



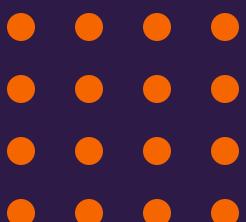
20
24



Model Selection and Cross-Validation

Dr. Aaron J. Masino

Associate Professor, School of Computing





Lecture Outline

- Bias vs. Variance (aka Capacity vs. Overfitting)
- Addressing bias and variance
- Systematic Model Selection / Evaluation

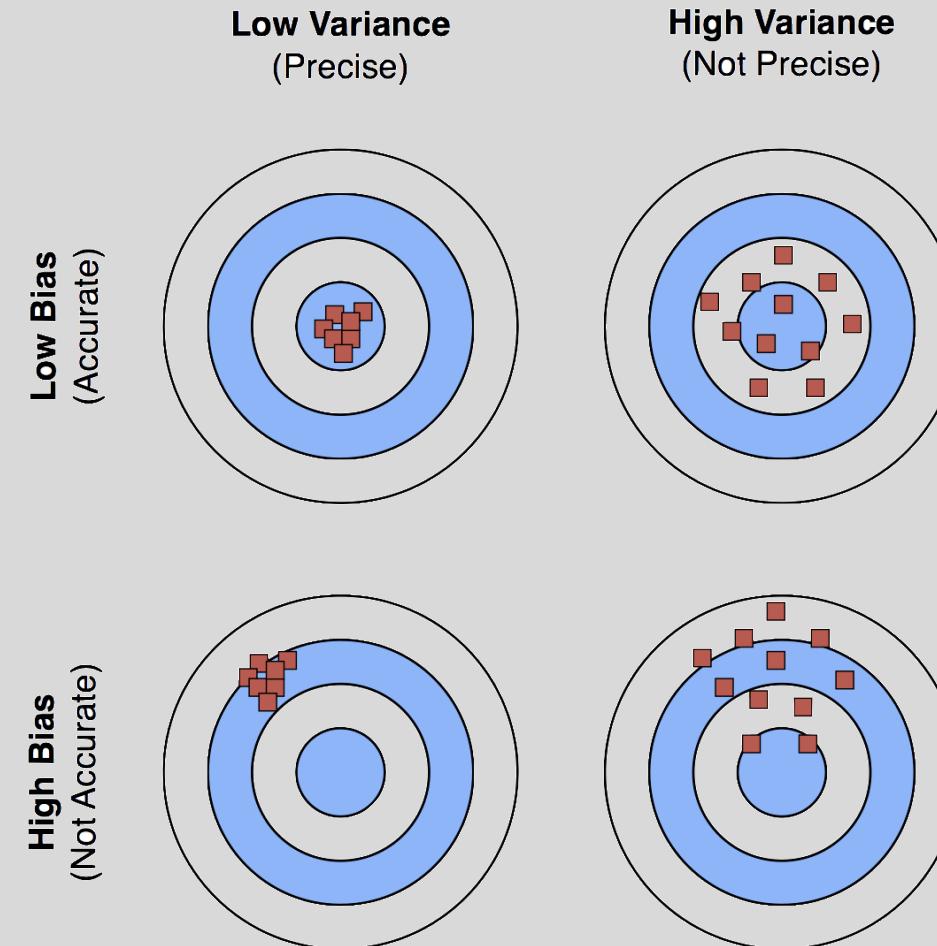


Bias vs. Variance

- Assume the true relation of interest is
$$Y = f(X) + \epsilon$$
- Model capacity refers to the space of functions, F , a model could perfectly represent. Usually difficult to establish, but some examples
 - Linear model $F = \{aX + b\} \forall a, b \in \mathbb{R}$
 - Neural network with arbitrary size: F is the space of ALL continuous functions
- Bias: the model error on the training data
 - Results from simplifying assumptions
 - High bias may be due to “underfitting” (insufficient capacity) in which the model cannot capture the true relationships
- Variance: the change in model parameter estimates due to training data variability
 - Usually evaluated through model performance difference on a test set varies
 - High variance indicates “overfitting” (excessive capacity)

