CSC 480/580 Principles of Machine Learning

02 Limits of Learning

Jason Pacheco

*some slides are from Daniel Hsu and Francesco Orabona with their permission

HW1 released (due next Thursday)

Instructions:

- Submit your homework on time to gradescope (there will be separate entries for 480/580). NO LATE DAYS, NO LATE SUBMISSIONS ACCEPTED.
- The submission must be one single PDF file (see the instructions from HW0 for details)
- You must copy-paste your code as part of the answer. Make sure it is formatted correctly (like indents). If I cannot read your code from the pdf, points will be deducted.
- Code must also be submitted to a separate gradescope entry dedicated for code.
- If you cannot answer a problem, describing what efforts you have put in to solve the problem and where you get stuck will receive partial credit. Also, feel free to post your questions on Piazza.
- Collaboration policy: do not discuss answers with your classmates. You can discuss HW for the clarification or any math/programming issues at a high-level. If you do get help from someone, please make sure you write their names down in your answer.
- If you cannot answer a problem, describing what efforts you have put in to solve the problem and where you get stuck will receive partial credit. Also, feel free to post your questions on Piazza.
- Each subproblem is worth 10 points unless noted otherwise.

So far..

- Decision trees serve as an example of ML method (supervised learning in particular)
- Machine learning is a general & useful framework…**but it's not "magic"**
- Understand when machine learning will and will not work

We will now get into the deeper into the foundations of supervised learning.

Optimal classification with known D

Suppose

- Binary classification: 0-1 loss $\ell(y, \hat{y}) = I(y \neq \hat{y})$
- Data Generating distribution D known for every (x, y)

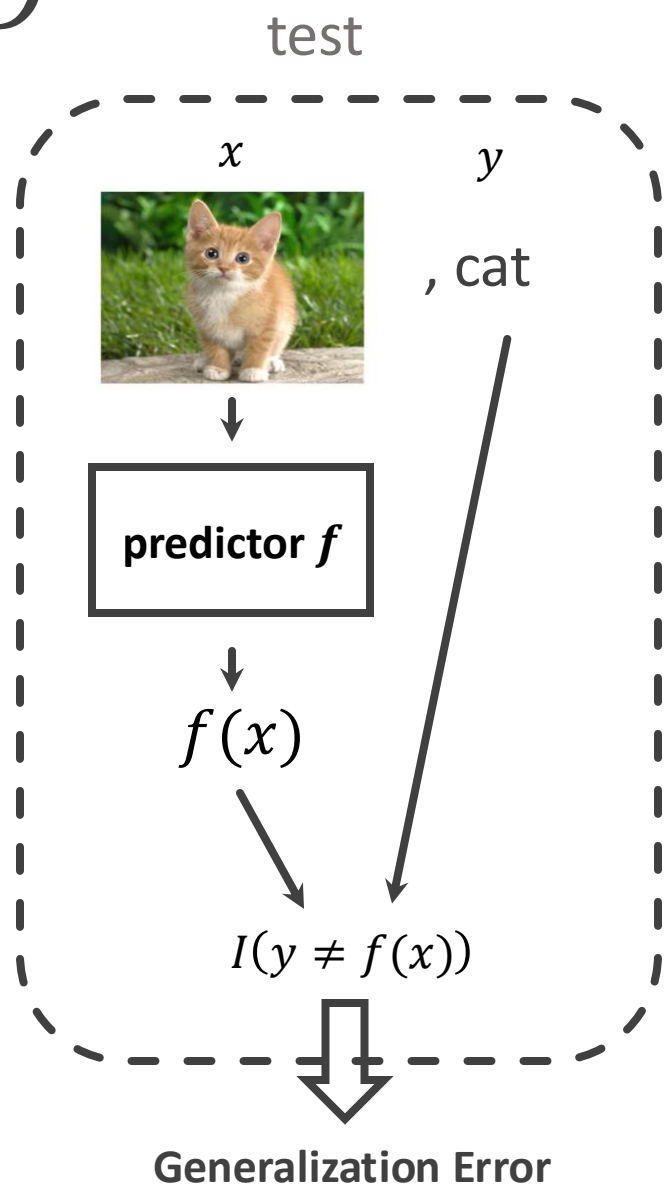
Generalization Error

$$err_{\mathcal{D}}(f) = \mathrm{E}_{(x,y)\sim D} I(y \neq f(x)) = P_{(x,y)\sim D}\,(y \neq f(x))$$

Question

What is the f that minimizes,

$$err_{\mathcal{D}}(f) = P_{(x,y)\sim D}(y \neq f(x))$$



Simple case: discrete domain $\mathcal{X}$

$P_D(x,y)$	$x=1$	$x=2$	$x=3$
$y=-1$	0.2	0.2	0.15
$y=+1$	0.1	0.3	0.05

Which classifier is better?

- $f_1(1)=-1, f_1(2)=-1, f_1(3)=-1 \quad \Rightarrow err_{\mathcal{D}}(f)(f_1)=0.1+0.3+0.05$
- $f_2(1)=-1, f_2(2)=+1, f_2(3)=-1 \quad \Rightarrow err_{\mathcal{D}}(f)(f_2)=0.1+0.2+0.05$

Is this the best classifier? Why?

- For any x, should choose y that has higher value of $P_D(x,y)$
- $f^*(1)=-1, f^*(2)=+1, f^*(3)=-1$

The Bayes classifier: The optimal classifier that we don't know.

