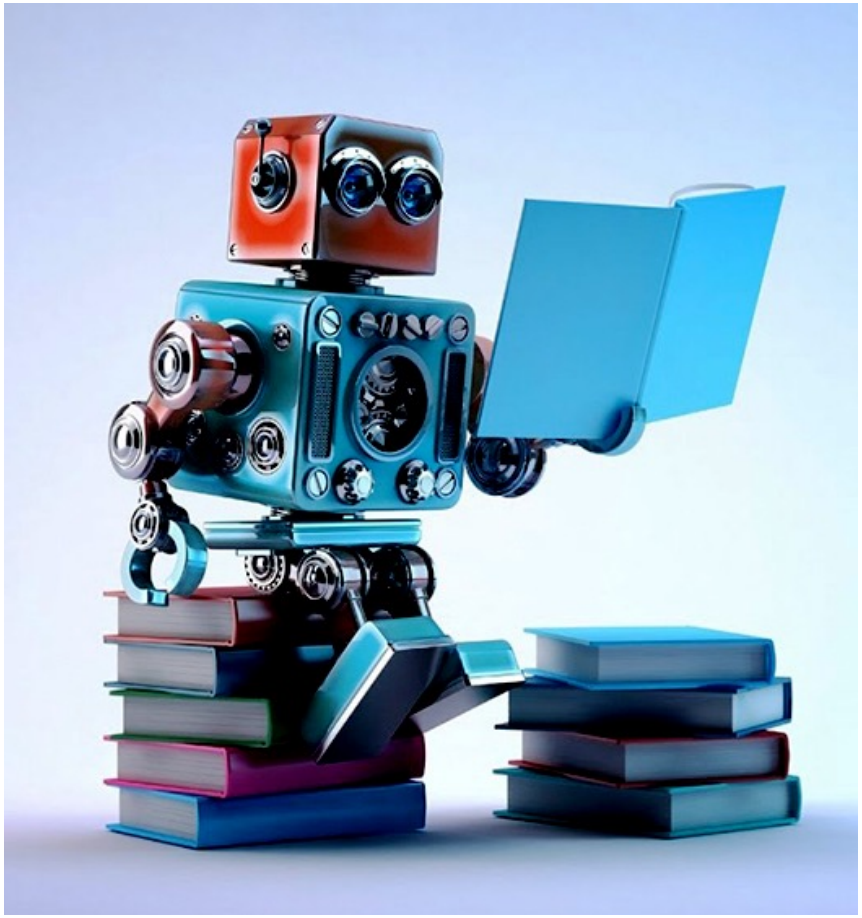


Machine Learning: Methodology

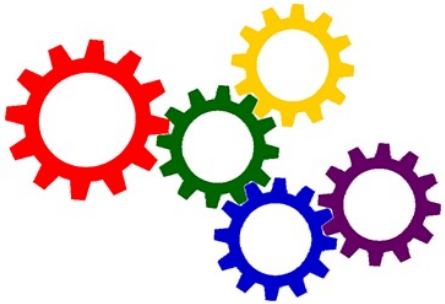
Chapter 19

ML is an experimental science

- Most ML work has an engineering or experimental flavor
  - it's being used as a tool to solve a problem
- **Methodology** is important
- As are approaches for evaluating results
- Common to try multiple ML methods, features, and parameters for a problem to find what works best
- Google's Rules of Machine Learning has more information

Martin Zinkevich introduces 10 of his favorite rules of machine learning. Read on to learn all 43 rules!

Many moving parts



Solving a problem with machine learning involves many decisions

- Selecting training data and deciding how much is needed
- Preprocessing the data, creating new **features** from it
- Selecting a machine learning algorithm
- Choosing its parameters
- Deciding on a metric to optimize
- Running evaluation experiments

Approaches

- Different classes of ML algorithms have different kinds of evaluation techniques
  - Some are common to most, however
- **Supervised ML** 

  - **Use some data with right answers for evaluation**
- Unsupervised ML
  - Some general metrics exist (e.g., for clusters)
  - May need human assessments
- Reinforcement learning
  - Problem determines good/bad outcomes (e.g., points won in a game)

UCI Machine Learning Repository: Zoo Data Set

http://archive.ics.uci.edu/ml/datasets/Zoo

Google

UCI

About Citation Policy Donate a Data Set Contact

Search

Repository Web

Google™

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

View ALL Data Sets

Zoo Data Set

Download: Data Folder, Data Set Description

Abstract: Artificial, 7 classes of animals

http://archive.ics.uci.edu/ml/datasets/Zoo

Data Set Characteristics:	Multivariate	**Number of Instances:**	101	**Area:**	Life
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	17	**Date Donated**	1990-05-15
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	18038

