- The more closely the model outputs a $Y$ that reflects this $X_{test}$ data, the more accurate the model is.
- Split: Time oriented

$\frac{1}{N} \sum_{i=1}^{N} |target_i - prediction_i|$

- Example:
  - Max depth 2 w/ 2 features
    - 181.28 bikes per prediction
      - Yikes!
  - Max depth 100 2/2 features
    - 0.0
      - YAY! (no error!) seems perfect!

- Model fits the data!
  - But it is data you already have.
- Idea: Test model on data that you have not seen yet?
Cross Validation

- Split data into two classes, – A training set and – testing set
- Calculate the difference
- Do it consecutive times
- Average their results (difference)

Example 10 Splits – with bikes.

What about KNN?
Fitting Data: What happens when K Varies?

Q1: Match to picture below:
- K = 2
- K = 4
- K = 10

What about KNN?
Fitting Data: What happens when K Varies?

Q1: Match to picture below:
- K = 2
- K = 4
- K = 10

Q2: As we increase k we are more likely to overfit?
True
False
What about **KNN**?

**Fitting Data: What happens when K Varies?**

Q1: Match to picture below:
- K = 1  **C**
- K = 3  **B**
- K = 10  **A**

Q2: As we increase k we are more likely to overfit?

True

False as we increase K less likely to overfit.

What about a **parametric model**

**Polynomial model of degree D**

**Fitting Data: What happens as D varies?**

- D = 1  \( y = m_1 x + b \)
- D = 4  \( y = m_4 x^4 + m_3 x^3 + m_2 x^2 + m_1 x + b \)
- D = 15

What about **a parametric model**

**Polynomial model of degree D**

**Fitting Data: What happens as D varies?**

- As we increase D are we more likely to overfit?
  - True
  - False
What about a **parametric model**

**Polynomial model of degree D**

**Fitting Data:** What happens as D varies?

- As we increase D are we more likely to overfit?
  - **True ✔** *(as we increase d we get closer to tagging actual data)*
  - **False**

- As the order of polynomial increases we start to match all the data points.
- Observation: as we increase d we are extrapolating in the direction of the data
  - This is not the case with KNN

---

**Revisit:** Comparing \( Y_{\text{test}} \) and \( Y_{\text{predict}} \)

- **Root Mean Square Error (RMSE)**
  - Squared before averaged
  - High weight to large errors
  - **Traditional**
- **Mean Absolute Error (MAE)**
  - Fixes problem with large residuals
  - All residuals have equal weight

- Both: Lower values are better
- Both: \( 0 \rightarrow \infty \)
- RMSE increases more than MAE as the sample size increases.
- MAE may be better in describing the average error across different error sample sizes.
- Still better to report both.

---

**Recall? Which is Larger?**

- Error in sample
- Error out of sample
Cross Validation

- Error in sample
- Error out of sample ✔

Recall we sliced the data and ran trials
- Training Set 60%
- Testing  Set 40 %
- May not be enough...

Slice it up differently, 1 Trial -> 5 Trials
- Training Set 80%
- Testing  Set 20%

Financial Data
- Cannot peak into the future

Implication:
- Can only ‘train’ before testing our data
- Roll forward.

Correlation

Divide Data:
- \( X_{\text{train}} \), \( Y_{\text{train}} \)
- \( X_{\text{test}} \), \( Y_{\text{test}} \) (true data)
- \( Y_{\text{predict}} \) (model prediction)

Correlation:
- -1, +1 strongly correlated to the data.
- Slope is not the correlation, correlation is fit.

Slope is not the correlation, correlation is fit.
What about Correlation vs. RMS Error?

- As RMS error increases
  A. Correlation decreases
  B. Correlation increases
  C. We can’t be sure either way

D. A & C

Over-fitting: Polynomials as \( D \) increases

- Polynomials
- In-Sample Errors
  - Decreases as \( D \) increases
- Out of sample Errors
  - Greater than In-Sample Errors
  - Error also decreases with \( D \)
  - Eventually increases opposite direction of in sample errors
There are a few other factors worth considering when evaluating a learning algorithm, and I've tallied a few of them here. I want you to think about each one of these and select which you think has better performance in that regard, linear regression or KNN?

Let's step through them.

- **Space for saving model**: KNN requires storing all the data, whereas linear regression requires storing only coefficients, which could be four numbers in the case of a linear regression with two features.
- **Compute time to train**: KNN requires comparing the test instance with all the training instances in the dataset, whereas linear regression can be computed much faster.
- **Compute time to query**: KNN requires computing distances for each training instance, whereas linear regression can be computed very quickly.
- **Ease to add new data**: KNN requires recalculating distances for new instances, whereas linear regression can be updated incrementally.

So, in terms of space for saving the model, linear regression is a hands-down winner. For compute time to train and query, KNN is bad in this regard. So, KNN is bad in terms of space for saving model and compute time to train and query.

Which is better?
Ensemble Learners

- Make learners better by assembling different learners
  - Lower Error
  - Less Overfitting
  - Avoid Biases of one model
    - Example: Linear regression bias on linear data.

Building Ensembles

- Train several parameterized polynomials of different degrees
- Train several KNN models using different subsets of data
- Combine the above into a super ensembles.

---

**Building Ensembles**

**Bootstrap aggregating** (Bagging)

- Combine the process of Training Data and Testing Data
  1. Create $K$ – bootstrap samples (with replacement)

<table>
<thead>
<tr>
<th>original</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap1</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>2</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>bootstrap2</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

2. Train on each sample
3. Vote (or average) the predictions of the K models

---

Breiman 1994
https://en.wikipedia.org/wiki/Bootstrap_aggregating

---

**Most Likely to Overfit**

A. Single 1NN Model Training on all the data
B. Ensemble of 10 1NN learners trained on 60% each?
Example

- Aggregating smoothes out the outcome.

Adaptive Boosting: Ada Boos

- Idea: Improve learners by focusing on data (areas) where models is not performing well.
- After building a ‘bag’, test it and spot data that has not been modeled well – give those data points more weight.
• Build by Breiman in late