

# 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** 
  - **We can use our data with the right answers**
- 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)

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

# 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)
  - Information retrieval (find relevant documents)
- One of the simplest is **accuracy**
  - Fraction of the answers that were correct
- It doesn't weigh different kinds of errors differently (e.g., false positive vs false negatives)



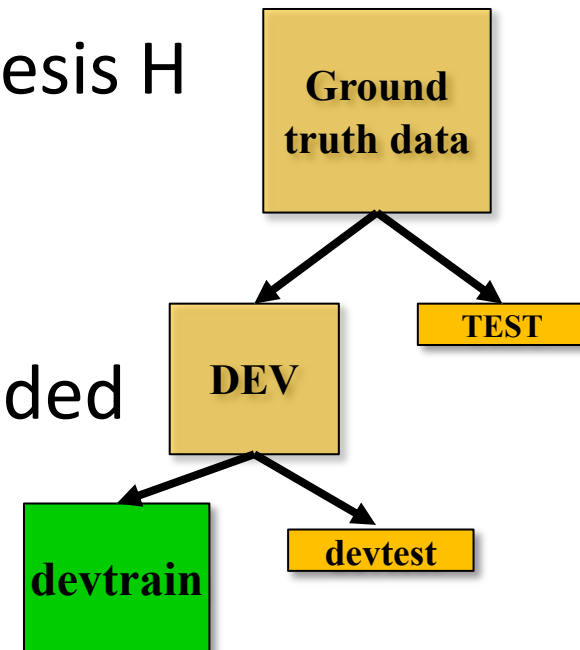
# Evaluation methodology (2)

- **Important: keep training and test sets disjoint!**
- 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:  
*development* & *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



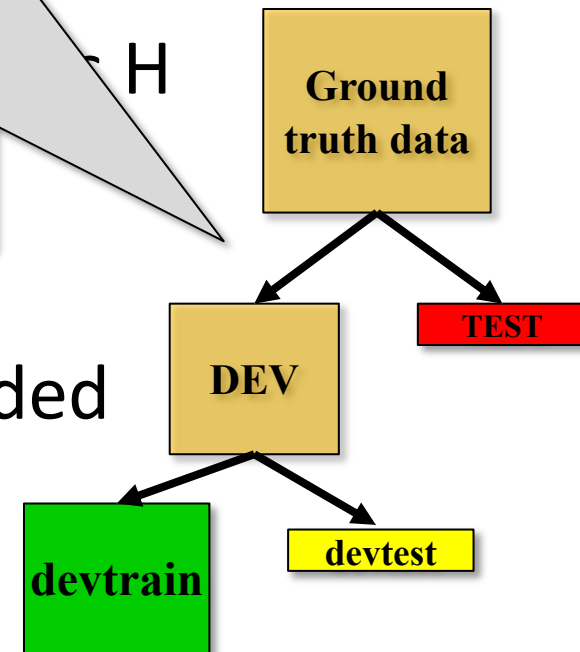
# Evaluation methodology (4)

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

classifications  
sets:  
development

*devtest* data

5. Modify approach, repeat 3-4 as needed
6. Final test on *test* data



# 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)
```

```
0.800000000000000000000004
```

```
>>> train_and_test(dtl(), zoo, 90, 101)
```

```
0.81818181818181823
```

```
>>> train_and_test(dtl(), zoo, 80, 90)
```

```
0.900000000000000000000002
```

*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:** need 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)
0.95500000000000007
```

