- We should train the model on a large portion of the dataset. Otherwise we'll fail to read and recognize the underlying trend in the data. This will eventually result in a higher bias
- We also need a good ratio of testing data points. As less amount of data points can lead to a variance error while testing the effectiveness of the model
- We should iterate on the training and testing process multiple times. We should change the train and test dataset distribution. This helps in validating the model effectiveness properly

k-fold cross validation

- In this method, we split the data-set into k number of subsets (known as folds) then we perform training on all the subsets but leave one ($k-1$) subset for the evaluation of the trained model
- We also iterate k times with a different subset reserved for testing purpose each time

Note:

- It is always suggested that the value of k should be 10 as the lower value of k takes towards validation and higher value of k leads to LOOCV method

k-fold cross validation

Example

The diagram below shows an example of the training subsets and evaluation subsets generated in k-fold cross-validation

Consider, we have total 25 instances:

- In first iteration we use the first 20 percent of data for evaluation, and the remaining 80 percent for training([1-5] testing and [5-25] training)
- In the second iteration we use the second subset of 20 percent for evaluation, and the remaining three subsets of the data for training([5-10] testing and [1-5 and 10-25] training), and so on



k-fold cross validation

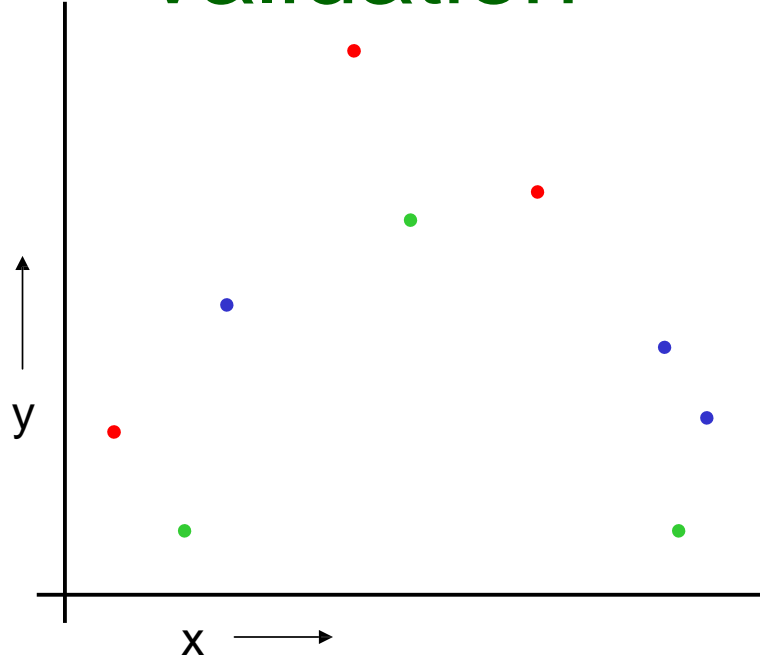
Total instances: 25

Value of k : 5

No. Iteration	Training set observations	Testing set observations
1	[5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]	[0 1 2 3 4]
2	[0 1 2 3 4 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]	[5 6 7 8 9]
3	[0 1 2 3 4 5 6 7 8 9 15 16 17 18 19 20 21 22 23 24]	[10 11 12 13 14]
4	[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 20 21 22 23 24]	[15 16 17 18 19]
5	[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]	[20 21 22 23 24]