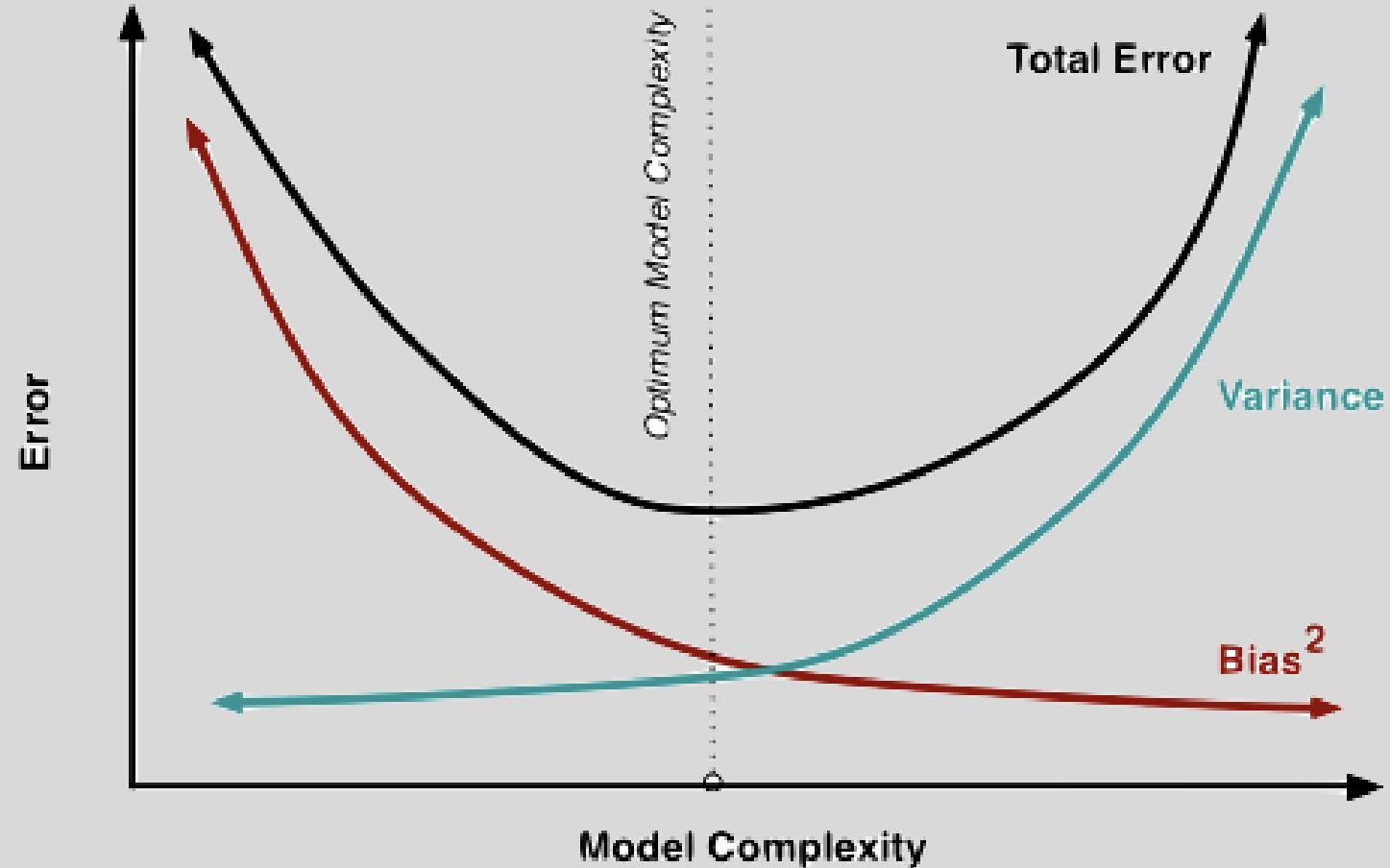


Bias vs. Variance





Bias/Variance Trade-off

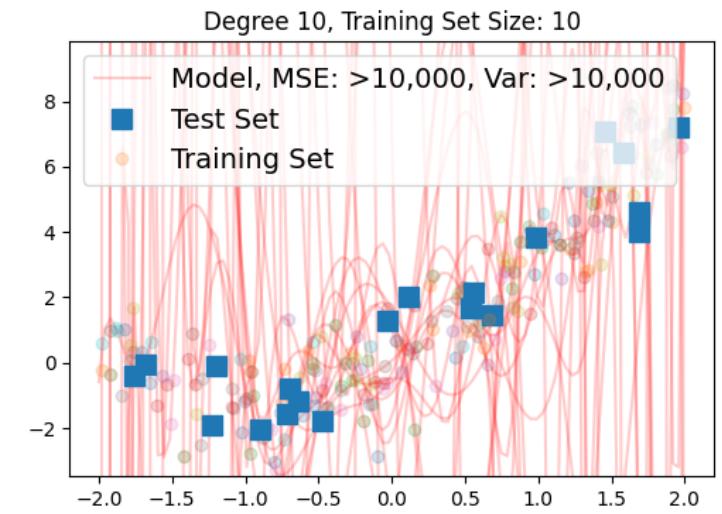
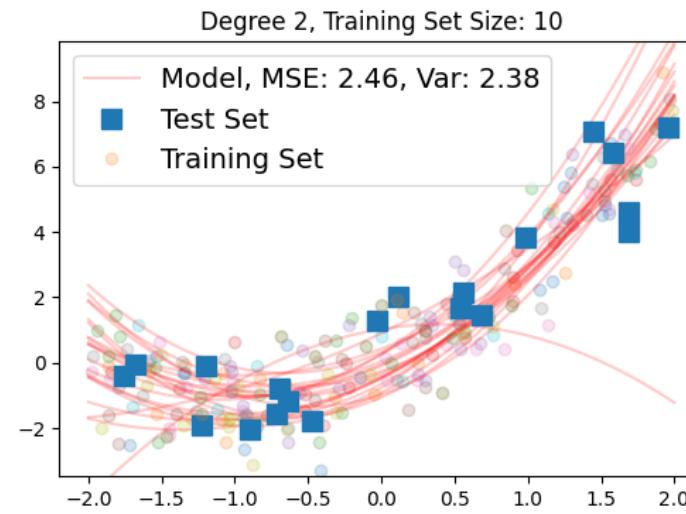
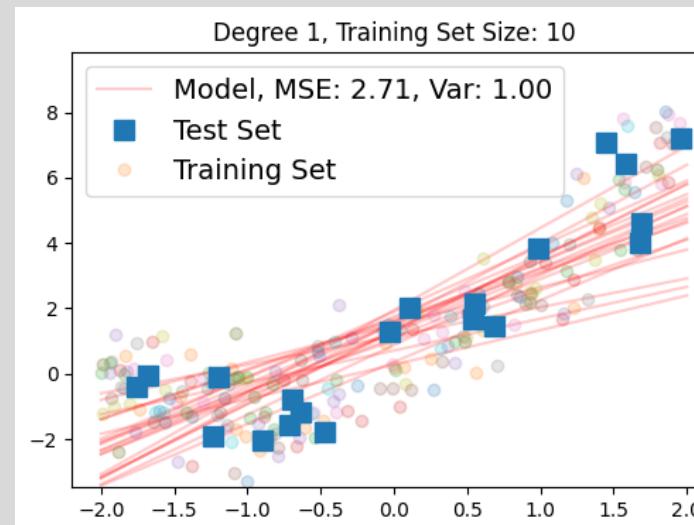


Bias vs. Variance impact of training data

- The true relation of interest is

$$Y = x^2 + \epsilon$$

- Train 20 models to 20 different training sets each with 10 samples
- Lower capacity models (degree 1, 2) have similar MSE mean and std
- High capacity model (degree 10) has very high MSE (overfitting)

