Computer Science

CSC380: Principles of Data Science

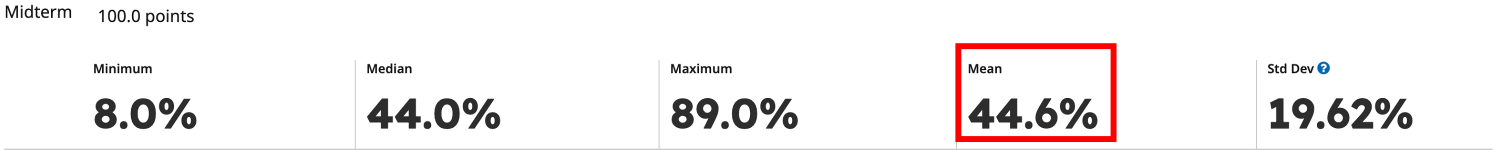
Introduction to Machine Learning / Basics of Predictive Modeling and Classification

Kyoungseok Jang

Announcement

- Delay: Due to the urgent circumstances of the TA in charge, we haven't finished the grading of the Midterm and HW4. We will have it done by next Tuesday. HW5 is out now (due : 3/24)
- Midterm Curving
  - I am thinking about $\sqrt{100\times(Your\ Score)}$ as the curved score.
    - E.g.) If your score is 50, your curved score is slightly over 70.

Because of the TA's circumstances,
One problem is not graded yet.



Midterm 100.0 points

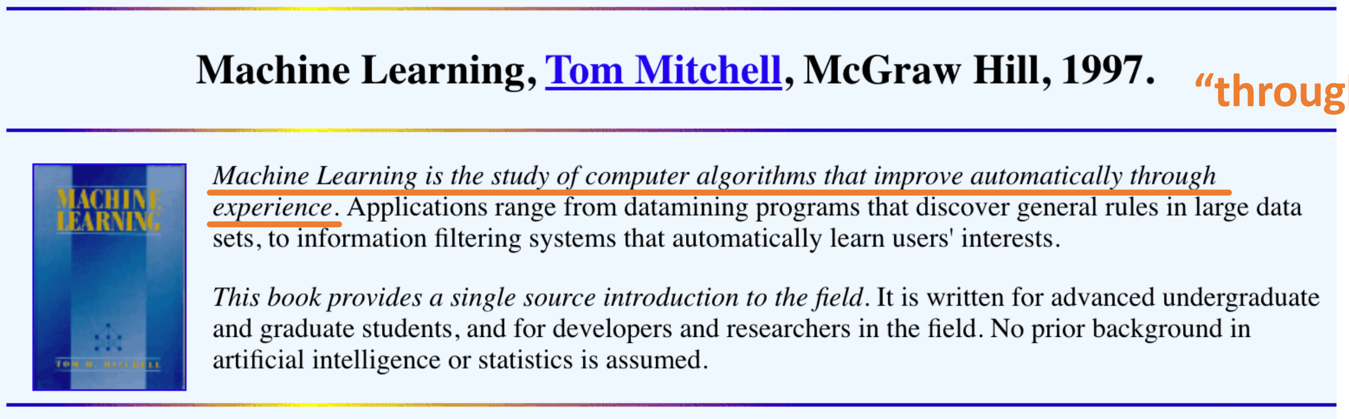
Minimum	Median	Maximum	Mean	Std Dev
8.0%	44.0%	89.0%	44.6%	19.62%

I expect around 50% after full grading...

- Self-Withdrawal deadline: 3/28

What is machine learning?

- **Tom Mitchell** established Machine Learning Department at CMU (2006).



Machine Learning, Tom Mitchell, McGraw Hill, 1997.

Machine Learning is the study of computer algorithms that improve automatically through experience. Applications range from datamining programs that discover general rules in large data sets, to information filtering systems that automatically learn users' interests.

This book provides a single source introduction to the field. It is written for advanced undergraduate and graduate students, and for developers and researchers in the field. No prior background in artificial intelligence or statistics is assumed.

"through experience"

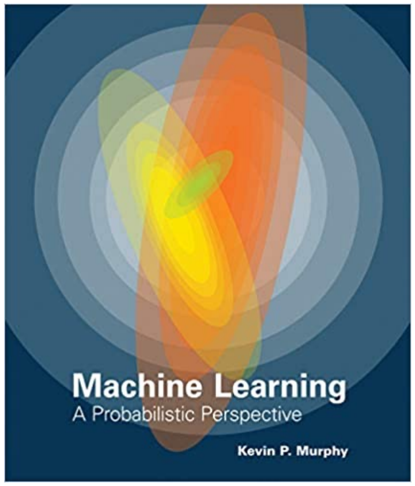
- A bit outdated with recent trends, but still has interesting discussion (and easy to read).
- A subfield of **Artificial Intelligence** – you want to perform nontrivial, smart tasks. The difference from the traditional AI is "**how**" you build a computer program to do it.

Textbooks

We will use a more recent textbook for readings

Takes a ***probabilistic approach*** *to machine learning*

Consistent with the goals of data science in this class



Murphy, K. "Machine Learning: A Probabilistic Perspective." MIT press, 2012

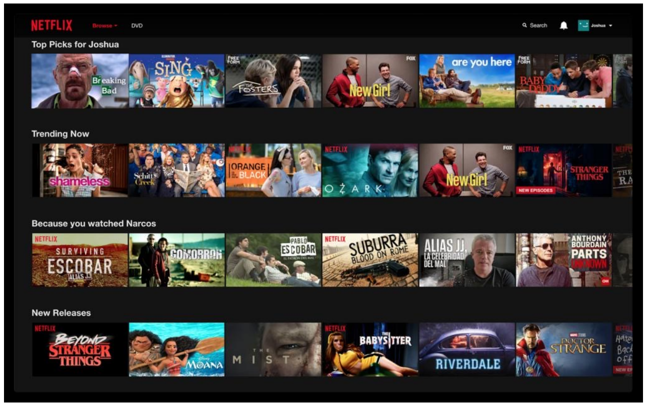
(UA Library)

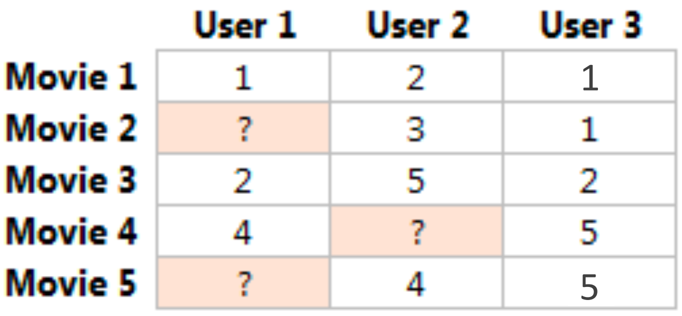
AI Task 1: Image classification

- Predefined categories: C = {cat, dog, lion, …}
- Given an image, classify it as one of the categories $c \in C$ with the highest accuracy.
- **Use**: sorting/searching images by category, medical imaging, object identification, traffic control, categorizing types of stars/events in the Universe (images taken from large surveying telescopes)

AI Task 2: Recommender systems

- Predict how user would rate a movie
- **Use**: For each user, pick an unwatched movie with high predicted ratings. (Youtube, Netflix, Amazon, etc.)
- **Idea**: compute user-user similarity or movie-movie similarity, then compute a weighted average.



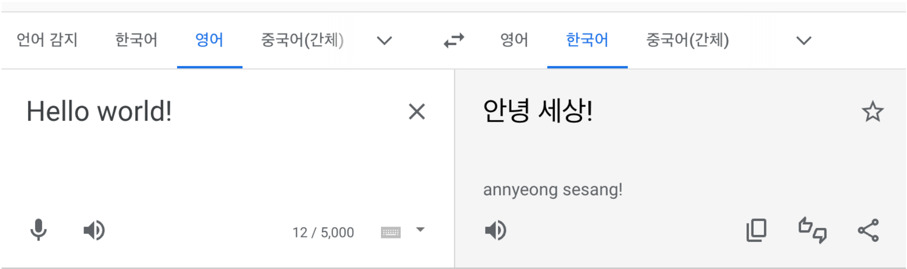


	User 1	User 2	User 3
Movie 1	1	2	1
Movie 2	?	3	1
Movie 3	2	5	2
Movie 4	4	?	5
Movie 5	?	4	5

“collaborative filtering”

AI Task 3: Machine translation

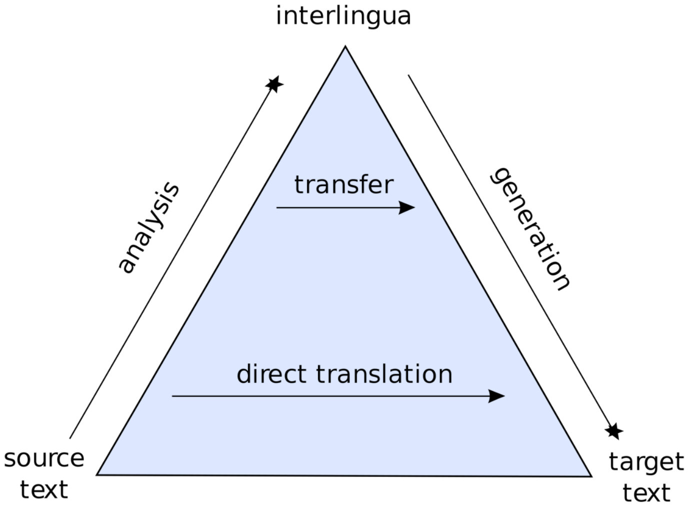
- No need to explain how useful it is.
- **Task**: 1) Transform a sentence to the interlingual language (analysis) and 2) create a sentence with another language with the same meaning, with appropriate grammar structure (generation).



Hello world!

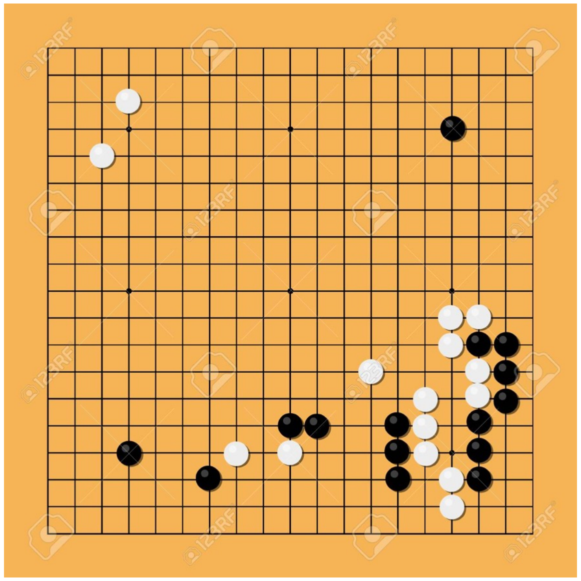
안녕 세상!

annyeong sesang!



AI Task 4: Board game

- Predict win probability of a move in a given game state (e.g., AlphaGo)
- Traditionally considered as a “very smart” task to perform.



Q: how will it be useful for us, though?

- **Use**: From the AI Go player, you can do practice play or even learn from it.
  - Now it’s a major trend in the field of Go

- **Potential use**: Board game (e.g., Catan) design, better AI
- Deeply related to robot AI and autonomous driving
  - Predict the future of your move

Traditional AI vs Machine Learning (ML)

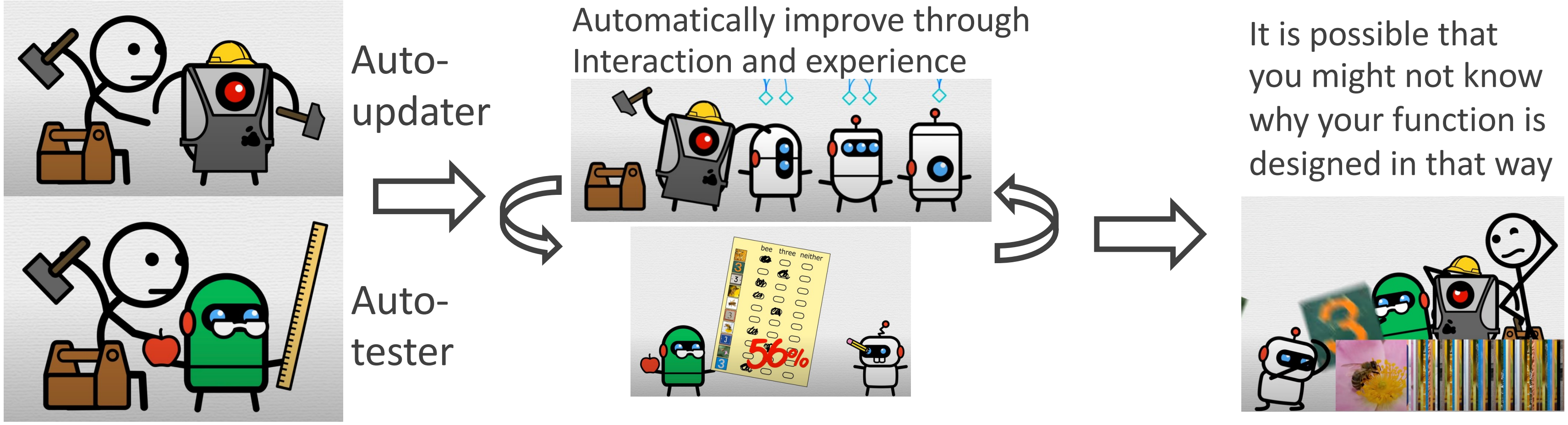
- **Traditional AI**: *you* encode the knowledge (e.g., logic statements/rules), and the *machine* executes it.
  - e.g., if there is feather-like texture with two eyes and a beak, classify it as a bird.
  - Advancements in automated '**inference**' like "if a -> b and b-> c, then a-> c". => 'expert system'
- **ML**: Given a set of input and output pairs (e.g., animal picture + label), and train a **function** (a set of logical statements / a neural network) that maps the input to the output accurately.
  - As the "big data" era comes, data is abundant => turns out, better than systems based on hand-coded domain knowledge!
  - "statistical" approach // data-driven approach

> *"Every time I fire a linguist, the performance of the speech recognizer goes up."*
> – *1988,* Frederick Jelinek, a Czech-American researcher in information theory & speech recognition.

