LECTURE 29

Cross Validation

Methods for ensuring the generalizability of our models to unseen data.

.

CSCI 3022, Fall 2023

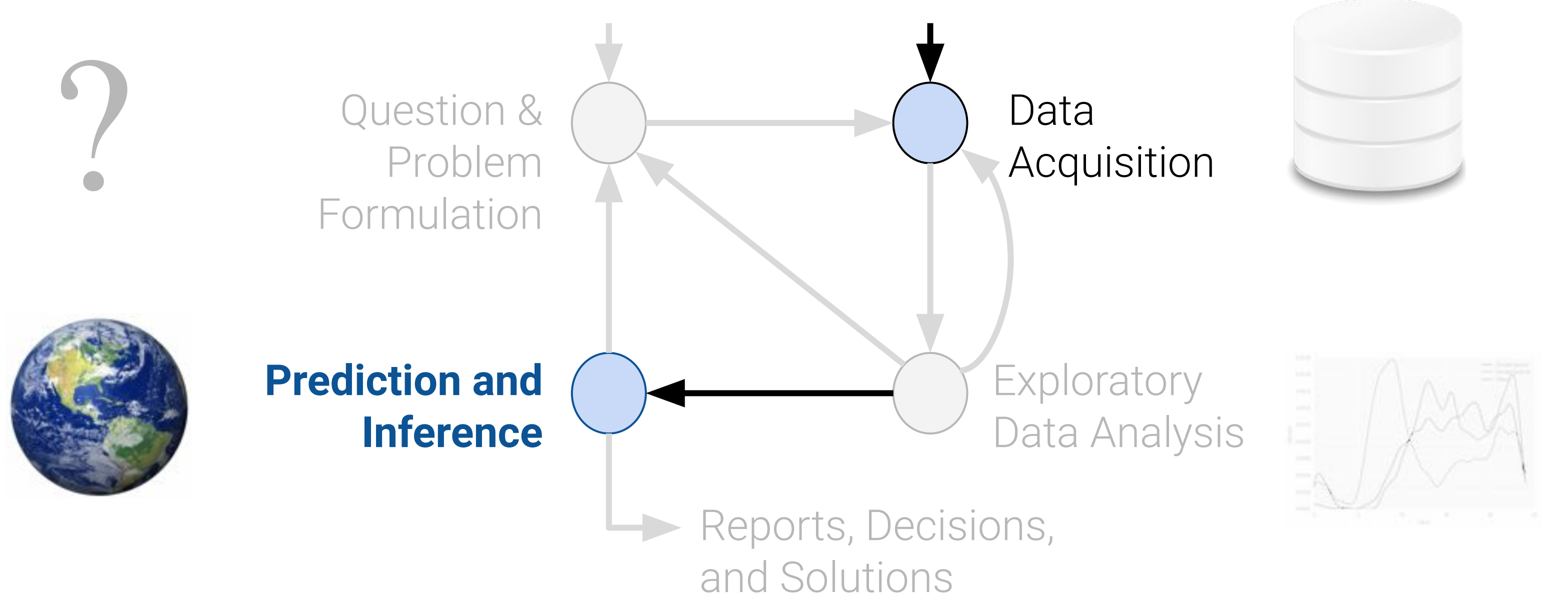
Maribeth Oscamou

Content credit: Acknowledgments

Course Logistics: 13th and 14th Weeks At A Glance

Mon 4/15	Tues 4/16	Wed 4/17	Thurs 4/18	Fri 4/19
Attend & participate in class	TA NB Discussion 5pm-6pm via Zoom **Project Part 2 Released**	Attend & participate in class	**Project Part 1 Due: 11:59pm MT** **No Late Submissions Accepted**	Attend & participate in class Quiz 7: Scope: L24-L26, HW 10, TA Discussion NB 12
Mon 4/22	Tues 4/23	Wed 4/24	Thurs 4/25	Fri 4/26
Attend & participate in class	TA NB Discussion 5pm-6pm via Zoom	Attend & participate in class	**Project Part 2 Due: 11:59pm MT** **No Late Submissions Accepted**	Attend & participate in class Quiz 8: Scope: L26-L30, HW 10, TA Discussion NB 12

Plan for Next Three Lectures: Model Selection



(today)

Model Selection Basics:
Cross Validation
Regularization

Model Selection Basics:
Regularization

Today's Roadmap

Finish lesson 28: Polynomial Features

Complexity and Overfitting

Cross-Validation

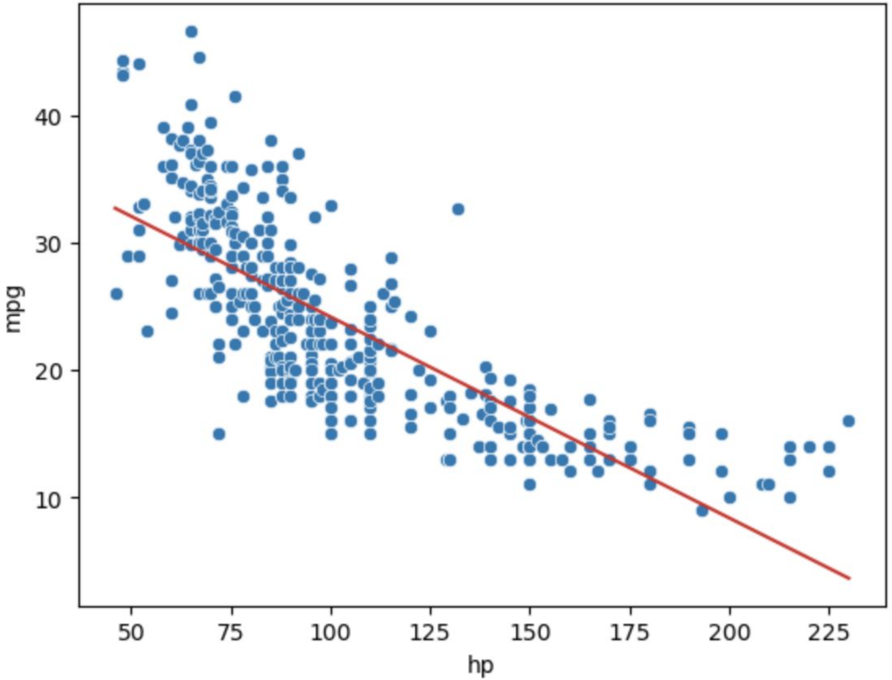
- Training, Test, and Validation Sets
- K-Fold Cross-Validation

Polynomial Features

- **Polynomial Features**

Recall: HW 10

We've seen a few cases now where models with linear features have performed poorly on datasets with a clear non-linear curve.



$$\hat{y} = \theta_0 + \theta_1(\text{hp})$$

MSE: 23.94

When our model uses only a single linear feature (**hp**), it cannot capture non-linearity in the relationship

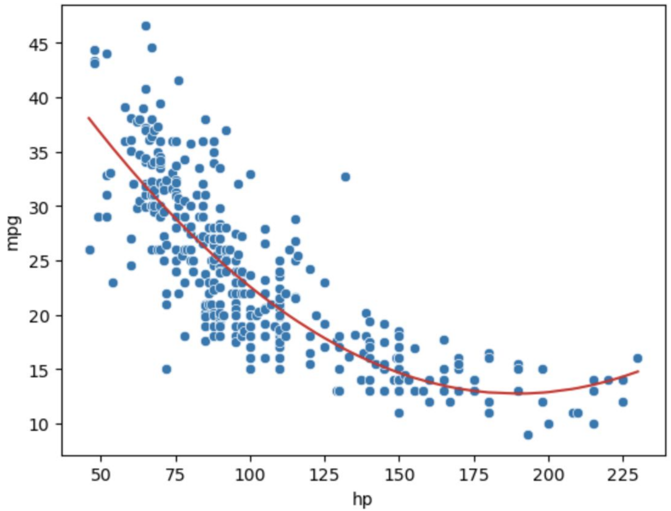
Solution: incorporate a non-linear feature!