- $f_{BO}(x) = \arg\max_{y \in \mathcal{Y}} \mathbb{P}_{\mathcal{D}}(Y = y \,|X = x)\,, \forall x \in \mathcal{X}$

we can never know what it is, but we can talk about it and reason about it.

- $err_{\mathcal{D}}(f_{BO})$ is called the "**Bayes error rate**"

- (Theorem) f_{BO} achieves the smallest error rate among all functions.

Proof

It suffices to show: $\forall g,\ err_{\mathcal{D}}(g) \geq err_{\mathcal{D}}(f_{BO})$

Fix a function g.

Recall: $f_{BO}(x) = \arg\max_{y \in \mathcal{Y}} \mathbb{P}_{\mathcal{D}}(Y = y \mid X = x), \forall x \in \mathcal{X}$

(1) State what we know

$$P(Y = f_{BO}(x) \mid X = x) \geq P(Y = y \mid X = x), \forall y \in \mathcal{Y}$$

$$\Rightarrow P(Y = f_{BO}(x) \mid X = x) \geq P(Y = g(x) \mid X = x)$$

(2) Start from $err_{\mathcal{D}}(g)$

$$\mathbb{E}_{\{A,B\}}[f(A,B)] = \mathbb{E}_B\left[\mathbb{E}_{A|B}[f(A,B)|B]\right]$$

$$err_{\mathcal{D}}(g) = \mathbb{E}_{X,Y}[I\{Y \neq g(X)\}] = \mathbb{E}_X\left[\mathbb{E}[I\{Y \neq g(X)\} \mid X]\right] \quad \text{// tower property}$$

$$= \mathbb{E}_X[\ P(Y \neq g(X) \mid X)\ ] \quad \text{// } \mathbb{E}_A[I\{F\}] = P(F)$$

$$= \mathbb{E}_X[1 - P(Y = g(x) \mid X)]$$

$$\geq \mathbb{E}_X[1 - P(Y = f_{BO}(x) \mid X)] \quad \text{// what we knew}$$

$$= err(f_{BO})$$

The cause of the limit

- Missing important features
  - Product search (e.g., amazon.com): The user poorly described what they want
    Or, some important user info is not available due to privacy
- Feature noise
  - e.g., images too dark to distinguish anything
- Label noise
  - The correct label is arguable.

cat vs dog

pitbull vs husky

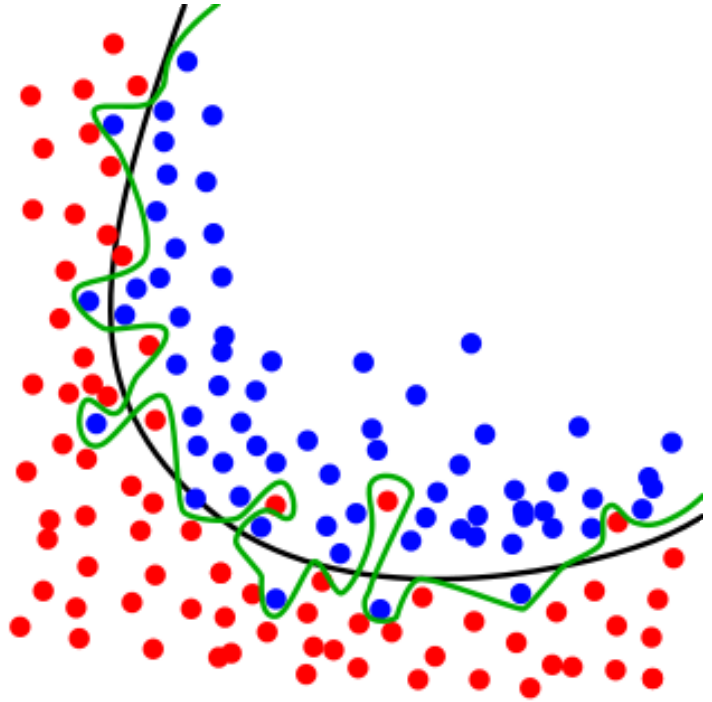
photos from websubstance/Bigstock, imgur.com/exoticearthporn

Challenges in ML

Why not learn a very complex function that can have 0 train set error and be done with it?

Extreme example: Let's memorize the data. To predict an unseen data, just guess a random label.

Doesn't generalize to unseen data – called *overfitting* the training data.



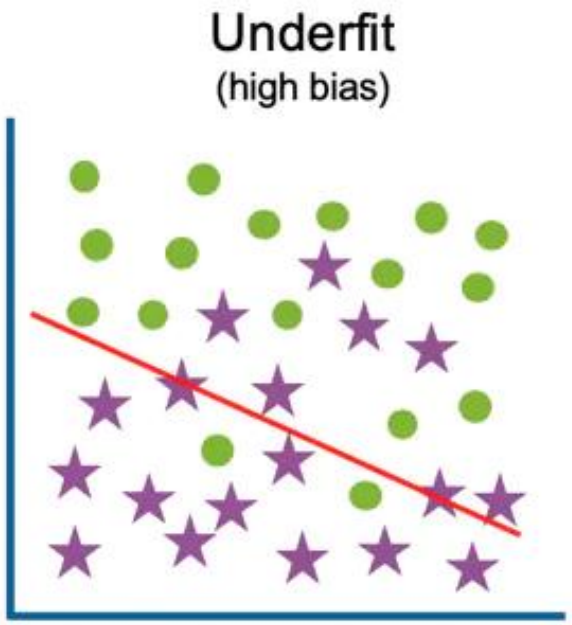
green: 1-NN: almost memorization
black: true decision boundary

Solution:

Try multiple ML methods with varying degree of "complexity" and choose the best one.