- 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(dt1(), zoo)
0.97029702970297027
```
- 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

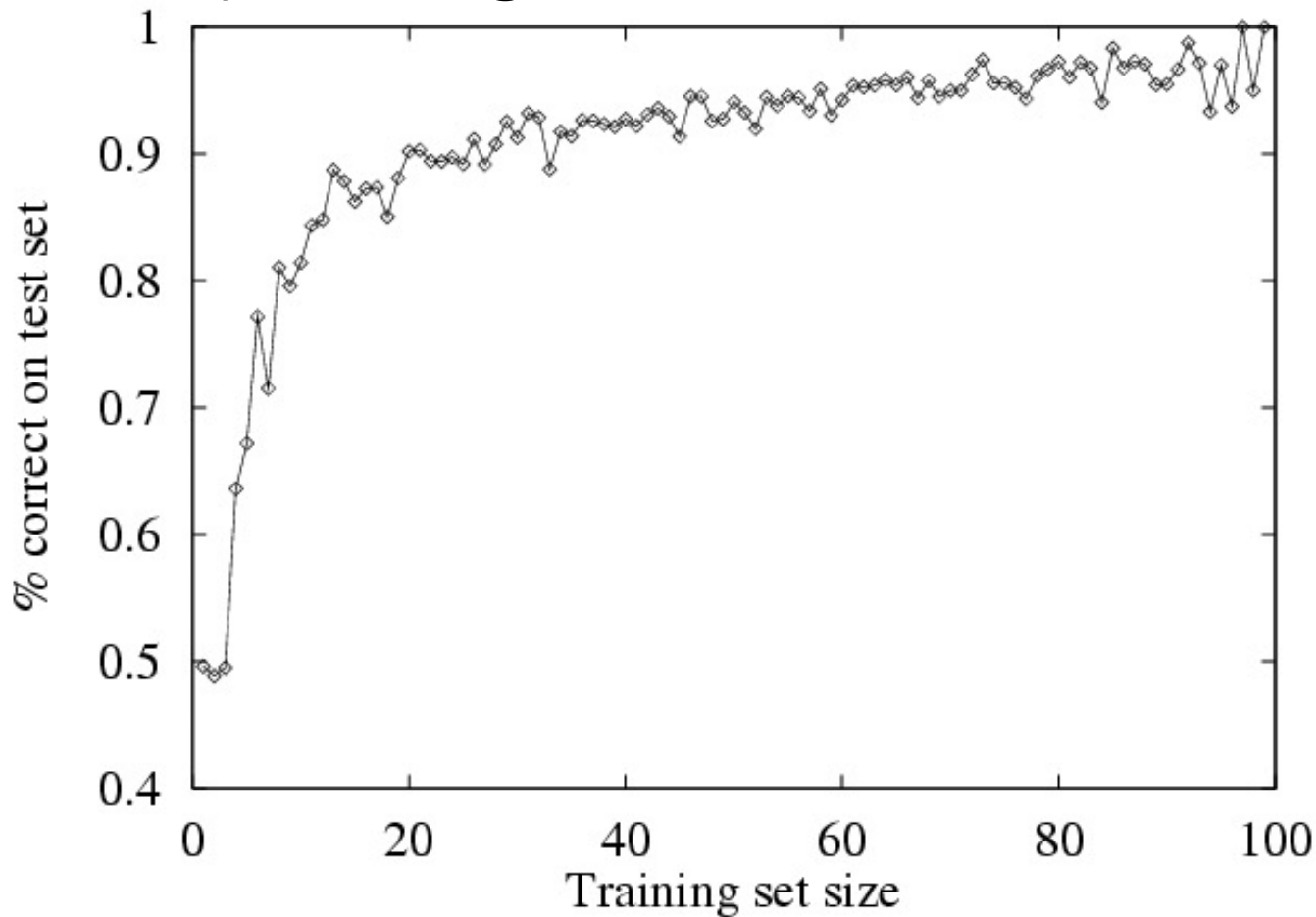


# Fast and slow learners

- 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 is generally better for many reasons (e.g., may want to apply it to many huge datasets)
- Learning curves give an intuitive way to assess

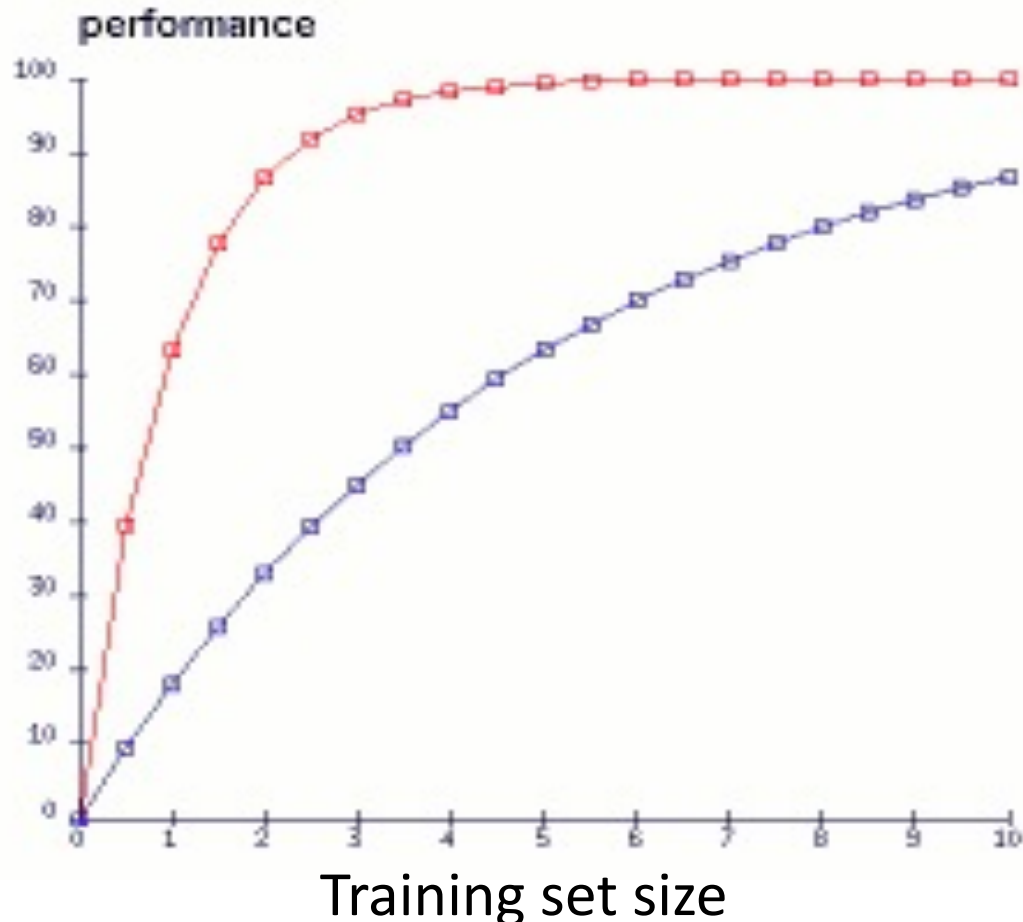
# Learning curve (1)

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



# Learning curve

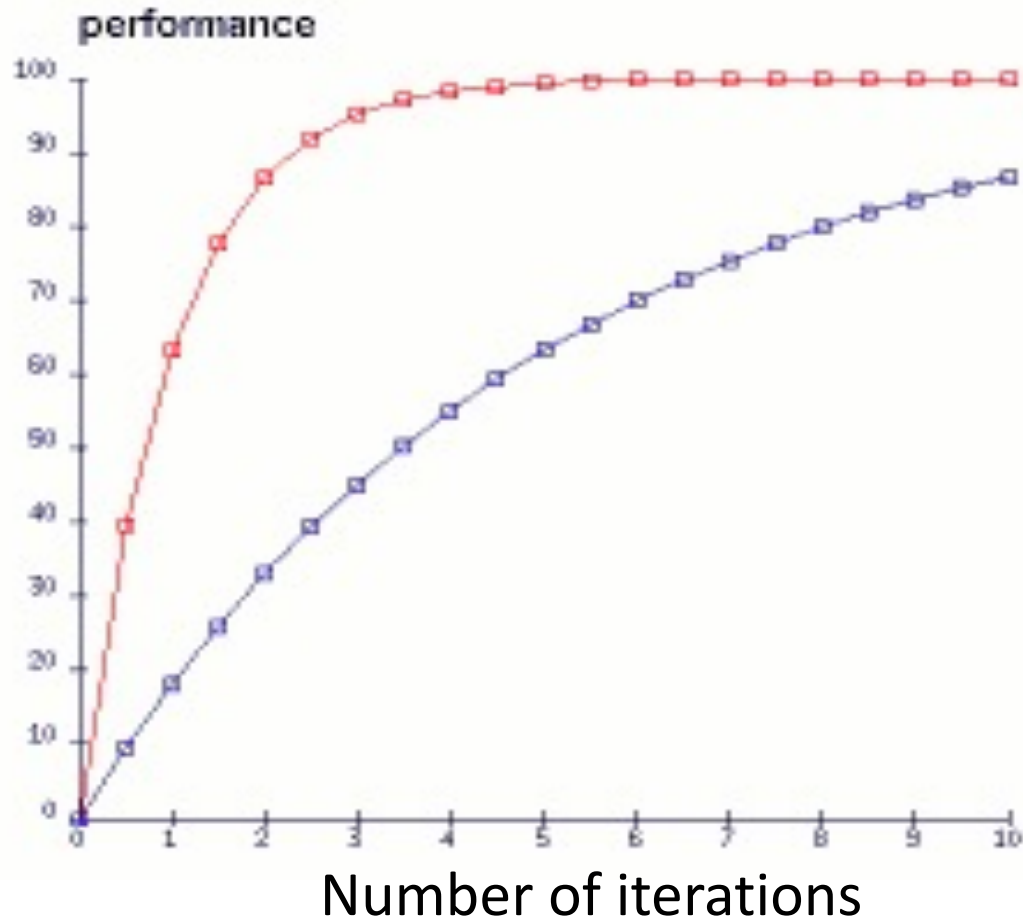
- When evaluating ML algorithms, steeper learning curves are better
- Represent faster learning with less data



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 less data to achieve a given accuracy



Search  
Repository Web

# Machine Learning Repository

Center for Machine Learning and Intelligent Systems

[View ALL Data Sets](#)

## Iris Data Set

Download: [Data Folder](#), [Data Set Description](#)

Abstract: Famous database; from Fisher, 1936



<http://archive.ics.uci.edu/ml/datasets/Iris>

<b>Data Set Characteristics:</b>	Multivariate	<b>Number of Instances:</b>	150	<b>Area:</b>	Life
<b>Attribute Characteristics:</b>	Real	<b>Number of Attributes:</b>	4	<b>Date Donated</b>	1988-07-01
<b>Associated Tasks:</b>	Classification	<b>Missing Values?</b>	No	<b>Number of Web Hits:</b>	386237

Source:

# Iris Data

- Three classes: Iris Setosa, Iris Versicolour, Iris Virginica
- Four features: sepal length and width, petal length and width
- 150 data elements (50 of each)