k-fold Cross Validation

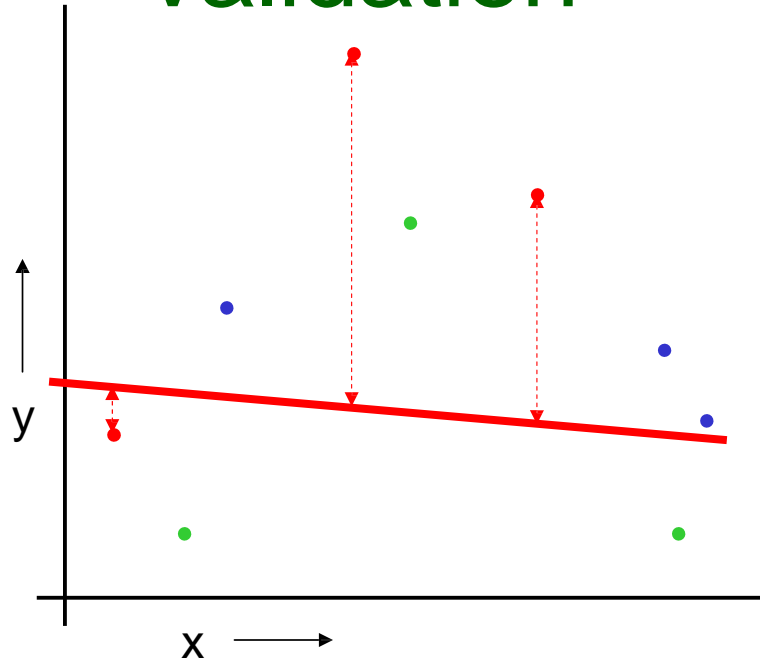
Randomly break the dataset into k partitions (in our example we'll have $k=3$ partitions colored Red Green and Blue)



k-fold Cross Validation

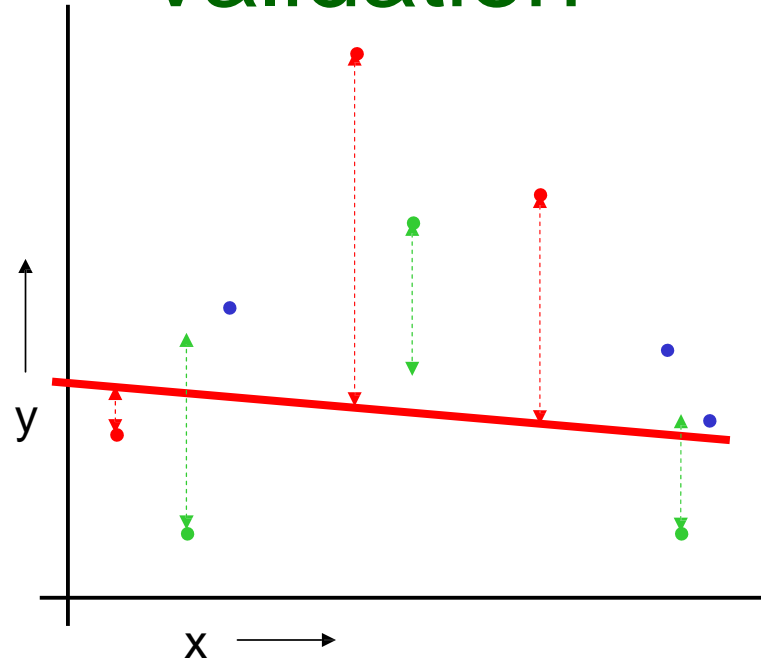
Randomly break the dataset into k partitions (in our example we'll have $k=3$ partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.



k-fold Cross Validation

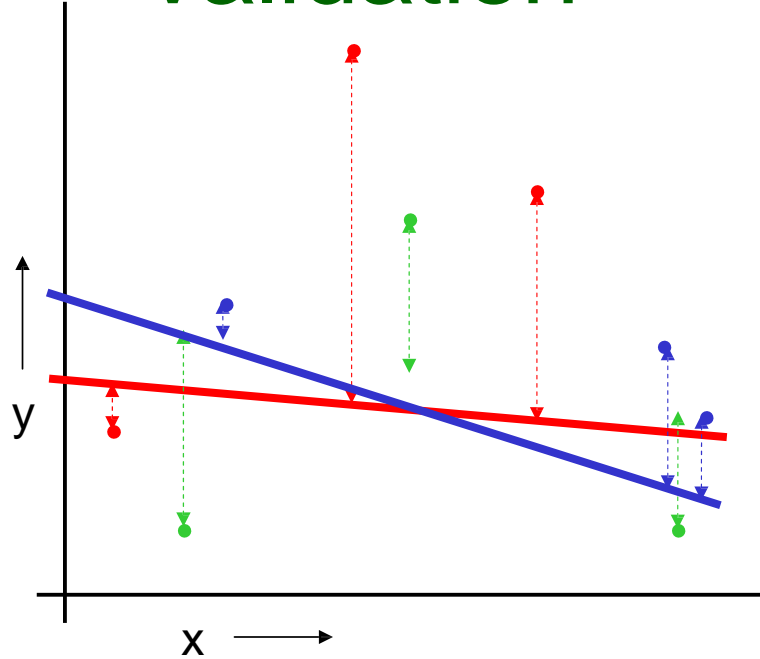
Randomly break the dataset into k partitions (in our example we'll have $k=3$ partitions colored Red Green and Blue)