Polynomial Features

We create a new feature: the square of the **hp**

$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2)$$

This is still a **linear model**. Even though there are non-linear *features*, the model is linear with respect to θ



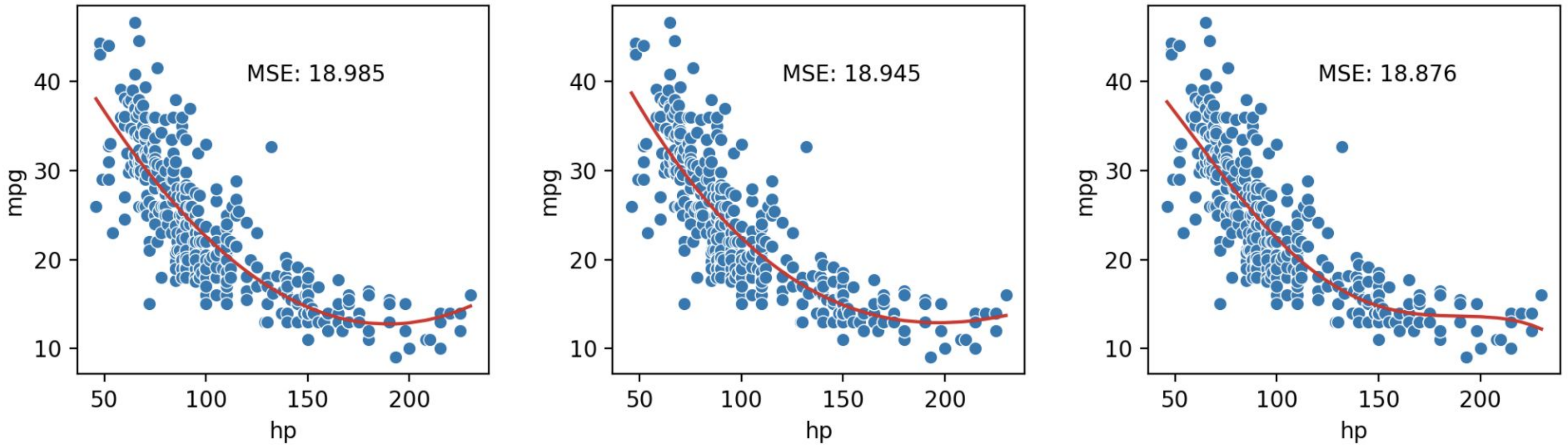
Degree of model: 2
MSE: 18.98

Looking a lot better: our predictions capture the curvature of the data.

Polynomial Features

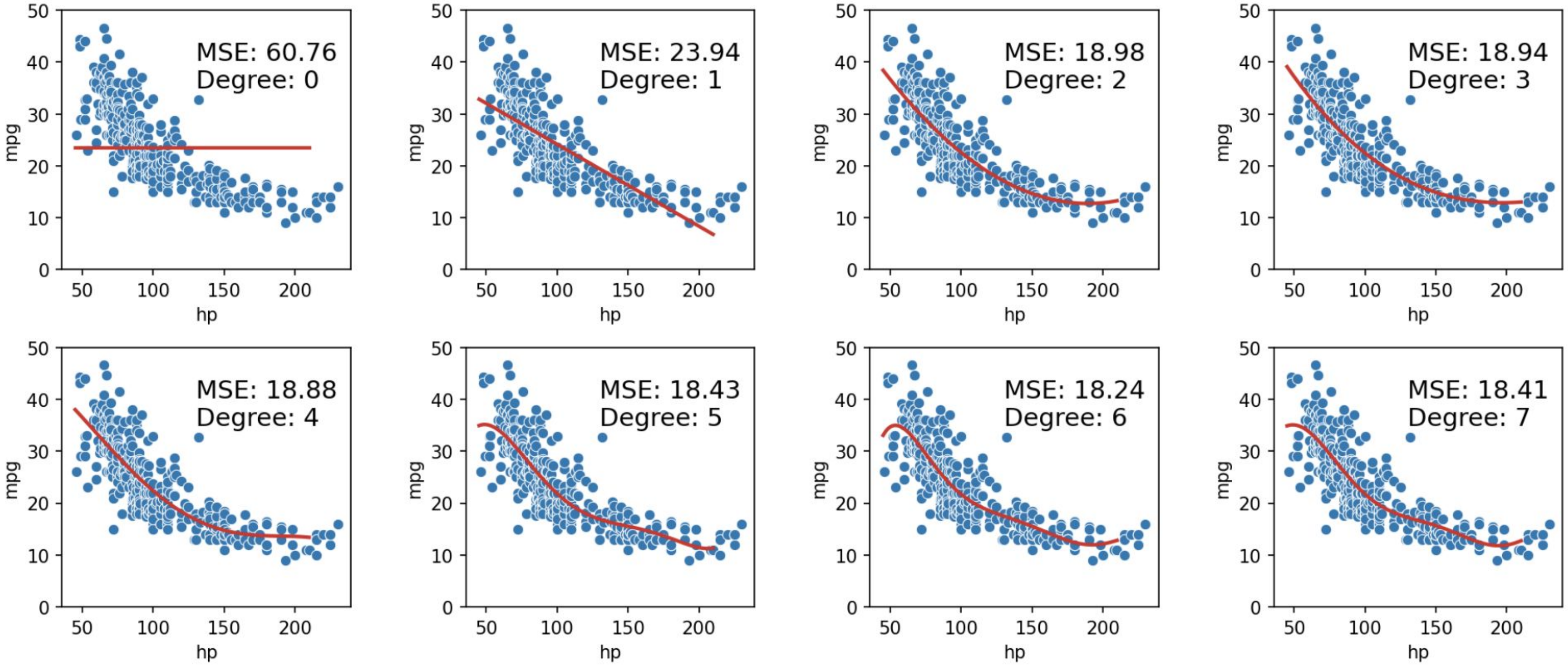
What if we add more polynomial features?

$$\hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2) \qquad \hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2) + \theta_3(\text{hp}^3) \qquad \hat{y} = \theta_0 + \theta_1(\text{hp}) + \theta_2(\text{hp}^2) + \theta_3(\text{hp}^3) + \theta_4(\text{hp}^4)$$



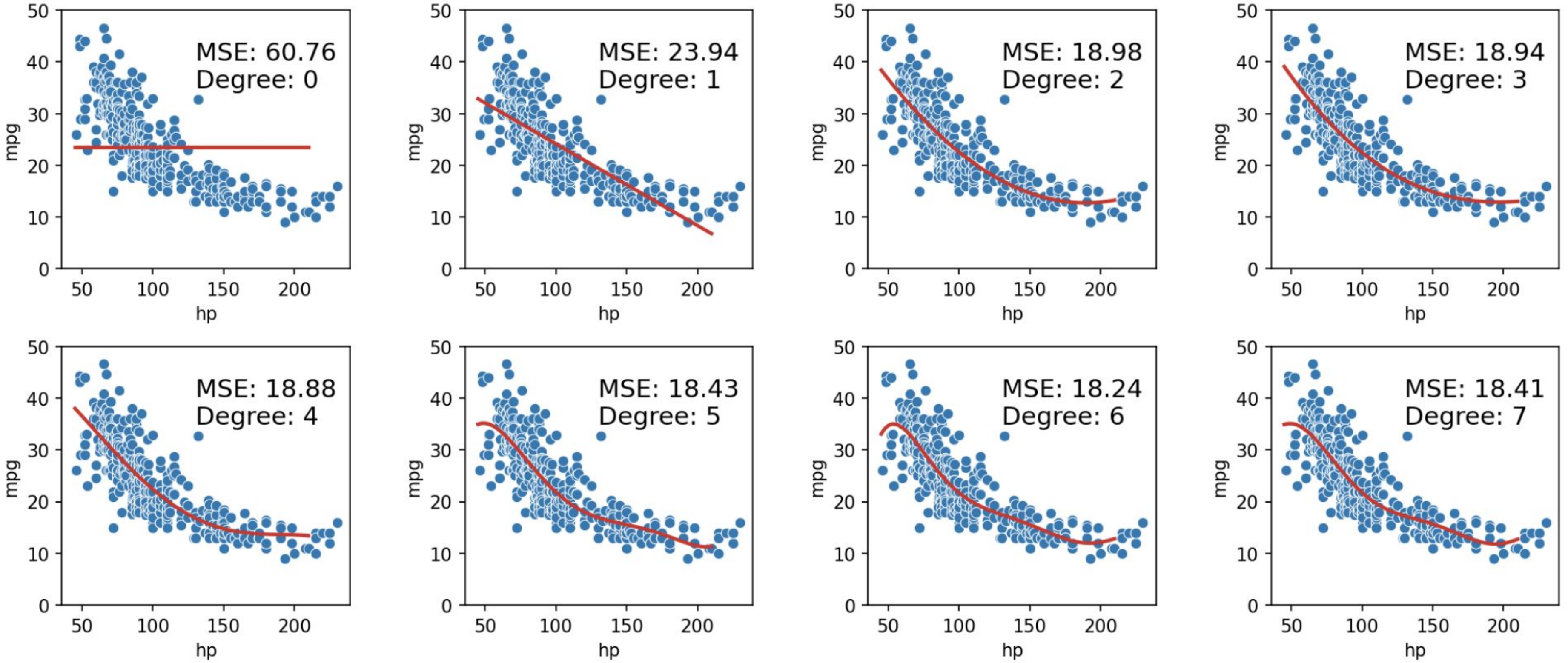
MSE continues to decrease with each additional polynomial term

How Far Can We Take This?