```
aima-python> more data/iris.csv
```

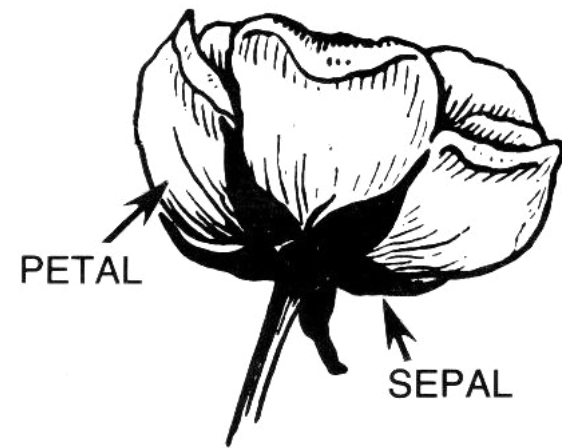
```
5.1,3.5,1.4,0.2,setosa
```

```
4.9,3.0,1.4,0.2,setosa
```

```
4.7,3.2,1.3,0.2,setosa
```

```
4.6,3.1,1.5,0.2,setosa
```

```
5.0,3.6,1.4,0.2,setosa
```



<http://code.google.com/p/aima-data/source/browse/trunk/iris.csv>

# 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	<b>0.92</b>	0.92
NearestNeighbor	0.85	<b>0.96</b>

# 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

		actual	
		C	$\sim C$
predicted	C	True positive	False positive
	$\sim C$	False negative	True negative

Type I

Type II



# Confusion Matrix (2)

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

actual

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

Correct answers

- This result suggests a system finds it easy to confuse dogs and cats

# 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

$$\text{Accuracy} = (\text{TP} + \text{TN}) / \text{All}$$

- **Error rate**:  $1 - \text{accuracy}$ , or  
 $\text{Error rate} = (\text{FP} + \text{FN}) / \text{All}$

## 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** =  $\text{TP} / (\text{TP} + \text{FN})$
- **Specificity**: True Negative recognition rate
  - **Specificity** =  $\text{TN} / (\text{TN} + \text{FP})$

# On Sensitivity and Specificity

- **High sensitivity:** few false negatives

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

- **High specificity:** few false positives