Traditional AI vs Machine Learning (ML)

- Traditional AI – watchmaker
  - You encode your knowledge (springs and parts) directly
  - You understand why those parts are necessary.

- Machine Learning (ML) – one example (from https://www.youtube.com/watch?v=R9OHn5ZF4Uo)



Auto-updater

Auto-tester

Automatically improve through Interaction and experience

It is possible that you might not know why your function is designed in that way

Overview of ML Methods

Supervised Learning

- Provide *training* data consisting of input-output pairs and learn mapping
- E.g., Spam prediction, object detection or image classification, machine translation, etc.

Unsupervised learning

- **No predefined categories**. Finds patterns in the data without the help of labels (outputs)
- E.g., clustering, dimensionality reduction, target tracking, image segmentation, etc.

Reinforcement learning ⇐ **We won’t cover this**

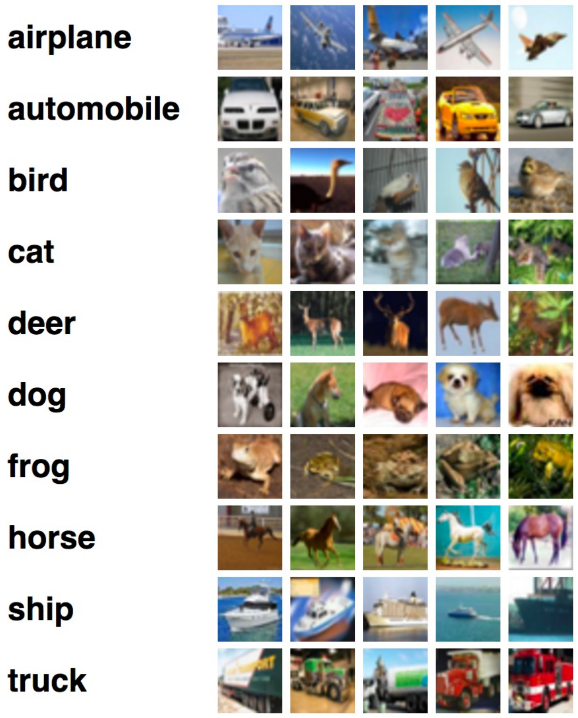
- The environment interacts with your action, transferring you to different states.
- E.g., autonomous driving, robot AI, recommendation system

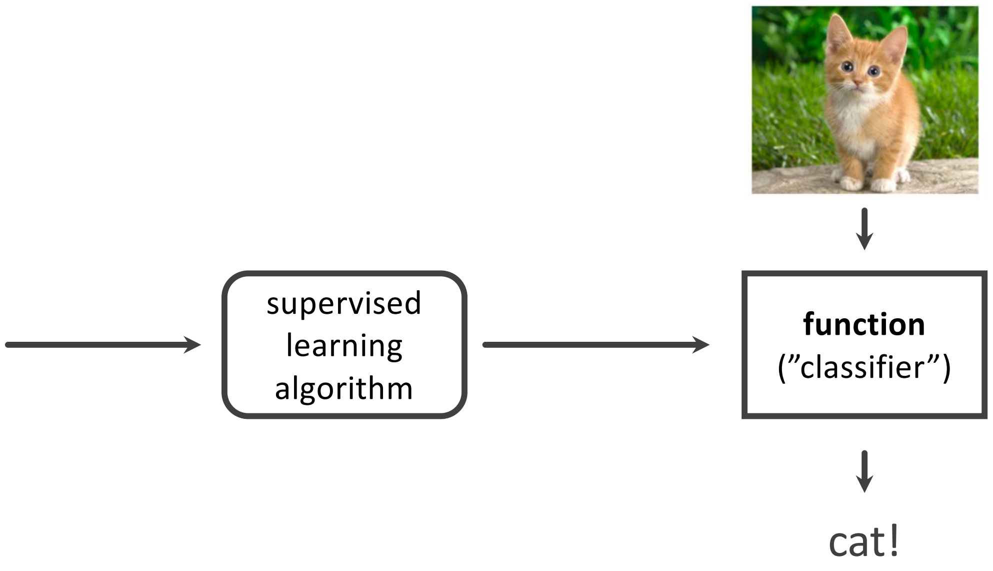
Supervised Learning

Basic setting: Supervised learning

example = data point
labeled = categorized

- Train data: dataset comprised of *labeled examples*: a pair of (input, label)





supervised learning algorithm

function (”classifier”)

cat!

training

testing

Example function 1: Decision tree

```
Task: predict the 5-star rating of a movie by a user

If age >= 60 then
   if genre = western then
      return 4.3
   else if release date > 1998 then
      return 2.5
   else ...
   ...
   end if
else if age < 60 then
...
end if
```

training:

- determine the shape of the tree
- which condition to have at each node
- what to output from each leaf node

Example function 2: Linear

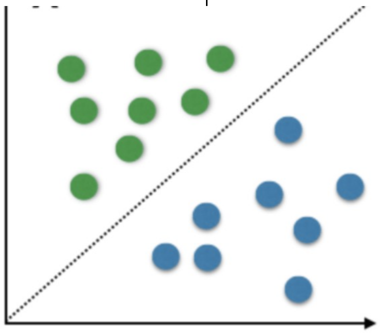
```
Task: Image classification

Let $x$ be a set of pixel values of a picture (30x30) =>
900-dimensional vector $x$.

If $\boxed{0.124 \cdot x_1 - 2.5 \cdot x_2 + \cdots + 2.31 \cdot x_{900} - 2.12} \geq 0$ then
   return cat
else
   return dog
end
```

called feature vector

“linear combination”/”inner product”

training:

- determine the coefficients & threshold

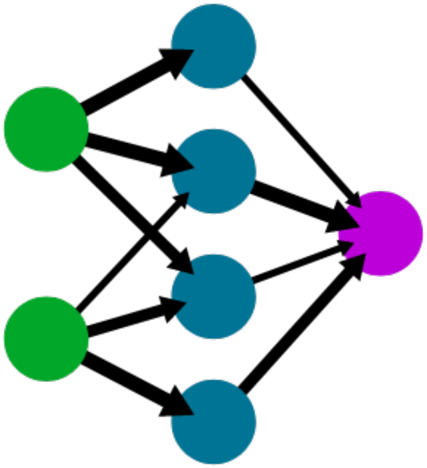


E.g., in 2d space, it induces a linear decision boundary:

$0.5 \cdot x_1 - 2.5 \cdot x_2 > 4.3$

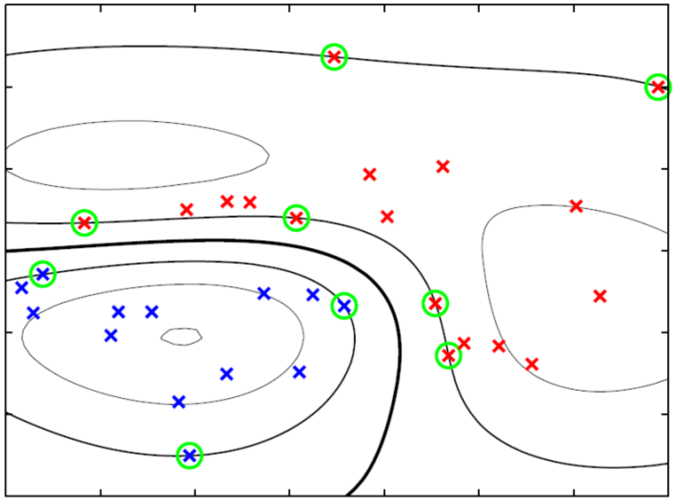
Example function 3: Nonlinear

Neural network



(stacked **linear** models with nonlinear **activation functions**)

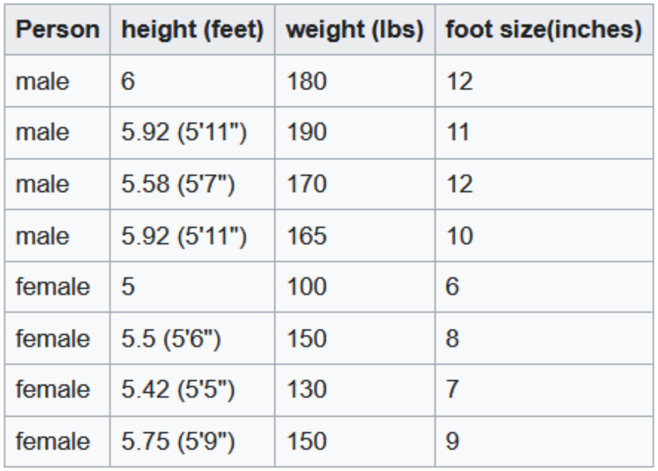
Support Vector Machine



(**linear** in the induced feature space)

Example: Naïve Bayes Classifier

Training Data:



Person	height (feet)	weight (lbs)	foot size(inches)
male	6	180	12
male	5.92 (5'11")	190	11
male	5.58 (5'7")	170	12
male	5.92 (5'11")	165	10
female	5	100	6
female	5.5 (5'6")	150	8
female	5.42 (5'5")	130	7
female	5.75 (5'9")	150	9

Features

Task: Observe feature vector $x = (x_1, \dots, x_n)$ and predict class label $y \in \{1, \dots, C\}$

Model: Treat features as *conditionally independent*, given class label:

$$p(x, y) = p(y) \prod_{i=1}^{n} p(x_i | y)$$

Doesn't capture correlation among features, but is easier to learn.

Classification: Bayesian model so classify by posterior,

$$p(y = c\ | x) = \frac{p(C = k) p(x | y = c)}{p(x)}$$

Supervised learning: Types of prediction problems

Binary classification: Choose between 2 classes

- Given an email, is it spam or not? (or the probability of it being a spam)

Multi-class classification: more than 2 categories.

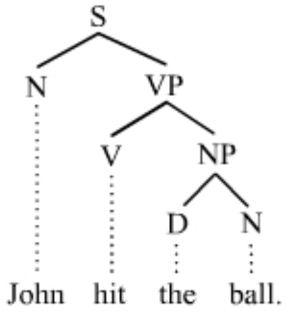
- Image classification with 1000 categories. (cat, dog, airplane, car, computer, …)

Regression: the label is real-valued (e.g., price)

- Say I am going to visit Italy next month. Given the price trends in the past, what would be the price given (the # of days before the departure, day of week)?
- Predict the stocks/bitcoin price in the future

Structured output prediction: more than just a number

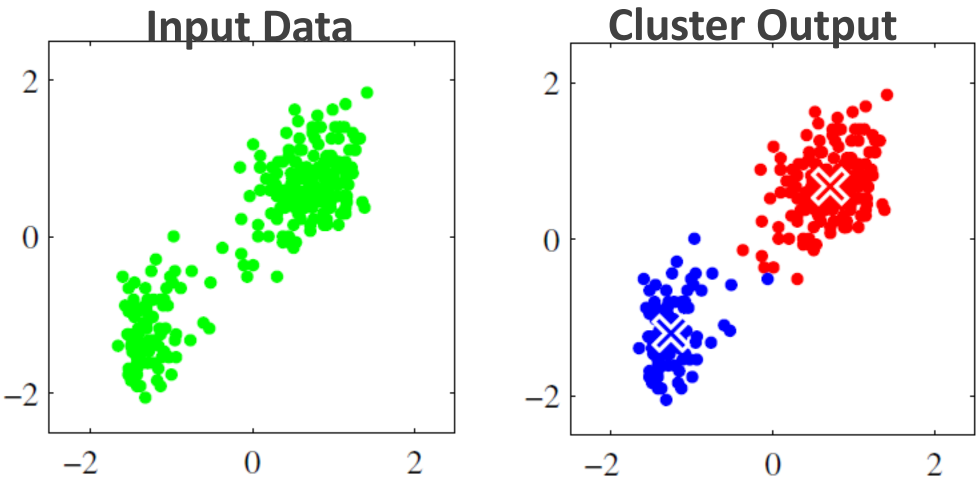
- Given a sentence, what is its grammatical parse tree?