How Far Can We Take This?

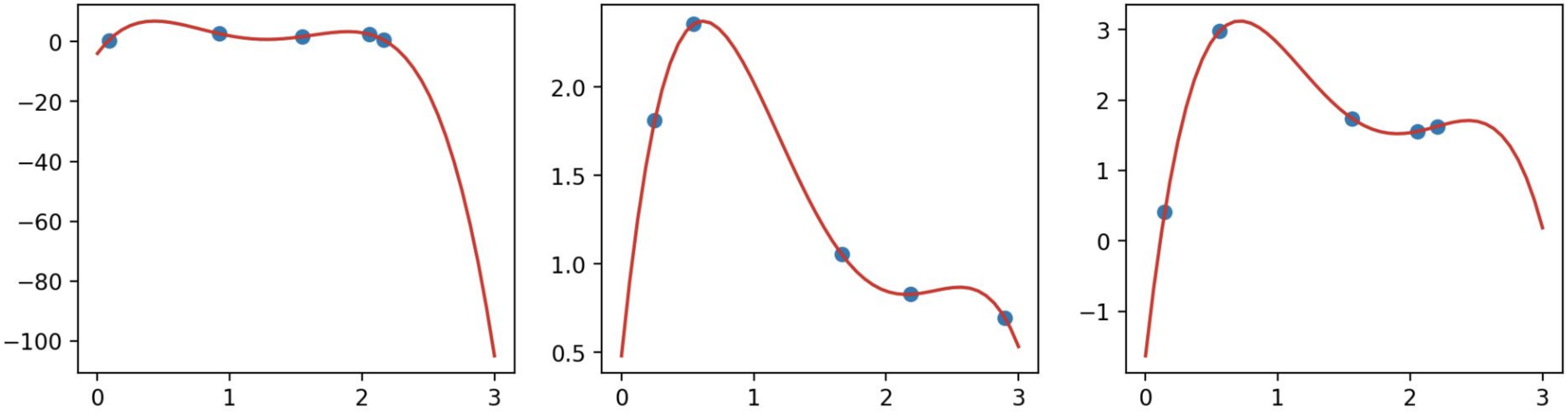
Poll: Which higher-order polynomial model do you think fits best?



An Extreme Example: Perfect Polynomial Fits

Math fact: given N non-overlapping data points, we can always find a polynomial of degree N-1 that goes through all those points.

For example, there always exists a degree-4 polynomial curve that can perfectly model a dataset of 5 datapoints



Complexity and Overfitting

Complexity and Overfitting

Cross-Validation

- Training, Test, and Validation Sets
- K-Fold Cross-Validation

Dataset

Today we will use the `mpg` dataset from the `seaborn` library.

Our task is to use some of the columns and their transformations to predict the value of the `mpg` column.

	mpg	cylinders	displacement	hp	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790	15.6	82	usa	ford mustang gl
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw pickup
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge rampage
396	28.0	4	120.0	79.0	2625	18.6	82	usa	ford ranger
397	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy s-10

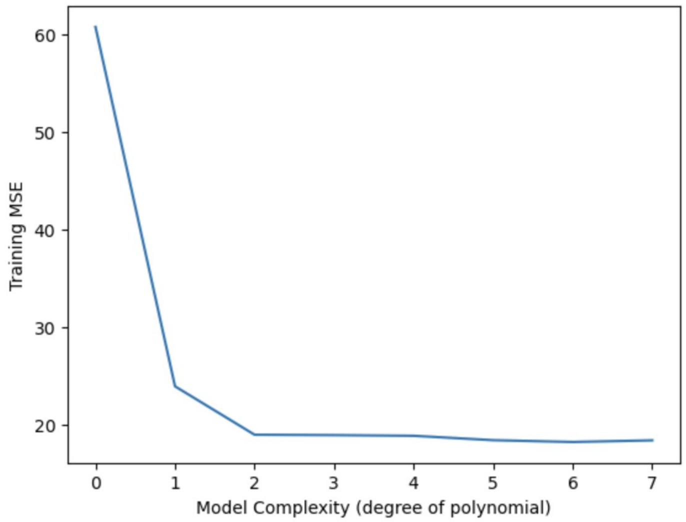
392 rows × 9 columns

Model Complexity

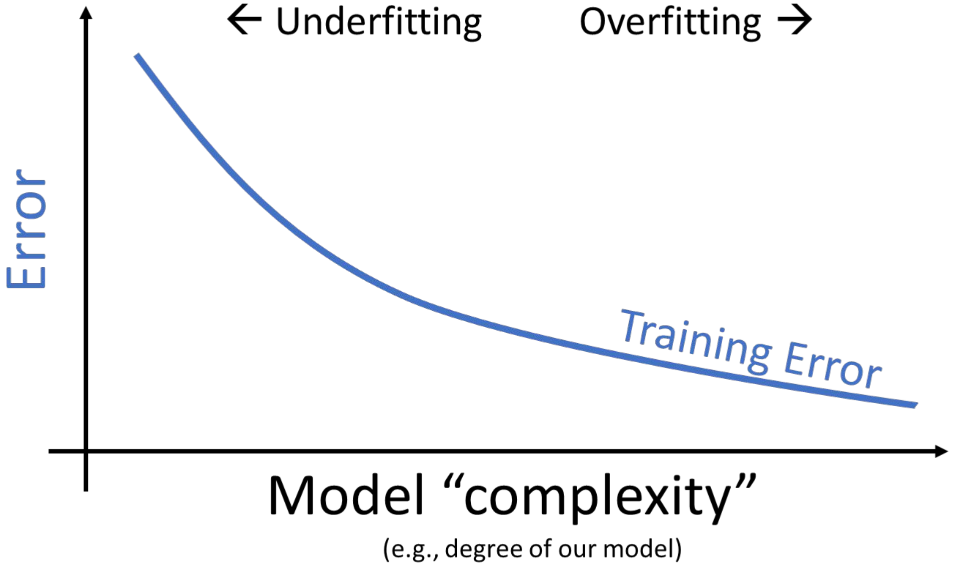
As we continue to add more and more polynomial features, the MSE continues to decrease

Equivalently: as the **model complexity** increases, its *training error* decreases

Our experiment using `vehicles`



General trend for an arbitrary dataset



Seems like a good deal?

Model Performance on Unseen Data

Our `vehicle` models from before considered a somewhat artificial scenario – we trained the models on the *entire* dataset, then evaluated their ability to make predictions on this same dataset

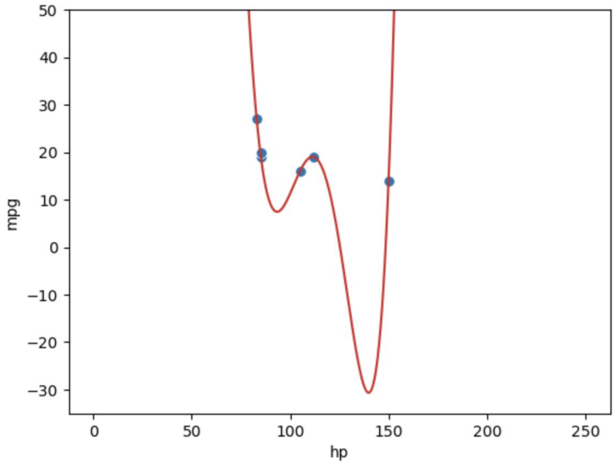
More realistic situation: we train the model on a *sample* from the population, then use it to make predictions on data it didn't encounter during training

