Model Evaluation

Many models enter ... only one leaves alive!!

We can now make many models even with just what we have learned so far. Considering each of the following models based on the target variable y and predictors x_1, x_2, and x_3:

- Linear Regression models:
  - Model 1A: $y = \beta_0 + \beta_1 x_1$
  - Model 1B: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$
  - Model 1C: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$
- Regression trees
  - Model 2A: Tree with Depth 2
  - Model 2B: Tree with Depth 3
  - Model 2C: Tree with Depth 4

How can we choose which model is the best?

Underfitting and Overfitting

Two major issues in a model are **Underfitting** and **Overfitting**

- Underfitting: Model is too simple to capture the underlying pattern.
- Overfitting: Model is too complex, capturing noise in the data.

Both are undesirable as they affect the model's generalization capability, meaning it will be hard to trust this model to be reliable on future data.

Linear Regression

Underfitting in Linear Regression models

- Example: Predicting house prices using only the number of rooms.
- Simplistic model will miss other factors like location, age, etc.
- In this case, adding more variables can improve the model.

Overfitting in Linear Regression

- Example: Predicting stock prices using hundreds of variables.
- Complex model may seem to fit the data well but will fail to generalize.
- Solution: Use fewer variables, or regularization techniques (tbd)

Regression Trees

Underfitting in Regression Trees

- Example: Predicting student grades using only attendance.
- A shallow tree will not capture other factors like study time, prior knowledge, etc.
- Solution: Increase the tree depth or use more variables.

Overfitting in Regression Trees

- Example: Predicting weather using minute-by-minute data for several years.
- A deep tree may capture noise rather than the actual pattern.
- Solution: Set a maximum depth.

Introduction to Model Tuning

You would like to find the best model that captures the underlying pattern in the data. One way to do this we have seen is called estimation. Another way is **tuning**

- Parameters: Internal configurations for the model learned from the data. We estimate parameters.
- Hyperparameters: Configurations external to the model, set prior to estimation. We will tune hyperparameters. Two examples are what variables to include in a linear regression model or what depth to fit a regression tree at.

Tuning is often as important as estimation. A well-tuned model will achieve better performance, generalization, and interpretability.

Measures of Goodness: R^2 and MSE

How do we tune? We need a few more tools in our tool kit. A **metric** is a numerical result from model output. A few popular metrics are

- R^2: Coefficient of determination. Higher values (closer to 1) are better.
- MSE: Mean Squared Error. Lower values are better.
- rMSE: Root Mean Squared Error. The square root of MSE.

Both are used to evaluate and compare models.

Mean Squared Error (MSE)

MSE is the average squared difference between the estimated values for the target variable and the actual values for the target variable.

