Indian Institute of Information Technology, Allahabad

Introduction to Machine Learning

Training and Test Dataset Split
Cross Validation

By

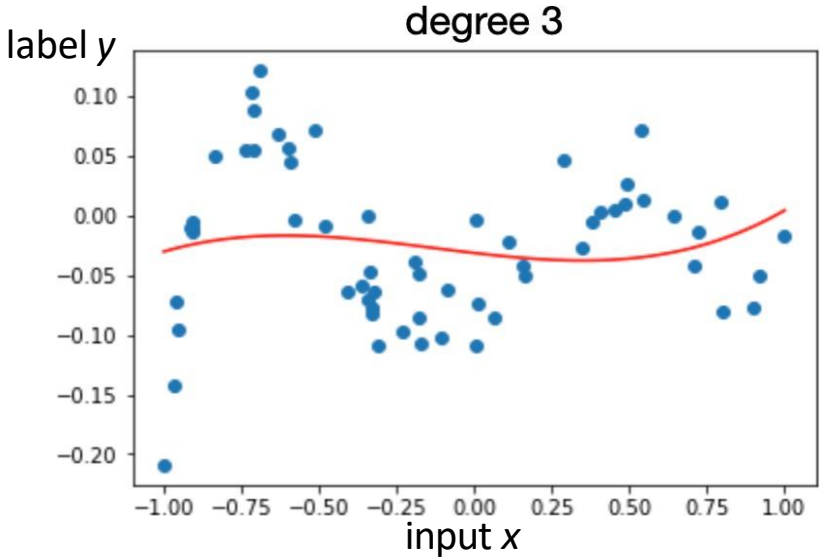
Dr. Shiv Ram Dubey
Associate Professor
Computer Vision and Biometrics Lab
Department of Information Technology
Indian Institute of Information Technology, Allahabad
Web: https://profile.iiita.ac.in/srdubey/

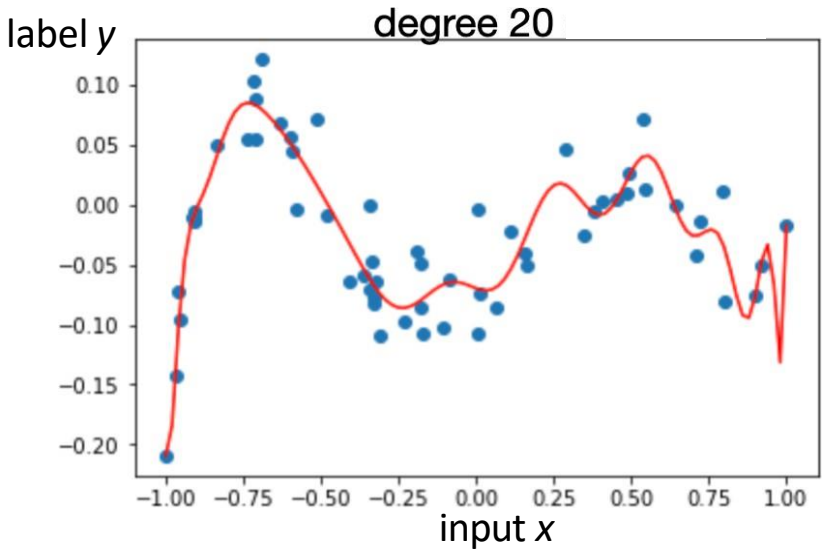
DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

Which p should we choose?

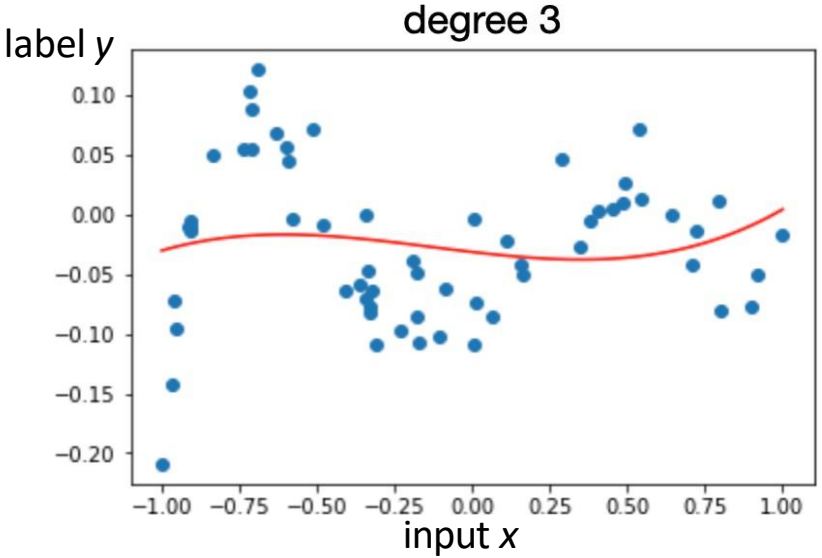
- First instance of class of models with different representation power = model complexity



