Cross-Validation of Machine Learning Models

Review of Overfitting

How good should we make our model?

Overfitting

Overfitting - when our model assumes that the available data says more about the real world than the data is actually capable of predicting

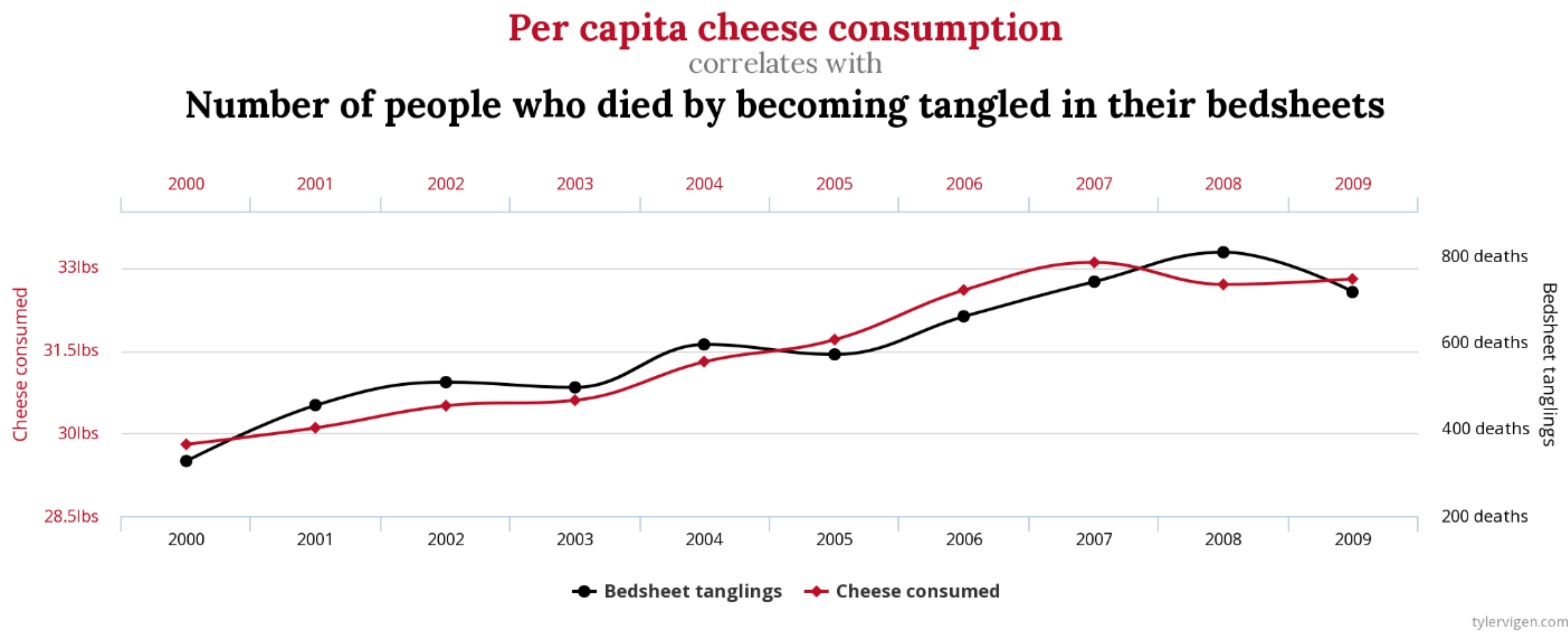
Underfitting

Underfitting - when our model has not yet learned all useful information available through the data

How to Overfit - Variable Choice

We can overfit by including variables in our data that *seem* relevant to the problem, but are not actually related to the outcome of interest

- Our model may not care about causation, but we should!



Remember this??