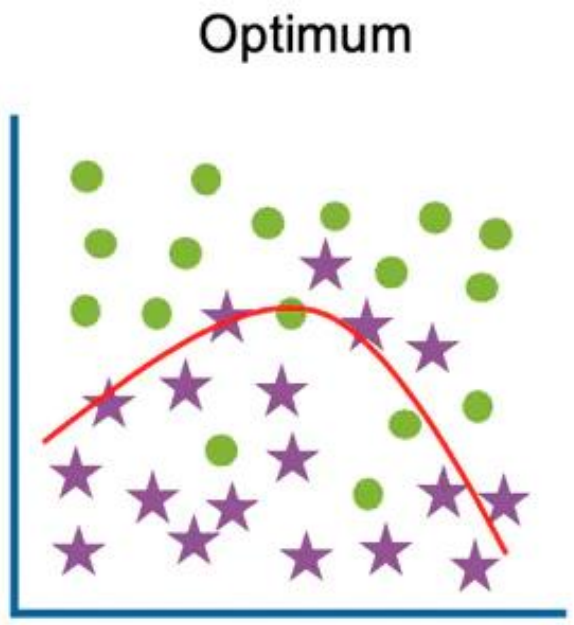
but how to choose?

Overfitting vs Underfitting



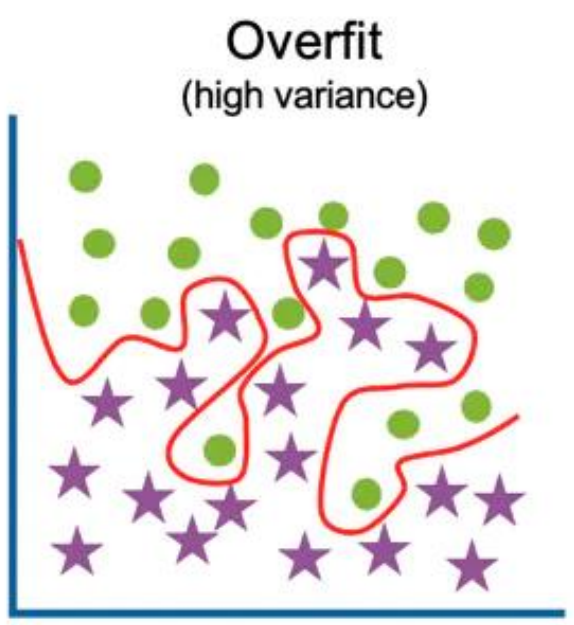
High training error
High test error

$\widehat{err}_S(\hat{f}) \approx err_{\mathcal{D}}(\hat{f})$
both high



Low training error
Low test error

$\widehat{err}_S(\hat{f}) \approx err_{\mathcal{D}}(\hat{f})$
both low



(very) Low training error
High test error

$\widehat{err}_S(\hat{f}) \ll err_{\mathcal{D}}(\hat{f})$

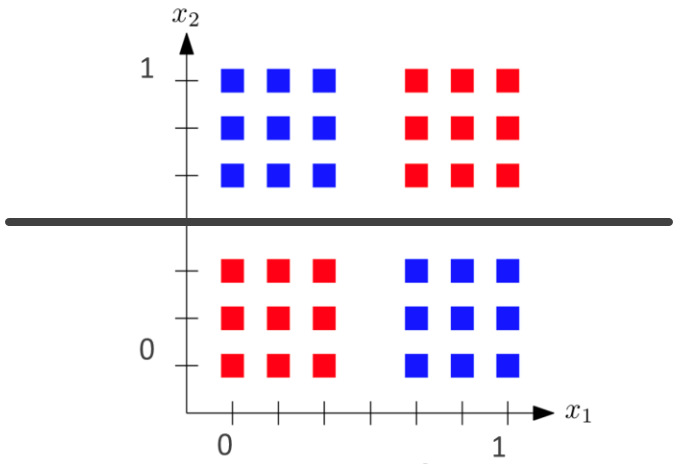
Note: These are loosely defined

Source: ibm.com

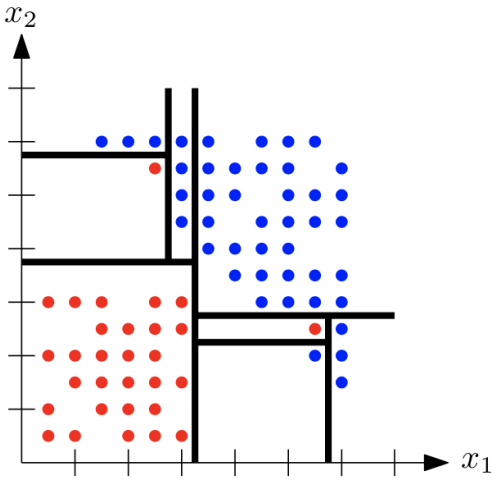
{Over,Under}-fitting

What is the inductive bias of a shallow decision tree?