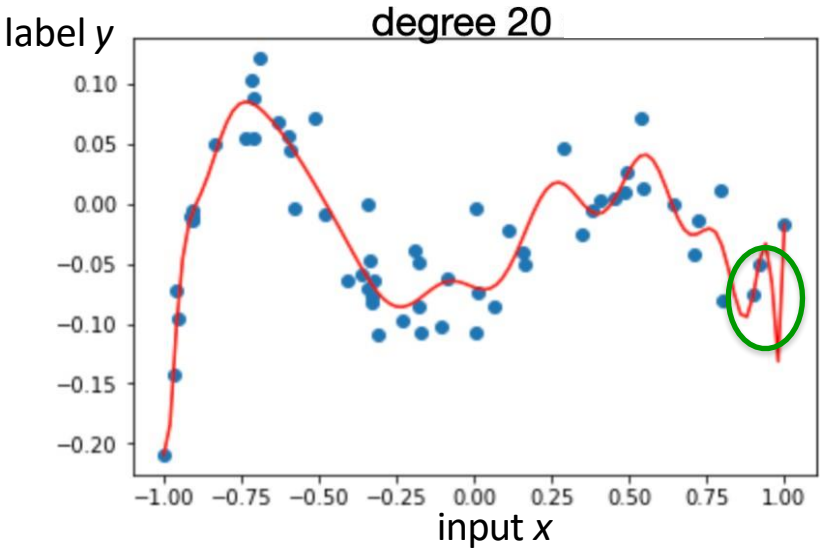


Which p should we choose?

- First instance of class of models with different representation power = model complexity



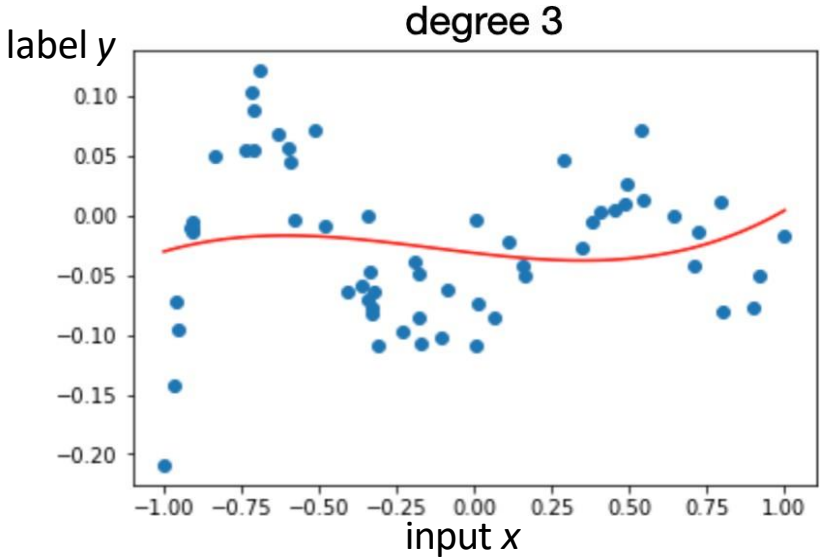
Underfitting: not fitting



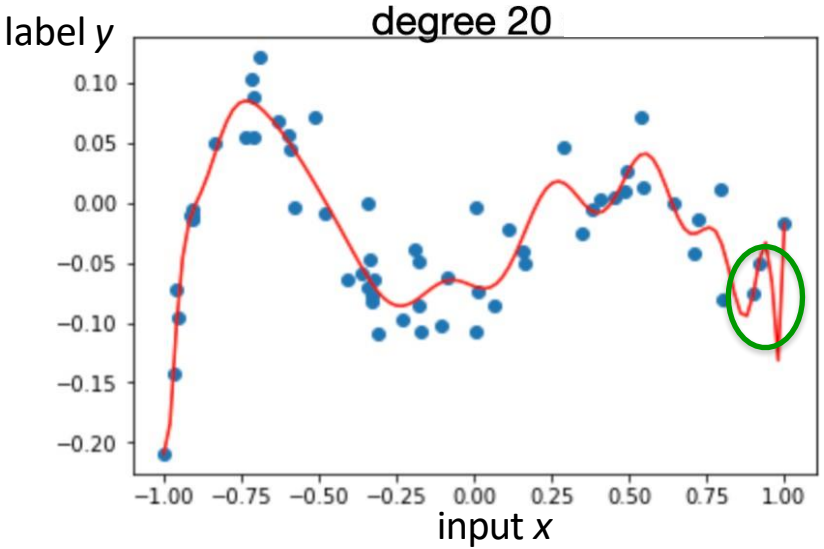
Overfitting: fitting "peculiarity" of training data that won't generalize

Which p should we choose?