Unsupervised Learning

Example: Clustering

Identify groups (clusters) of similar data



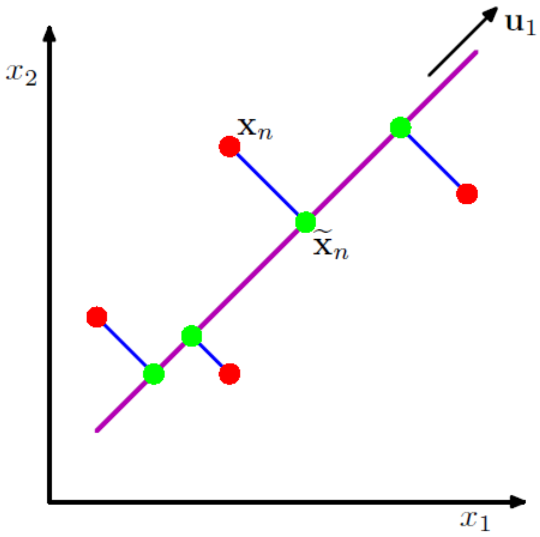
Useful for interpreting large datasets

Clusters are assigned arbitrary labels (e.g. 1, 2, …, K).
=> afterwards, you may look at the data and name each group.

Common clustering algorithms: K-means, Expectation Maximization (EM)

Example: Principal Component Analysis (PCA)

Reduce dimension of high-dimensional data using linear projection



Identify directions of **maximum variation** in the data by computing ***eigenvectors***

Easier explanation: Identify important directions

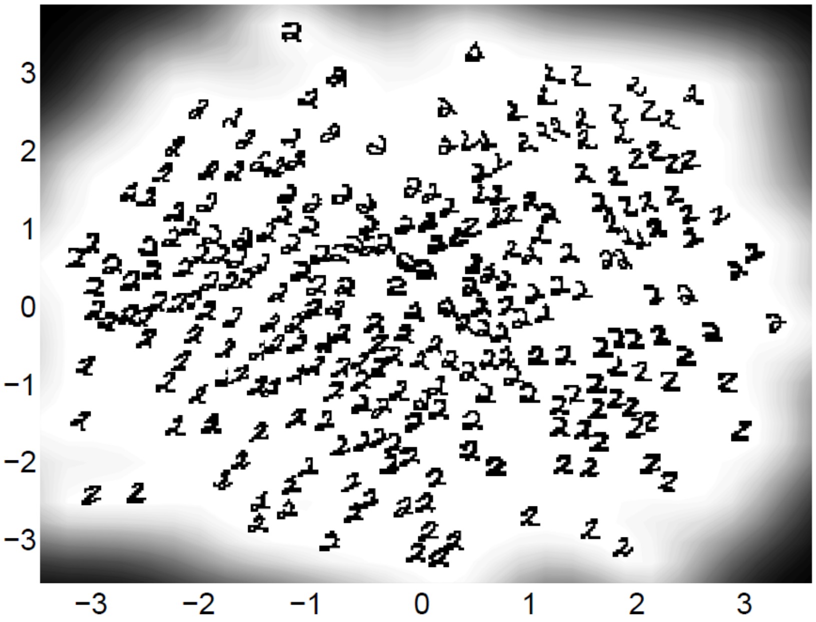
Linear projection onto K-dimensional subspace spanned by top K eigenvalues

Can be used for visualization (project to 2D) or for compressing images.

Source: Bishop, C. PRML

Example: Principal Component Analysis (PCA)

Reduce dimension of high-dimensional data using linear projection



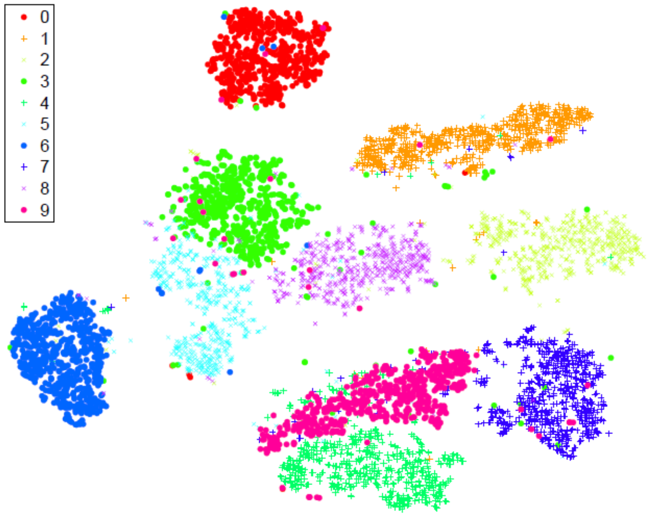
Source: Lawrence, N. (2005)

Example for modeling / visualizing handwritten digits

Each digit is a black/white image with 28x28 pixels (784 dimensions) projected down to 2D

Example: Nonlinear Dimensionality Reduction

t-SNE



Nonlinear reduction can (potentially) amplify clustering properties

t-Distributed Stochastic Neighbor Embedding (t-SNE) Models similarity between data as a t distribution and strives to find projection that preserves similarity.

Example: Generative models

We won't cover this

- AI image generators
- It is hard to define how 'good' the generated image is.
    - How can we explain the 'painting style' to computers? Mostly impossible… → Unsupervised!

Summary

- **Supervised Learning** - Training data consist of inputs and outputs
  - Classification, regression, translation, …

- **Unsupervised Learning** – Training data only contain inputs
  - Clustering, dimensionality reduction, segmentation, …

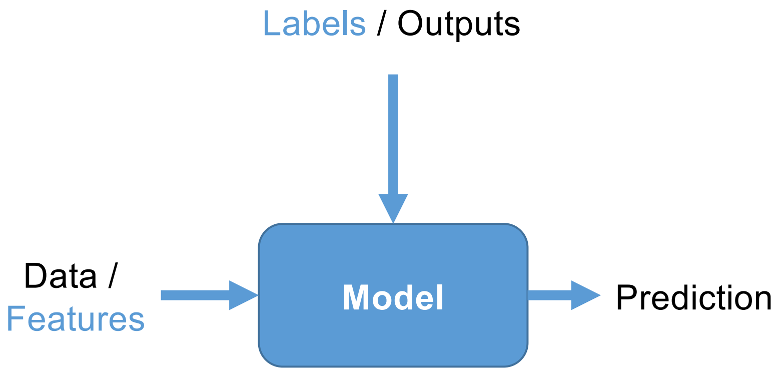
- **Linear** models generate output as a *linear combination* of inputs,
  - E.g. $y = w_1 x_1 + w_2 x_2 + \ldots + w_d x_d$
  - PCA, linear regression, etc.

- **Nonlinear** models fit an arbitrary nonlinear function to map inputs-outputs
  - Neural networks, support vector machine, nonlinear dimensionality reduction

Training Machine Learning Models

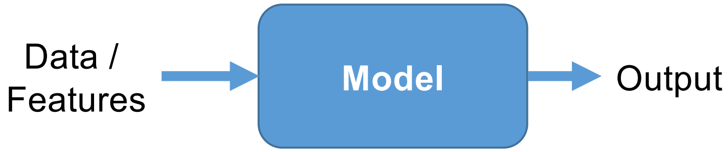
Supervised Learning

works with labeled data



Unsupervised Learning

works with unlabeled data



ML models distinguished by a number of factors

- Number of parameters needed (parametric / nonparametric)
- Whether they model uncertainty (probabilistic / nonprababilistic)
- Do they model the data generation process? (generative / discriminative)

CSC380: Principles of Data Science

Basics of Predictive Modeling and Classification 1:
Decision Tree

Kyoungseok Jang

Decision Trees

figures/examples from “A Course in Machine Learning” by Hal Daume III http://ciml.info/

Majority Vote Classifier

The most basic classifier you can think of.

How to train:

- Given: A (train) dataset with m data points $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ with C classes.
- Compute the most common class c^* in the dataset.

$$c^* = \arg \max_{c \in \{1,\ldots,C\}} \sum_{i=1}^{m} \mathbf{I}\{y^{(i)} = c\}$$

- Output a classifier $f(x) = c^*$.

Example:
Data: m=10
$x^{(i)}$: images of cats and dogs
$y^{(i)}$: label (cat/dog)
Suppose that there are 6 dogs and 4 cats.
After 'training', your classifier always outputs 'dog', even without looking at the input.

Stupid enough classifier! Always try to beat this classifier.

Often, state-of-the-art ML algorithms perform barely better than the majority vote classifier..

⇒ happens when there is no association between features and labels in the dataset

Train set accuracy

- Suppose the ML algorithm has trained a function f using the dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ where $x^{(i)}$ is input and $y^{(i)}$ is label.
- Train set accuracy:

$$\widehat{acc}(f) := \frac{1}{m} \sum_{i=1}^{m} \mathbf{I}\{f(x^{(i)}) = y^{(i)}\}$$

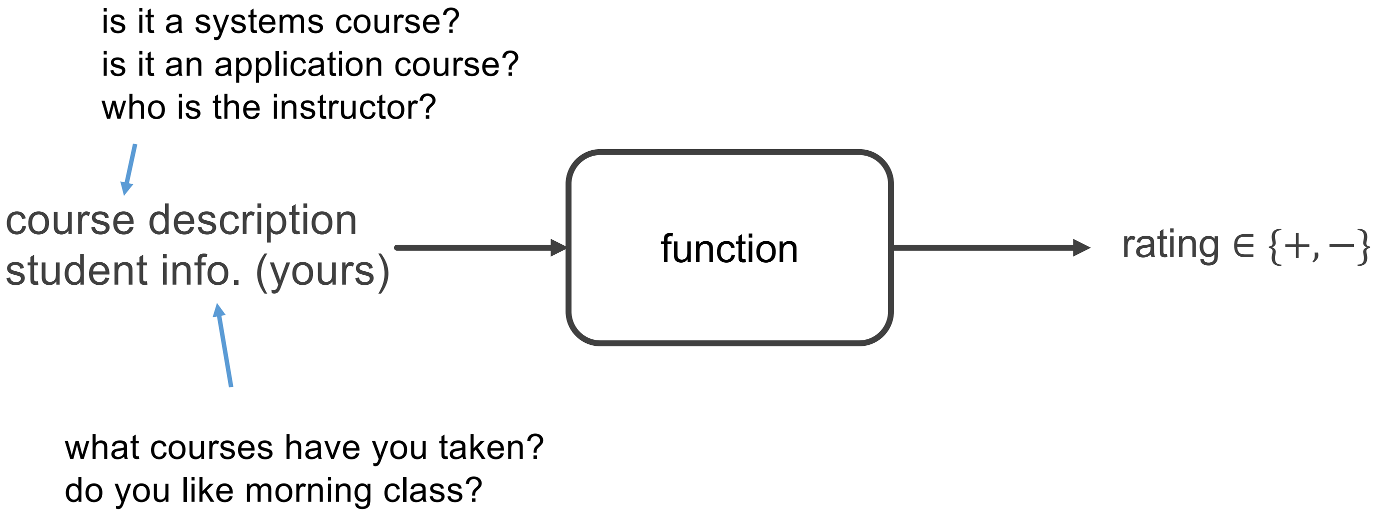
It is the number of times the function got the answer right divided by m.

- Q: We have 100 data points (images) with 5 cats, 80 dogs, and 15 lions. What is the train set accuracy of the majority vote classifier?

.80

Decision tree (example: course recommendation)

- Build software: recommend a set of courses for you
  - More precisely, given a course, predict its rating



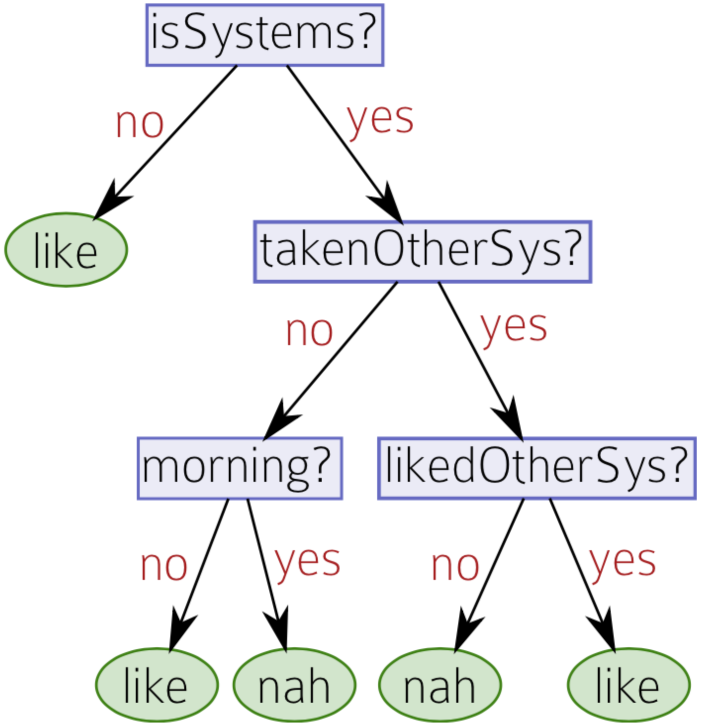
is it a systems course?
is it an application course?
who is the instructor?

course description
student info. (yours)

function

rating ∈ {+, −}

what courses have you taken?
do you like morning class?