animal name: string

hair: Boolean

feathers: Boolean

eggs: Boolean

milk: Boolean

airborne: Boolean

aquatic: Boolean

predator: Boolean

toothed: Boolean

backbone: Boolean

breathes: Boolean

venomous: Boolean

fins: Boolean

legs: {0,2,4,5,6,8}

tail: Boolean

domestic: Boolean

catsize: Boolean

type: {mammal, fish, bird, shellfish, insect, reptile, amphibian}

Zoo data

101 examples

aardvark,1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,mammal
antelope,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,mammal
bass,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,fish
bear,1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,mammal
boar,1,0,0,1,0,0,1,1,1,1,0,0,4,1,0,1,mammal
buffalo,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,mammal
calf,1,0,0,1,0,0,0,1,1,1,0,0,4,1,1,1,mammal
carp,0,0,1,0,0,1,0,1,1,0,0,1,0,1,1,0,fish
catfish,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,fish
cavy,1,0,0,1,0,0,0,1,1,1,0,0,4,0,1,0,mammal
cheetah,1,0,0,1,0,0,1,1,1,1,0,0,4,1,0,1,mammal
chicken,0,1,1,0,1,0,0,0,1,1,0,0,2,1,1,0,bird
chub,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,fish
clam,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,shellfish
crab,0,0,1,0,0,1,1,0,0,0,0,0,4,0,0,0,shellfish
...

Zoo example

```
aima-python> python
>>> from learning import *
>>> zoo
<DataSet(zoo): 101 examples, 18 attributes>
>>> dt = DecisionTreeLearner()
>>> dt.train(zoo)
>>> dt.predict(['shark',0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0])
'fish'
>>> dt.predict(['shark',0,0,0,0,0,1,1,1,1,0,0,1,0,1,0,0])
'mammal'
```

Evaluation methodology (1)

Standard methodology:

1. Collect large set of examples with correct classifications (aka ground truth data)
2. Randomly divide collection into two disjoint sets: ***training*** & ***test*** *(e.g., via a 90-10% split)*
3. Train a model using your algorithm on the **training** set giving hypothesis H
4. Measure performance of the model (and H) on the held-out **test** set, comparing model's prediction to correct class

Accuracy: a simple metric

- What measure of performance can we use?
- It depends on the kind of task, e.g.,
  - Classification (e.g., which species of iris is this)
  - Information retrieval (find relevant documents)
- One of the simplest is **accuracy**
  - Fraction of the answers that are correct
- Doesn't weigh different kinds of errors differently (e.g., false positive vs. false negatives)

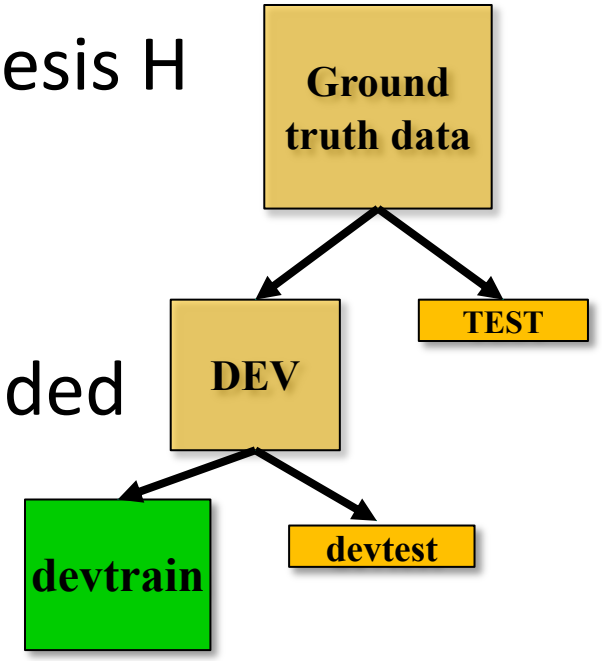
Evaluation methodology (2)

- **Important: keep training and test sets disjoint!**
  - Don’t train on all the data & then test on some of it
- Study efficiency & robustness of algorithm: repeat steps 2-4 for different training sets & training set sizes
- On modifying algorithm or its parameters, restart with step 1 to avoid evolving algorithm to work well on just this collection

Better evaluation methodology

Common variation on methodology:

1. Collect set of examples with correct classifications
2. Randomly divide it into two disjoint sets: ***developmen*t** & ***test;*** further divide development into ***devtrain*** & ***devtest***
3. Apply ML to *devtrain,* giving hypothesis H
4. Measure performance of H w.r.t. *devtest* data
5. Modify approach, repeat 3-4 as needed
6. Final test on *test* data

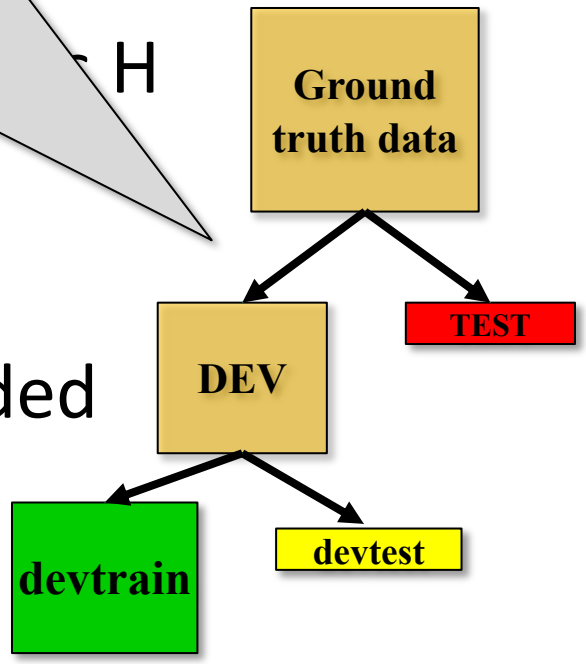


Evaluation methodology (4)

Co

1. classifications
2. sets:
   development
3. H
4. *devtest* data
5. Modify approach, repeat 3-4 as needed
6. Final test on *test* data

- Only **devtest** data used for evaluation during system **development**
- When all development has ended, **test** data used for **final evaluation**
- Ensures final trained system not influenced by test data
- If more development needed, get new dataset!