- First instance of class of models with different representation power = model complexity



Underfitting: not fitting



Overfitting: fitting "peculiarity" of training data that won't generalize

- How do we determine which is better model?

Generalization

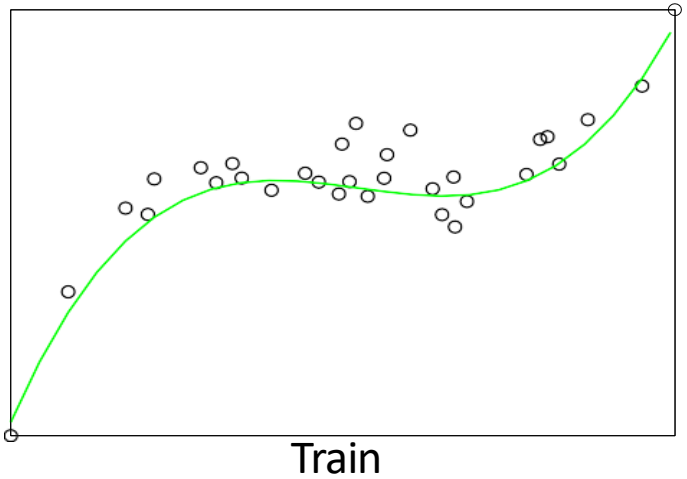
Test data

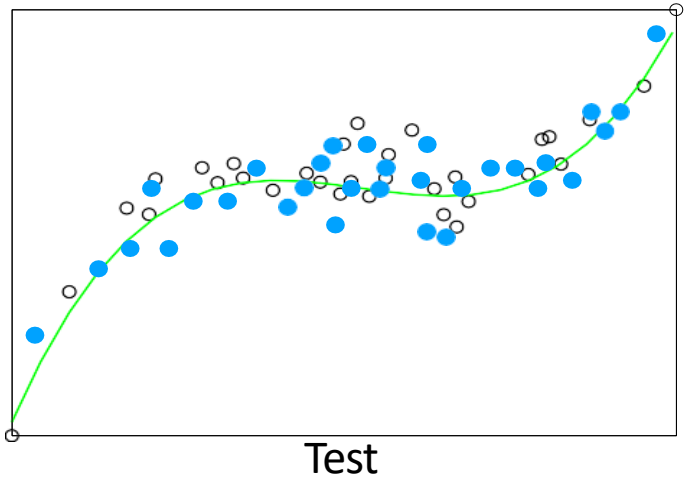
- We say a predictor **generalizes** if it performs as well on unseen data as on training data
- The data used to train a predictor is **training data** or **in-sample data**
- We want the predictor to work on **out-of-sample data**
- We say a predictor **fails to generalize** if it performs well on in-sample data but does not perform well on out-of-sample data

Generalization

Consider the following:

- **train** a cubic predictor on 32 (**in-sample**) white circles: Mean Squared Error (MSE) 174
- **predict** label y for 30 (**out-of-sample**) blue circles: MSE 192
- **Would you conclude this predictor generalizes effectively?**