specificity=1  $\Rightarrow$  TN=N  $\Rightarrow$  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

# COVID-19 Sensitivity & Specificity

- COVID-19: test sensitivity and specificity both 0.99 (i.e., 99% accuracy)
- Assume 1% of population infected (pos)
- Test 10,000 people where 100 pos, 9900 neg
  - 99 + 99 will test positive (half right, half wrong)
  - 01 + 9801 will test negative (virtually all correct)
- Dr. Birx, April 2020: “I want to be very clear to the American people, none of our tests are 100% sensitive and specific. What do I mean by that? None of our tests that we use in medicine and diagnose 100% of the people who are positive, and correctly diagnose 100% of the people who are negative”

# Precision and Recall

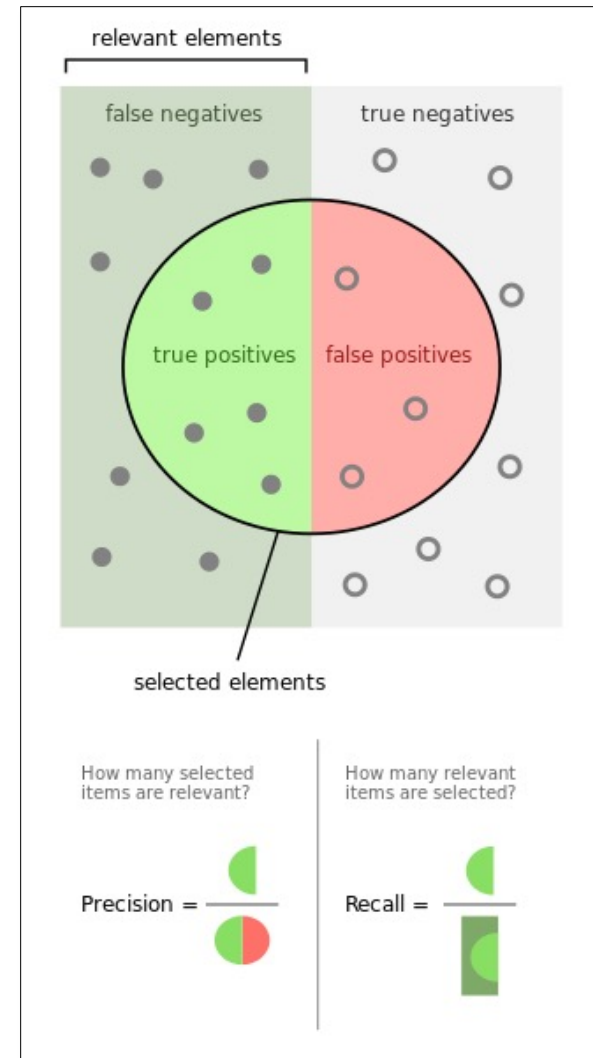
Information retrieval uses similar measures, [precision & recall](#), to characterize retrieval effectiveness

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

–**Recall:** % of positive items classifier labels as positive

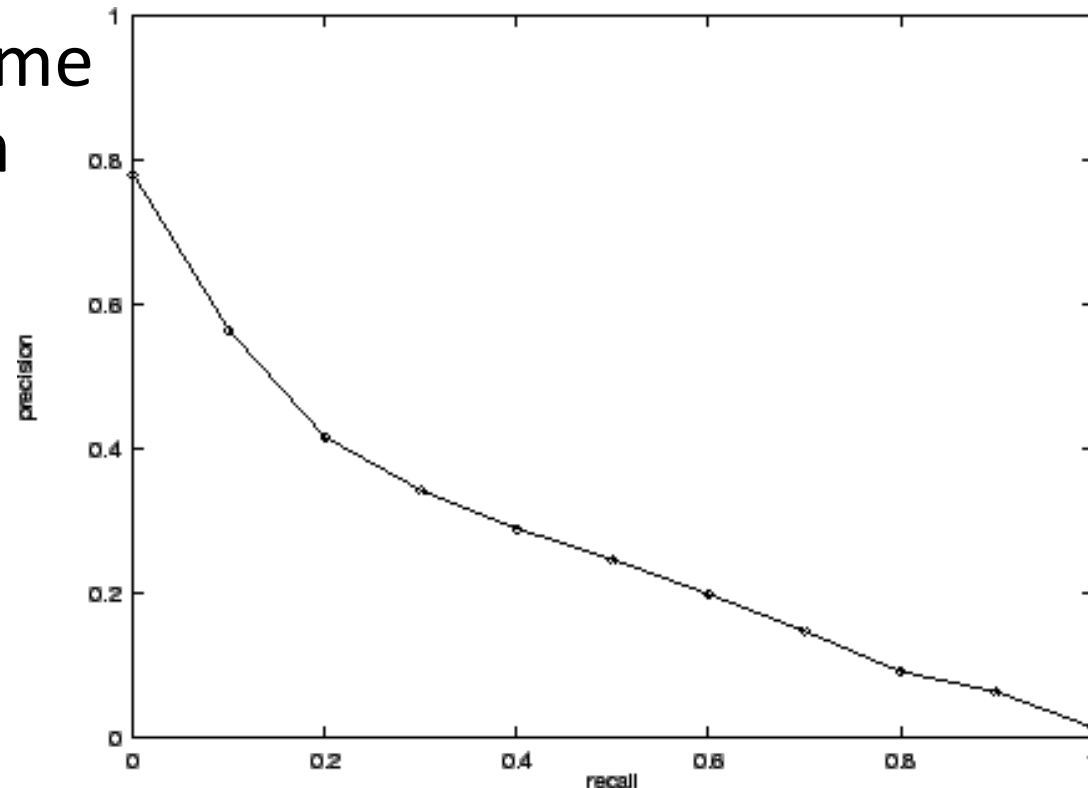
$$\textit{precision} = \frac{TP}{TP + FP}$$

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



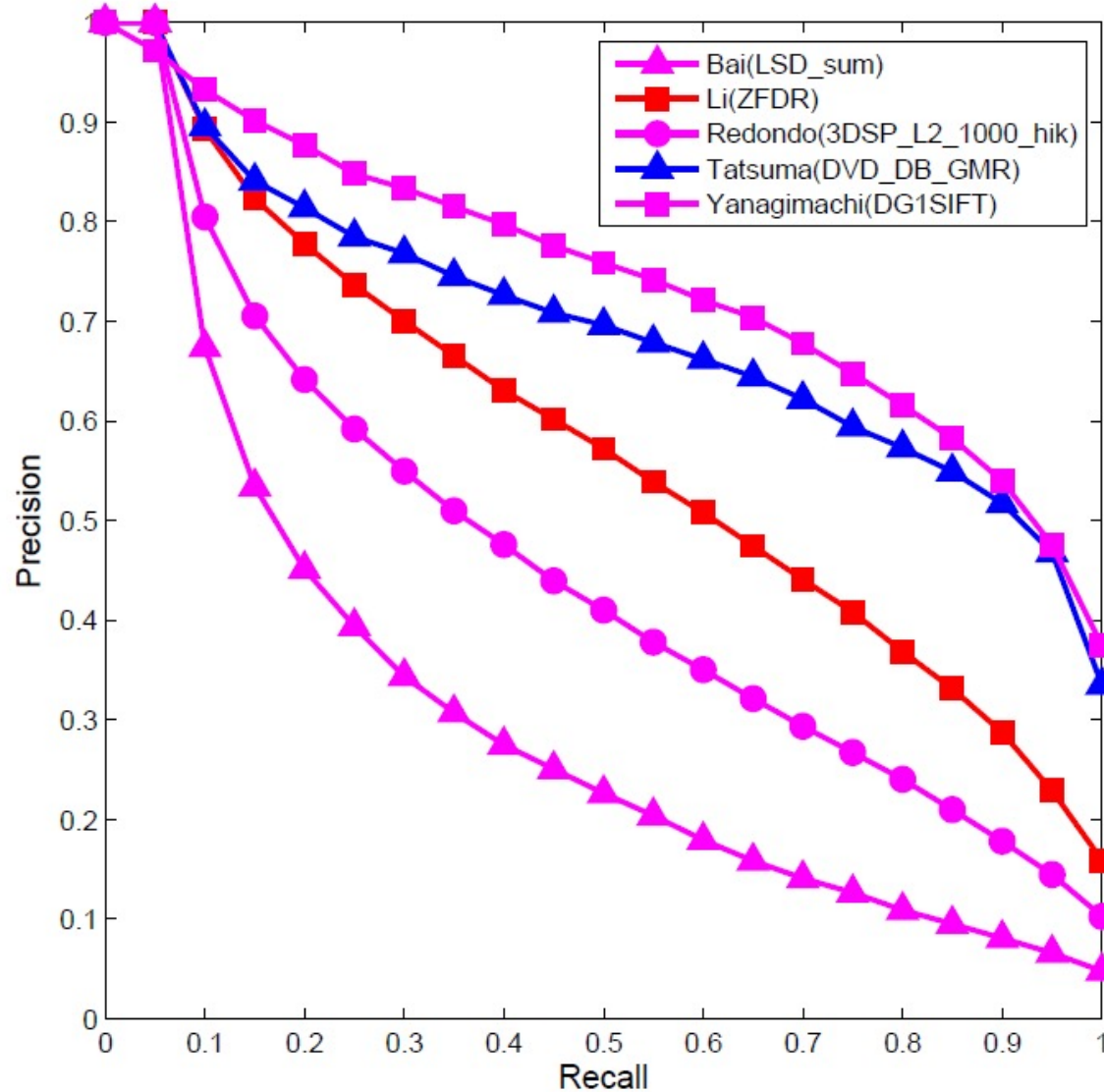
# Precision and Recall

- 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're most certain of
- We usually want some balance of precision and recall
- Studying the precision-recall curve is informative



# Precision and Recall

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