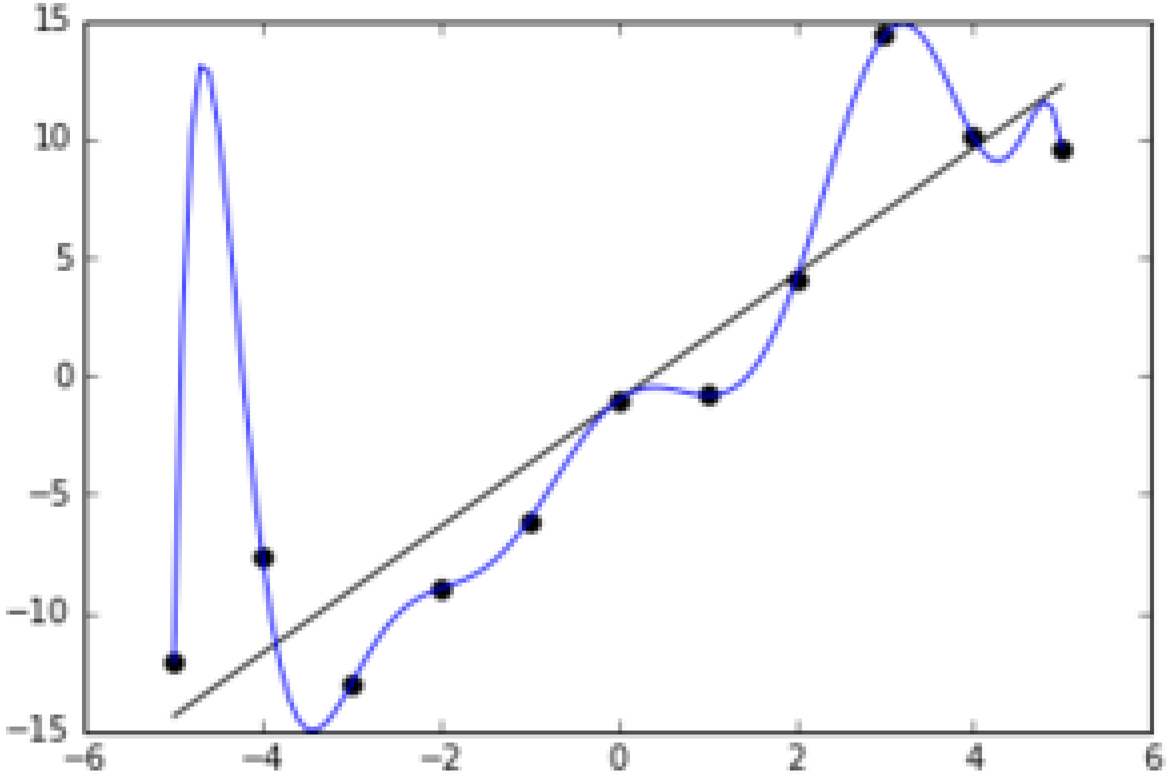
Thought Exercise: How similar are twins, really?

How to Overfit - Model Complexity

We can also overfit a model by choosing a model of high complexity

- Remember that not all variation can be modeled, and we may have to accept some inaccuracy in a realistic model of the world

Which model is better? Blue or black?