Zoo evaluation

train_and_test(learner, data, start, end) uses data[start:end] for test and rest for train

- Hold out 10 data items for test; train on the other 91; show the **accuracy** on the test data
- Doing this four times for different test subsets shows accuracy from 80% to 100%
- What's the true accuracy of our approach?

Zoo evaluation

train_and_test(learner, data, start, end) uses data[start:end] for test and rest for train

```
>>> dtl = DecisionTreeLearner
>>> train_and_test(dtl(), zoo, 0, 10)
1.0
>>> train_and_test(dtl(), zoo, 90, 100)
0.8000000000000004
>>> train_and_test(dtl(), zoo, 90, 101)
0.8181818181818183
>>> train_and_test(dtl(), zoo, 80, 90)
0.9000000000000002
```

We might use the average accuracy of the experiments as the overall metric, in this case 0.9

K-fold Cross Validation

- **Problem:** getting *ground truth* data expensive
- **Problem: n**eed different test data for each test
- **Problem:** experiments needed to find right *feature space* & parameters for ML algorithms
- **Goal:** minimize training+test data needed
- **Idea:** split training data into K subsets; use K-1 for *training* and one for *development testing*
- Repeat K times and average performance
- Common K values are 5 and 10

N-fold Cross Validation

- AIMA code has a cross_validation function that runs K-fold cross validation
- **cross_validation(learner, data, K, N)** does N iterations, each time randomly selecting 1/K data points for test, leaving rest for train

```
>>> cross_validation(dtl(), zoo, 10, 20)
0.9550000000000007
```

- Very common approach to evaluating model accuracy during development
- Best practice: hold out a final test data set

Leave one out validation

- AIMA code also has a **leave1out** function that runs experiments to estimate model accuracy
- leave1out(learner, data) does len(data) trials, each using **one element for test**, rest for train

```
>>> leave1out(dtl(), zoo)
0.9702970297029702
```

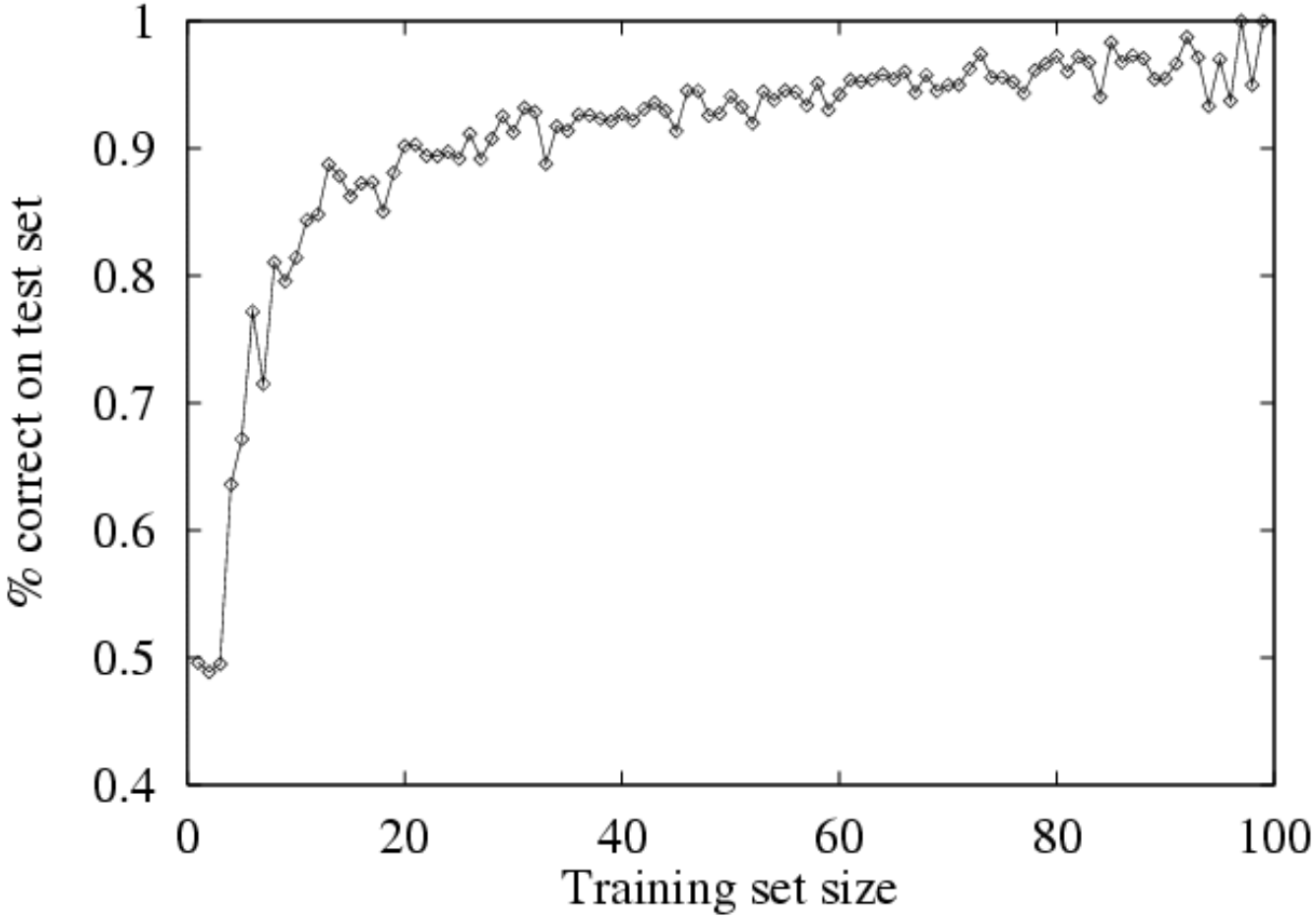
- K-fold cross validation can be *too pessimistic*, since it only trains with 80% or 90% of the data
- The leave one out evaluation is an alternative