# F1 measure

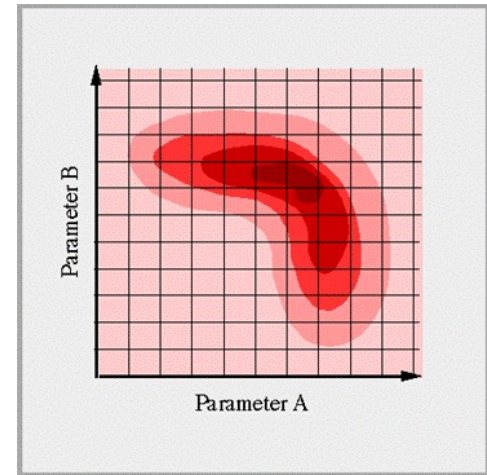
- We often want just one measure to compare two systems to decide which is best overall
- F1 measure combines both into a useful single metric
- It's the harmonic mean of precision & recall

$$H = \frac{2x_1x_2}{x_1 + x_2}, \quad F = \frac{2 \times \textit{precision} \times \textit{recall}}{\textit{precision} + \textit{recall}}$$



# 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
- ...and tests each combination, to get a metric (e.g., accuracy, F1)
- See this scikit learn colab example

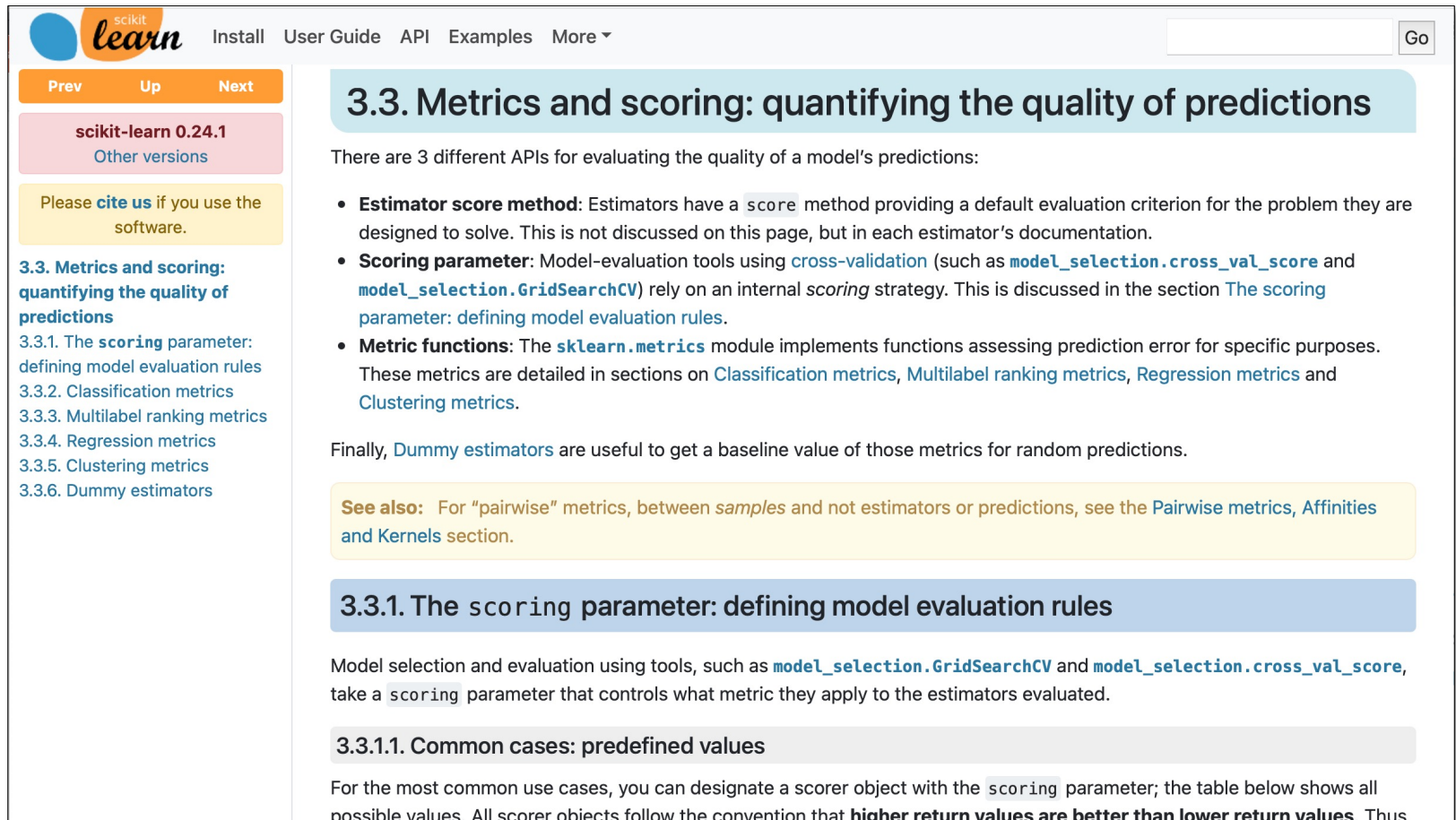


# Precision at N

- 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

# Model evaluation in scikit learn

[scikit.metrics](#) is an evaluation module that supports most of its models in a fairly uniform way



The screenshot shows the scikit-learn documentation page for the section '3.3. Metrics and scoring: quantifying the quality of predictions'. The page includes a navigation bar with links for 'Install', 'User Guide', 'API', 'Examples', and 'More'. A search bar is located in the top right corner. The left sidebar contains navigation links for 'Prev', 'Up', and 'Next', and a section for 'scikit-learn 0.24.1' with a link to 'Other versions'. A yellow box in the sidebar asks the user to 'Please cite us if you use the software.' The main content area features a blue header for the section title, followed by a paragraph stating that