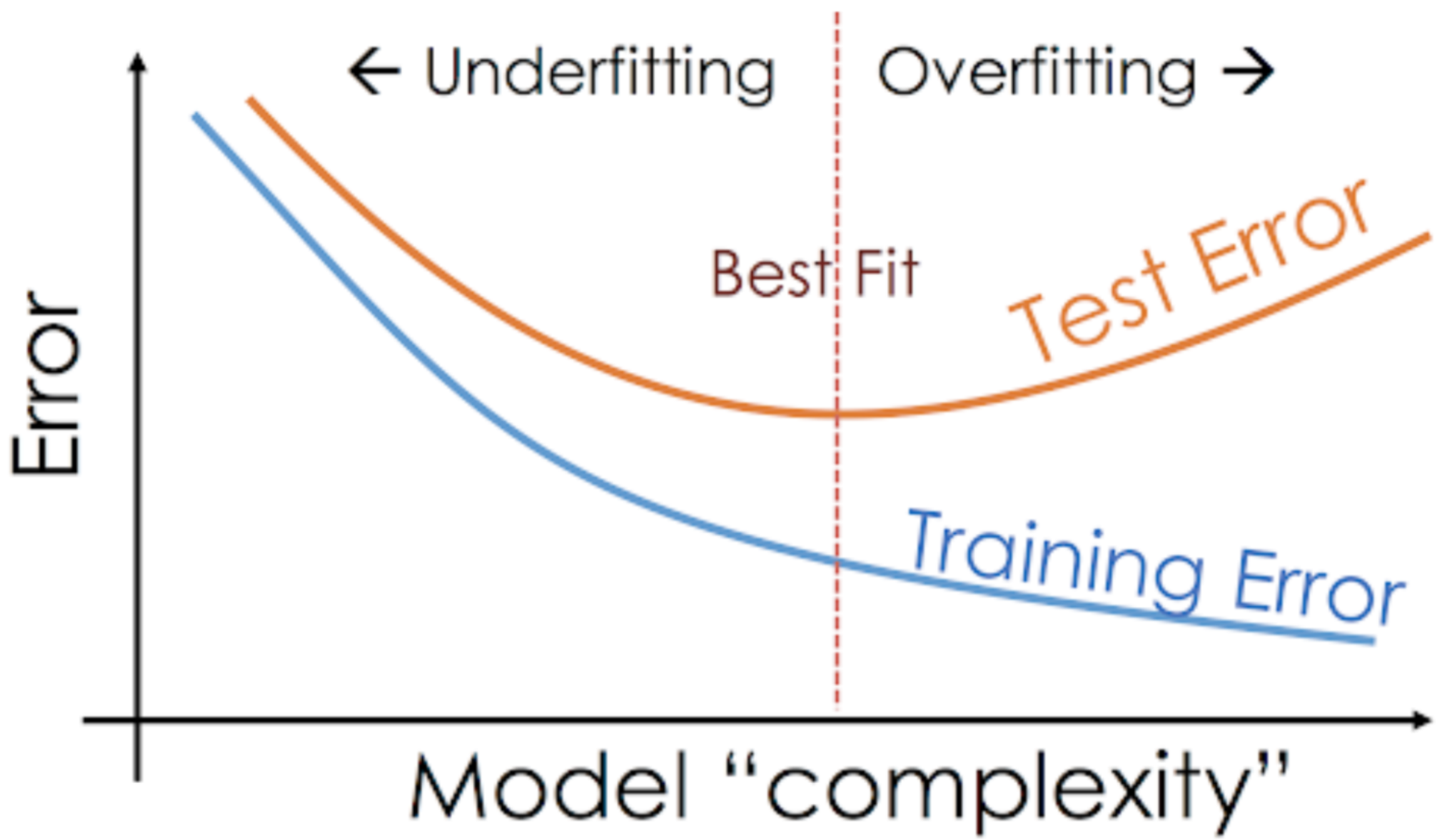
Finding the sweet spot

In order to create a high-quality model, we need to

1. Choose the right variables
2. Choose the right model complexity

Knowing how to do this will take practice! We just need to keep trying to refine* our models!

$*$ refine does **not** necessarily mean improving accuracy scores



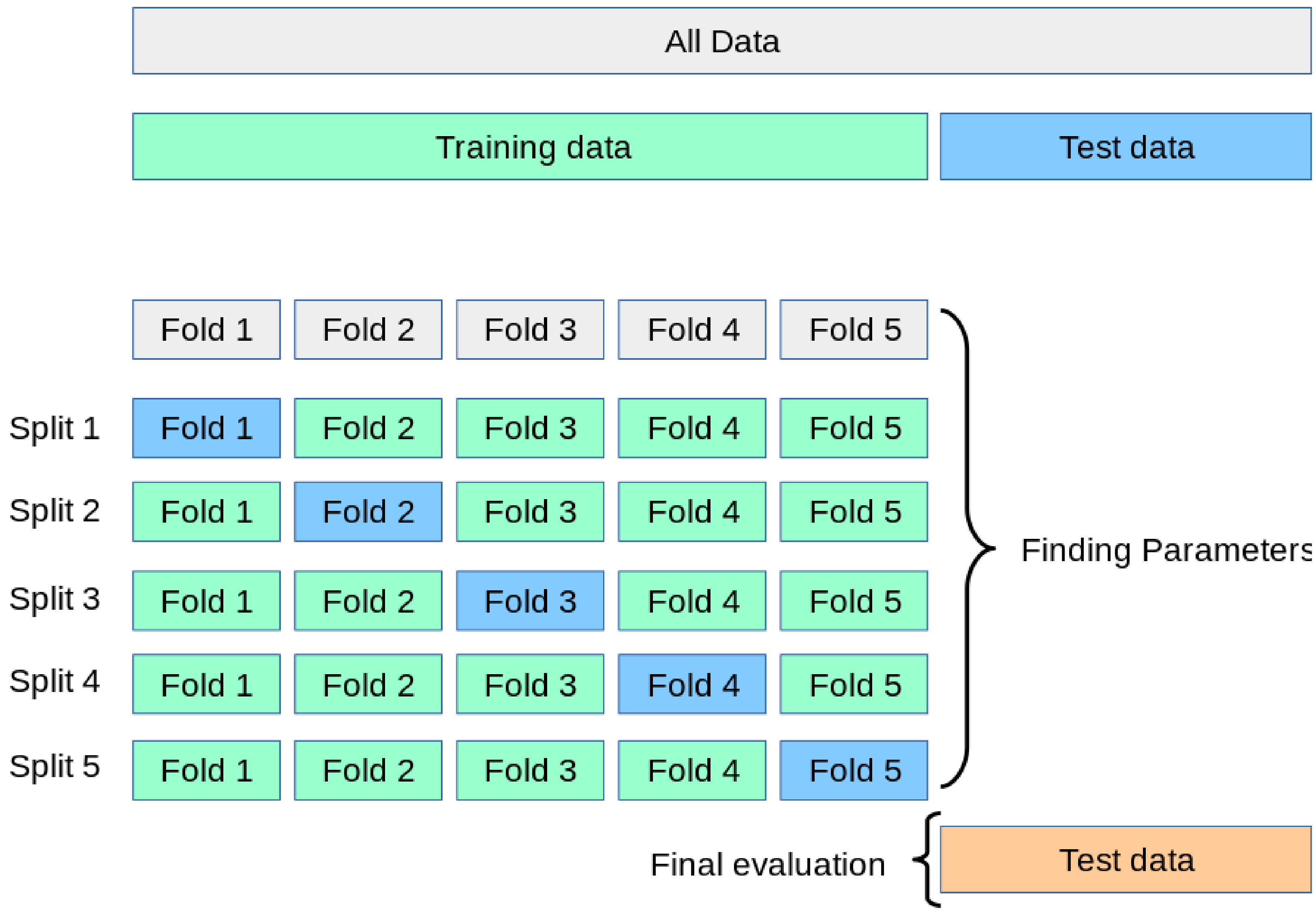
← Underfitting
Overfitting →
Best Fit
Test Error
Training Error
Error
Model "complexity"