Fast and slow learners

- We might want to evaluate a ML system w.r.t. how fast it can learn
- Some approaches require less training data to reach a given performance level than others
- We can think of them as **faster learners**
- Differences can be due to data preprocessing, algorithm choice, and/or parameter settings
- Faster generally better for many reasons, e.g., may want to apply it to many huge datasets
- **Learning curve** give an intuitive way to assess

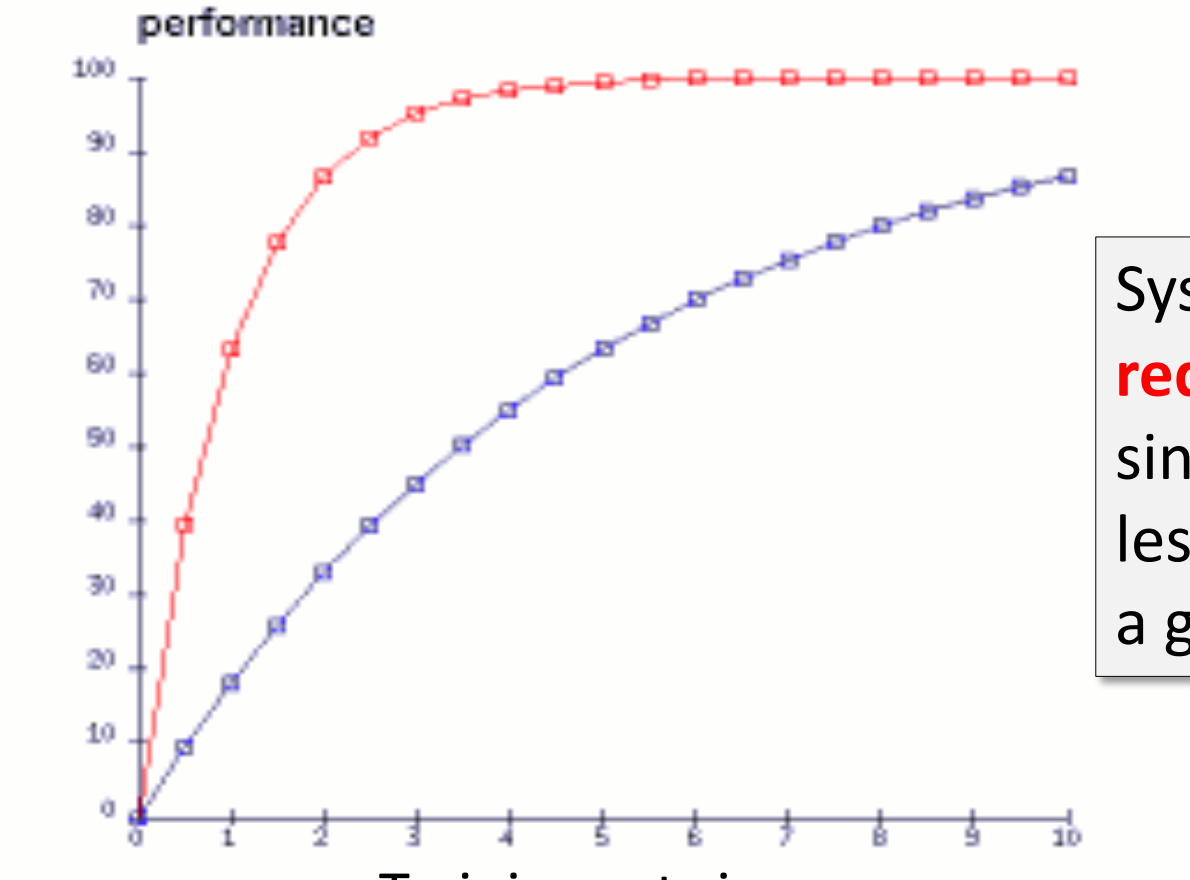
Learning curve (1)

A learning curve shows accuracy on test set as a function of training set size or (for neural networks) running time