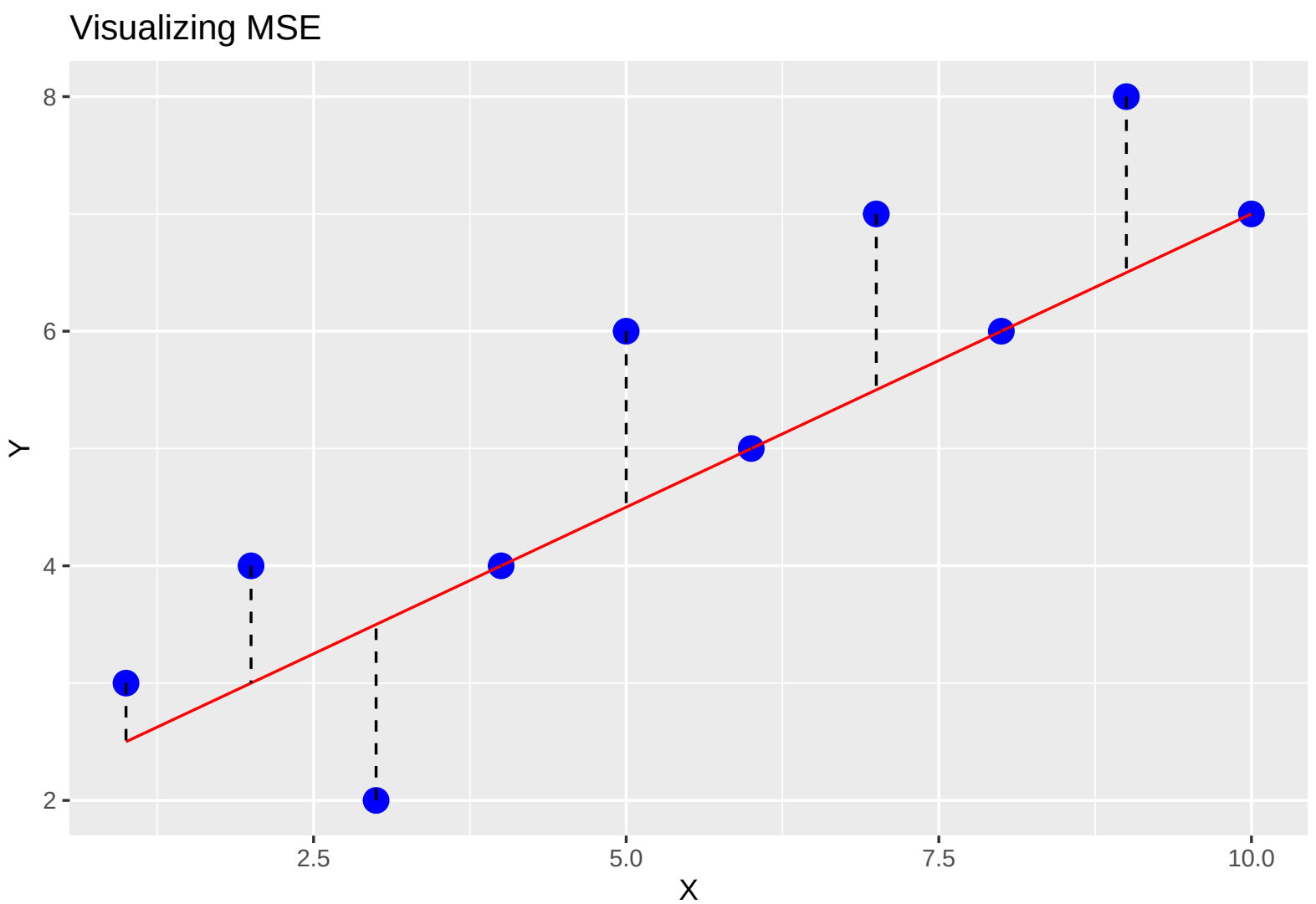
- Formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  where $\hat{y}_i$ are the predicted values for observation i.

- Logic: Averaged squared difference between observed and predicted values.
- Importance for Tuning:
  - Lower MSE implies the model is closer to the true values.
  - Useful for comparing configurations of your model

Mean Squared Error (MSE)



Blue points: Actual y, Red line: Predicted y_hat, Dashed lines: Error

Coefficient of Determination (R^2)

R^2 is a measure of percent of variation explained. Essentially, if the variance of the target variable with no predictors is 100 and the variance of the regression model errors is 50, we explained 50% of the variance. You could think of this as how thin predicted errors are compared to the full variance.