Model Performance on Unseen Data

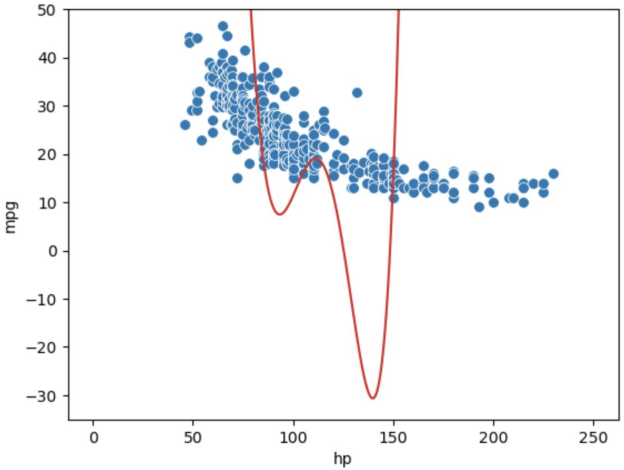
New (more realistic) example:

- We are given a training dataset of just 6 datapoints
- We want to train a model to then make predictions on a *different* set of points

We may be tempted to make a highly complex model (eg degree 5)



Complex model makes perfect predictions on the training data…

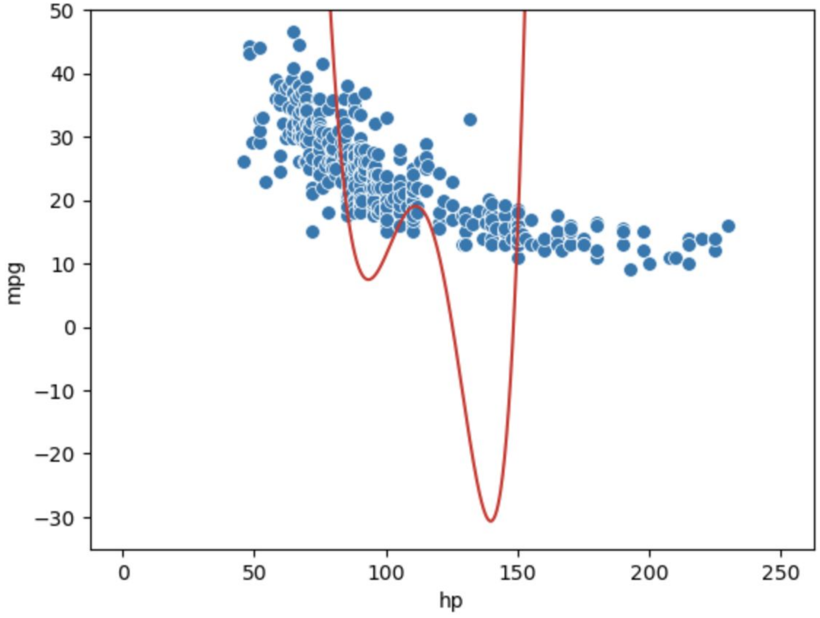


…but performs *horribly* on the rest of the population!

Model Performance on Unseen Data

What went wrong?

- The complex model **overfit** to the training data – it essentially “memorized” these 6 training points
- The overfitted model does not **generalize** well to data it did not encounter during training

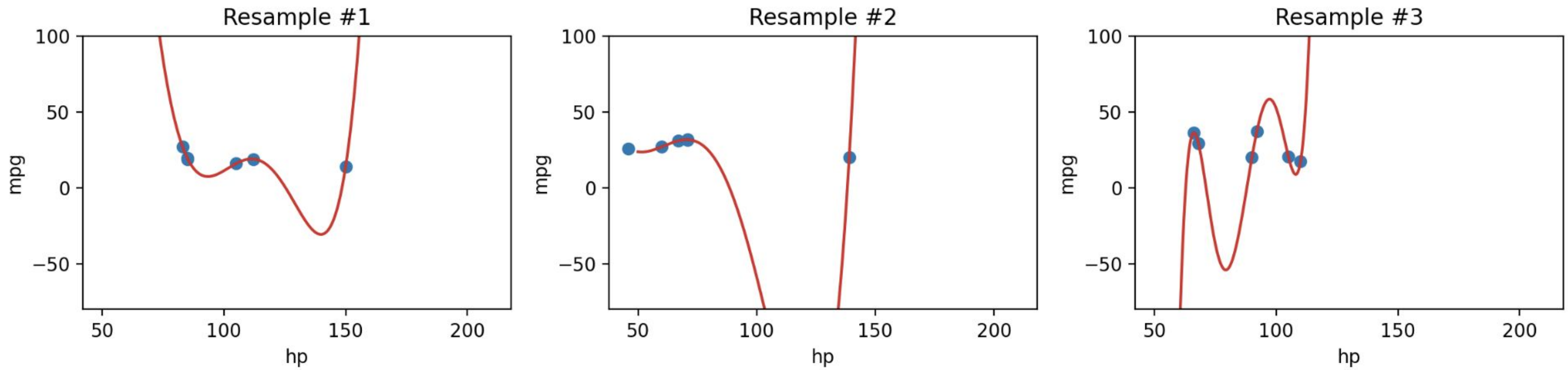


This is a problem: we want models that are generalizable to “unseen” data

Model Variance

Complex models are sensitive to the specific dataset used to train them – they have high **variance**, because they will *vary* depending on what datapoints are used for training them

Our degree-5 model varies erratically when we fit it to different samples of 6 points from `vehicles`

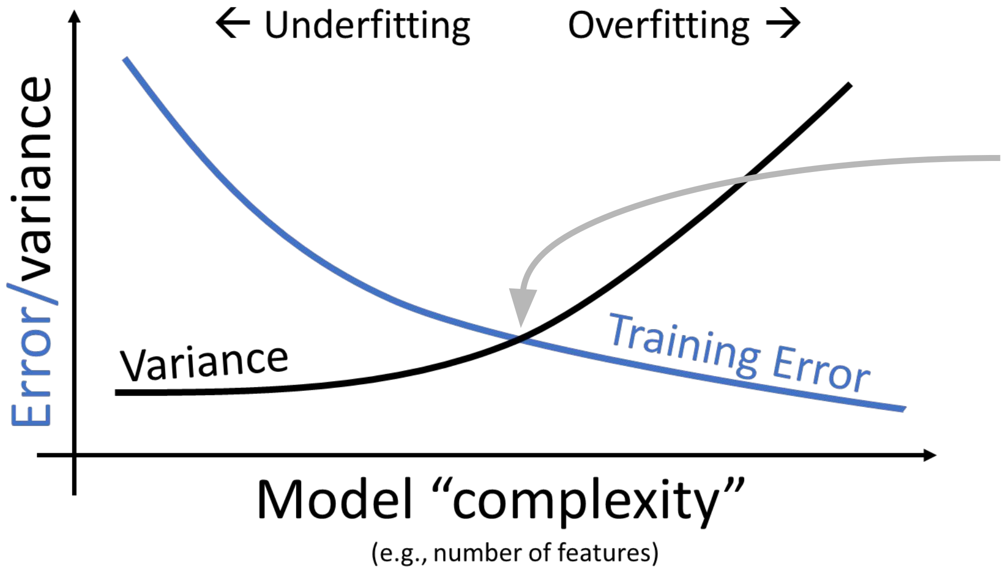


In Machine Learning, this **sensitivity** to data is known as **model variance**.

Error, Variance, and Complexity

We face a dilemma:

- We know that we can **decrease training error** by increasing model complexity
- However, models that are *too* complex start to overfit and do not generalize well – their **high variance** means they can't be reapplied to new datasets



Our goal: find this "sweet spot"

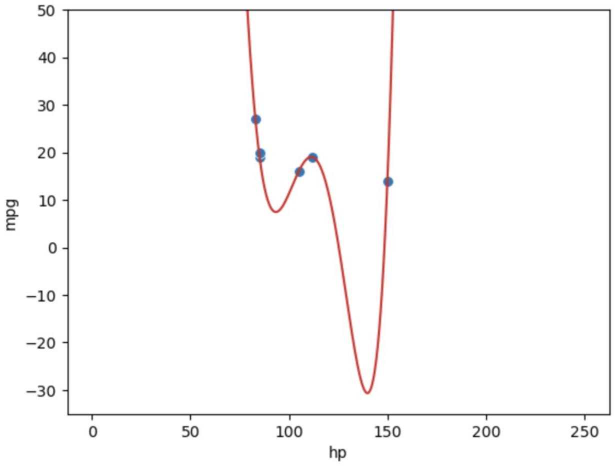
Training, Test, and Validation Sets

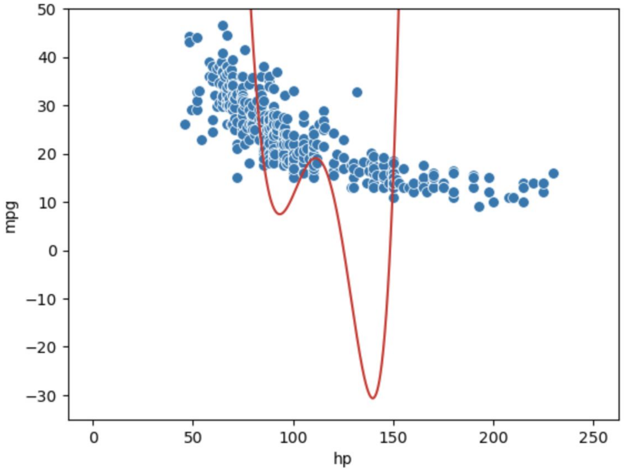
Cross-Validation

- **Training, Test, and Validation Sets**
- K-Fold Cross-Validation

Overfitting

A complex model may not perform well on data it did not encounter during training.