Wouldn’t it be nice to construct such a tree automatically by a computer algorithm?

Wouldn’t it be nice if it accurately predicts?

You can, if you have data!

HasTakenPrereqs (=: Prereq)

HasTakenACourseFromTheSameLecturer (=: Lecturer)

HasLabs

Rating	Easy?	~~AI?~~	~~Sys?~~	~~Thy?~~	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

consider it to be ‘like’

consider it to be ‘dislike’

For example, this table is data D.
Each row is a course you’ve rated.
$x^{(i)}$ is a sequence of 5 yes/no (d=5) for i-th course.
$y^{(i)}$ is the sign of the rating for i-th course.

Define the data $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$

$\in \{y, n\}^d$

$\in \{+, -\}$

Each dimension of $x^{(i)}$ is called a **feature**.
$x^{(i)}$ is called a **feature vector**.

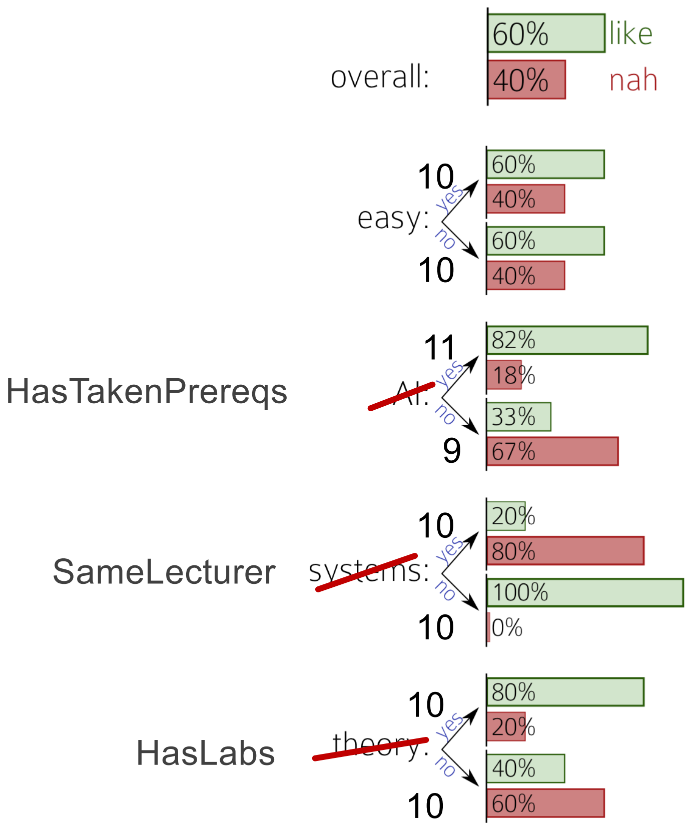
How to Train a Tree

- Main principle: Find a tree that has a high train set accuracy

  $$\widehat{acc}(f) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{I}\{f(x^{(i)}) = y^{(i)}\}$$

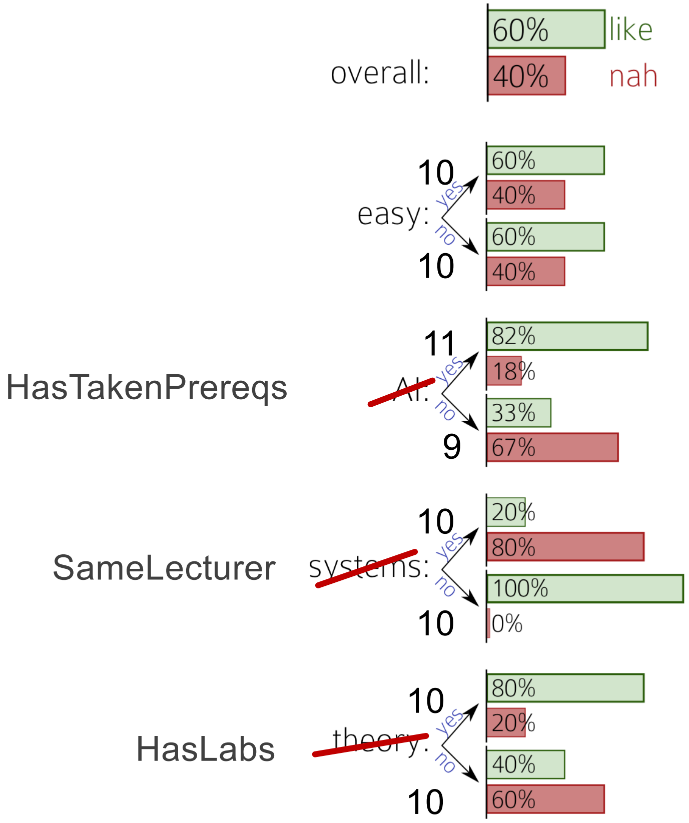
- This is essentially the main principle governing pretty much all the machine learning algorithms!
  - "Empirical risk minimization" principle
    (empirical risk := 1 – train_accuracy)

How to construct a tree



		Prereqs	Lecturer	HasLabs	
Rating	**Easy?**	~~**AI?**~~	~~**Sys?**~~	~~**Thy?**~~	**Morning?**
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

How to construct a tree



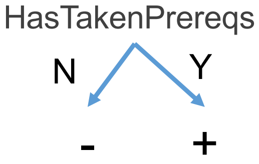
Baseline: majority vote classifer

Major

(+)

Q: What is the train set accuracy? 0.60

Suppose we place the node HasTakenPrereqs at the root.
Set the prediction at each leaf node as the majority vote.

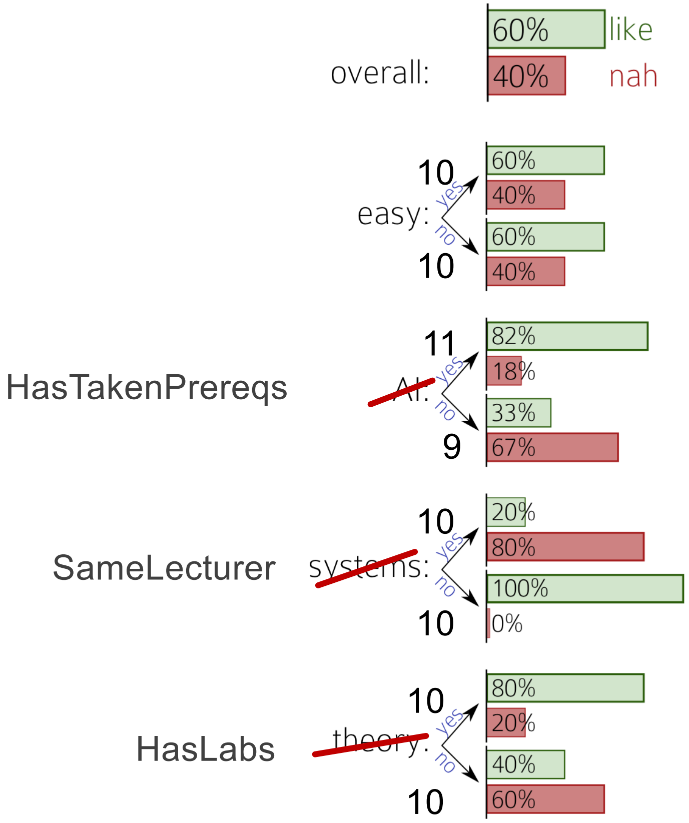


What is the train set accuracy now?

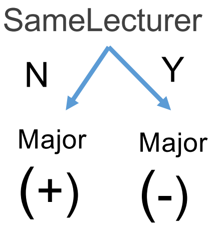
$$\frac{9}{20}\cdot\frac{6}{9}+\frac{11}{20}\cdot\frac{9}{11}=\frac{15}{20}=0.75$$

improved!

How to construct a tree



Suppose placing the node SameLecturer at the root.


SameLecturer

N

Y

Major

(+)

Major

(-)

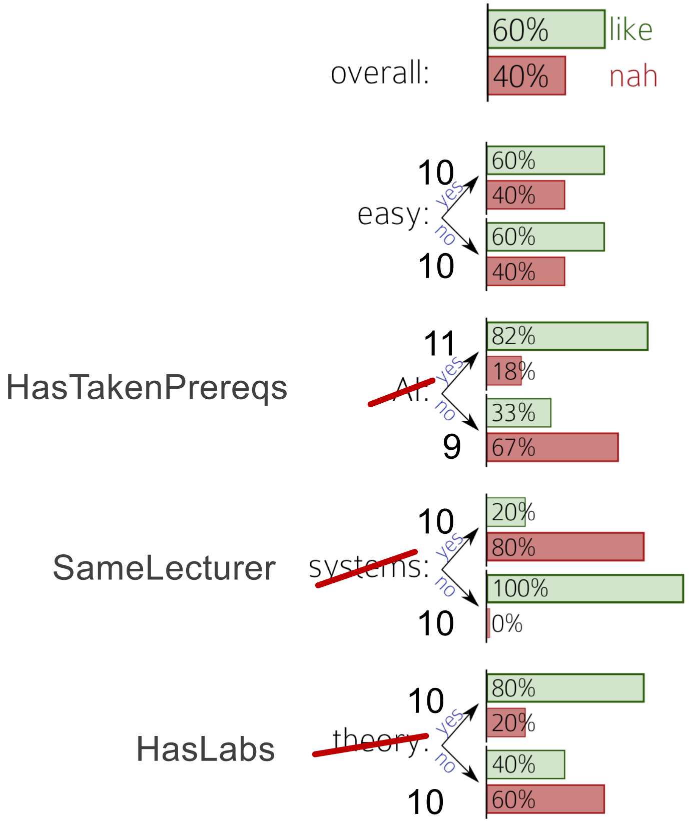
What is the train set accuracy now?

$$\frac{10}{20}\cdot\frac{10}{10}+\frac{10}{20}\cdot\frac{8}{10}=\frac{18}{20}=0.9$$ even better!

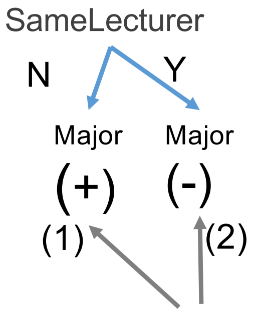
What would you do to build a depth-1 tree?

try out each feature and choose the one that leads to the largest accuracy!

How to construct a tree



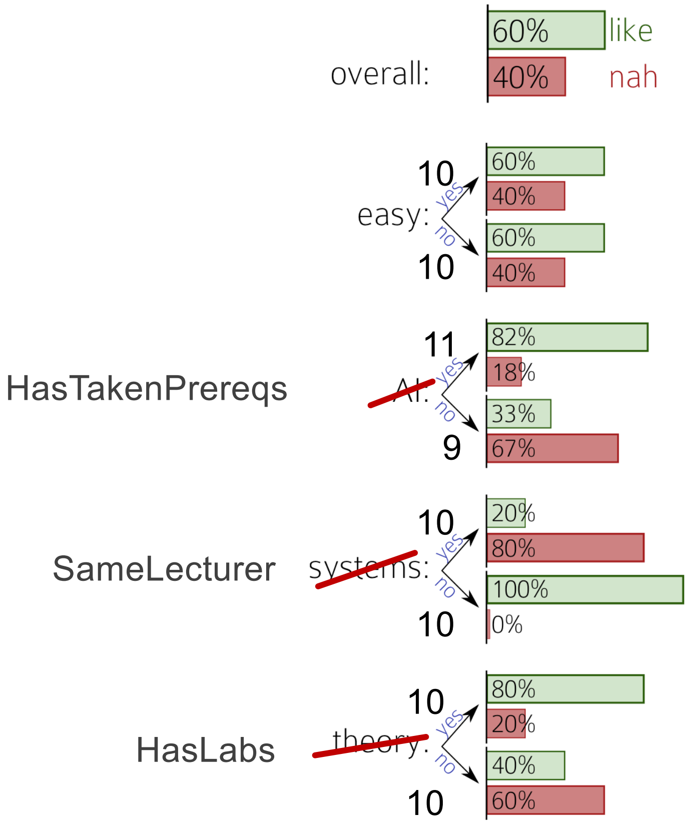
What about depth 2?

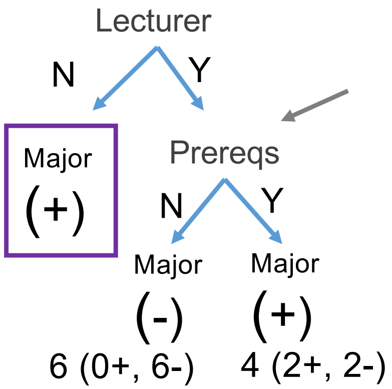


Which nodes to put at each leaf node?