Learning curve

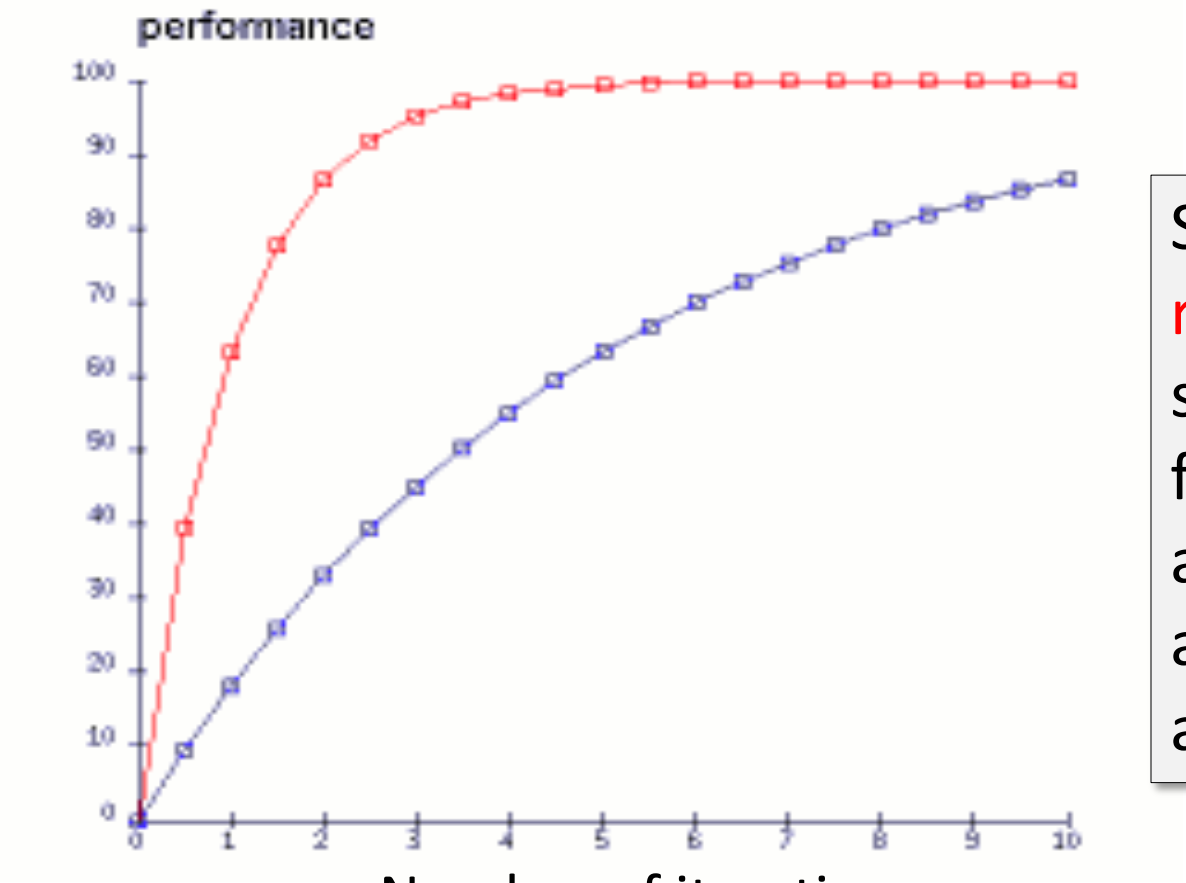
- When evaluating ML algorithms, **steeper learning curves** are better
- Represent faster learning with less data



System with the **red curve** is better since it requires less data to achieve a given accuracy

Neural network learning curves

For neural networks, the **x axis** is usually the **number of iterations** of the training algorithm



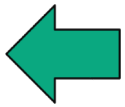
System with the red curve is better since it requires fewer iterations and less time to achieve a given accuracy

Comparing ML Approaches

- Effectiveness of ML algorithms varies depending on problem, data, and features used
- You may have intuitions, but run experiments
- Average accuracy (% correct) is a standard metric

```
>>> compare([DecisionTreeLearner, NaiveBayesLearner,
NearestNeighborLearner], datasets=[iris, zoo], k=10, trials=5)
```

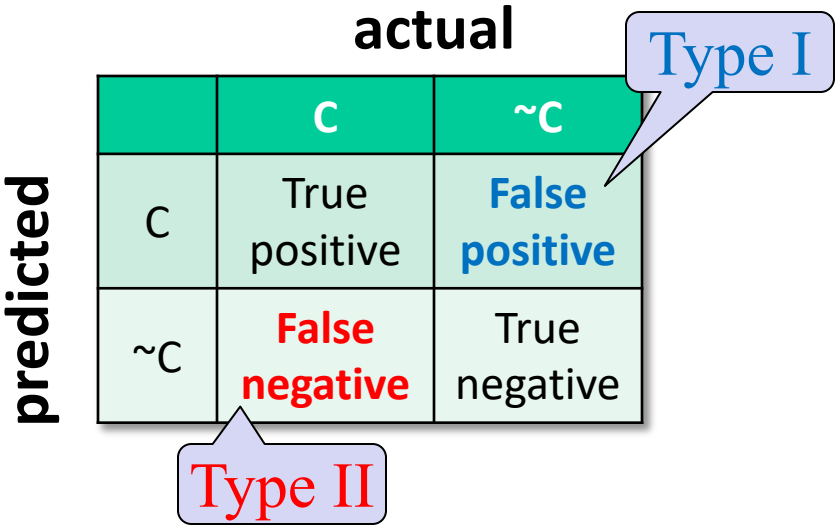
	iris	zoo
DecisionTree	0.86	0.94
NaiveBayes	**0.92**	0.92
NearestNeighbor	0.85	**0.96**