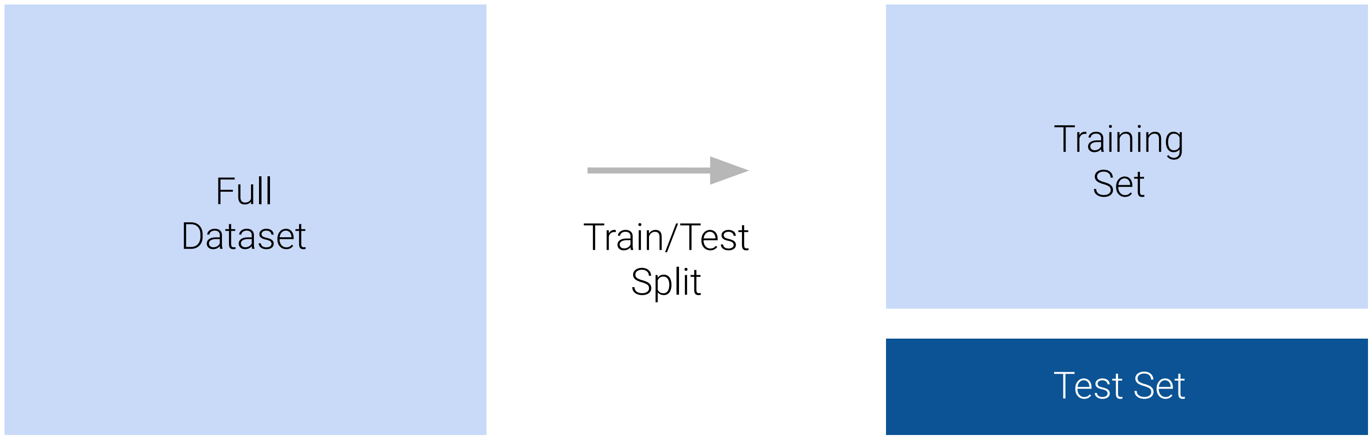
How to quantify performance on this "unseen" data? Introduce a **test set**.

Test Sets

A **test set** is a portion of our dataset that we set aside for testing purposes.

- We do *not* consider the test set when fitting/training the model.
- The test set is only ever touched once: to compute the performance (MSE, RMSE, etc) of the model *after* all fine-tuning has been completed.

Our new workflow for modeling: First, perform a **train-test split** (see documentation). Consider only the training set when designing the model. Then, evaluate on the test set.



Test Sets

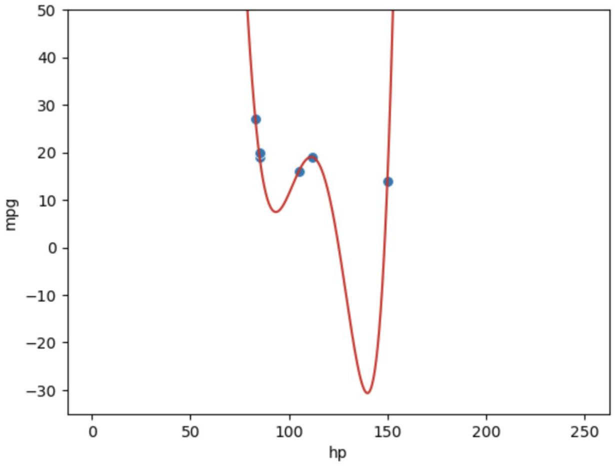
Test sets can be something that we generate ourselves. Or they can be a common dataset whose solution is unknown.

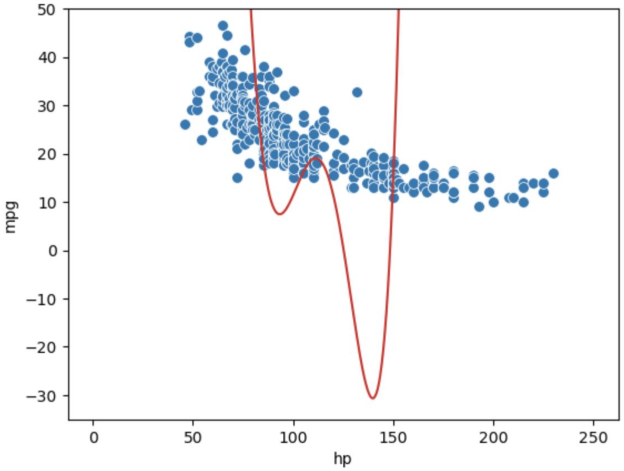
In real world machine learning competitions, competing teams share a **common test set**.

- To avoid accidental or intentional overfitting, the **correct predictions for the test set are never seen by the competitors**.

Where We Left Things

A complex model may not perform well on data it did not encounter during training.

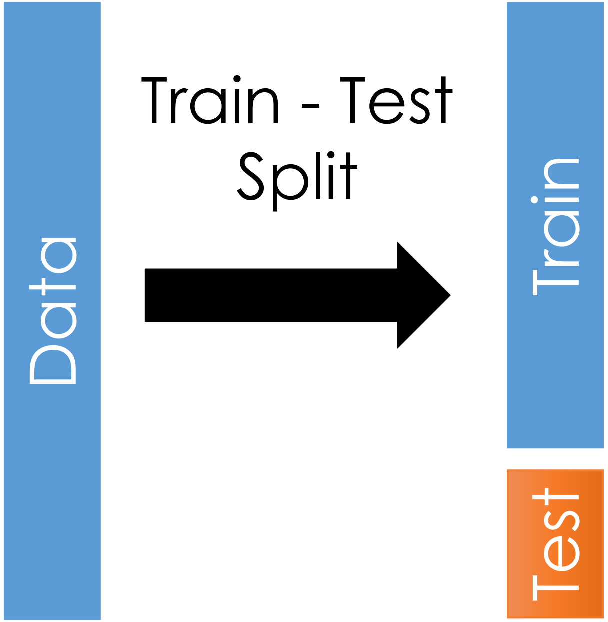




How to quantify performance on this "unseen" data? Introduce a **test set**.

Generalization: *The Train-Test Split*

- **Training Data:** used to fit model
- **Test Data:** check generalization error
- How to split?
  - Randomly, Temporally, Geo…
  - Depends on application (usually randomly)
- What size for training vs test?
  - Larger training set – more complex models
  - Larger test set – better estimate of generalization error
  - Typically use between 75%-90% of the data for the training set



You can only use the test dataset once after deciding on the model.

Validation Sets

What if we were dissatisfied with our test set performance?

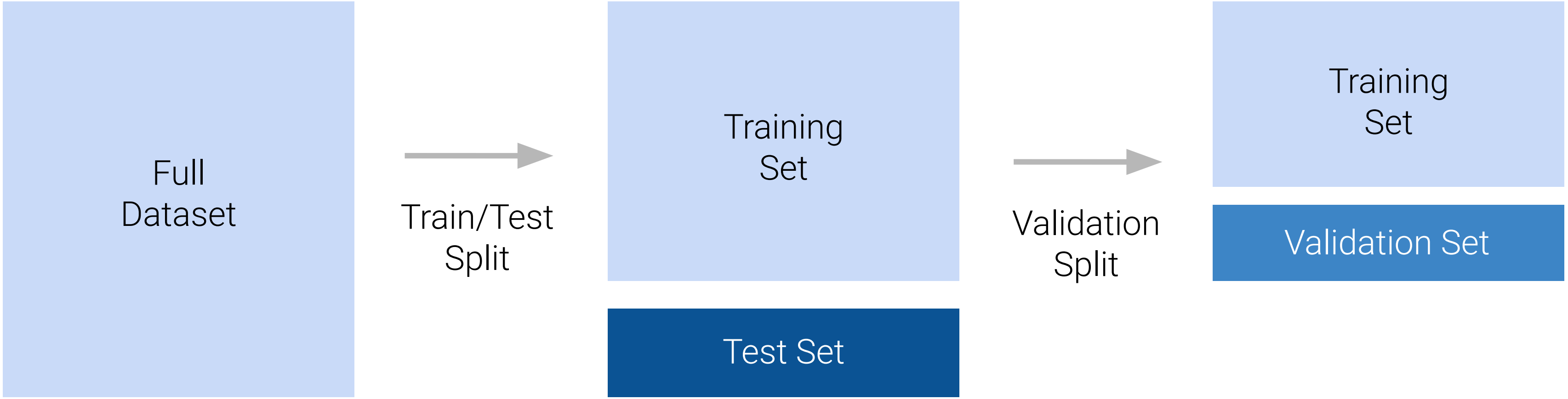
In our current framework, we'd be stuck – we can't then go back and adjust our model, because we'd be *factoring in information from the test set* to design our model. The test set would no longer represent performance on unseen data.