there are three different APIs for evaluating model quality. A bulleted list describes the 'Estimator score method', 'Scoring parameter', and 'Metric functions'. A yellow box provides a 'See also' reference to 'Pairwise metrics, Affinities and Kernels section'. Below this, a blue header introduces '3.3.1. The scoring parameter: defining model evaluation rules', followed by a paragraph explaining that model selection tools like `model_selection.GridSearchCV` and `model_selection.cross_val_score` take a `scoring` parameter. A grey header introduces '3.3.1.1. Common cases: predefined values', followed by a paragraph stating that a scorer object can be designated with the `scoring` parameter and that higher return values are better than lower return values.

scikit learn

Install User Guide API Examples More ▾

Prev Up Next

scikit-learn 0.24.1  
Other versions

Please cite us if you use the software.

## 3.3. Metrics and scoring: quantifying the quality of predictions

There are 3 different APIs for evaluating the quality of a model's predictions:

- **Estimator score method:** Estimators have a `score` method providing a default evaluation criterion for the problem they are designed to solve. This is not discussed on this page, but in each estimator's documentation.
- **Scoring parameter:** Model-evaluation tools using [cross-validation](#) (such as `model_selection.cross_val_score` and `model_selection.GridSearchCV`) rely on an internal *scoring* strategy. This is discussed in the section [The scoring parameter: defining model evaluation rules](#).
- **Metric functions:** The `sklearn.metrics` module implements functions assessing prediction error for specific purposes. These metrics are detailed in sections on [Classification metrics](#), [Multilabel ranking metrics](#), [Regression metrics](#) and [Clustering metrics](#).

Finally, [Dummy estimators](#) are useful to get a baseline value of those metrics for random predictions.

**See also:** For "pairwise" metrics, between *samples* and not estimators or predictions, see the [Pairwise metrics, Affinities and Kernels](#) section.

### 3.3.1. The scoring parameter: defining model evaluation rules

Model selection and evaluation using tools, such as `model_selection.GridSearchCV` and `model_selection.cross_val_score`, take a `scoring` parameter that controls what metric they apply to the estimators evaluated.

#### 3.3.1.1. Common cases: predefined values

For the most common use cases, you can designate a scorer object with the `scoring` parameter; the table below shows all possible values. All scorer objects follow the convention that **higher return values are better than lower return values**. Thus

# Model evaluation in scikit learn

- The [scikit.metrics](#) 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?
  - etc.
- Many ML algorithms have specialized evaluation techniques
- There is a lot more to the topic