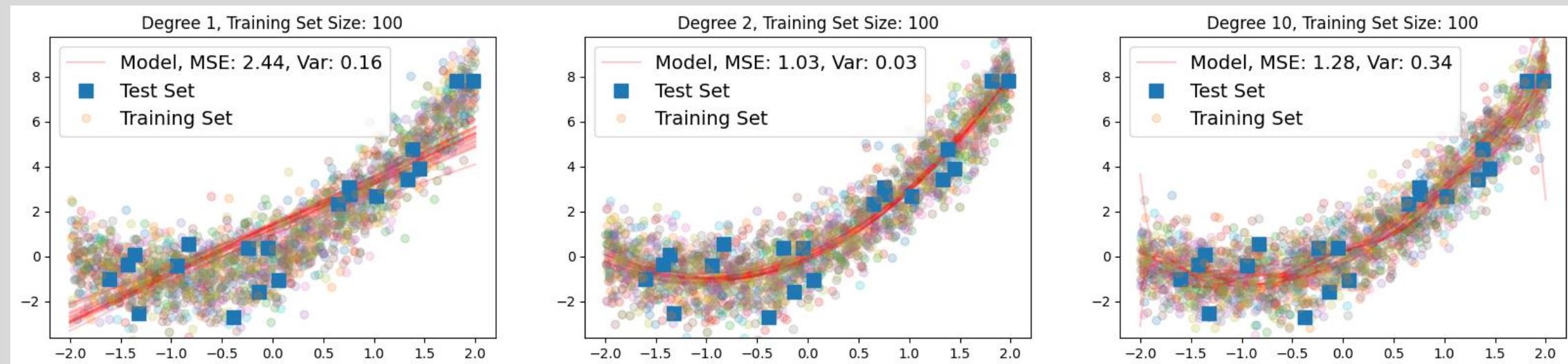




Bias vs. Variance impact of training data

- The true relation of interest is

$$Y = x^2 + \epsilon$$
- Train 20 models to 20 different training sets each with 100 samples
- Notice degree 1 model has an average MSE approximately equal to 10 samples – can't fix bias with more data
- Notice degree 10 model variance has dropped at least 4 orders of magnitude – is this model overfit?





Lecture Outline

- Bias vs. Variance (aka Capacity vs. Overfitting)
- **Addressing bias and variance**
- Systematic Model Selection / Evaluation



Overcoming model (statistical) bias

- Increasing capacity of linear models
 - We are free to transform an independent feature by any function, f , and subsequently create a linear model
$$y = \beta_0 + \sum_{i=1}^p \beta_i x_p + \beta_{p+1} f(x_{p+1})$$
 - Is this still a linear model?
 - Yes, in the sense that the relation between y and the transformed feature, $f(x_{p+1})$, is linear with slope β_{p+1} .
 - We can use the same methods to solve for the coefficients, including β_1
 - In a similar way, we can add non-additive interaction terms to the model, for example
$$y = \beta_0 + \sum_{i=1}^p \beta_i x_i + \beta_{p+1} x_1 x_2 = \beta_0 + \sum_{i=2}^p \beta_i x_i + (\beta_1 + \beta_{p+1} x_2)x_1$$

- Use a non-linear model. There are many options
 - Ensembles of decision trees
 - Neural networks
 - Many, many more

The relation between y and x_1 is no longer constant as a change in x_2 will change the slope



Overcoming model variance (overfitting)

- Reduce the number of input features
- **Regularization** – Modify the loss function L to restrict the capacity, typically by constraining the model parameter values

$$L(y_i, f(\mathbf{x}_i; \boldsymbol{\theta})) + \lambda G(\boldsymbol{\theta})$$

where λ is a scalar that gives the weight (or importance) of the regularization term and $G(\boldsymbol{\theta})$ is a function of the model parameters, $\boldsymbol{\theta}$, that if minimized constrains the model parameters.

- Commonly used regularization terms include
 - L¹ norm: $\lambda \|\boldsymbol{\beta}\|_1 = \lambda \sum_{i=1}^p |\beta_i|$ - referred to as *Lasso* when applied for linear regression
 - L² norm: $\lambda \|\boldsymbol{\beta}\|_2 = \lambda \sqrt{\sum_{i=1}^p \beta_i^2}$ - referred to as *Ridge* when applied for linear regression
- Reduce model capacity through structural choices (**we'll see this later**)



Selecting features with forward selection

Example: 3 predictors (X_1, X_2, X_3)

Models with 0 predictor:

- M0:

Models with 1 predictor:

- M1: X_1
- M2: X_2
- M3: X_3

Models with 2 predictors:

- M4: $\{X_1, X_2\}$
- M5: $\{X_2, X_3\}$
- M6: $\{X_3, X_1\}$

Models with 3 predictors:

- M7: $\{X_1, X_2, X_3\}$

2^J Models

Wrapper methods for feature selection

- Assume there are p input features, then there are $\Phi = 2^p$ possible feature subsets
- *Best subset:* Test all Φ subsets. This is usually computationally impractical
- In practice, we use a stepwise approach, though even these can be impractical (e.g., deep learning models)

Stepwise Selection Methods

Forward Sequential Feature Selection (F-SFS):

1. Start with 0 features in selected features, ϕ
2. Repeat for each feature not in ϕ
 - a. Add 1 feature to ϕ to form ϕ^*
 - b. Train the model with ϕ^* and record performance
3. Find ϕ^* that gave best model performance. If performance is better than for ϕ , replace ϕ with ϕ^* and repeat step 2, otherwise stop.

Backward Sequential Feature Selection (B-SFS): The same idea as F-SFS but proceeds in opposite direction. All features are initially included, and one feature is removed at a time.



Stepwise Variable Selection: Computational Complexity

How many models did we evaluate?