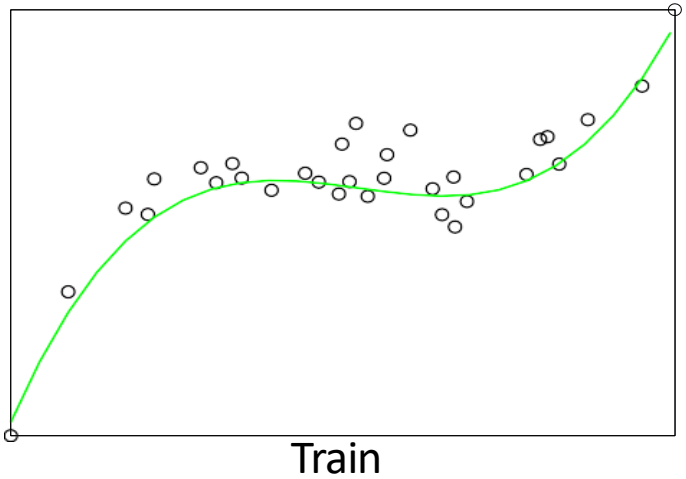


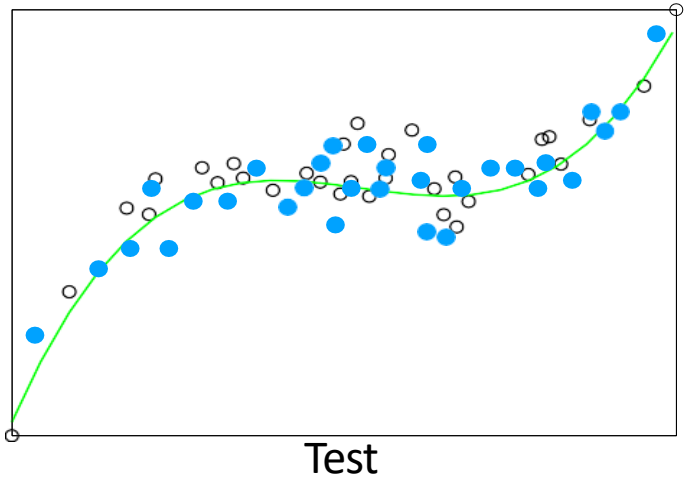
Generalization

Consider the following:

- **train** a cubic predictor on 32 (**in-sample**) white circles: Mean Squared Error (MSE) 174
- **predict** label y for 30 (**out-of-sample**) blue circles: MSE 192
- **Would you conclude this predictor generalizes effectively?**

Yes, as in-sample MSE ≅ out-of-sample MSE



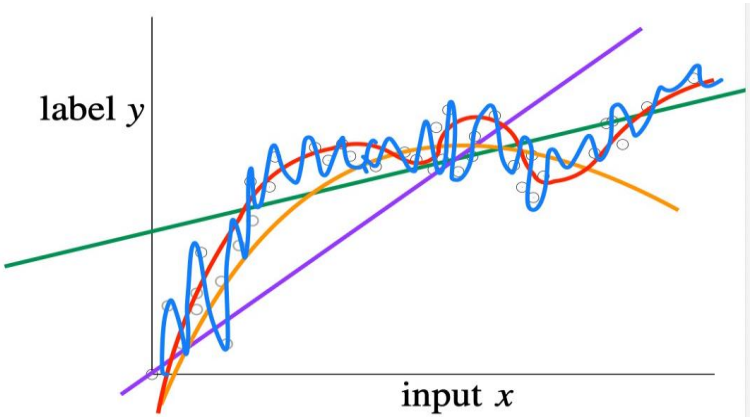


Generalization

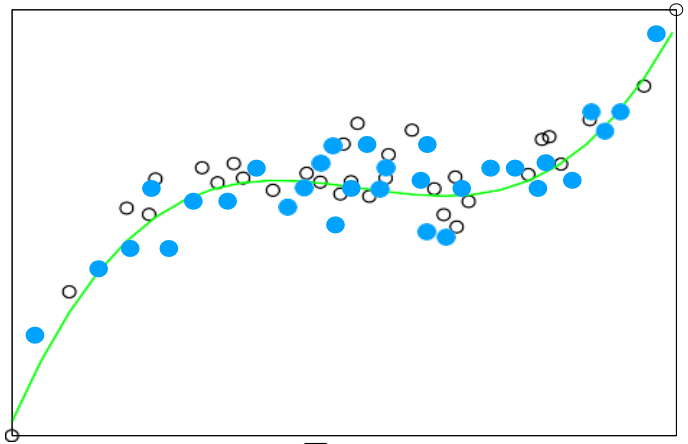
Consider the following:

- **train** a cubic predictor on 32 (**in-sample**) white circles: Mean Squared Error (MSE) 2
- **predict** label y for 30 (**out-of-sample**) blue circles: MSE 356
- **Would you conclude this predictor generalizes effectively?**