Focus on (2). Try placing HasTakenPrereqs

How to construct a tree





Q: How many training data points fall here? 10

Q: How many training data points arrive at these two leaves? How many for each label?

Q: what prediction should we use for each leaf?

Q: What is the train set accuracy, conditioning on SameLecturer=Y?

'local' train set accuracy

$$\frac{6}{10}\cdot\frac{6}{6}+\frac{4}{10}\cdot\frac{2}{4}=\frac{8}{10}$$

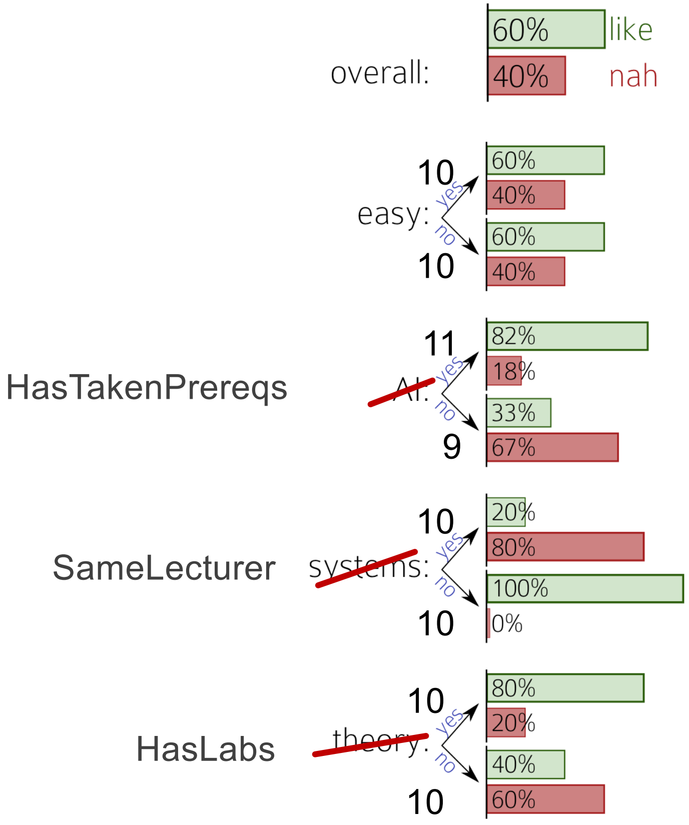
Try all the other nodes and pick the one with the largest acc.!

Then, repeat the same for SameLecturer=N branch!

=> but this has 1 local train set acc. So leave it be!

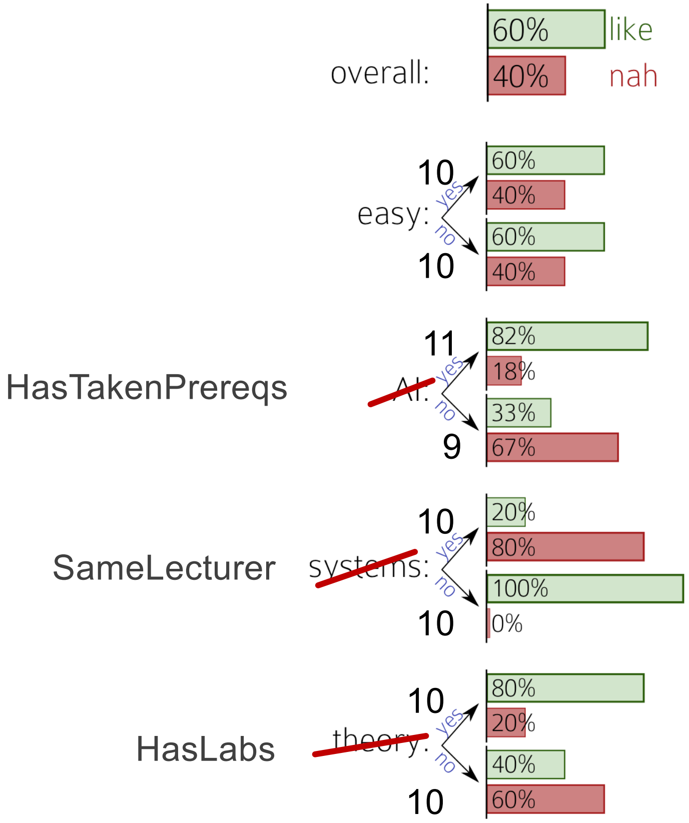
Move onto expanding nodes at depth 2!

How to construct a tree



		Prereqs	Lecturer	HasLabs	
Rating	**Easy?**	**~~AI?~~**	**~~Sys?~~**	**~~Thy?~~**	**Morning?**
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

How to construct a tree



Overall idea:

1. Set the root node as a leaf node.
2. Grab a leaf node for which its ‘local’ train accuracy is not 1.
3. Find a feature that maximizes the ’local’ train accuracy and replace the leaf node with a node with that feature; add leaf nodes and set their predictions by majority vote.
4. Repeat 2-3.

Algorithm 1 DecisionTreeTrain(*data*, *remaining features*)

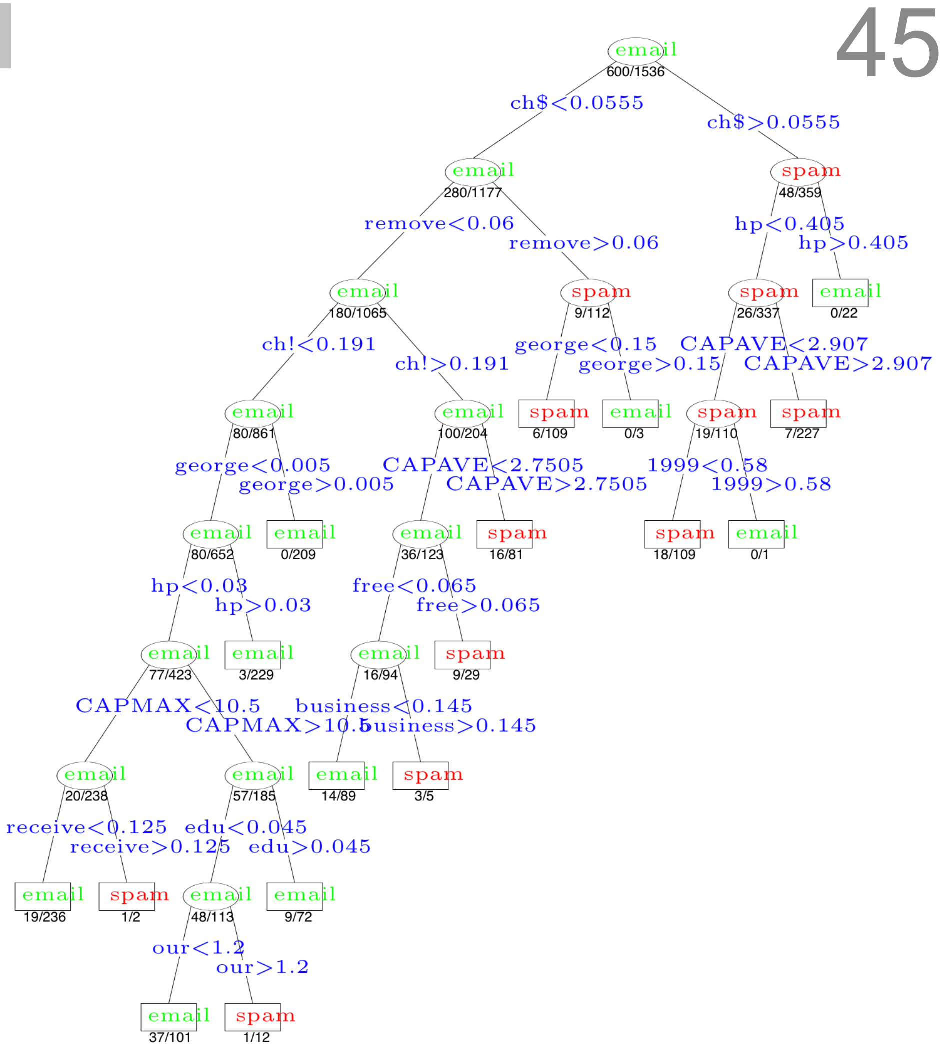
```
guess ← most frequent answer in data                      // default answer for this data
if the labels in data are unambiguous then                    <= i.e., all data points have the same label
    return Leaf(guess)                                    // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                    // base case: cannot split further
else                                                      // we need to query more features
    for all $f \in$ remaining features do                     <= there is no point in adding a feature
        NO ← the subset of data on which $f$=no                      that appeared in its parent!
        YES ← the subset of data on which $f$=yes
        score[$f$] ← ( # of majority vote answers in NO                       <= answer = label
                     + # of majority vote answers in YES ) / size(data)

    end for
    $f$ ← the feature with maximal score($f$)
    NO ← the subset of data on which $f$=no
    YES ← the subset of data on which $f$=yes
    left ← DecisionTreeTrain(NO, remaining features $\setminus \{f\}$)
    right ← DecisionTreeTrain(YES, remaining features $\setminus \{f\}$)
    return Node($f$, left, right)
end if
```

Algorithm 2 DecisionTreeTest(*tree*, *test point*)

```
if tree is of the form Leaf(guess) then
    return guess
else if tree is of the form Node(f, left, right) then
    if $f = no$ in test point then
        return DecisionTreeTest(left, test point)
    else
        return DecisionTreeTest(right, test point)
    end if
end if
```

Example: spam filtering I

- Spam dataset
- 4601 email messages, about 39% are spam
- Classify message by spam and not-spam
- 57 features
  - 48 are of the form “percentage of email words that is (WORD)”
  - 6 are of the form “percentage of email characters is (CHAR)”
  - 3 other features (e.g., “longest sequence of all-caps”)
- Final tree after pruning has 17 leaves, 9.3% test error rate



Background: Train Error vs Test Error

Error := 1 – accuracy.

Suppose we have trained a function $\hat{f}$ on $D = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{m}$ using a supervised learning algorithm.

- Train error: Evaluate on D.

$$\widehat{err}_D(f) := \frac{1}{|D|} \sum_{(x,y)\in D} \mathbf{I}\{f(x) \neq y\}$$

- Test error: Evaluate on $D' = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{m'}$ not used for training.
  - It can be possible that our function just 'memorized' the training data and doesn't do well in real life. (overfitting)

Q: Choose one:
(1) train error $\geq$ test error (2) train error $\approx$ test error (3) train error $\leq$ test error

Background: Workflow of Training a Classifier

Standard practice:

- Given a data set D, split it into train set D_{train} and D_{test}
  - large data: 90-10 ratio
  - medium data: 80-20 ratio
  - small data: 70-30 ratio

(these are guidelines only)

- Train on D_{train} and evaluate error rate on D_{test}. You trust that D_{test} will be the performance when you deploy the trained classifier.

Discussion: What would be reasonable logics behind such a trust?

Computer Science

CSC380: Principles of Data Science

Basics of Predictive Modeling and Classification 2

Decision Trees / k-Nearest Neighborhood

Kyoungseok Jang

Announcements: Midterm

- Final curving: you will recover 66% of the score you lost.
  - E.g.) If your original score was 40, your curved score will be $40 + (100 - 40) \times \frac{2}{3} = 80$
  - A bit more beneficial for the students who didn't have basic knowledge in probability and statistics.
  - New average: 83.3

- Final exam: I will spend a lecture for the final review, and I will try to 'describe' the problems more explicitly.
  - I will reuse several midterm problems with a bit of variation.

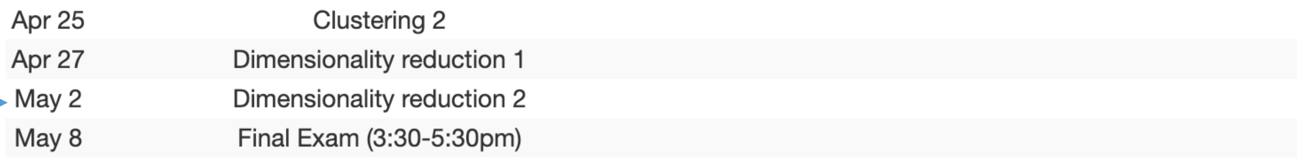
Announcement: Midterm

- Regrade request
  - Problem 7(4): We decided to give everyone the score. Please check your answer, and if your answer was ‘False’, please send us the regrade request.
  - For the student who used the back side of the paper for your answer, please let us know.