Shallow tree:



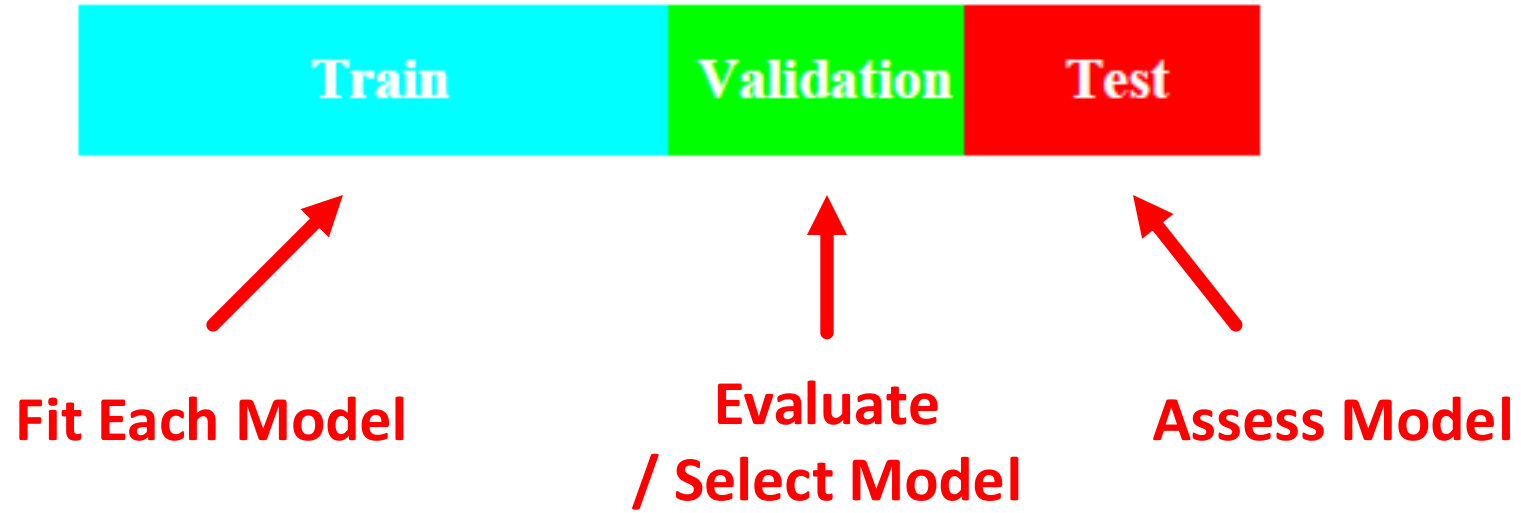
Deep tree:



- Underfitting: Can learn something but didn't
- Overfitting: Pay too much attention to idiosyncrasies to training data, and do not generalize well
- A model that neither overfits nor underfits is expected to do best

Model Selection / Assessment

Partition your data into Train-Validation-Test sets

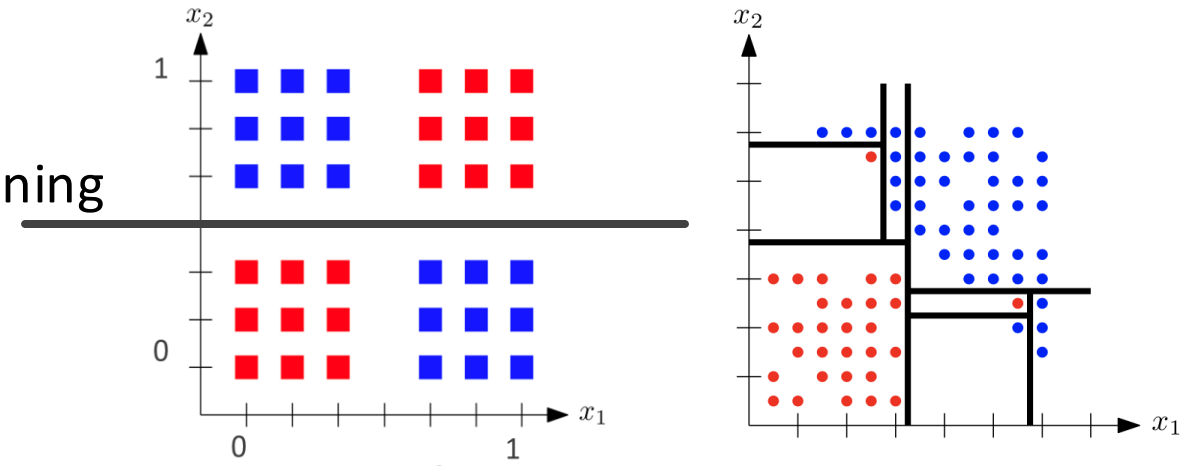


- Ideally, Test set is kept in a “vault” and **only peek at it once model is selected**
- Small dataset: 50% Training, 25% Validation, 25% Test (very loose rule)
- For large data (say a few thousands), 80-10-10 is usually fine.

Hyperparameter tuning using validation set

- E.g. in decision tree training, how to choose tree depth $h \in \{1, \dots, H\}$?

