

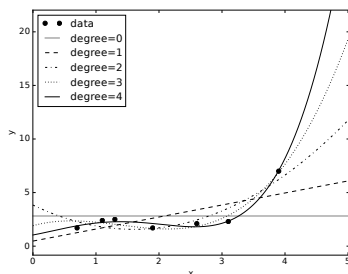
- **Overfitting** occurs when the learner finds regularities in the training set that do not occur in the world (or in the test set).
- Model tries to find signal in randomness.
- Often results in
  - ▶ more **complex models** than can be justified by the limited data.
  - ▶ **overconfidence**: more extreme probabilities than is justified.
- Fitting the training set better does not mean fitting the test set or better predictions of future cases

# Example of Overfitting: Restaurant Ratings

- We have a web site where people rate restaurants with 1 to 5 stars.
- We want to report the most liked restaurant(s) — the one predicted to have the best future ratings.
- How can we determine the most liked restaurant?
- Are the restaurants with the highest average rating the most liked restaurants?
- Which restaurants have the highest average rating?
- Which restaurants have a rating of 5?
  - ▶ Only restaurants with few ratings have an average rating of 5.
  - ▶ Restaurants with few ratings but all high are unlikely to be as good as the ratings indicate.
- Ratings are a mix of quality and luck. Lots of data averages out luck.

# Example of Overfitting: Model complexity

Polynomials of various degrees can be used to fit data:



- Can use standard linear regression with  $1, x, x^2, x^3 \dots$  as features. The maximum power with non-zero coefficient is the **degree** of the polynomial.
- Higher-degree polynomials can always(?) fit data better.
- What happens with extrapolation?  
How do polynomials of odd and even degrees differ?

Test set error is caused by:

- **Bias:** error due to the algorithm finding an imperfect model. The bias can be divided into:
  - ▶ **representation bias:** the representation doesn't contain a hypothesis close to the ground truth
  - ▶ **search bias:** the algorithm doesn't search enough of the space of hypotheses to find the best hypothesis.
- **Variance:** error due to a lack of data. A fixed amount of data has a **bias–variance trade-off**:
  - ▶ a complicated model could be accurate, but there is not enough data to estimate it accurately
  - ▶ a simple model may not be accurate, but you can estimate the parameters reasonably well given the data
- **Noise:** inherent error due to the data depending on features not modeled or the process generating the data is inherently stochastic.

- For many of the prediction measures, the optimal prediction on the training data is the mean.
- In the restaurant example, the mean rating wasn't a good measure (too extreme for restaurants with few ratings).
- A simple solution is to start with some pseudo-examples:
  - ▶ initially a restaurant is assumed to be average
  - ▶ As data comes in, the observed ratings are added to the pseudo-examples
  - ▶ Don't need to store pseudo-examples, just the sufficient statistics: pseudocounts.

# Pseudocounts

- Suppose the examples are the values  $v_1, \dots, v_n$
- You want to make a prediction for the next  $v$ , written as  $\hat{v}$ .
- When  $n = 0$  – there is no data – use prediction  $a_0$
- For the other cases, use

$$\hat{v} = \frac{c * a_0 + \sum_i v_i}{c + n}$$

where  $c$  is a nonnegative real-value constant.

- The value of  $c$  controls the relative importance of the initial hypothesis (the **prior**) and the data.
- $a_0$  and  $c$  can be estimated from other data (e.g., other restaurants)

A theoretical justification of pseudocounts is given in Chapter 10.

# Regularization

*“What can be done with fewer [assumptions] is done in vain with more.”*

*William of Ockham (1285–1349)*

- Prefer simpler hypotheses over more complex ones.
- **Regularization**: optimize fit-to-data plus a term that penalizes complexity
- Find a predictor  $\hat{Y}$  to minimize

$$\left( \sum_e \text{loss}(\hat{Y}(e), Y(e)) \right) + \lambda * \text{regularizer}(\hat{Y})$$

- ▶  $\text{loss}(\hat{Y}(e), Y(e))$  is the loss of example  $e$  for predictor  $\hat{Y}$
- ▶  $\text{regularizer}(\hat{Y})$  is a penalty term that penalizes complexity.
- ▶ The **regularization parameter**,  $\lambda$ , trades off fit-to-data and model simplicity

# Regularization

- In **decision tree learning**, one complexity measure is the number of leaves in a decision tree.
- When building a decision tree, you could optimize the sum of a loss plus a function of the size of the decision tree, minimizing

$$\left( \sum_{e \in Es} \text{loss}(\hat{Y}(e), Y(e)) \right) + \gamma * |tree|$$

where  $|tree|$  is the number of leaves in a tree representation of  $\hat{Y}$ .