Solution: introduce a **validation set**.

Validation Sets

A **validation set** is a portion of our *training set* that we set aside for assessing model performance while it is *still being developed*.

- Randomly shuffle the data. Then select training/validation and test sets.
- Train model on the training set. Assess performance on the validation set. Adjust the model, then repeat.
- After *all* model development is complete, assess final performance on the test set.

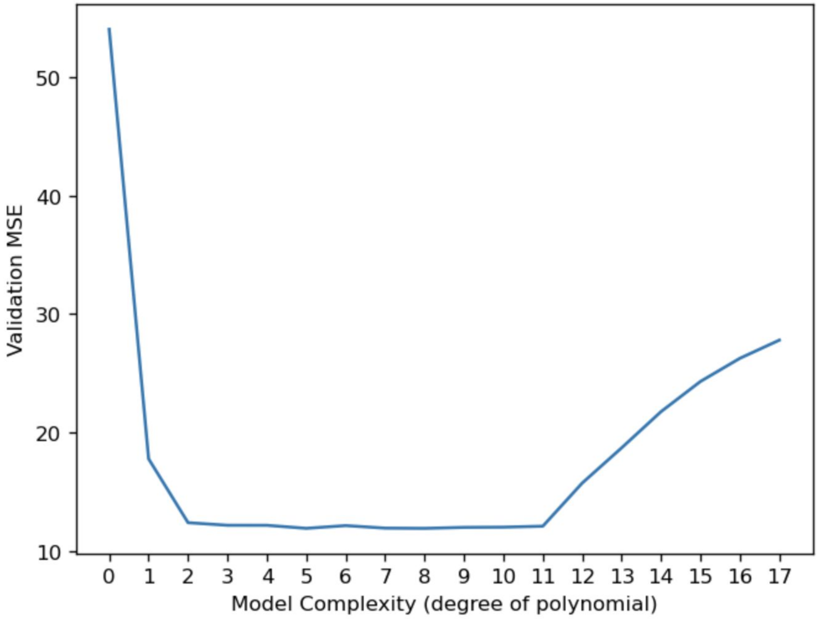


Why do we need to randomly shuffle the data before selecting the training/test/validation sets?

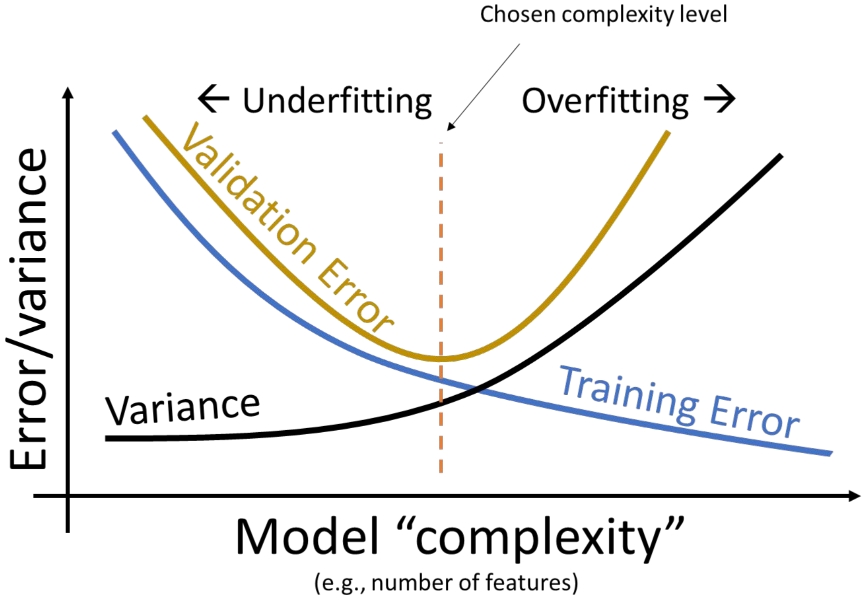
Updating Our Understanding of Model Complexity

Computing the validation error allows us to visualize under- and overfitting.

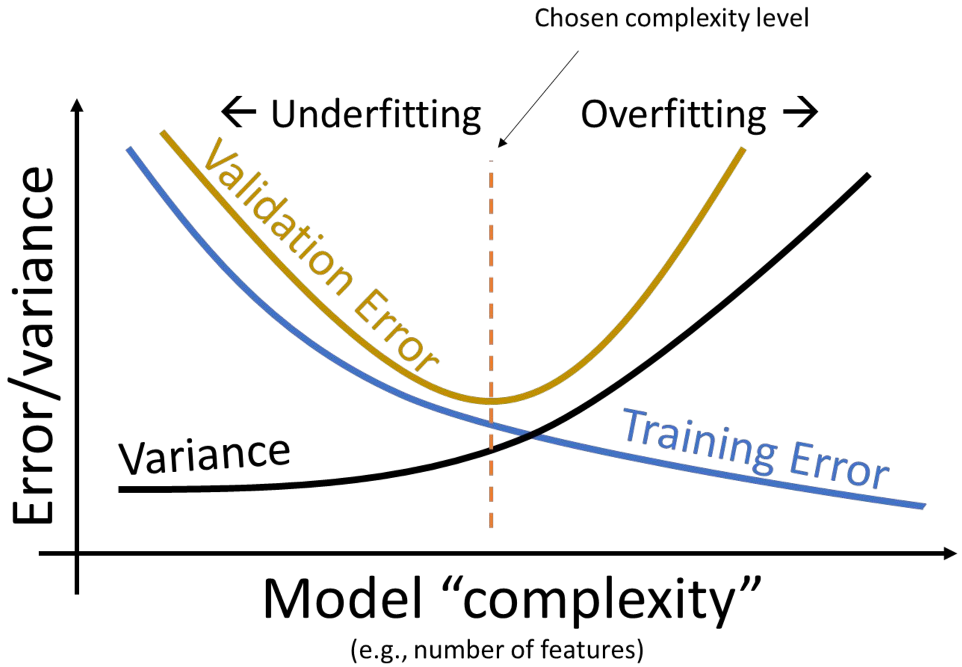
Our experiment using `vehicles`



General trend for an arbitrary dataset



Updating Our Understanding of Model Complexity



Typically, as model complexity increases:

- Training error decreases
- Variance increases
- Error on validation set decreases, then increases

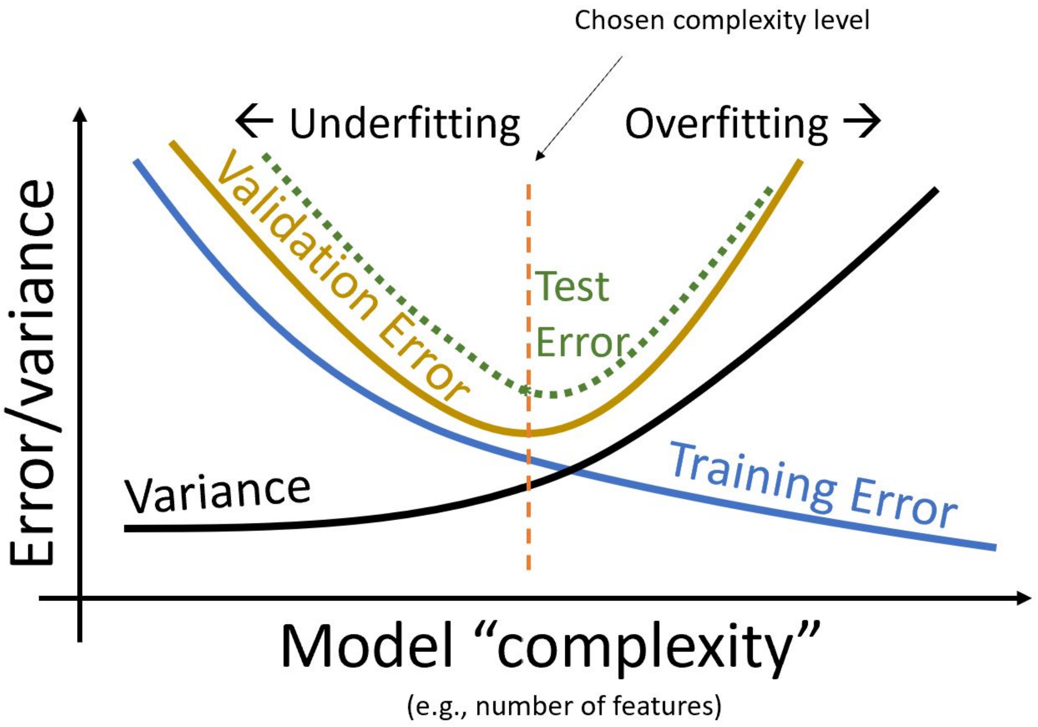
Our goal: Choose the model complexity that minimizes validation error.