Announcement: Prerequisite

- Some students asked me about prerequisites.
  - Especially about the dimensionality reduction part
  - We will not evaluate you based on those prerequisites
    - We will not ask you like, how to calculate the eigenvectors or eigenvalues on your exam, or in your final project.
    - We will teach you the basic knowledge to understand. (E.g. inner product)
    - We will introduce you to some scipy functions for eigenvector computations.
    - (I am not sure whether we can cover the dimensionality reduction part)

I guess I will spend this lecture for the final review…



Apr 25	Clustering 2
Apr 27	Dimensionality reduction 1
May 2	Dimensionality reduction 2
May 8	Final Exam (3:30-5:30pm)

Announcement: Homeworks

- There was a gap between our scheduled progress and our current progress
  - For example, we should have finished the ‘sampling bias’ part before the midterm.
  - Therefore, now HW5 was posted too early.
    - We will learn k-Nearest Neighborhood today and Naïve-Bayes Classifier on Thursday. (both of them were included in HW5…)

- Therefore, we will extend the due date for HW5 to Mar. 31st.

Announcement: TA

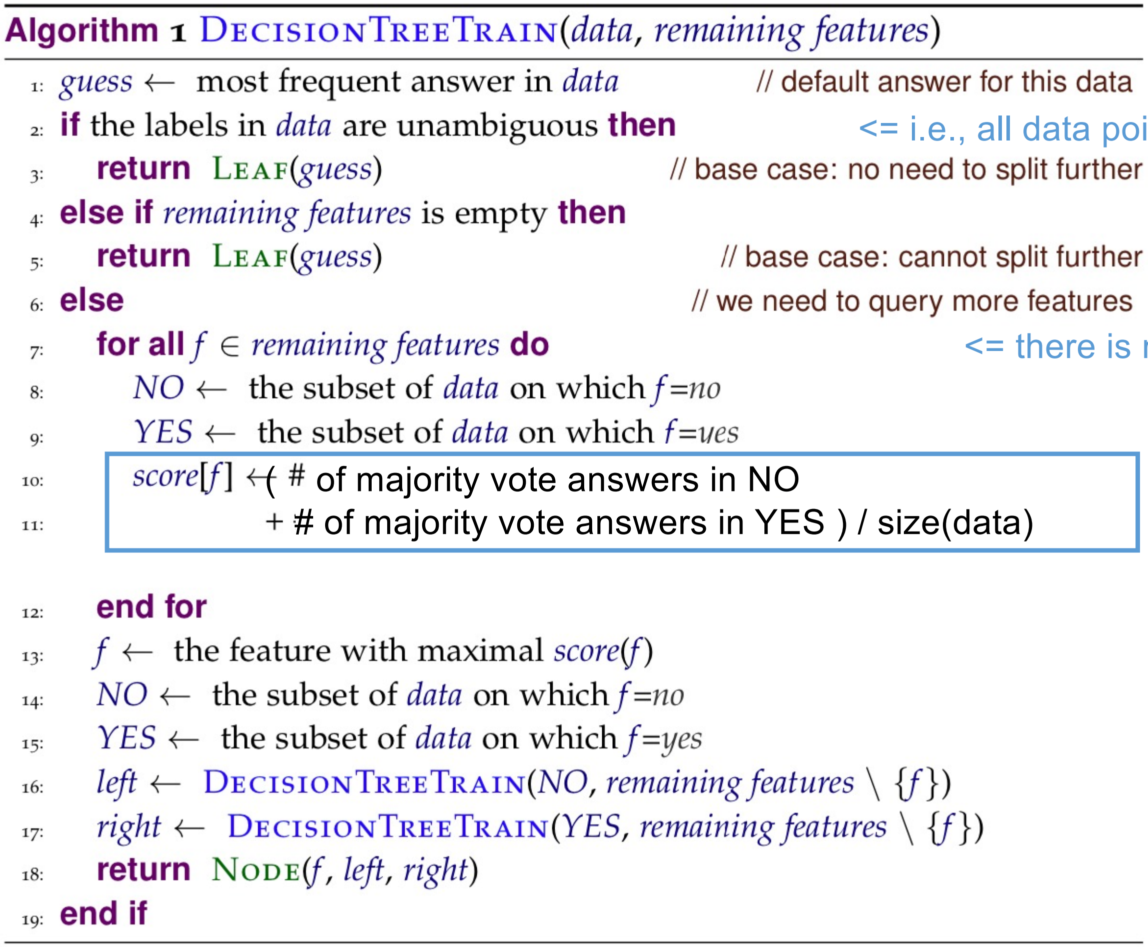
- Due to his personal circumstances, TA Saiful will no longer be in charge of this class.
- Temporarily, we will not be able to provide the following services.
  - His office hour: Wed
  - His piazza hour: Wed/ Thu/ Fri

Outline

- Decision Tree
  - Review
  - Variations - Different criterions
  - Different types of features / labels
  - Regression
  - Pruning

- K-Nearest Neighborhood
  - Main concepts
  - Feature scaling
  - Variations / Issues

Review: Decision Tree method



Algorithm 1 DecisionTreeTrain(*data*, *remaining features*)

```
guess ← most frequent answer in data                    // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                                   // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                   // base case: cannot split further
else                                                     // we need to query more features
    for all $f \in$ remaining features do
        NO ← the subset of data on which $f$=no
        YES ← the subset of data on which $f$=yes
        score[$f$] ←( # of majority vote answers in NO
                    + # of majority vote answers in YES ) / size(data)

    end for
    $f$ ← the feature with maximal score($f$)
    NO ← the subset of data on which $f$=no
    YES ← the subset of data on which $f$=yes
    left ← DecisionTreeTrain(NO, remaining features \ {$f$})
    right ← DecisionTreeTrain(YES, remaining features \ {$f$})
    return Node($f$, left, right)
end if
```

<= i.e., all data points have the same label

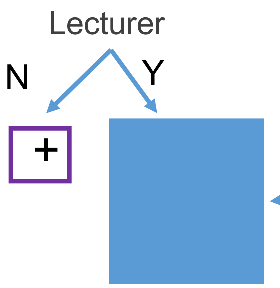
<= there is no point in adding a feature that appeared in its parent!

<= score[f]='local' train set acc.

- Main question: How to calculate the 'local' train set accuracy? (score[f])

Review: Decision tree - How to construct a tree

		Prereqs	Lecturer	HasLabs	
Rating	Easy?	~~AI?~~	~~Sys?~~	~~Thy?~~	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y



Suppose that we already have this structure
And need to construct more on the RHS.

According to the algorithm, we need to calculate score[f], which means the 'local' train set accuracy for each feature f.

Suppose that now our f is 'Prereqs'

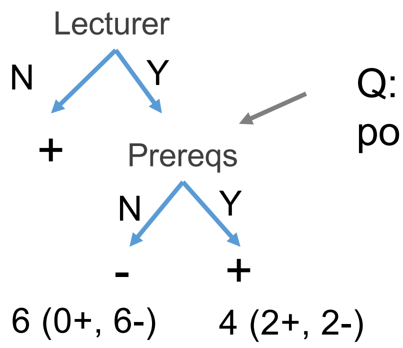
Main trick for calculating score[f]:

$$\widehat{acc}(f) = \frac{1}{m}\sum_{i=1}^{m} \mathbf{I}\{f(x^{(i)}) = y^{(i)}\}$$

$$= \frac{1}{m}(\Sigma_{i\in YES}\, \mathbf{I}\{f(x^{(i)}) = y^{(i)}\}+\Sigma_{i\in NO}\, \mathbf{I}\{f(x^{(i)}) = y^{(i)}\})$$

Review: Decision tree - How to construct a tree

		Prereqs	Lecturer	HasLabs	
Rating	Easy?	~~AI?~~	~~Sys?~~	~~Thy?~~	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y



Q: How many training data points fall here? 10

Q: How many training data points arrive at these two leaves? How many for each label?

Q: what prediction should we use for each leaf?
A: Majority vote for each leaf

Q: How many samples will your current function outputs the 'correct' rating (sign) for each leaf?

Hint: Majority, since it is based on the majority vote

6 for left
2 for right

Q: What is the train set accuracy, conditioning on SameLecturer=Y?

'local' train set accuracy $\frac{1}{10}(6+2)=\frac{8}{10}$ or $(\frac{6}{10}\frac{6}{6}+\frac{4}{10}\frac{2}{4})=\frac{8}{10}$

Sum of (fraction of sub group * fraction of correct answer in sub group)

Variations

- Recall the previous ‘score[f]’
- $\frac{6}{10}\frac{6}{6}+\frac{4}{10}\frac{2}{4}$
- Sum of (fraction of subgroup * fraction of correct answer in subgroup)
- What if we change it to Sum of (fraction of a subgroup * some function on that subgroup)

Variations

Notions of uncertainty: binary case ($\mathcal{Y} = \{0,1\}$)

Suppose in a set of examples $S \subseteq \mathcal{X} \times \{0,1\}$, a p fraction are labeled as 1

What we did: Majority fraction = 1 – minority fraction

1. **Classification error**: (red)

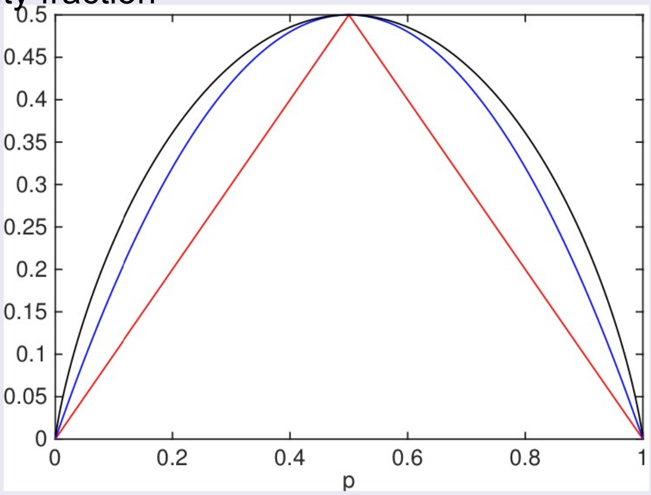
$$u(S) := \min\{p, 1-p\}$$

2. **Gini index**: (blue)

$$u(S) := 2p(1-p)$$

3. **Entropy**: (black)

$$u(S) := p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$$



Gini index and entropy (after some rescaling) are concave upper-bounds on classification error

Let q is the fraction of data points with feature=Y.

Modification:
Set score[f] as
$q \cdot (-u(YES)) + (1-q) \cdot (-u(NO))$

Set score[f] as $-u$ where u is one of these tree measure.

Using classification error is equivalent to using the accuracy.

For example,
in the previous example,
Lecturer=Y Prereqs=Y has
p=2/4

Check: When u is the classification error, $q \cdot \big(-u(YES)\big) + (1-q) \cdot \big(-u(NO)\big) = (score\ we\ knew) \ - 1$

Decision tree – different types of features

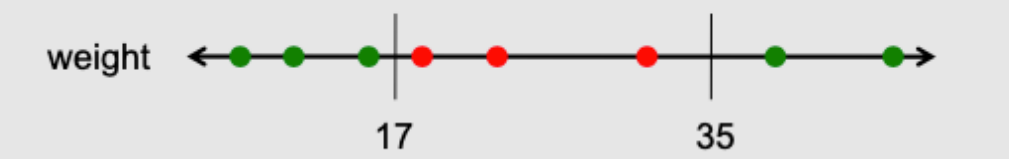
- Binary
- Categorical: values in $\{1, \ldots, C\}$ e.g., occupation, blood type
  - Option 1: Instead of 2 children, have C children.
  - Option 2: Derive C features of the form "feature=c?" for every $c \in C$.

    ↑ binary features!

  Q: How about features of the form "feature$\in C'$" for every $C' \subset C$?

  computational complexity ↑ Because there are 2^C subsets!

- Real value e.g., weight, age
  - Sort the values.
  - Find the **breakpoints**: For every two adjacent points with opposite labels, compute the midpoint.
  - Derive features like "weight ≤ breakpoint"



Types of labels

- Binary

- Multiclass: What changes do we need to make?
  - Almost none! Just extend the computation of accuracy to multiclass.

If the number of classes is >2

Notions of uncertainty: general case

Suppose in $S \subseteq \mathcal{X} \times \mathcal{Y}$, a p_k fraction are labeled as k (for each $k \in \mathcal{Y}$).

1. **Classification error**:
$$u(S) := 1 - \max_{k \in \mathcal{Y}} p_k$$
2. **Gini index**:
$$u(S) := 1 - \sum_{k \in \mathcal{Y}} p_k^2$$
3. **Entropy**:
$$u(S) := \sum_{k \in \mathcal{Y}} p_k \log \frac{1}{p_k}$$

Each is *maximized* when $p_k = 1/|\mathcal{Y}|$ for all $k \in \mathcal{Y}$ (i.e., equal numbers of each label in S)

Each is *minimized* when $p_k = 1$ for a single label $k \in \mathcal{Y}$ (so S is **pure** in label)

Regression: when the labels are real numbers

- Classification vs Regression
  - Both supervised learning
  - Regression has real-valued labels.
- Examples: Price prediction. Property value prediction.
- Standard measure of performance: mean squared error: $\frac{1}{m}\sum_{i=1}^{m}\left(f\left(x^{(i)}\right) - y^{(i)}\right)^2$

Q: why are we using squared error rather than absolute error?

my opinion: convenience & tradition

- What changes needed for decision tree?
  - How to make predictions at the leaf node?