- 1st step, **J Models**
- 2nd step, **$J-1$ Models** (add 1 predictor out of $J-1$ possible)
- 3rd step, **$J-2$ Models** (add 1 predictor out of $J-2$ possible)
- ...

$$O(J^2) \ll 2^J \text{ for large } J$$



Model hyperparameters

- Nearly all machine learning models have hyperparameters (aka tuning parameters) that are not learned from the data
- Examples - regularization coefficient λ , number of trees, number of layers in a neural network, learning rate, ... (many more)

$$\widehat{y}_n = \sum_{i=1}^p \beta_i x_{i,n} + \lambda \sum_i \beta_i^2$$

Linear regression coefficients, β are learned from the data, however, regularization coefficient, λ , is a hyperparameter that must be specified.

- Model performance can be very sensitive to hyperparameter selection
- Ideally, a systematic search and evaluation over multiple candidate hyperparameters should be used



Hyperparameter grid search

- Assume the model has H hyperparameters $\{\lambda_1, \dots, \lambda_H\}$
- User specifies values (the grid) to test for each hyperparameter
- For each hyperparameter λ_h , if we specify g_h values, then there are

$$G = \prod_{i=1}^H g_i$$

hyperparameter combinations specified by the grid

- We must train a new model for each of the G combinations to identify the best one
- Note, G grows geometrically as g_h increases so care must be taken in grid selection
- No theoretical grounds for grid selection



Improving on grid search

We won't cover these, but you may see them in real world

- Randomized search
 - Hyperparameters are randomly sampled from a specified probability distribution
 - Each hyperparameter may have its own sampling distribution
 - Advantage is that the search is **not** a cross product (i.e., avoids geometric growth) and a budget (number of combinations to test) can be set
- Bayesian and quasi-Bayesian search
 - Similar to randomized search, but successive hyperparameter candidates are conditionally selected based on the model performance on previous candidates
 - Various Python implementations:
 - Fully Bayesian (e.g., *bayes_opt*) – user specifies bounds for each hyperparameter
 - quasi-Bayesian (e.g., *hyper_opt* based on Tree of Parzen Estimators) – user specifies a distribution for each hyperparameter



We have a problem

How do we control model bias and (especially) overfitting and simultaneously

- perform necessary data preprocessing (e.g., missing data imputation)
- pick the best features
- pick the best model hyperparameters
- pick the best model structure (more on this later)



Lecture Outline

- Bias vs. Variance (aka Capacity vs. Overfitting)
- Addressing bias and variance
- **Systematic Model Selection / Evaluation**



Model Selection

Model selection is the application of a principled method to select among features and model form

A strong motivation for performing model selection is to avoid overfitting while still picking the best performing model

Model training process

Let's put everything together into an overall process for training and selecting a model

1. Split the data into a training set (~80%) and a test set (~20%) – split should be random and stratified
2. Using only the training data, perform any necessary preprocessing steps (imputation, filter-based feature selection/dimensionality reduction)
3. Select hyperparameter search method
4. **Select training and evaluation approach**
 - a. Train / validation / test split
 - b. K-fold cross validation
 - c. Bootstrap

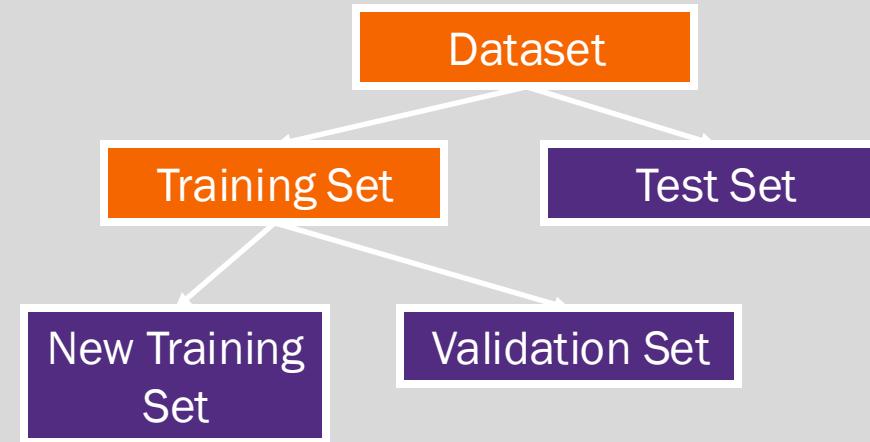
Test set data pre-processing should be completed using training set parameters (e.g., for imputing missing test set values should be based on training data such as KNN imputation)