- For each hyperparameter $h \in \{1, \dots, H\}$:
  - Train Tree_h using DecisionTreeTrain by constraining the tree depth to be h
- Choose one from $\mathrm{Tree}_1, \dots, \mathrm{Tree}_H$

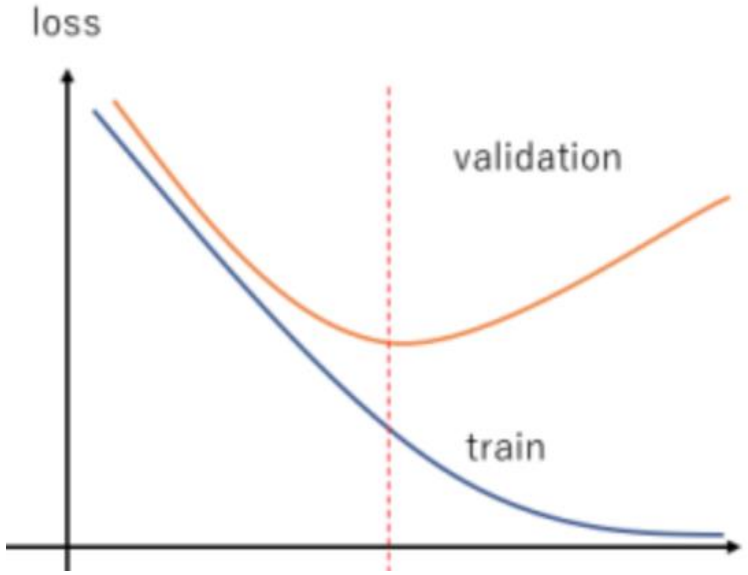


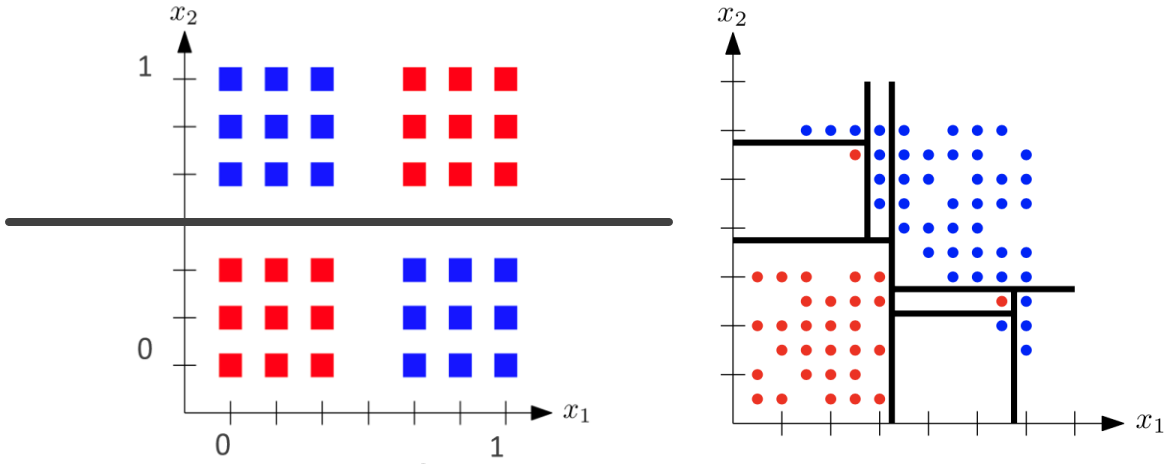
- Idea 1: choose Tree_h that minimizes training error
- Idea 2: choose Tree_h that minimizes test error
- Idea 3: further split training set to training set and validation set (development/hold-out set), (1) train Tree_h's using the (new) training set; (2) choose Tree_h that minimizes validation error

Training: 700 examples	Val:100 examples	Test: 200 examples

Hyperparameter tuning using validation set

- E.g. in decision tree training, how to choose tree depth $h \in \{1, \dots, H\}$?