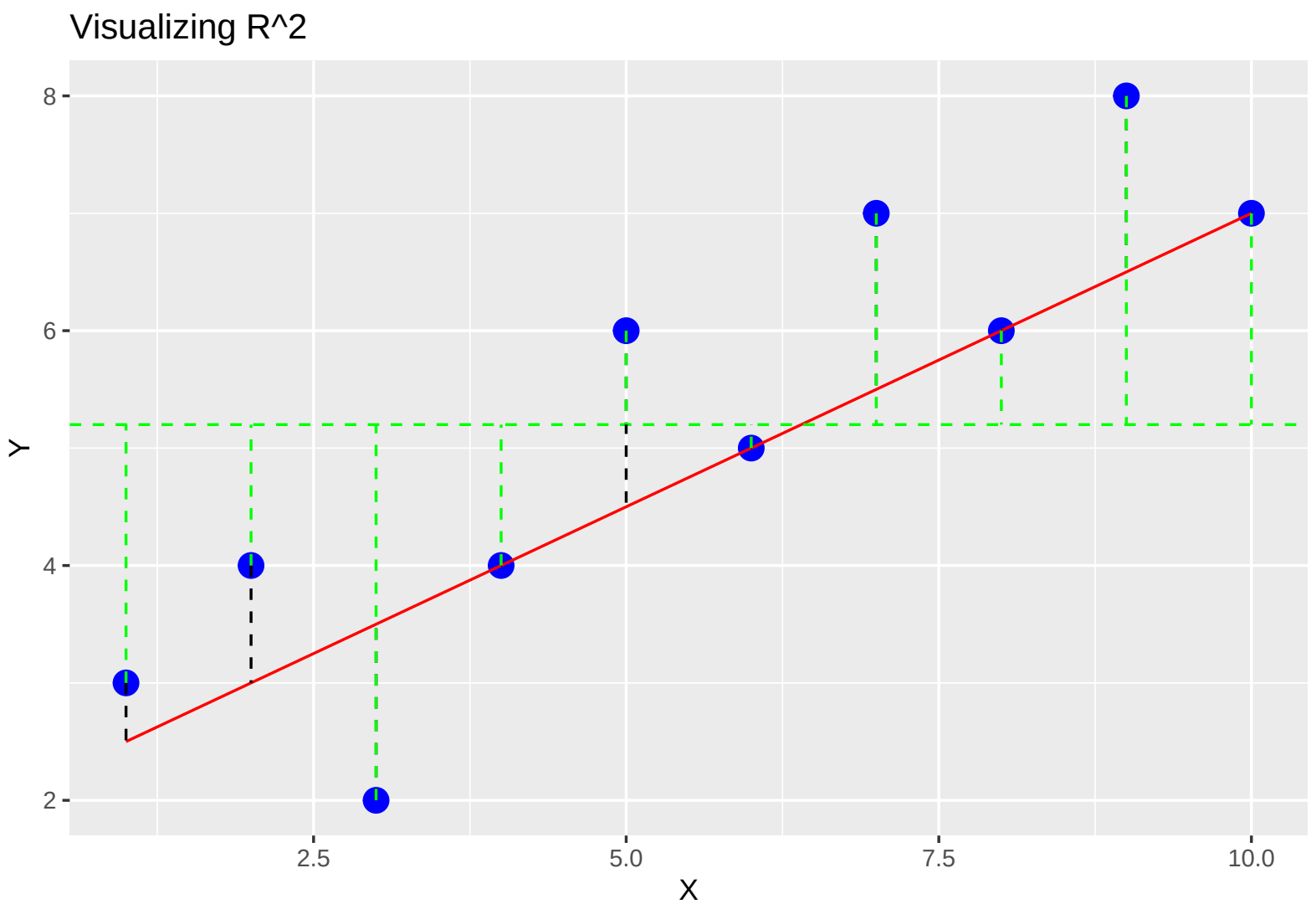
- Formula:

$$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$$

  where $\text{SS}_{\text{res}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  and $\text{SS}_{\text{tot}} = \sum_{i=1}^{n}(y_i - \bar{y})^2$

- Logic: Proportion of the variance in the dependent variable explained by the model.
- Importance for Tuning:
  - Higher R^2 (closer to 1) implies a better fit to the data.
  - Useful for assessing the explanatory power of the model.

Mean Squared Error (MSE)



Actual y, Red line: Predicted y_hat, Green line: Mean y_bar, Black Dashed lines: Error, Green Dashed lines: Total Variability

Linear Regression

- Simple linear regression focuses on modeling the relationship between a dependent variable and one or more independent variables.
- You can adjust features or regularization to improve the model.