No, as in-sample MSE $\ll$ out-of-sample MSE



Train



Test

Split the Data into Training and Testing

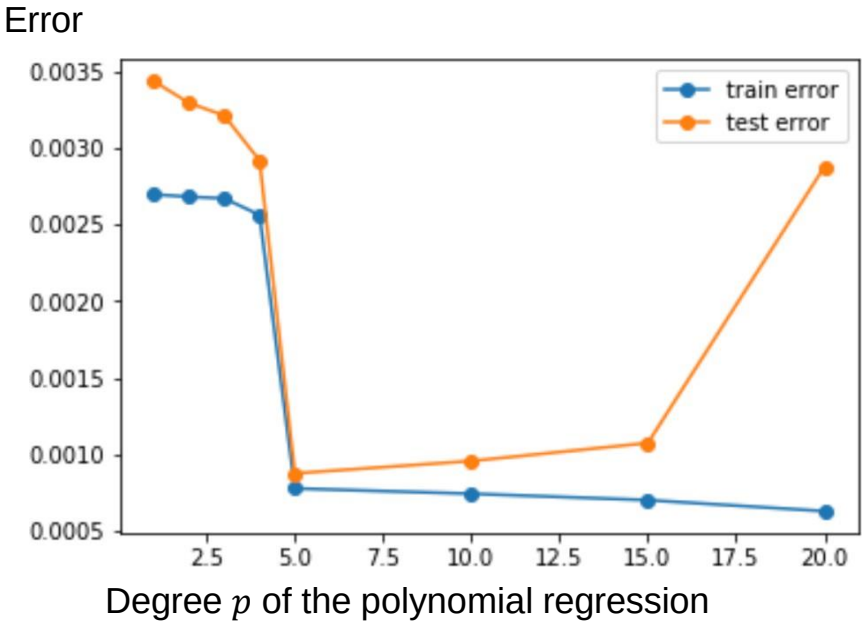
- a way to mimic how the predictor performs on unseen data
- given a single dataset $S = \{x_i, y_i\}_{i=1}^{n}$
- we split the dataset into two: training set and test set (e.g., 90/10)
- **training set** used to train the model
  - minimize $\mathscr{L}_{\text{train}}(w) = \frac{1}{|S_{\text{train}}|} \sum_{i \in S_{\text{train}}} (y_i - x_i^T w)^2$ # Fit the params of the model
- **test set** used to evaluate the model
  - $\mathscr{L}_{\text{test}}(w) = \frac{1}{|S_{\text{test}}|} \sum_{i \in S_{\text{test}}} (y_i - x_i^T w)^2$
- this assumes that test set is similar to unseen data

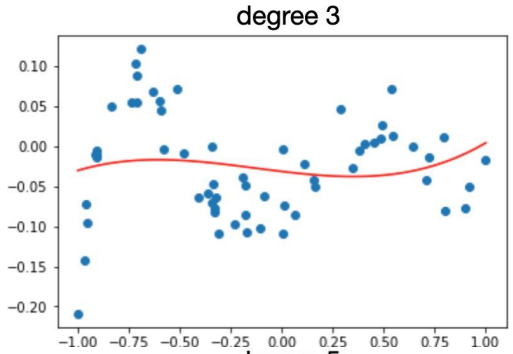
Split the Data into Training and Testing

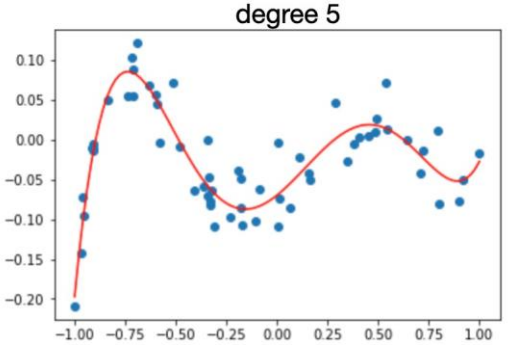
- a way to mimic how the predictor performs on unseen data
- given a single dataset $S = \{x_i, y_i\}_{i=1}^{n}$
- we split the dataset into two: training set and test set (e.g., 90/10)
- **training set** used to train the model
  - minimize $\mathscr{L}_{\text{train}}(w) = \frac{1}{|S_{\text{train}}|} \sum_{i \in S_{\text{train}}} (y_i - x_i^T w)^2$ # Fit the params of the model
- **test set** used to evaluate the model
  - $\mathscr{L}_{\text{test}}(w) = \frac{1}{|S_{\text{test}}|} \sum_{i \in S_{\text{test}}} (y_i - x_i^T w)^2$
- this assumes that test set is similar to unseen data
- **test set should never be used in training or picking unknowns!!**

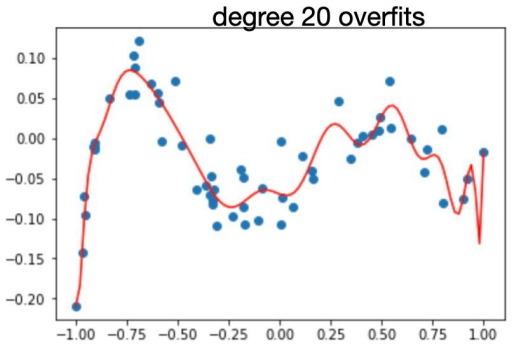
This is cheating and will lead you to the wrong conclusions.
Big statistical no no