If feasible (i.e., if time and resources are sufficient) repeat steps 1-4 to multiple times

Train / validation / test split training and evaluation

- From initial train and test sets in step 1:
 - Further split the training data into a new training (~80%) and a validation set (~20%)
 - Ensure the splits are stratified
- For each combination of hyperparameters:
 - Train the model on the new training set
 - Record performance on the validation set
- Select the model (hyperparameter combination) with the best validation set performance
- Retrain the model on the original training set
- Report performance on the test set

This approach is typically used for very large datasets and/or complex models requiring long training times (e.g., deep learning models)





Cross Validation

Using a single validation set to select amongst multiple models can be problematic - **there is the possibility of overfitting to the validation set.**

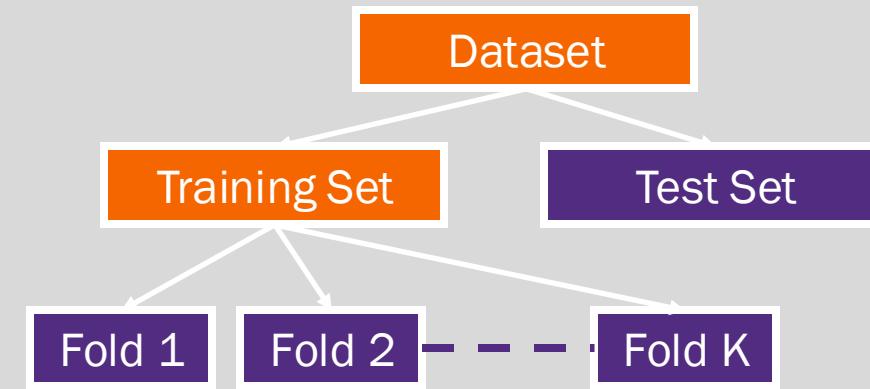
One solution to the problems raised by using a single validation set is to evaluate each model on **multiple** validation sets and average the validation performance.

One can randomly split the training set into training and validation multiple times **but** randomly creating these sets can create the scenario where important features of the data never appear in our random draws.

K-fold cross validation

- From initial train and test sets in step 1:
 - Split the training data in K stratified folds
 - Ensure the splits are stratified
- For each combination of hyperparameters:
 - For $k \in [1, K]$:
 - Combine data from all folds except the k^{th} fold
 - Train the model on the combined folds
 - Compute performance on the k^{th} fold
 - Record the average model performance (e.g., on the loss function) over the k iterations (cross validation performance)
- Select the model (hyperparameter combination) with the best average cross validation
- Retrain the model on the original training set
- Report performance on the test set

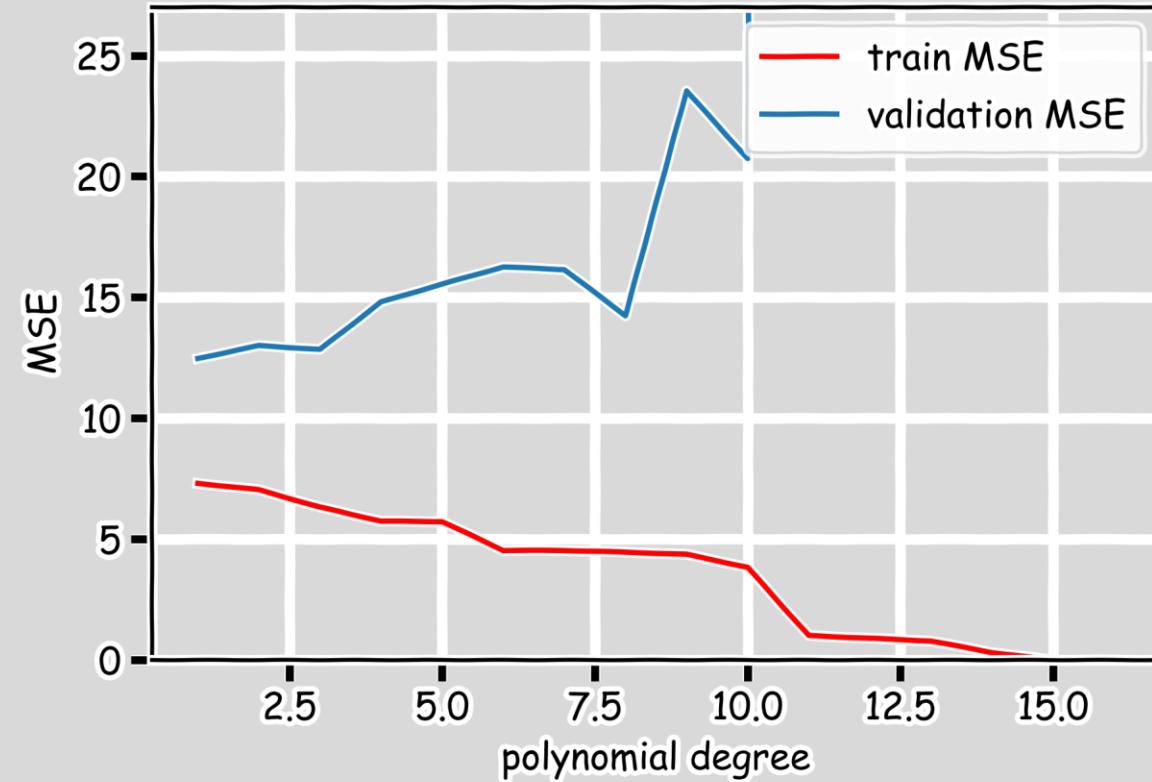
- This approach is advantageous in that it accounts for model variance by obtaining validation performance on all the training data.
- However, it requires K times as many model training runs compared to the single validation set approach.
- Taking K equal to the number of samples is the *leave one out* cross validation approach.