Average labels of the data at the leaf; denote by $\bar{y}_{YES}$ and $\bar{y}_{NO}$.

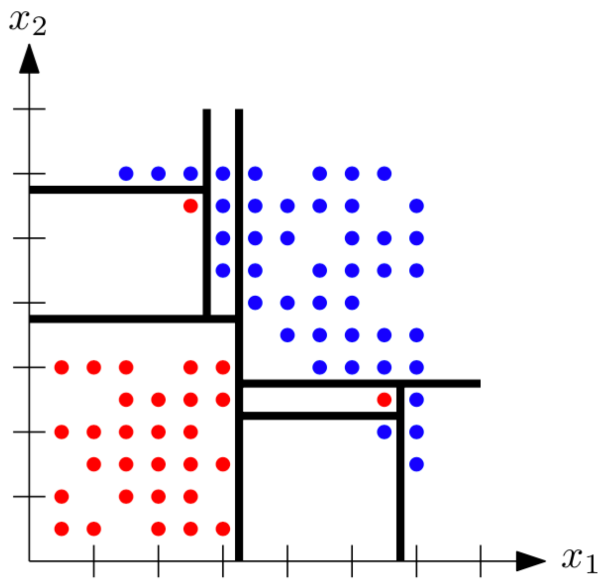
  - How to adjust score[f]?

Use negative squared error

$$\frac{1}{|data|}\cdot\left(-\sum_{i\in YES}(y_i - \bar{y}_{YES})^2 - \sum_{i\in NO}(y_i - \bar{y}_{NO})^2\right)$$

(notations from the decision tree pseudocode)

“Spurious” patterns can be learned

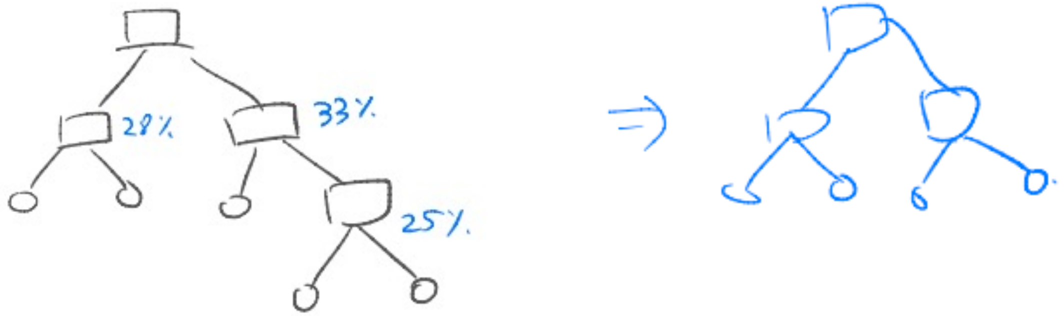


by the way, note axis-parallel decision boundaries

Unlearn spurious patterns by pruning

Split the data into **train set** and **validation set**

- Build a decision tree based on the **train set**; compute the **validation set** error
- While true
  - For each non-leaf node, pretend that it is a leaf node and then compute the validation set error (but do not make it a leaf node yet)
  - If none reduces the validation set error
    - Break
  - Else
    - **Prune** the one that reduces the validation set error the most



original validation set error: 35%

k-Nearest Neighbors (k-NN)

k-nearest neighbor: main concept

- Train set: $S = \{ (x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

- **Idea**: given a new, unseen data point x, its label should resemble the labels of **nearby points**

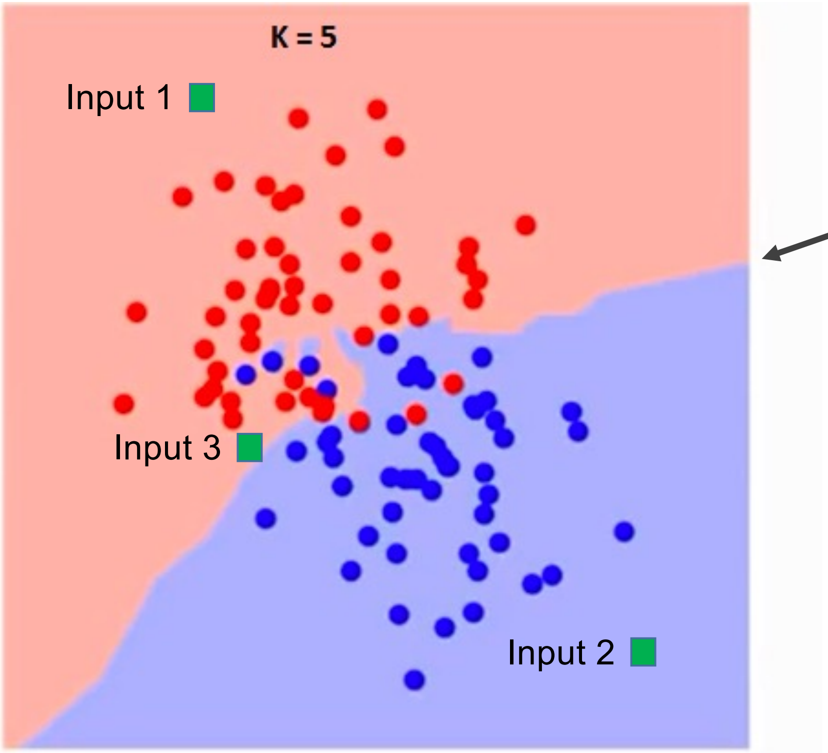
- What function?
  - Input: $x \in \mathbb{R}^d$

  - From S, find the k nearest points to x from S; call it $N(x)$

    E.g., Euclidean distance

  - Output: the majority vote of $\{y_i : i \in N(x)\}$
    - For regression, take the average label.

k-NN example



decision boundary

Basics

How to extract features as **real values**?

- Binary features: Take 0/1
- Categorical {1,…,C} (e.g., movie genres)
  - Binary vector of length C. Set c-th coordinate 1 and 0 otherwise. one-hot encoding

Q: Why don't we just take 1,…,C as a real-valued feature?

Distance:

- (popular) Euclidean distance: $d(x, x') = \sqrt{\sum_{i=1}^{d} \left(x_i - x_i'\right)^2}$
- Manhattan distance : $d(x, x') = \sum_{i=1}^{d} |x_i - x_i'|$

Q: If we shift a feature, would the distance change? no

Q: What about scaling a feature? yes

Make sure features are scaled fairly

- Features having different scale can be problematic. (e.g., weights in lbs vs shoe size)
- [Definition] **Standardization**
  - For each feature f, compute $\mu_f = \frac{1}{m}\sum_{i=1}^{m} x_f^{(i)}$, $\sigma_f = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(x_f^{(i)} - \mu_f\right)^2}$
  - Then, transform the data by $\forall f \in \{1,\dots,d\}, \forall i \in \{1,\dots,m\},\ x_f^{(i)} \leftarrow \frac{x_f^{(i)} - \mu_f}{\sigma_f}$

after transformation, each feature has mean 0 and variance 1

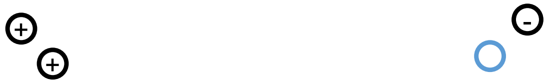
- Be sure to keep the “standardize” function and apply it to the test points.
  - Save $\{(\mu_f, \sigma_f)\}_{f=1}^{d}$
  - For test point x^*, apply $x_f^* \leftarrow \frac{x_f^* - \mu_f}{\sigma_f}, \forall f$

k-NN Summary

- Given: labeled data D
- Training
  - Compute and save $\{(\mu_f, \sigma_f)\}_{f=1}^{d}$
  - Compute and save standardization of D
- Test
  - Given x^*, apply standardization $x_f^* \leftarrow \frac{x_f^* - \mu_f}{\sigma_f}, \forall f$
  - Compute k nearest neighbors $N(x^*)$
  - Predict by majority vote label in $N(x^*)$ (average label for regression tasks)

Variations

Recall the majority vote rule: $\hat{y} = \arg\max_{y \in \{1,\ldots,C\}} \sum_{i \in \mathcal{N}(x)} 1\{y_i = y\}$



Q: Blue dot is the test point. If k=3, which label would it predict?

Q: Which label do you think we should predict?

Weighted version

- $\hat{y} = \arg\max_{y \in \{1,\ldots,C\}} \sum_{i \in \mathcal{N}(x)} w_i 1\{y^{(i)} = y\}$

weights that sum to 1

$$w_i \propto \exp\left(-\beta \cdot d\left(x, x^{(i)}\right)\right), \beta > 0$$

$$\cancel{w_i \propto \frac{1}{d(x, x^{(i)})^\beta}}$$

$$w_i \propto \frac{1}{1 + d(x, x^{(i)})^\beta}$$

Q: What would be the downside of using weighted version?

tuning β is cumbersome!

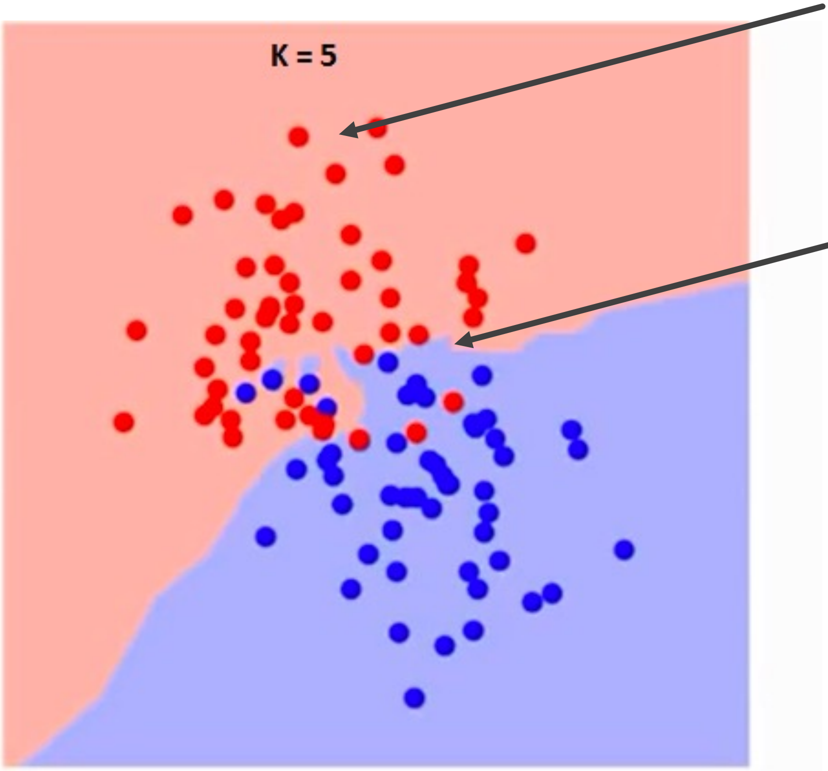
Confidence

Confidence

- $P(Y = y|X = x) \propto \sum_{i \in \mathcal{N}(x)} 1\{y^{(i)} = y\}$
- $P(Y = y|X = x) \propto \sum_{i \in \mathcal{N}(x)} w_i 1\{y^{(i)} = y\}$ // weighted version

Same thing applies to decision tree – (number of majority points in that leaf node / number of points in that leaf node)

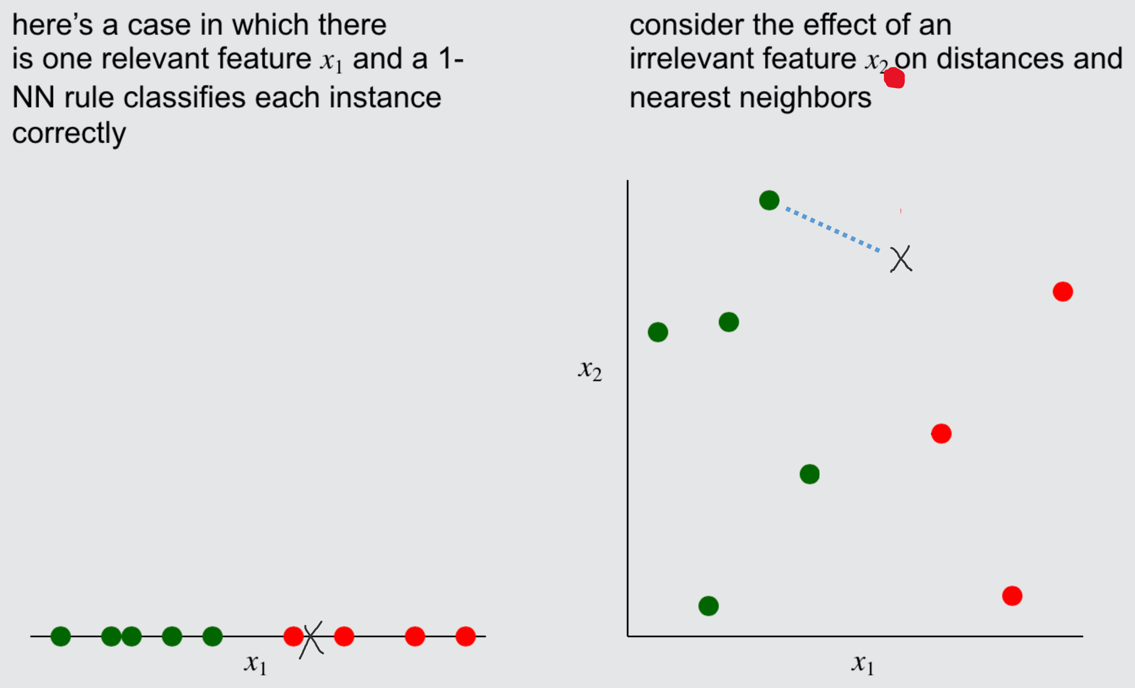
Confidence



High confidence

Low confidence

Issues 1: irrelevant features



here's a case in which there is one relevant feature x_1 and a 1-NN rule classifies each instance correctly

consider the effect of an irrelevant feature x_2 on distances and nearest neighbors