Train/test Error vs. Complexity









- Degree $p = 5$, since it achieves minimum **test error**
- **Train error** monotonically decreases with model complexity
- **Test error** has a U shape. There is a best value

test set should never be used in training or picking degree

Picking Basis Function

- **How do we pick the number of basis functions…**

- **We could use the test data, but…**

Picking Basis Function

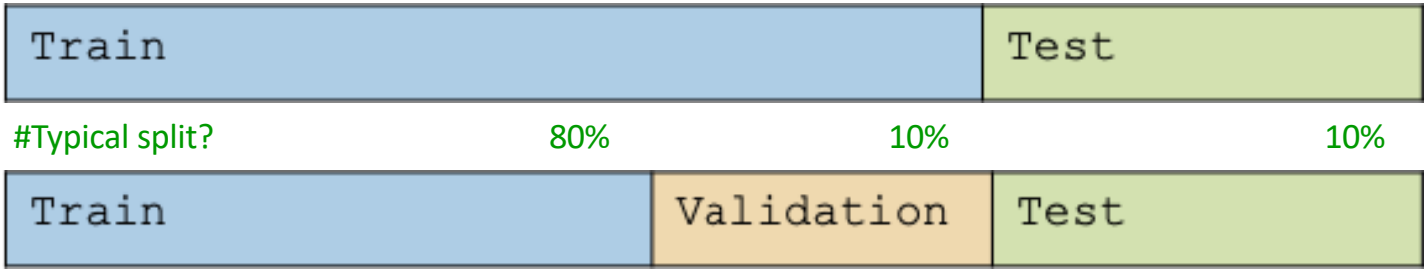
- How do we pick the number of basis functions...

- We could use the test data, but...

  - Never ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever ever
    train on the test data

Validation Set

- Simplest approach is to add another dataset partition called the **validation set**



- Each set has its own role # Fit model params like w
  - **Train:** Used to train a model of each model complexity
  - **Validation:** Used as a hold-out to evaluate each model. Generally choose model complexity with lowest validation error.

    # Choose *hyper*parameters, like polynomial degree p
  - **Test:** Once the model is chosen, can get an estimate of future error by test. This would be what you report. **Only do this once!**

(LOO) Leave-one-out Cross Validation

- **Consider a validation set with 1 example:**
  - **D – training data** # n samples
  - **D\j – training data with j^{th} data point (x_j, y_j) moved to validation set**
- **Learn model $f_{D\setminus j}$ with $D\setminus j$ dataset** # n-1 samples
- **Estimate true error** as squared error on predicting y_j:
  - **Unbiased estimate of $error_{true}(f_{D\setminus j})$!**

(LOO) Leave-one-out Cross Validation

- **Consider a validation set with 1 example:**
  - **D – training data** # n samples
  - **D\j – training data with j^{th} data point (x_j, y_j) moved to validation set**
- **Learn model $f_{D\setminus j}$ with $D\setminus j$ dataset** # n-1 samples
- **Estimate true error** as squared error on predicting y_j:
  - **Unbiased estimate of $error_{true}(f_{D\setminus j})$!**

- **LOO cross validation**: Average over all data points *j*:
  - **For each data point you leave out, learn a new model $f_{D\setminus j}$**
  - **Estimate error** as:

$$\text{error}_{LOO} = \frac{1}{n}\sum_{j=1}^{n}(y_j - f_{\mathcal{D}\setminus j}(x_j))^2$$

LOO Cross Validation is (almost) Unbiased Estimate!

- **When computing LOOCV error, we only use *N-1* data points**
  - **So it's not estimate of true error of learning with *N* data points**
  - **Usually pessimistic, though – learning with less data typically gives worse answer**

- **LOO is almost unbiased**! Why not **use LOO error for model selection!**
  - **E.g., picking degree**

LOO Cross Validation is (almost) Unbiased Estimate!

- **When computing LOOCV error, we only use *N-1* data points**
  - **So it's not estimate of true error of learning with *N* data points**
  - **Usually pessimistic, though – learning with less data typically gives worse answer**

- **LOO is almost unbiased**! Why not **use LOO error for model selection!**
  - **E.g., picking degree**

You just fit n models!!

Computational Cost of LOO

- Suppose you have 100,000 data points
- You implemented a great version of your learning algorithm
  - Learns in only 1 second
- Computing LOO will take about 1 day!!!

Use *k*-fold Cross Validation

- Randomly divide training data into *k* equal parts: $D_1,\ldots,D_k$
- For each *i*
  - Learn model $f_{D\setminus Di}$ using data point not in D_i
  - Estimate error of $f_{D\setminus Di}$ on validation set D_i:

1	2	3	4	5
Train	Train	Validation	Train	Train

$$\text{error}_{\mathcal{D}_i} = \frac{1}{|\mathcal{D}_i|} \sum_{(x_j,y_j)\in\mathcal{D}_i} (y_j - f_{\mathcal{D}\setminus\mathcal{D}_i}(x_j))^2$$