Cross Validation

- We need tools to help us determine whether or not we have overfit our model
- Cross-validation provides that toolkit
- It can be used with ANY model!

k-Fold Cross Validation

1. Separate training and testing data
2. Randomly assign training data to k equally-sized portions
3. Train the model k times using identical model parameters, using each fold as test data **once**
4. Record the performance of each iteration
5. Calculate the average performance of the k models
6. If performance is satisfactory, apply model to testing data for final validation of predictive ability
    - Otherwise, refine the model parameters and go back to step 2.


All Data
Training data
Test data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 1
Split 2
Split 3
Split 4
Split 5
Finding Parameters
Final evaluation
Test data

Why?

Using cross-validation

- I might see that the accuracy of one split is very different from the accuracy of another split
- This reveals overfitting!
- It also provides more realistic expectations for the performance of a model than we might otherwise have.

A word of caution

Ensure that the data we use to train our model actually resembles the data that we will observe in practice

No matter how well-trained our model is, the model **will fail** if it has not been trained on representative data!

Implementing Cross Validation

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.tree import DecisionTreeClassifier as dt
from sklearn.metrics import accuracy_score

mnist = pd.read_csv(
    "https://github.com/dustywhite7/pythonMikkeli/blob/"
    +"master/exampleData/mnistTrain.csv?raw=true")

# Separate our features from our labels
y = mnist['Label']
x = mnist.drop('Label', axis=1)
```

Implementing Cross Validation

```
# Make 5 folds in the data
skf = StratifiedKFold(n_splits=5)

# Create the model
clf = dt(max_depth=15)

# Create a list to store accuracy values
accuracy = []
n=1
```

Implementing Cross Validation

```
# For loop to train the model on each fold
for train_index, test_index in skf.split(x, y):
    # Store the folded data
    x_train = x.loc[train_index, :]
    x_test = x.loc[test_index, :]
    y_train =  y[train_index]
    y_test = y[test_index]

    # Fit the model
    clf.fit(x_train, y_train)

    # Calculate model accuracy on left-out data
    acc = accuracy_score(clf.predict(x_test), y_test)

    # Print results
    print("Fold {0} Accuracy: {1}%".format(n, round(acc*100, 2)))

    # Store results
    accuracy.append(acc)

    # Add one to our label count
    n+=1
```

Implementing Cross Validation

```
# Print overall results
print("\nAverage Accuracy: {}%".format(
    round(np.mean(accuracy)*100, 2)))
print("Accuracy Standard Deviation: {}%".format(
    round(np.std(accuracy)*100), 2))
```

Output:

```
Fold 1 Accuracy: 74.08%
Fold 2 Accuracy: 74.55%
Fold 3 Accuracy: 75.3%
Fold 4 Accuracy: 76.35%
Fold 5 Accuracy: 72.62%

Average Accuracy: 74.58%
Accuracy Standard Deviation: 1.0%
```

After Cross-Validation

Once your cross-validation demonstrates that you have minimized overfitting, it's time to retrain your model

- We dont USE our cross-fitting models
- Instead, retrain the model on the ENTIRE training dataset
- This maximizes the information your model can use to predict future observations
- Now it is time to use the testing data that you reserved from the original data set to evaluate performance!

Lab Time!