*Common practice: make best result **bold** for each experiment, e.g., NaiveBayes worked best for IRIS and NearestNeighbor was best for zoo*

Confusion Matrix (1)

- A confusion matrix can be a better way to show results for many problems
- For binary classifiers it's simple and related to type I and type II errors (i.e., false positives and false negatives)
- We may have different costs for each error
- So, we must understand their frequencies



Confusion Matrix (2)

- For multi-way classifiers, a confusion matrix is even more useful
- It lets you focus in on where the errors are

Consider a system trained to identify an images as a dog, cat, or rabbit

actual

predicted	Cat	Dog	rabbit
Cat	5	3	0
Dog	2	3	1
Rabbit	0	1	9

Correct

- Result shows system finds it easy to confuse dogs & cats
- Overall accuracy is 17/24 (71%) but for dogs vs. cats it is 8/13 (62%)

Accuracy, Error Rate, Sensitivity, Specificity

P/A	C	¬C	
C	**TP**	**FP**	**P'**
¬C	**FN**	**TN**	**N'**
	P	**N**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

Class Imbalance Problem:

- One class may be *rare*, e.g. fraud, HIV-positive, ebola
- Significant *majority in negative class &* rest in positive class
- **Sensitivity**: True Positive recognition rate
  - **Sensitivity = TP/TP+FN**
- **Specificity**: True Negative recognition rate
  - **Specificity = TN/TN+FP**

On Sensitivity and Specificity

- **High sensitivity:** few false negatives

  sensitivity=1 => TP=P => you correctly identify all positives, but may include many negatives

- **High specificity:** few false positives

  specificity=1 => TN=N => you correctly identify all negatives but may include many positives

- TSA security scenario:

  Scanners set for *high sensitivity* & low specificity (e.g., trigger on keys) reducing risk of missing dangerous objects

- Web search scenario:

  Set for *high specificity* so first page has nearly all relevant documents

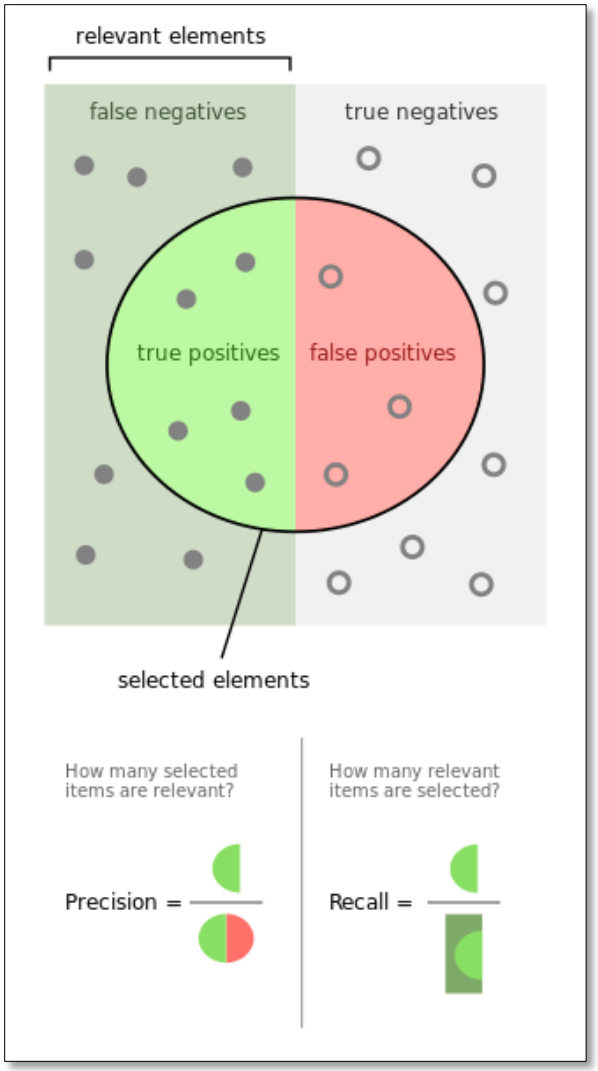
Precision and Recall

Information retrieval uses similar measures, precision & recall, to characterize retrieval effectiveness

- **Precision**: % of items classifier labels as positive that are actually positive
- **Recall:** % of positive items classifier labels as positive