- k-fold cross validation error is average over data splits:

$$error_{k-fold} = \frac{1}{k}\sum_{i=1}^{k} error_{\mathcal{D}_i}$$

Use *k*-fold Cross Validation

> Randomly divide training data into *k* equal parts: $D_1,\ldots,D_k$

> For each *i*

- Learn model $f_{D\setminus Di}$ using data point not in D_i
- Estimate error of $f_{D\setminus Di}$ on validation set D_i:

$$\text{error}_{\mathcal{D}_i} = \frac{1}{|\mathcal{D}_i|} \sum_{(x_j,y_j)\in\mathcal{D}_i} (y_j - f_{\mathcal{D}\setminus\mathcal{D}_i}(x_j))^2$$

> k-fold cross validation error is average over data splits:

$$error_{k-fold} = \frac{1}{k}\sum_{i=1}^{k} error_{\mathcal{D}_i}$$

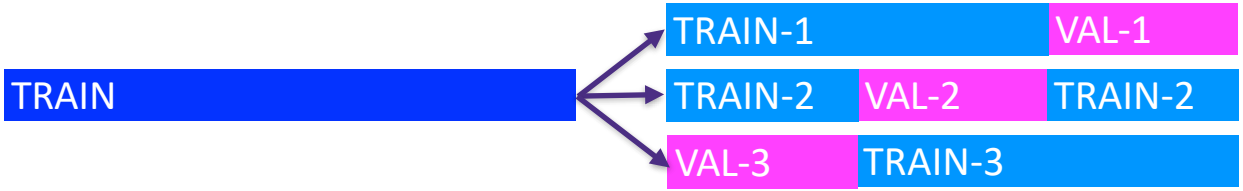
> k-fold cross validation properties:

- Much faster to compute than LOO
- More (pessimistically) biased – using much less data, only *n(k-1)/k*
- Usually, k = 10

Recap

- Given a dataset, begin by splitting into

- Model selection: Use k-fold cross-validation on TRAIN to train predictor and choose magic parameters such as degree



- Model assessment: Use TEST to assess the accuracy of the model you output
  - Never ever train or choose parameters based on the test data

Example-1

- You wish to predict the stock price of zoom.us given historical stock price data
- You use all daily stock price up to Jan 1, 2020 as TRAIN and Jan 2, 2020 - April 13, 2020 as TEST
- What's "wrong" with this procedure?

  - ML assumes data is i.i.d.; is this?
    - Nope! Future data is o.o.d. (out of distribution) from the training data

  - Why is "wrong" in square quotes?
    - You might still want to do this. It's economically valuable, if you could.

Example-2

- Given d=10,000-dimensional data and n=1000 examples, we pick a subset of 50 dimensions that have the highest correlation with labels in the entire dataset:

50 indices j that have largest $\frac{|\sum_{i=1}^{n} x_{i,j} y_i|}{\sqrt{\sum_{i=1}^{n} x_{i,j}^2}}$

- After picking our 50 features, we then break data into train and test dataset.
- We train linear regression on these selected features on the training set. We compute the test error and report it.
- What's wrong with this procedure?
  - Top 50 features might be redundant with each other.
    - e.g. sqft & square meters

Example-3 The “Clever Hans” Effect

- Computer vision researchers trained a classifier to distinguish huskies from wolves.
- It performed amazingly well on the test set.
- However, the model ignored the animals!
- It had learned that ***grass = husky (pet)*** and ***snow = wolf (wild)***.

Ribeiro, M. T., Singh, S., & Guestrin, C.. “Why should i trust you? Explaining the predictions of any classifier”. SIGKDD (2016).

Recap

- Machine learning pipeline
  - Collect some data
    - E.g., housing info and sale price
  - Randomly split dataset into TRAIN, VAL, and TEST
    - E.g., 80%, 10%, and 10%, respectively
  - Choose a hypothesis class or model
    - E.g., linear with non-linear transformations
  - Choose a loss function
    - E.g., least squares on TRAIN
  - Choose an optimization procedure
    - E.g., set derivative to zero to obtain estimator
  - Choose hyperparameters
    - cross-validation on VAL to pick num. features
  - Justify the accuracy of the estimate
    - E.g., report TEST error

Regression Evaluation Metrics

- For regression tasks, where we predict continuous values, evaluation metrics quantify how close our model's predictions ($\hat{y}$) are to the actual target values (y).
- The difference between an actual value and a predicted value ($y - \hat{y}$) is called the *residual* or *error* for that data point.
- Regression metrics are typically based on aggregating these residuals across the entire dataset (or a specific subset like a test set).