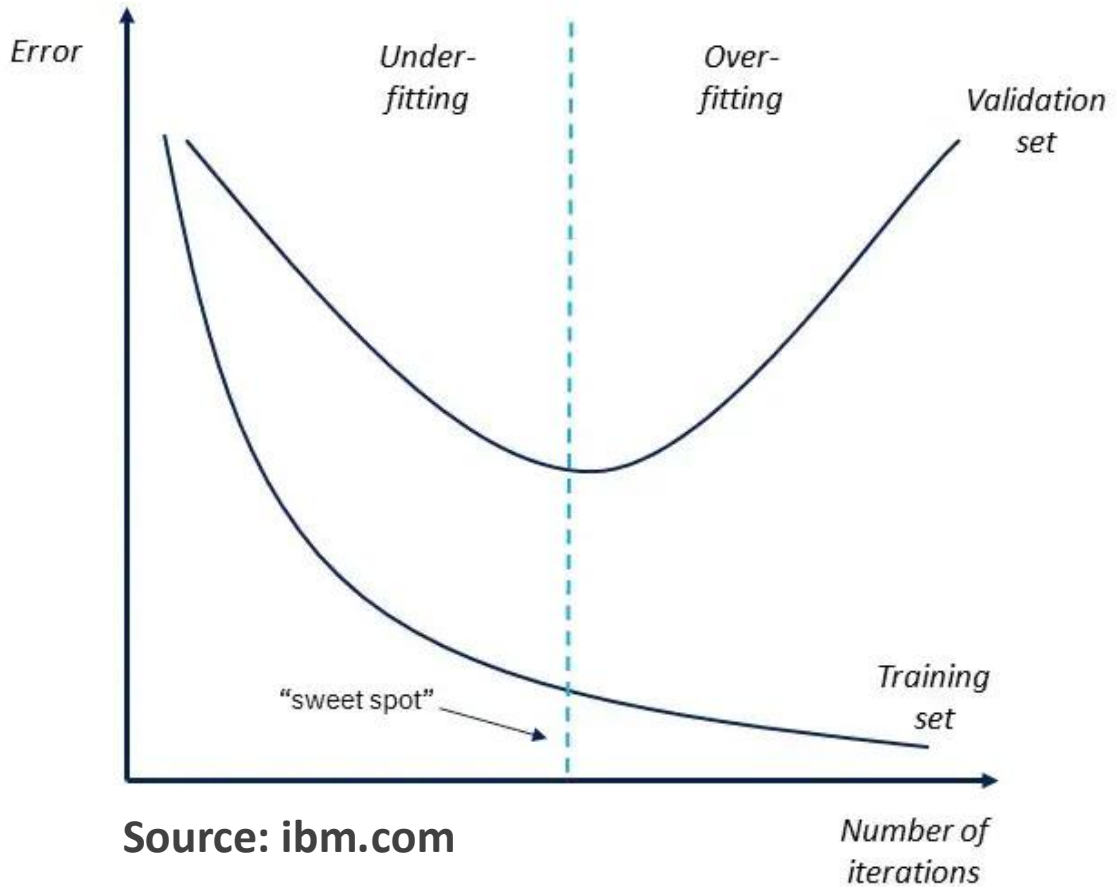




- Law of large numbers => Validation error closely approximates test error & generalization error

Overfitting vs Underfitting

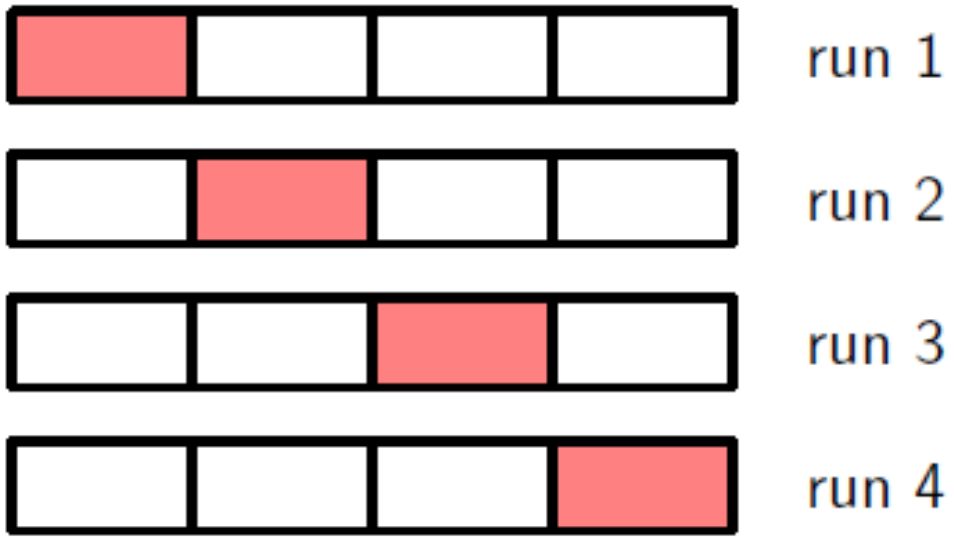
Underfitting performs poorly on *both* training and validation…



Source: ibm.com

…overfitting performs well on training but not on validation

Cross-Validation



N-fold Cross Validation Partition training data into N “chunks” and for each run select one chunk to be validation data

For each run, fit to training data (N-1 chunks) and measure accuracy on validation set. Average model error across all runs.

Drawback Need a lot of training data to partition.

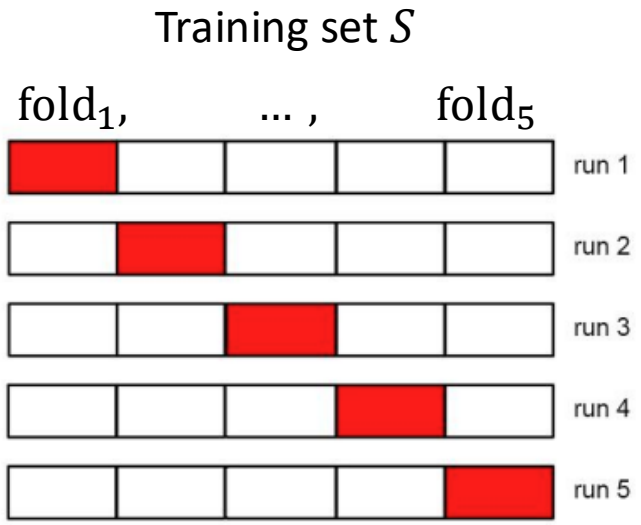
Source: Bishop, C. PRML

Hyperparameter tuning: cross-validation

- Main idea: split the training / validation data in multiple ways

- For hyperparameter $h \in \{1, \dots, H\}$
  - For $k \in \{1, \dots, K\}$
    - train $\hat{f}_k^h$ with $S \setminus \text{fold}_k$
    - measure error rate $e_{h,k}$ of $\hat{f}_k^h$ on fold_k
  - Compute the average error of the above: $\widehat{\text{err}}^h = \frac{1}{K}\sum_{k=1}^{K} e_{h,k}$
- Choose $\hat{h} = \arg\min_h \widehat{\text{err}}^h$
- Train $\hat{f}$ using S (all the training points) with hyperparameter $\hat{h}$

- $k = |S|$: leave one out cross validation (LOOCV)



Stratification in k-CV

- Issue: Say we have few positive labels (=imbalanced class)
  The error rates in CV can be unstable.
  (or even, some folds may have no positive points!)