- A single split on a leaf increases the number of leaves by 1.
- When greedily splitting, a single split is worthwhile if it reduces the sum of losses by  $\gamma$ .



# L1 and L2 Regularization

Linear/logistic regression, minimize sum-of-squares:

$$\text{minimize } Error_E(\bar{w}) = \sum_{e \in E} \left( Y(e) - f\left(\sum_i w_i X_i(e)\right) \right)^2.$$

L2 regularization (ridge regression):

$$\text{minimize } \sum_{e \in E} \left( Y(e) - f\left(\sum_i w_i X_i(e)\right) \right)^2 + \lambda \sum_i w_i^2$$

L1 regularization (lasso):

$$\text{minimize } \sum_{e \in E} \left( Y(e) - f\left(\sum_i w_i X_i(e)\right) \right)^2 + \lambda \sum_i |w_i|$$

$\lambda$  is a parameter given a priori and/or learned.

# SGD with L1 and L2 Regularization

- An  $L2$  regularization is implemented in stochastic gradient descent by updating each weight  $w_i$  after a batch by:

$$w_i := w_i - \eta * \lambda * b/|Es| * w_i$$

where  $b$  is batch size.

The  $m/|Es|$  is because the regularization is  $\lambda$  for the whole dataset, but the update occurs for each batch.

- An  $L1$  regularizer can be implemented by updating each weight after a batch by:

$$w_i := \text{sign}(w_i) * \max(0, |w_i| - \eta * \lambda * m/|Es|).$$

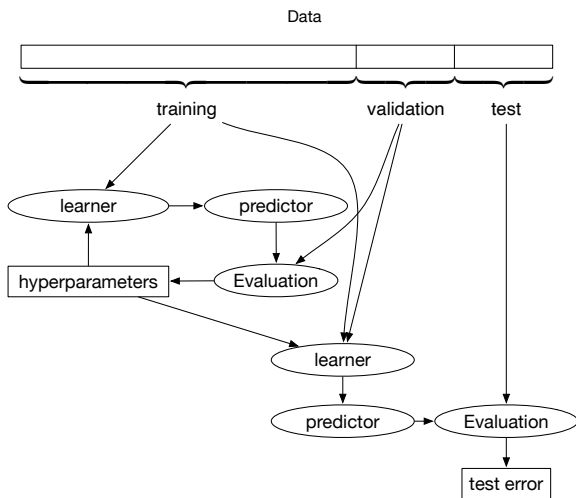
This is called **iterative soft-thresholding**

# Cross Validation

- To evaluate an algorithm some of the data is used as *test data* (random data or latest in time). The test set *must not* be used for any part of training or choosing parameters.
- Idea: split the remaining data into:
  - ▶ training set
  - ▶ validation set
- A **hyperparameter** is a parameter used to define what is being optimized, or how it is optimized.
- Use the new training set to train on. Select the hyperparameters that work best on the validation set.

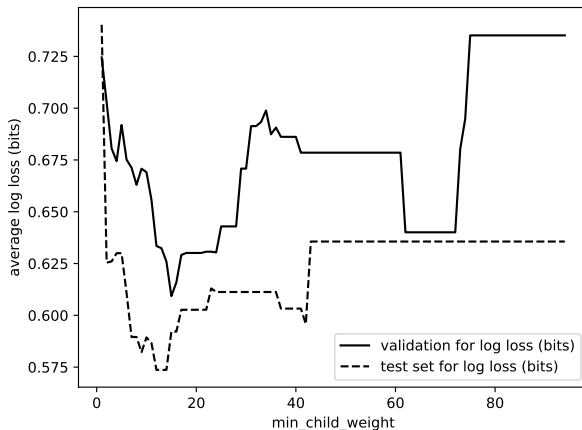
# Cross Validation

Cross validation: use some of the non test set as a surrogate for test data:



# Cross Validation

Cross validation assumptions: hyperparameter values that are best for validation examples will be best for test examples.



minimum number of examples that needs to be in a child for decision-tree learning.

# k-Fold Cross Validation

- With limited data, either training data or validation data will be small (or both).
- How can we avoid overfitting to these small datasets?
- k-Fold Cross Validation:
  - ▶ partition non-test data  $E_s$  into  $k$  folds,  $E_1, \dots, E_k$  ( $k = 10$  is common for 10-fold cross validation)
  - ▶ For  $i$  from 1 to  $k$ :
    - train on  $E_s \setminus E_i$  evaluate on  $E_i$
  - ▶ Select the hyperparameter settings with lowest average error on  $E_1, \dots, E_k$
  - ▶ Train a model on  $E_s$  with these settings
- If  $k = 10$ , during hyperparameter tuning, 90% of the training examples are used for training and 10% of the examples for validation.  
It does this 10 times, so each example is used once in a validation set.