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

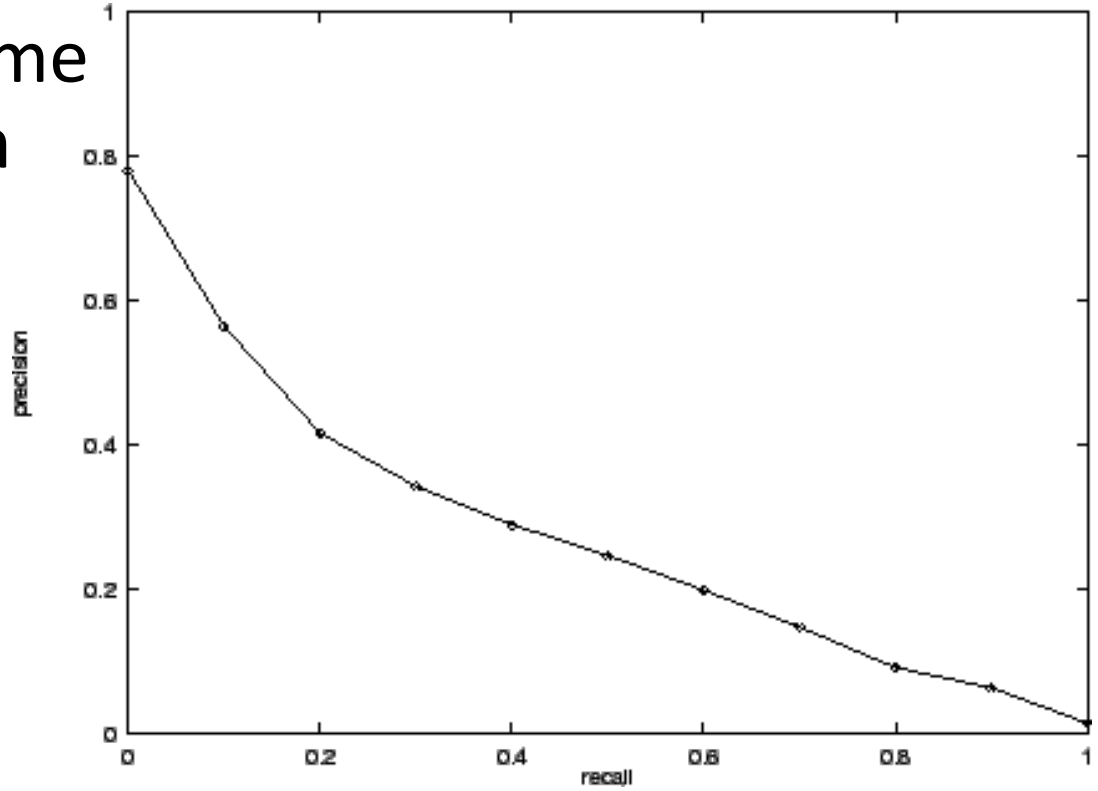


Learning to Rank

- Information retrieval might not seem like a machine learning task, but it is
- It's an example of a ranking task
  - Rather than classifying a document as relevant or nor given a query a query
  - We want a list of all relevant documents ordered by how relevant how relevant they are
- Precision & recall are relevant metrics for ranking tasks, e.g., ranking candidates for a job