Cross Validation



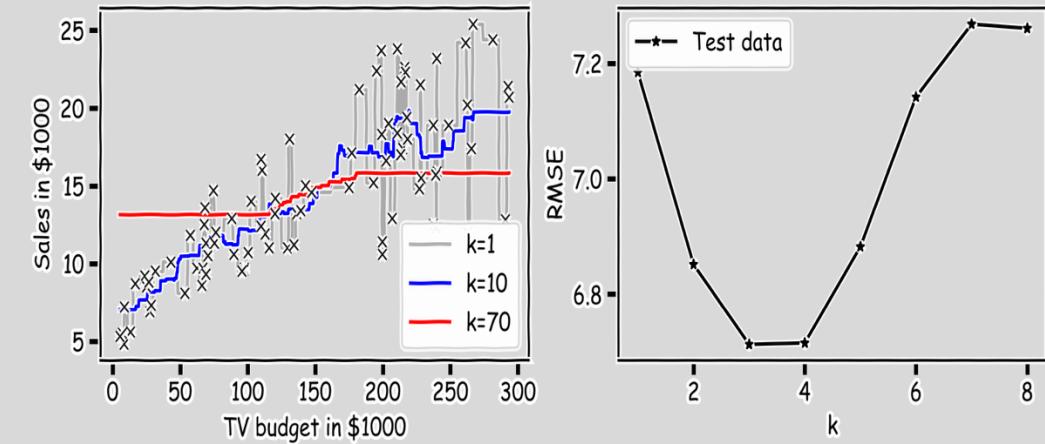
Cross Validation





kNN Revisited

Recall our first simple, intuitive, non-parametric model for regression – the kNN model. We saw that it is vitally important to select an appropriate k for the data.