- Goal: ensure each fold receives the same fraction of pos/neg labels.

- E.g., |S|=100. 20 positive/80 negative. K=10
  - Each fold should have 2 $\oplus$, 8 $\ominus$.
  - Pool positive data points, randomly shuffle them; place 2 data points for each fold.
  - Perform the same with negative data points.

The role of test set

- Your boss says: I will allow your recommendation system to run on our website only if the accuracy is $\geq$ 90% accuracy!

- Again, never allow test set to be part of train/validation set!!

Review of terminologies

- Parameter

- Hyperparameter

- Model = ‘model class/family’ + parameter

Implementation tips

Useful python packages

- **numpy**: numerical computing (only includes the 'essential' part)
  - Classes for vectors and matrices

- **scipy**: scientific computing
  - Based on numpy
  - Includes numerical optimization (=minimize a given function), integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics

- **scikit-learn**: numpy-based machine learning library

- **pytorch**: python package for ML (mainly for deep learning)

An example real-world machine learning pipeline

- Any step can go wrong
  - E.g. data collection, data representation
- Debugging pipeline: run oracle experiments
  - Assuming the downstream tasks are perfectly done, is this step achieving what we want?
- General suggestions:
  - Build the stupidest thing that could possibly work
  - Decide whether / where to fix it

1	real world goal	increase revenue
2	real world mechanism	better ad display
3	learning problem	classify click-through
4	data collection	interaction w/ current system
5	collected data	query, ad, click
6	data representation	bow^2, $\pm$ click
7	select model family	decision trees, depth 20
8	select training data	subset from april'16
9	train model & hyperparams	final decision tree
10	predict on test data	subset from may'16
11	evaluate error	zero/one loss for $\pm$ click
12	deploy!	(hope we achieve our goal)

Debugging

- Take debugging seriously
  - You can get stuck indefinitely! – it's better to take some time to do a systematic approach
  - It's a real problem solving/research skill!
  - Debugging is harder in ML since verifying correctness is nontrivial.

You implemented an algorithm to get error rate 45%..
It's relatively high for binary classification.
You think: *Oh, maybe I did not implement it correctly?*

- pdb/ipdb: debugging tools
  - Examining values are more important in numerical programming!!
  - pdb.set_trace(): insert this to the place where you want to stop briefly and check values or run some functions with the variables right at that point in runtime.
  - pdb.pm(): when error happens, it gets you back to the place where the error happened, so you can easily check values and go up and down in the stack trace
    - Read: https://wil.yegelwel.com/pdb-pm/

Data splitting for train/validation/test or k-CV

- I am given a dataset from someone else. It has 10 categorical labels.
- Total 1000 points.
- I am going to take the first 800 points for training, 100 points for validation, and the rest for test.
- What's wrong with this approach?

The data points could have been sorted in certain way.
(e.g., first 100 points are label 1, the next 100 are label 2, …)

Or, labels are mixed, but the instances may come sorted by certain feature (e.g., age, timestamp of the tweet from a day worth of tweets).

Dataset split with numpy

Example: (k=5)-fold CV

```
import numpy as np
permidx = np.random.permutation(12)
```

```
array([ 6,  1,  8,  7,  3,  4,  2,  5, 11, 10,  0,  9])
```

array = vector in math

```
idx = np.arange(12) % 5
```

```
array([0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1])
```

idx==0 is array([True, False, False, False, False, True, False, False, False, False, True, False])

```
folds = [permidx[idx == i] for i in np.arange(5)]
```

```
[array([6, 4, 0]),
 array([1, 2, 9]),
 array([8, 5]),
 array([7, 11]),
 array([3, 10])]
```

```
folds_except = [permidx[idx != i] for i in np.arange(5)]
```

```
[array([ 1,  8,  7,  3,  2,  5, 11, 10,  9]),
 array([ 6,  8,  7,  3,  4,  5, 11, 10,  0]),
 array([ 6,  1,  7,  3,  4,  2, 11, 10,  0,  9]),
 array([ 6,  1,  8,  3,  4,  2,  5, 10,  0,  9]),
 array([ 6,  1,  8,  7,  4,  2,  5, 11,  0,  9])]
```

If the data is feature X (n by d array; n data points, d dimension) and label Y (length-n array)

- For $i = 1..k$
  - train set: trainX = X[folds_except[i],:]
    trainY = Y[folds_except[i]]
  - validation set: valiX = X[folds[i],:]
    valiY = Y[folds[i]]
  - …

Debugging under randomness

- During the development phase, you should use random seed.
- Otherwise, your code could have a ‘stochastic bug’.

```
In [28]: import numpy as np

In [29]: np.random.seed(3)

In [30]: np.random.rand()
Out[30]: 0.550797902574575 5

In [31]: np.random.rand()
Out[31]: 0.708147822618104 8
```

```
In [28]: import numpy as np

In [32]: np.random.seed(31)

In [33]: np.random.rand()
Out[33]: 0.28605382166051563

In [34]: np.random.rand()
Out[34]: 0.958105566519
```