We will discuss how in our next lesson

Idealized Training, Validation, and Test Error

As we increase the complexity of our model:

- **Training error** decreases.
- Variance increases.
- Typically, **validation error** decreases, then increases.
- The **test error** is the essentially the same thing as the **validation error**! Only difference is that we are much restrictive about computing the **test error**
    - Don't get to see the whole curve!



K-Fold Cross-Validation

Cross-Validation

- Training, Test, and Validation Sets
- **K-Fold Cross-Validation**

Another View of Validation

Introducing a validation set gave us one "extra" chance to assess model performance.

Specifically, now we understand how the model performs on *one* particular set of unseen data.

- It's possible that we may have, by random chance, selected a set of validation points that was *not* representative of other unseen data that the model might encounter.

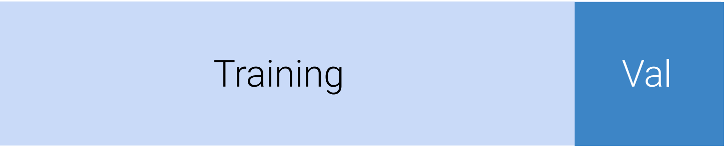
```
Val error from train/validation split #1: 14.6104005581132
Val error from train/validation split #2: 24.755706579814404
Val error from train/validation split #3: 22.2320832995848
```

Ideally: Assess model performance on *several* different validation sets before touching the test set.

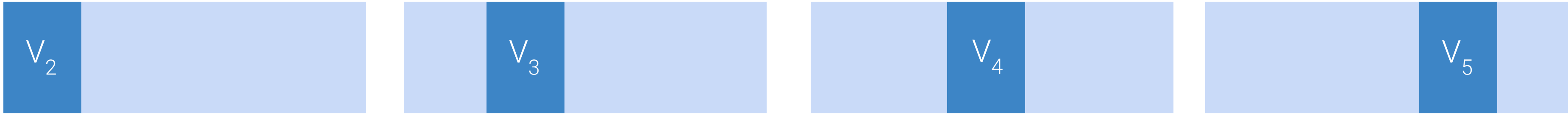
Validation Folds

In our original validation split, we set aside x% of the training data to use for validation.

- For example, 20% of the training data is used for validation



We could have selected *any* 20% portion of the training data for validation.

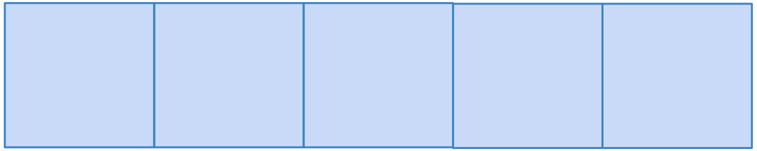


In total, there are 5 non-overlapping “chunks” of datapoints we could set aside for validation.

Validation Folds

The common term for one of these chunks is a "fold".

- Our training data has 5 folds, each containing 20% of the datapoints.



Another perspective: we actually have 5 validation sets "hidden" in our training set.

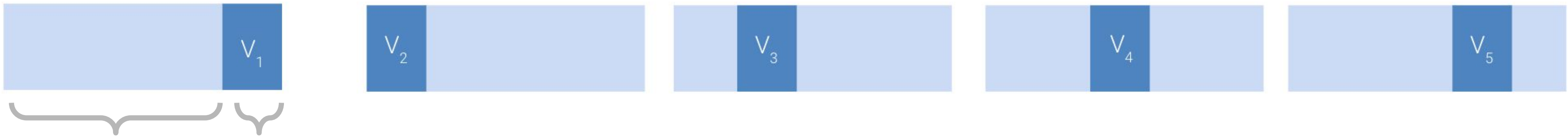
In **cross-validation**, we perform validation splits for *each* of these folds.

K-Fold Cross-Validation

For a dataset with K folds:

- Pick one fold to be the validation fold.
- Train model on data from every fold *other* than the validation fold.
- Compute the model's error on the validation fold and record it.
- Repeat for all K folds.

The **cross-validation error** is the average error across all K validation folds.



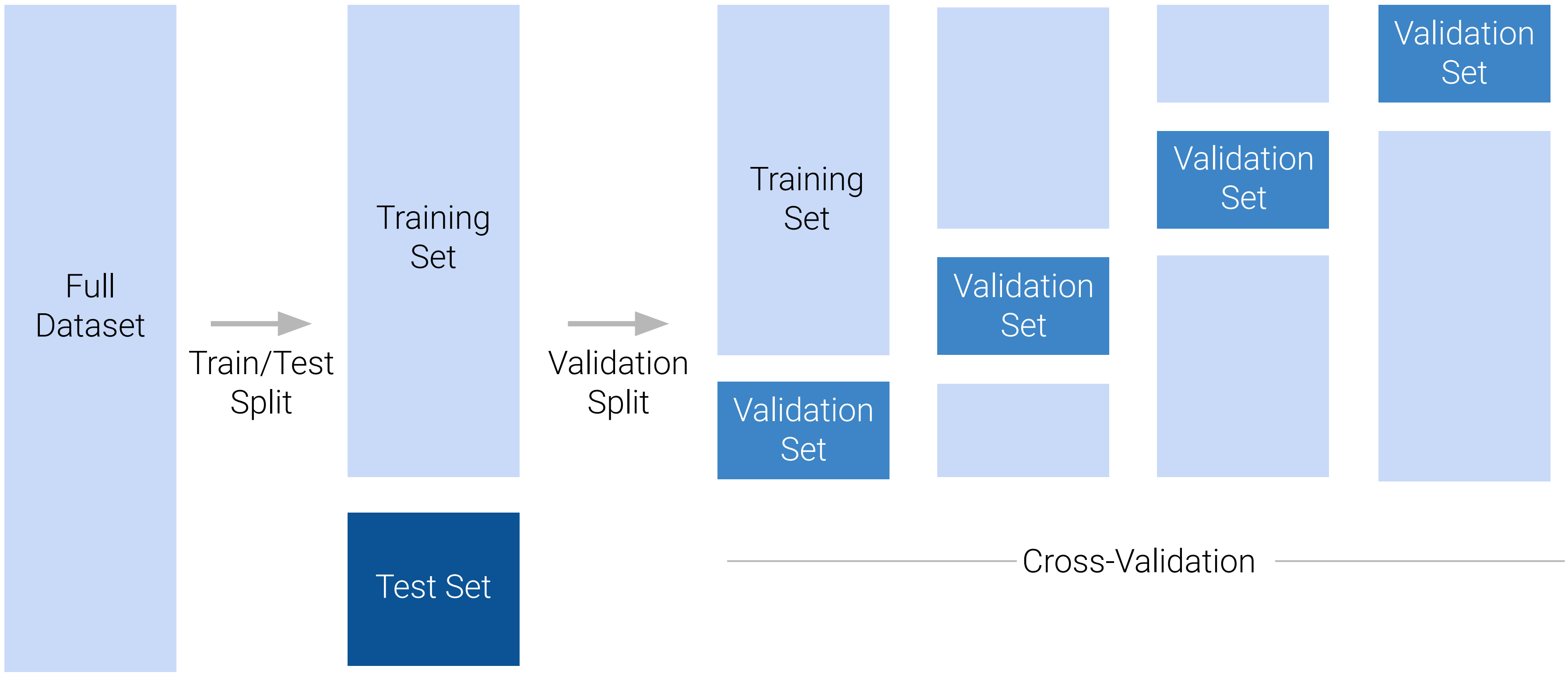
Train model on all other folds

Compute MSE on V_1

Validation error #1 | Validation error #2 | Validation error #3 | Validation error #4 | Validation error #5

Cross-validation error = mean of validation errors #1 to #5

Model Selection Workflow



Hyperparameter: Terminology

Cross-validation is often used for **hyperparameter** selection.