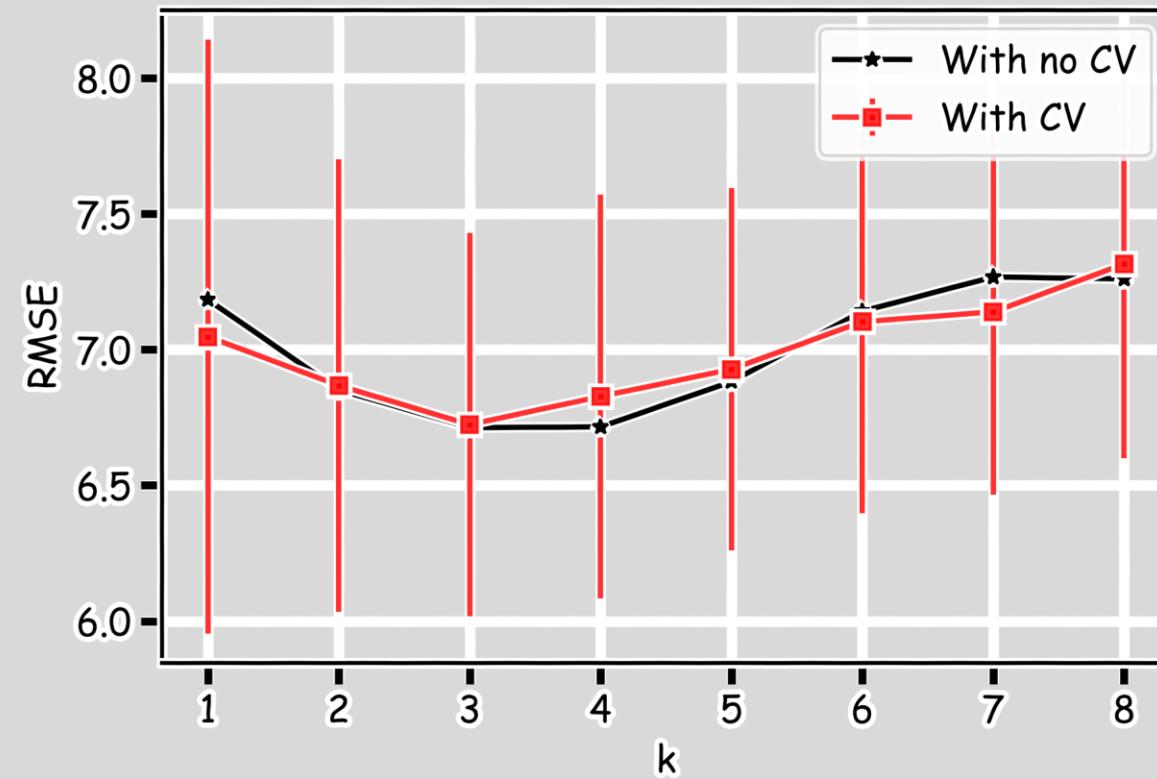


If the k is too small then the model is very sensitive to noise (since a new prediction is based on very few observed neighbors), and if the k is too large, the model tends towards making constant predictions.

How should we select the value for k ?



K-Fold with k=100





Train-Validation-Test

Question:

How would you report the performance of the model?

$R^2_{\text{test}} = 0.52$

$R^2_{\text{train(degree=1)}} = 0.83$





Leave-One-Out Cross Validation

Or using the **leave one out** method:

- validation set: $\{X_i\}$
- training set: $X_{-i} = \{X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n\}$

for $i = 1, \dots, n$:

We fit the model on each training set, denoted $\hat{f}_{X_{-i}}$, and evaluate it on the corresponding validation set, $\hat{f}_{X_{-i}}(X_i)$.

The **cross validation score** is the performance of the model averaged across all validation sets:

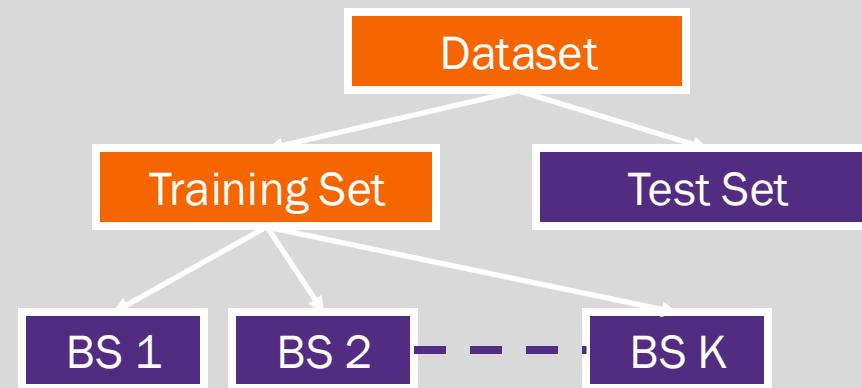
$$CV(\text{Model}) = \frac{1}{n} \sum_{i=1}^n L(\hat{f}_{X_{-i}}(X_i))$$

where L is a loss function.

Bootstrap

- Select the number of bootstrapped training sets, B , and the number of samples per bootstrapped set, M
- For each combination of hyperparameters (and random learning parameter initializations if applicable):
 - For $b \in [1, B]$:
 - Randomly select M samples from the training data
 - Train the model bootstrapped sample
 - Record the training performance
 - Record the average model performance over the B bootstrapped sets
- Select the model (hyperparameter combination) with the best average performance on the bootstrapped sets
- Retrain the model on the original training set
- Report performance on the test set

- This approach is advantageous in that it accounts for model variance by obtaining validation performance many distinct training sets
- Particularly useful when the available data is limited and cross validation may be limited
- It requires B times as many model training runs compared to the single validation set approach.





Questions?