Scikit-Learn

Models can be fit using the `fit()` function. E.g., Random Forest Classifier,


```
>>> from sklearn.ensemble import RandomForestClassifier
>>> clf = RandomForestClassifier(random_state=0)
>>> X = [[ 1,  2,  3],  # 2 samples, 3 features
...      [11, 12, 13]]
>>> y = [0, 1]  # classes of each sample
>>> clf.fit(X, y)
RandomForestClassifier(random_state=0)
```

`fit()` Generally accepts 2 inputs

- Sample matrix X— 2d array (n_samples, n_features)
- Target values Y— real numbers for regression, integer for classification

k-Nearest Neighbors

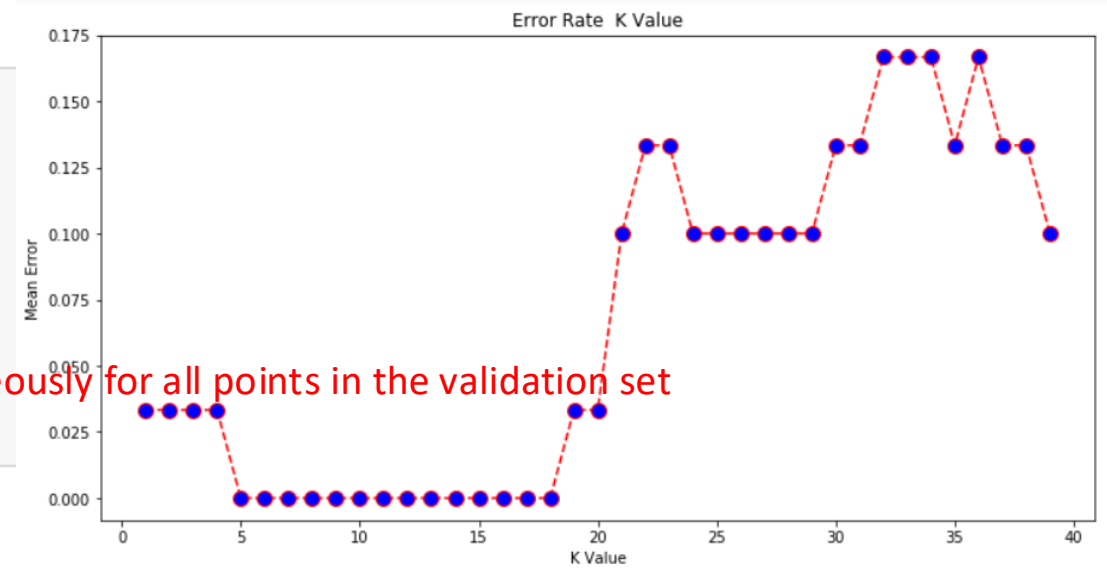
Train / evaluate the KNN classifier for each value k,

```
from sklearn.neighbors import KNeighborsClassifier
```

```
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_val) // make predictions simultaneously for all points in the validation set
    error.append(np.mean(pred_i != y_val))
```

↑ vector operation!
(element-wise)



Plot error vs k:

For large dataset, use geometric grid like 1,2,4,8,…

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Preprocessing : Z-Score

Typical ML workflow starts with *pre-processing* or *transforming* data into some useful form, which Scikit-Learn calls *transformers*:

```
>>> from sklearn.preprocessing import StandardScaler
>>> X = [[0, 15],
...      [1, -10]]
>>> # scale data according to computed scaling values
>>> StandardScaler().fit(X).transform(X)
array([[-1.,  1.],
       [ 1., -1.]])
```

Example use this to standardization in k-NN.

fit(X) returns the object created by StandardScaler() so you can use a series of dot operations!

- Features are standardized independently (columns of X)
- Other transformers live in `sklearn.preprocessing`

Preprocessing : Encoding Labels

Oftentimes, categorical labels come as strings, which aren't easily modeled (e.g., with Naïve Bayes).

```
>>> le = preprocessing.LabelEncoder()
>>> le.fit(["paris", "paris", "tokyo", "amsterdam"])
LabelEncoder()  0       1        2          3
>>> list(le.classes_)
['amsterdam', 'paris', 'tokyo']
>>> le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1]...)
>>> list(le.inverse_transform([2, 2, 1]))
['tokyo', 'tokyo', 'paris']
```

`LabelEncoder` transforms these into integer values, e.g. for categorical distributions

fit() is doing the heavy work: create the mapping from string to integers

Can *undo* using `inverse_transform` so we don't have to store two copies of the data

Cross-Validation

Easily do cross validation for model selection / evaluation…

```
>>> from sklearn.datasets import make_regression
>>> from sklearn.linear_model import LinearRegression
>>> from sklearn.model_selection import cross_validate
...
>>> X, y = make_regression(n_samples=1000, random_state=0)
>>> lr = LinearRegression()
...
>>> result = cross_validate(lr, X, y)  # defaults to 5-fold CV
>>> result['test_score']  # r_squared score is high because dataset is easy
array([1., 1., 1., 1., 1.])
```

& score being R^2 for regression

- `sklearn.model_selection`
  - Many split functions: K-fold, leave-one-out, etc.

The **cross_validate** function differs from **cross_val_score** in two ways:

- It allows specifying multiple metrics for evaluation.
- It returns a dict containing fit-times, score-times (and optionally training scores as well as fitted estimators) in addition to the test score.