Regression Evaluation Metrics

- For regression tasks, where we predict continuous values, evaluation metrics quantify how close our model's predictions ($\hat{y}$) are to the actual target values (y).
- The difference between an actual value and a predicted value ($y - \hat{y}$) is called the *residual* or *error* for that data point.
- Regression metrics are typically based on aggregating these residuals across the entire dataset (or a specific subset like a test set).
- Common metrics include:
  - Mean Absolute Error (MAE)
  - Mean Squared Error (MSE)
  - Root Mean Squared Error (RMSE)
  - R-squared (R^2) Score (Coefficient of Determination)

Mean Absolute Error (MAE)

- The Mean Absolute Error (MAE) is one of the simplest and most intuitive metrics. It measures the average absolute difference between the predicted values and the actual values.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Interpretation:

- **Units:** MAE is in the same units as the target variable (e.g., if you're predicting house prices in dollars, the MAE will also be in dollars).
- **Scale:** MAE ranges from 0 to ∞. A value closer to 0 indicates better model performance, meaning the average prediction error is small.
- **Sensitivity to Outliers:** Because it uses the absolute difference, MAE is less sensitive to large errors (outliers) compared to metrics that square the errors. It treats all errors linearly based on their magnitude.

Mean Squared Error (MSE)

- The Mean Squared Error (MSE) is another widely used metric.
- Instead of taking the absolute difference, it calculates the average of the squared differences between predictions and actual values.

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

Interpretation:

- **Units:** MSE is in the *square* of the units of the target variable (e.g., dollars squared for house prices). So, hard to interpret directly in the context of the original problem.
- **Scale:** MSE ranges from 0 to ∞, with values closer to 0 being better.
- **Sensitivity to Outliers:** Squaring the errors gives much more weight to larger errors. A prediction that is off by 10 units contributes 100 to the sum, while one off by 2 units contributes only 4. So, MSE heavily penalizes models that make large mistakes.
- **Mathematical Properties:** The squared term makes MSE differentiable, which is advantageous for certain optimization techniques used in training some models.

Root Mean Squared Error (RMSE)

- The Root Mean Squared Error (RMSE) is simply the square root of the MSE.
- It addresses the primary interpretation issue of MSE by bringing the metric back into the original units of the target variable.

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

Interpretation:

- **Units:** RMSE is in the same units as the target variable, similar to MAE.
- **Scale:** RMSE ranges from 0 to ∞, with lower values indicating a better fit.
- **Sensitivity to Outliers:** Like MSE, RMSE penalizes large errors more heavily than MAE due to the squaring step before the square root. However, its interpretation is often easier than MSE because the units match the target.
- RMSE is arguably the most common regression metric. It provides a measure of the typical magnitude of the errors, expressed in the original units, while still being sensitive to larger deviations.

R-squared (R2) Score

- The R-squared (R^2) score, also known as the coefficient of determination, measures the proportion of the variance in the target variable that is explained by the model's features.

- It compares the model's performance (using the sum of squared errors, SSE, which is $n \times$ MSE) to that of a baseline model that simply predicts the mean of the target variable ($\bar{y}$) for all inputs (whose error is the total sum of squares, SST).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{\text{SSE}}{\text{SST}} \text{ where } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

R-squared (R2) Score

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} = 1 - \frac{\text{SSE}}{\text{SST}} \text{ where } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Interpretation:

- **Units:** R^2 is a unitless metric, representing a proportion.
- **Scale:**
  - An R^2 of 1 indicates that the model perfectly predicts the target variable (explains 100% of the variance).
  - An R^2 of 0 indicates that the model performs no better than the baseline model that always predicts the mean of the target values.
  - An R^2 score can be negative. This happens when the model performs *worse* than the baseline mean model, meaning the model's predictions are actively harmful compared to just guessing the average.
- **Context:** R^2 provides a relative measure of fit. A "good" R^2 value is context-dependent. In some fields (like physics), high R^2 values (e.g., > 0.95) are expected. In others (like social sciences), an R^2 of 0.3 might be informative. It doesn't tell you the average error magnitude, only how much better your model is than a naive mean prediction, relative to the total variance.

Choosing the Right Metric

The choice of evaluation metric depends on your specific goals and the characteristics of your data:

- Use **MAE** if you want an easily interpretable metric in the original units and if outliers are not a major concern or should not be disproportionately penalized.
- Use **RMSE** if you want a metric in the original units but need to penalize larger errors more significantly. It's often a good default choice.
- Use **MSE** primarily when its mathematical properties are advantageous (less common for final evaluation due to unit interpretation difficulty).
- Use **R^2** when you need to understand the proportion of variance explained by the model, providing a relative measure of goodness-of-fit. It's useful for comparing models' explanatory power but doesn't directly indicate the typical prediction error size.

Often, it's beneficial to look at multiple metrics to get a more complete picture of your model's performance. For instance, a high R^2 indicates your model captures the trend well, while a low MAE or RMSE confirms the typical prediction error is small.

Acknowledgement

Most of the slides are based on the ML course of:

- Natasha Jaques, University of Washington

THANK YOU