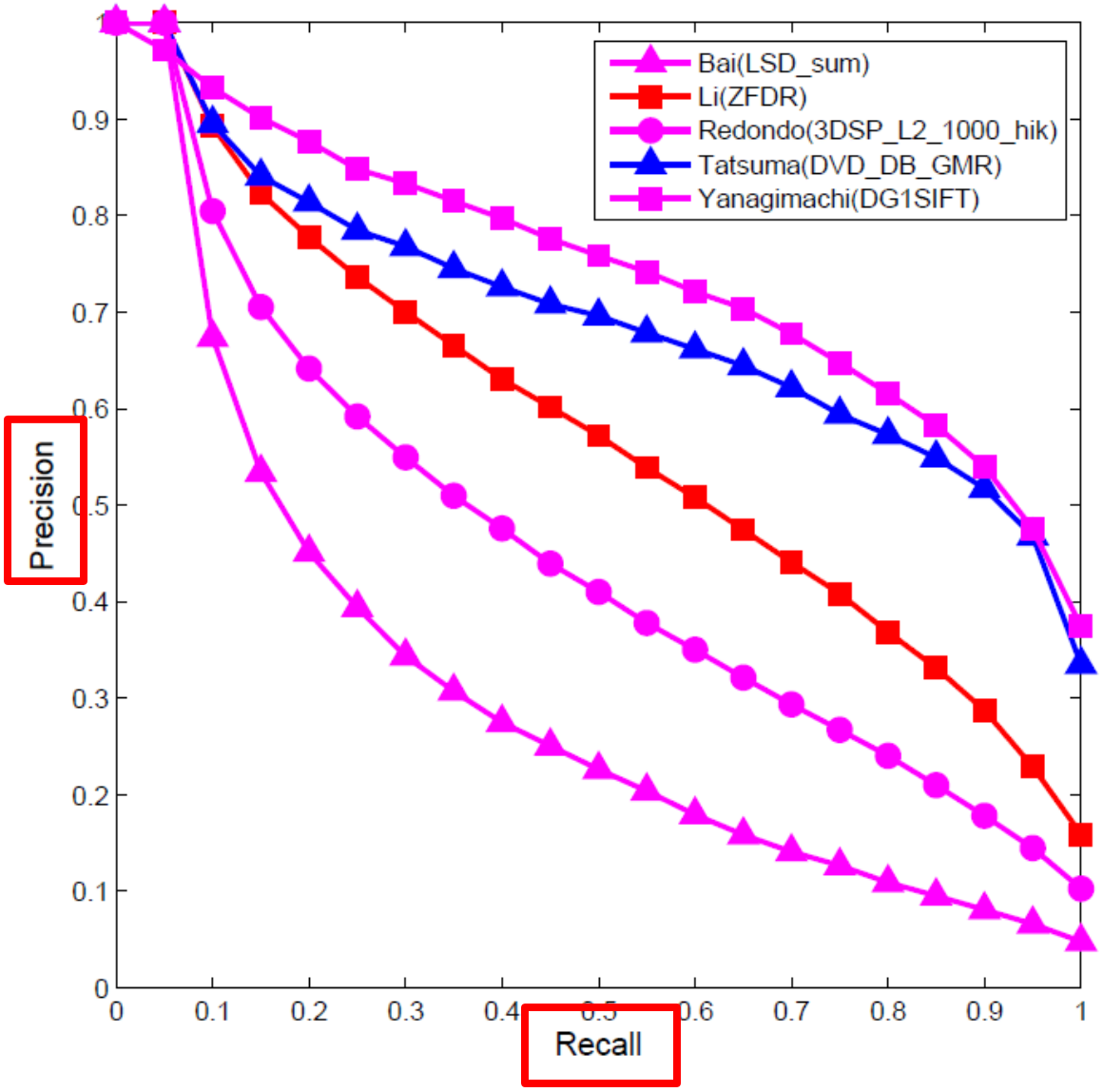
Precision and Recall

- In general, increasing one causes other to decrease
- Get **recall=1** by marking every item as positive
- Get **highest precision** by marking only one item positive, the one you are most certain of
- We usually want some **balance** of precision and recall
- Studying the precision-recall curve is informative



Precision and Recall Curves

If one system's curve is always above the other, it's better



F1 measure

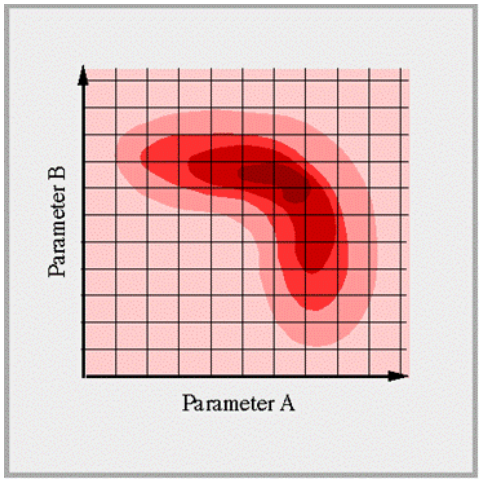
- We often want a **single measure** to compare two systems to decide which is best overall
- F1 measure combines both into a useful single metric
- It's the harmonic mean of precision & recall

$$H = \frac{2x_1x_2}{x_1 + x_2}. \qquad F = \frac{2 \times precision \times recall}{precision + recall}$$

Precision at N

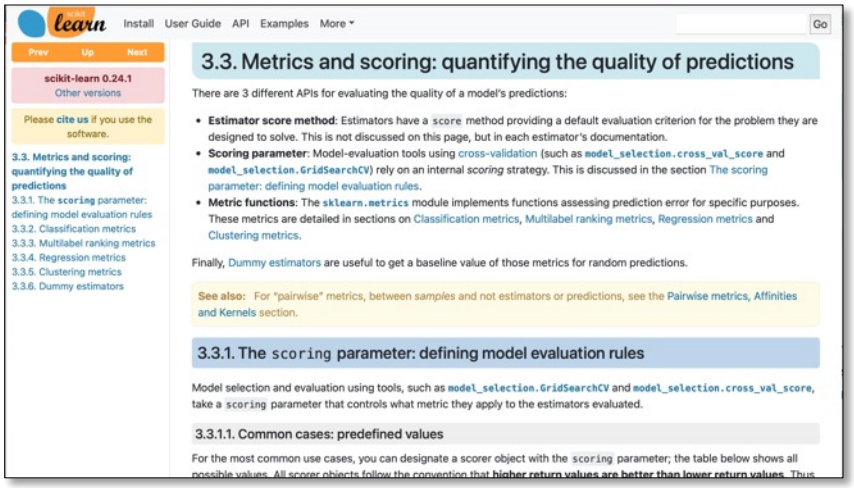
- **Ranking tasks** return a set of results ordered from best to worst
  - E.g., documents about “barack obama”
  - Best knowledge graph type for “Barack Obama”
- Learning to rank systems do this using a variety of algorithms (including SVM)
- Precision at K is the fraction of top K answers that are correct

Grid search



- ML algorithms tend to have many parameters
- How can we effectively find the best setting for all of them?
- A grid search takes a list of possible values for each of a set of parameters parameters
- …and tests each combination, to get a metric (e.g., accuracy, F1)
- See this scikit learn colab example

Model evaluation in scikit learn



[scikit.metrics](#)'s evaluation module supports most of its models in a uniform way

- It has functions that make it easy to
  - Split the data into train and test subsets
  - Do cross validation
  - Get various metrics
  - Do a [grid search](#) for a set of parameters and their possible values
- See our [colab notebooks](#) for examples

Summary

- Evaluating the results of a ML system is very important!
- Part of the development process to decide
  - What parameters maximize performance?
  - Is one system better?
  - Do we need more data?
  - Do we need different data?
  - etc.
- Many ML algorithms have specialized evaluation techniques
- There is a lot more to the topic