For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

k-fold Cross Validation



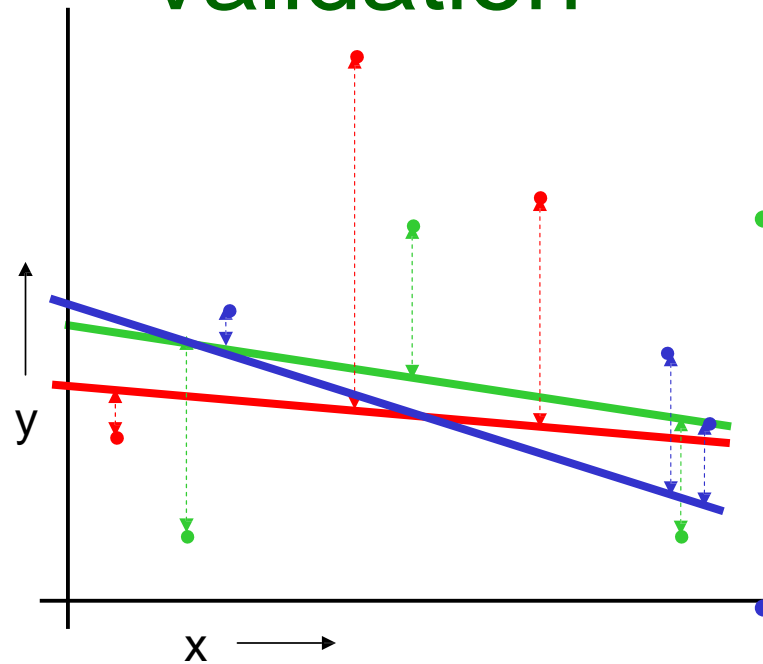
Randomly break the dataset into k partitions (in our example we'll have $k=3$ partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

k-fold Cross Validation

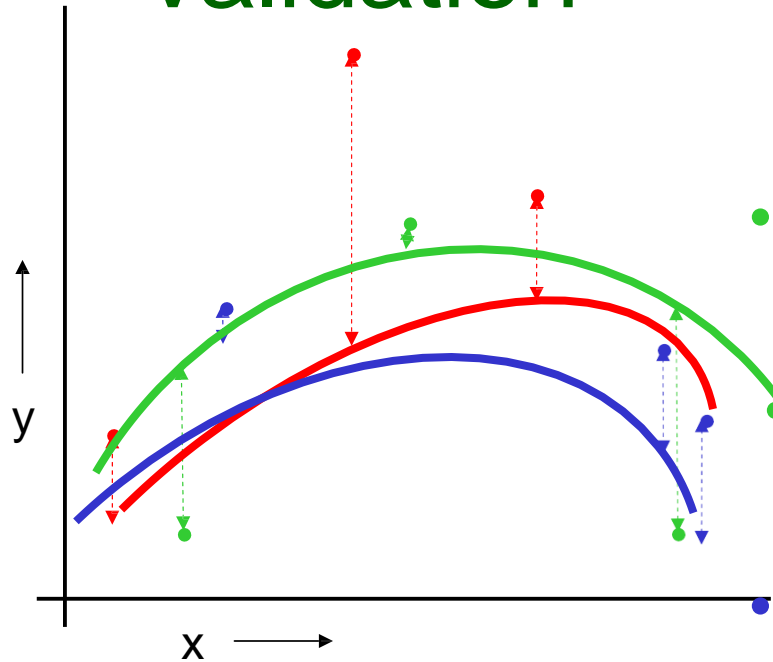


Linear Regression
 $MSE_{3FOLD}=2.05$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