In machine learning, a **hyperparameter** is a value that controls the learning process itself.

Hyperparameter: Value in a model chosen *before* the model is fit to data.

- Cannot solve for hyperparameters via calculus/numerical optimization - instead we must choose it ourselves.
- Examples
  - For our example today, we built a series of models each with increasing orders of horsepower. In this case, the hyperparameter is the degree or k that controlled the order of our polynomial.
  - Regularization penalty,(to be introduced in next lesson)

We use:

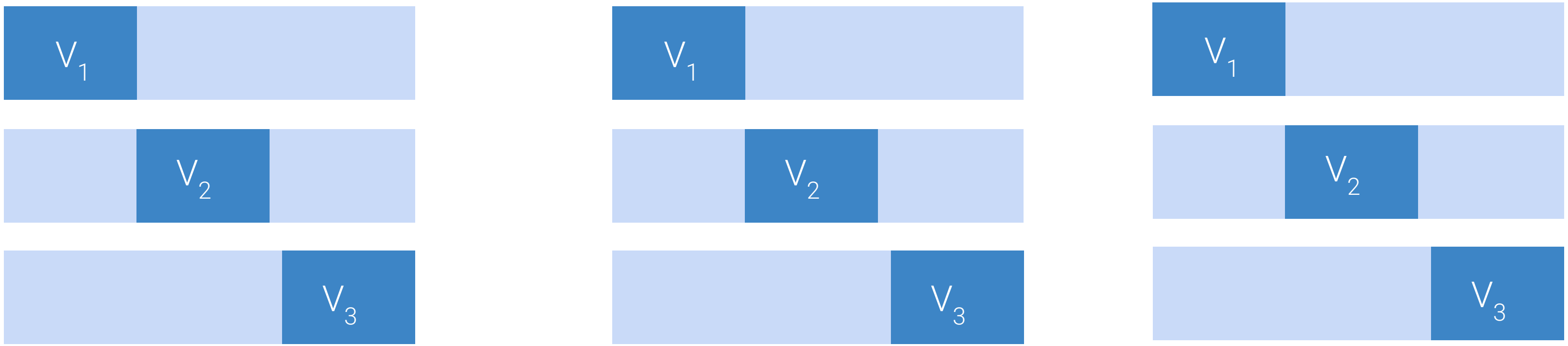
- The **training set** to **select parameters**.
- The **validation set** (a.k.a. development set) (a.k.a. cross validation set) to **select hyperparameters**, or more generally, between different competing models.

Hyperparameter Tuning

To select a hyperparameter value via cross-validation:

- List out several different “guesses” for the best hyperparameter.
- For each guess, run cross-validation to compute the CV error for that choice of hyperparameter value.
- Select the hyperparameter value with lowest CV error.

Example: Guesses for “best” hyperparameter (for example degree of polynomial) are 2, 4, and 10. We decide to apply 3-fold cross-validation.



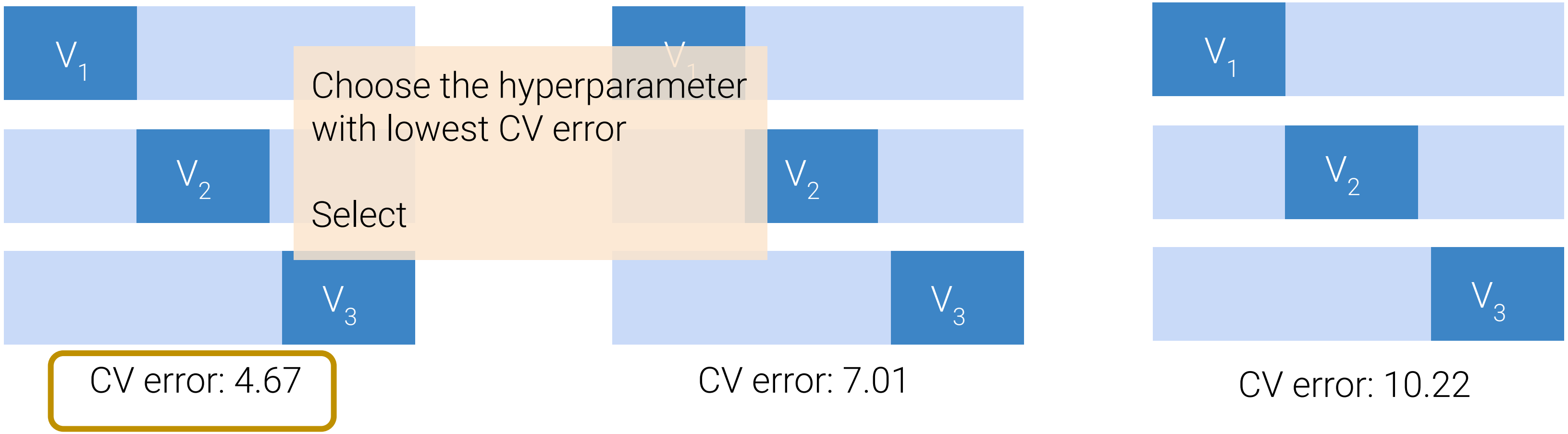
CV error: 4.67 CV error: 7.01 CV error: 10.22

Hyperparameter Tuning

To select a hyperparameter value via cross-validation:

- List out several different “guesses” for the best hyperparameter
- For each guess, run cross-validation to compute the CV error for that choice of hyperparameter value
- Select the hyperparameter value with lowest CV error

Example: Guesses for “best” hyperparameter (for example degree of polynomial) are 2, 4, and 10. We decide to apply 3-fold cross-validation.



To select the best polynomial degree for our model out of 5 possible options, how many times will we compute the MSE during 4-fold cross-validation?

To select the best polynomial degree for our model out of 5 possible options, how many times will we compute the MSE during 4-fold cross-validation?

Solution: 5 models with 4 folds each = 20 total MSEs

A More Complex Example

Suppose we have a dataset with 9 features.

- We want to decide which of the 9 features to include in our linear regression.

hp	weight	displacement	hp^2	hp weight	hp displacement	weight^2	weight displacement	displacement^2
130.0	3504.0	307.0	16900.0	455520.0	39910.0	12278016.0	1075728.0	94249.0
165.0	3693.0	350.0	27225.0	609345.0	57750.0	13638249.0	1292550.0	122500.0
150.0	3436.0	318.0	22500.0	515400.0	47700.0	11806096.0	1092648.0	101124.0
150.0	3433.0	304.0	22500.0	514950.0	45600.0	11785489.0	1043632.0	92416.0
140.0	3449.0	302.0	19600.0	482860.0	42280.0	11895601.0	1041598.0	91204.0
...	...	...	...	...	...	...	...	...
86.0	2790.0	140.0	7396.0	239940.0	12040.0	7784100.0	390600.0	19600.0
52.0	2130.0	97.0	2704.0	110760.0	5044.0	4536900.0	206610.0	9409.0
84.0	2295.0	135.0	7056.0	192780.0	11340.0	5267025.0	309825.0	18225.0
79.0	2625.0	120.0	6241.0	207375.0	9480.0	6890625.0	315000.0	14400.0
82.0	2720.0	119.0	6724.0	223040.0	9758.0	7398400.0	323680.0	14161.0

Tweaking Complexity via Feature Selection

With 9 features, there are 2^9 different models.One approach:

- For each of the 2^9 linear regression models, compute the **validation MSE**.
- Pick the model that has the lowest **validation MSE**.

Runtime is exponential in the number of parameters!

	hp	w	dis	hp^2	hp w	hp dis	w^2	w dis	dis^2	MSE
Least complex model	no	no	no	no	no	no	no	no	no	172.2
	no	no	no	no	no	no	no	no	yes	77.3
	no	no	no	no	no	no	no	yes	no	85.3
	no	no	no	no	no	no	no	yes	yes	77.2
	no	no	no	no	no	no	yes	no	no	81.1
	no	no	no	no	no	no	yes	no	yes	74.6
	...									
Most complex model	yes	yes	yes	yes	yes	yes	yes	yes	yes	195.3