Linear Regression: Basic Code Example

```
# Import LinearRegression
from sklearn.linear_model import
    LinearRegression
from sklearn.metrics import mean_squared_error,
    r2_score

# Initialize model
model = LinearRegression()

# Fit the model
model.fit(X, y)

# Evaluate model
y_pred = model.predict(X)
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
```

Regression Trees

- Regression Trees partition the feature space into regions and fit a simple model (like a constant) in each region.
- You can adjust the depth of the tree or other constraints to tune the model.

Regression Trees: Basic Code Example

```
# Import DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor

# Initialize model with max depth 2
tree_model = DecisionTreeRegressor(max_depth=2)

# Fit the model
tree_model.fit(X, y)

# Evaluate model
tree_y_pred = tree_model.predict(X)
tree_mse = mean_squared_error(y, tree_y_pred)
tree_r2 = r2_score(y, tree_y_pred)
```

MSE and R^2

SUPER IMPORTANT!!!!

- Models with a *higher* R^2 are better than a model with a lower R^2
- Models with a *lower* MSE are better than a model with a higher MSE.

MSE and R^2 problem

We have a big problem though with MSE and R^2. They will always favor a more complex model if you test them on the same data you estimated. Recall our models:

- Linear Regression models:
  - Model 1A: $y = \beta_0 + \beta_1 x_1$
  - Model 1B: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$
  - Model 1C: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$
- Regression trees
  - Model 2A: Tree with Depth 2
  - Model 2B: Tree with Depth 3
  - Model 2C: Tree with Depth 4

MSE for Model 1C is *always* lower than for 1B which is *always* lower than for 1A. Similarly MSE for 2C is lower than 2B, which is lower than 2A.

Train-Test Split

Solution: Don't use the data for MSE that was used to fit the model. Here's the strategy:

- Split the data into two groups. The first group is called the **training** data set and the second group is called the **test** data set.
- Fit the model to the training data set
- Make predictions on the test data set
- Calculate metrics on the predictions for the test data

Importance of Out-of-Sample Metrics

- Out-of-sample evaluation helps to detect overfitting.
- A model that performs well on in-sample but poorly on out-of-sample data is likely overfitting.
- Use train-test split for a more robust evaluation.

Train-Test Split: Code Example

```
from sklearn.model_selection import
    train_test_split

# Split the data into training and test sets
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.2)

# Fit the model on the training set
model.fit(X_train, y_train)

# Evaluate the model on the test set
y_pred_test = model.predict(X_test)
mse_test = mean_squared_error(y_test,
    y_pred_test)
r2_test = r2_score(y_test, y_pred_test)
```

Example

Suppose we have calculated the in and out of sample metrics for our 6 models:

Model	Train MSE	Test MSE	Train R^2	Test R^2
1A	40	50	0.60	0.55
1B	30	45	0.70	0.60
1C	20	55	0.85	0.50
2A	35	60	0.65	0.40
2B	25	35	0.80	0.70
2C	15	70	0.90	0.30

Detecting Underfitting Using Metrics

Principle:

- Underfitting occurs when a model performs poorly on both in-sample (training) and out-of-sample (testing) data.
- High MSE and low R^2 values are indicators.

Example:

- Models 1A and 2A show signs of underfitting.
- Both have higher in-sample and out-of-sample MSEs and lower R^2 values.

Detecting Overfitting Using Metrics

Principle:

- Overfitting occurs when a model performs well on in-sample data but poorly on out-of-sample data.
- Low in-sample MSE but high out-of-sample MSE can be a clue.

Example:

- Models 1C and 2C are likely overfitting.
- They perform well in-sample but poorly out-of-sample.

Model Selection Using Metrics

Principle:

- The best model usually performs well on both in-sample and out-of-sample data.
- A balance between MSE and R^2 is often desirable.

Example:

- Model 2B appears to be the best for out-of-sample performance.
- It has the lowest out-of-sample MSE and a reasonably high R^2 value.

Wrapping it up

- There are decisions that need to be made outside of algorithms like MLE or target function minimization for estimation.
- You will often propose many models, some of which may be in the same family of models, but you will tune the hyperparameters differently
- Evaluating model performance on out of sample data using metrics like MSE and R^2 is the most accurate way to do this
- The most complicated model is not always the best model.