- For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.
- For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.
- For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.
- Then report the mean error

k-fold Cross Validation

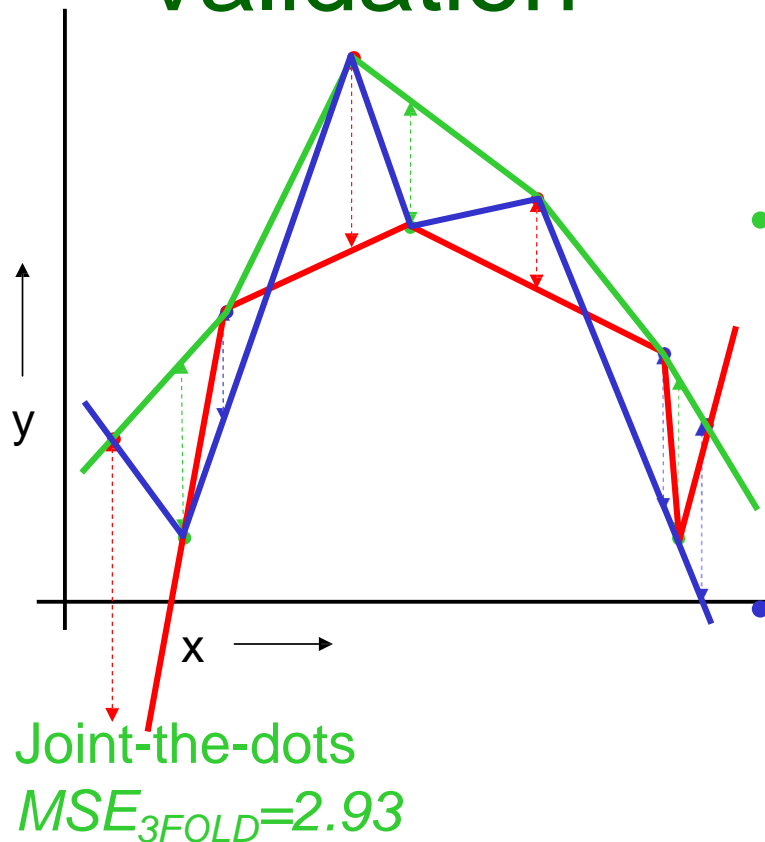


Quadratic Regression
 $MSE_{3FOLD}=1.11$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

- For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.
- For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.
- For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.
- Then report the mean error

k-fold Cross Validation



Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

- For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.
- For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.
- For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.
- Then report the mean error

How many folds are needed?

■ With a large number of folds

- + The bias of the true error rate estimator will be small (the estimator will be very accurate)
- The variance of the true error rate estimator will be large
- The computational time will be very large as well (many experiments)

■ With a small number of folds

- + The number of experiments and, therefore, computation time are reduced
- + The variance of the estimator will be small
- The bias of the estimator will be large (conservative or smaller than the true error rate)

■ In practice, the choice of the number of folds depends on the size of the dataset

- For large datasets, even 3-Fold Cross Validation will be quite accurate
- For very sparse datasets, we may have to use leave-one-out in order to train on as many examples as possible

■ A common choice for K-Fold Cross Validation is $K=10$

Which kind of Cross Validation?

	Downside	Upside
Test-set	Variance: unreliable estimate of future performance	Cheap
Leave-one-out	Expensive. Has some weird behavior	Doesn't waste data
10-fold	Wastes 10% of the data. 10 times more expensive than test set	Only wastes 10%. Only 10 times more expensive instead of R times.
3-fold	Wastier than 10-fold. Expensivier than test set	Slightly better than test-set
R-fold	Identical to Leave-one-out	

END of LECTURE