Q: how did we deal with irrelevant features in decision trees?

not all features are used because
(i) we stop adding features when they are unnecessary (e.g. having zero local accuracies, subset is already pure)
(ii) pruning

Issues 2: test time complexity

- How a k-NN function work:
  - Compute distance to m points $O(dm)$
  - Sort distances $O(m \log m)$
  - Pick k smallest. $O(k)$
  - Overall $O\big(m(d + \log m)\big)$

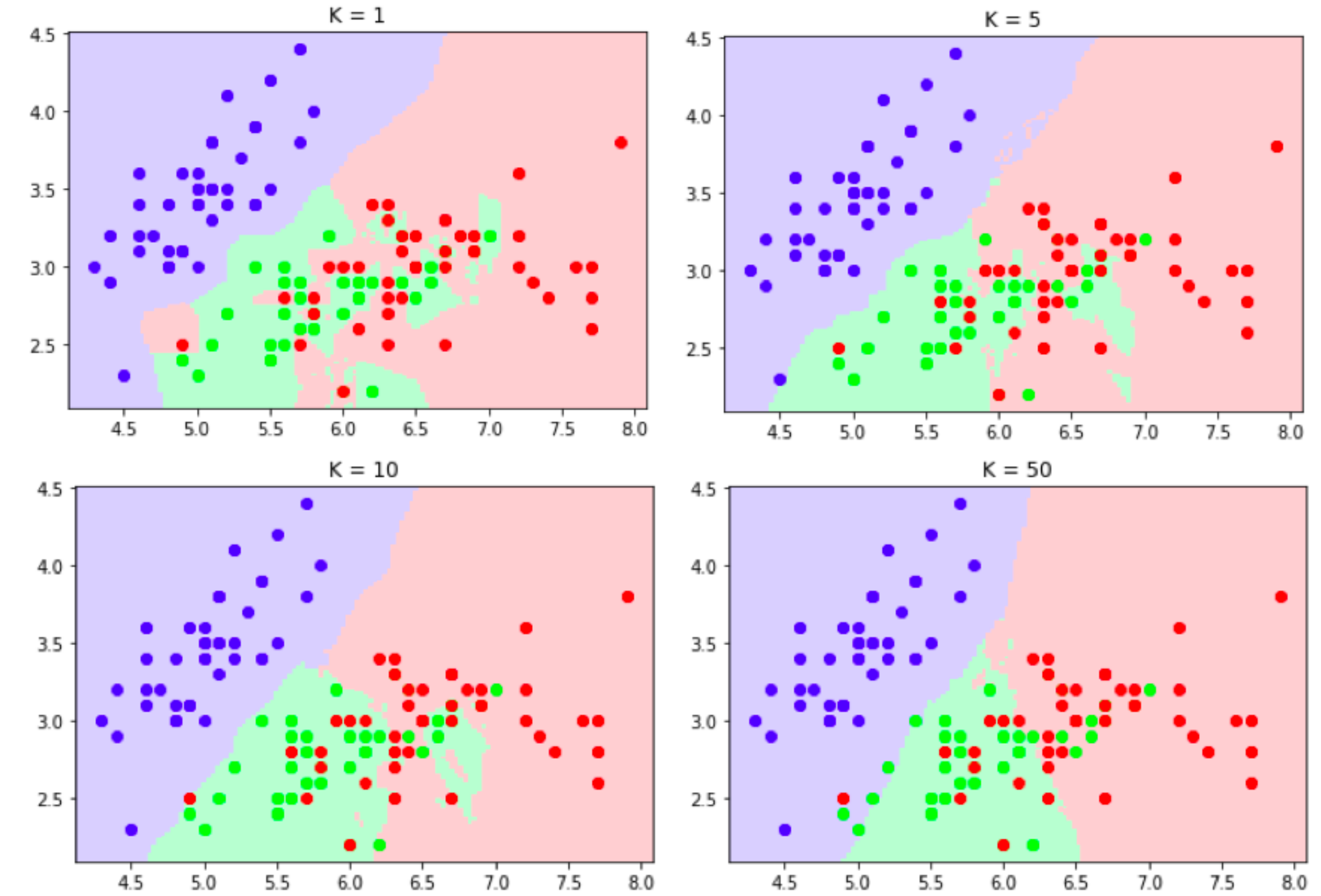
- Issue: test time complexity scales linearly with m!!
- Solutions
  - k-d tree: Exact search

    for large d very likely to hit the worst case
    - Best case: $O(\log(m))$ Worst case: $O(m)$
  - Locality-sensitive hashing: approximate search, $O(m^{\rho})$ with $\rho \in (0,1)$

Issue 3: choosing k

- Q: If we set $k = m$, then which classification rule does it look like?

- Q: If we set $k = 1$, what would be the train set error (assume there is no repeated train data point)?



K = 1

K = 5

K = 10

K = 50

Comparison

	Decision Tree	k-NN
• Interpretability	good	bad
• Sensitivity to irrelevant features	low	high
• train time	$O(dm^2 + dm \log m)$	$O(dm)$
• test time	depth of the tree worst: $\mathrm{O}(\min\{d, m\})$ best: $\log(m)$	$O(m(d + \log(m)))$ bad
• test time space complexity	worst: $O(m)$ in general: much smaller	$\Theta(dm)$

Next time

- Model selection
  - How to choose k?
  - Overfitting
- Naïve Bayes Classifier

Thank you!

Computer Science

CSC380: Principles of Data Science

Basics of Predictive Modeling and Classification 3 :

Model Selection / Naïve Bayes Classifier

Kyoungseok Jang

Announcement: Midterm

- Regrade request
  - Question 7(4): We decided to give everyone the score. Please check your answer, and if your answer is ‘False’, please send us the regrade request.
  - For the student who used the back side of the paper for your answer, please let us know.

Announcement: TA

- Due to his personal circumstances, TA Saiful will no longer be in charge of this class.
- Temporarily, we will not be able to provide the following services.
  - His office hour: Wed
  - His piazza hour: Wed/ Thu/ Fri

Announcement: Recording

- There was a problem with Lecture 17 (Mar. 21) recording.
  - The slides were missing in the video.
- I will re-record that lecture this weekend.

Overfitting and Model Evaluation

Challenges in ML

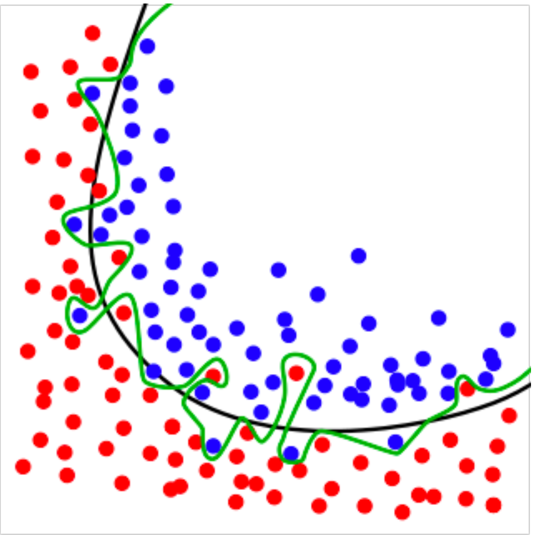
Train set error is an important score to measure the performance of your function,

But it's not enough.

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

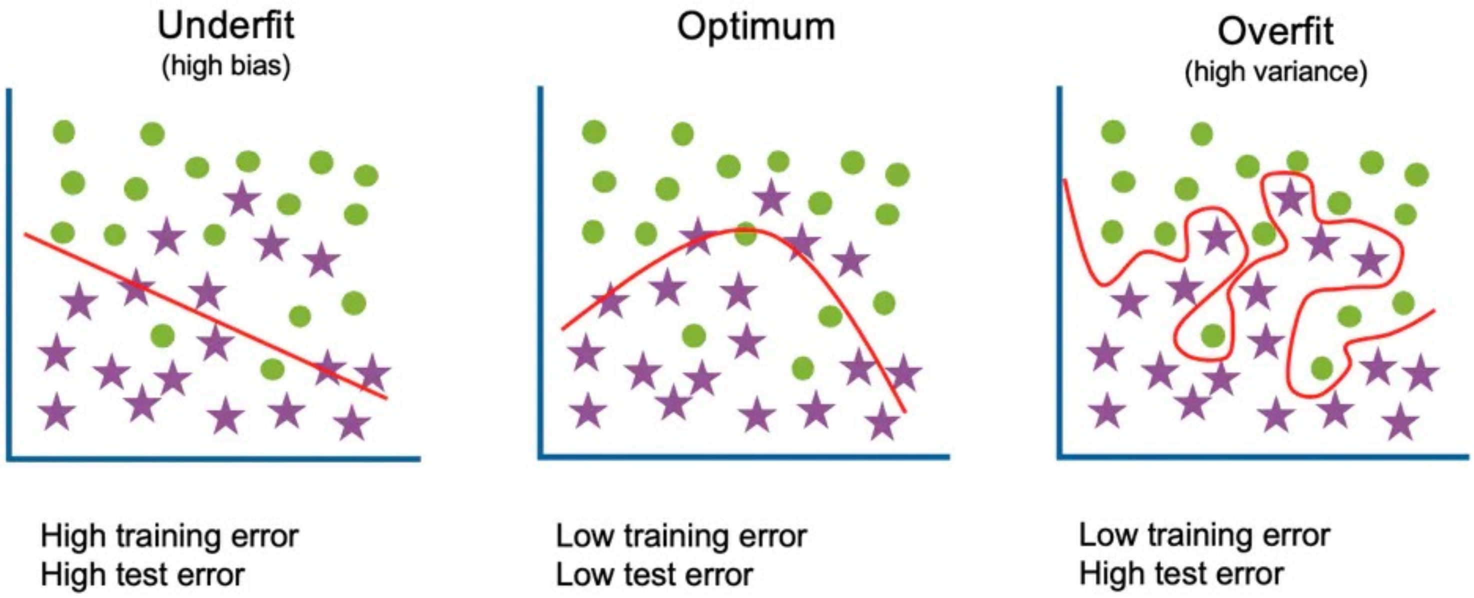
This function will not work well on real life – called *overfitting*

Solution: Fit our model based on the train set but shouldn't "over-do" it. This is called **regularization**.



green: almost memorization
black: true decision boundary

Overfitting vs Underfitting



Underfit
(high bias)

High training error
High test error

Optimum

Low training error
Low test error

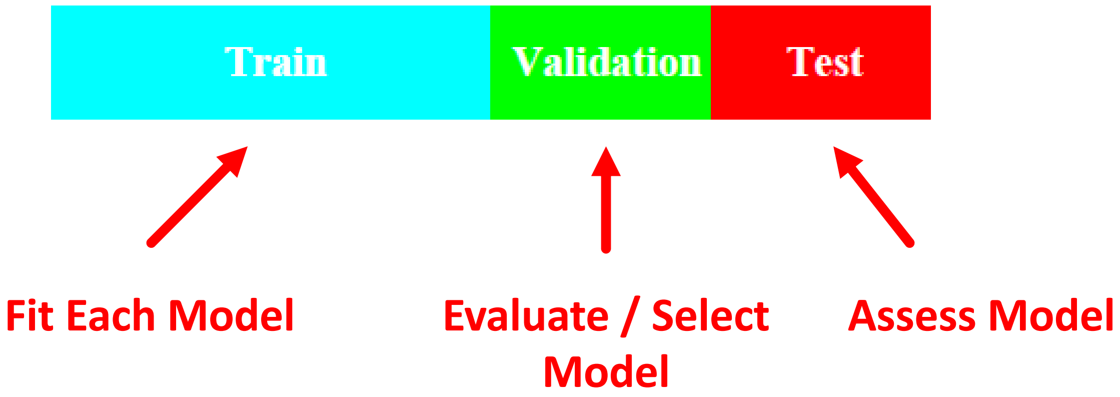
Overfit
(high variance)

Low training error
High test error

Source: ibm.com

Model Selection / Assessment

Partition your data into Train-Validation-Test sets



Fit Each Model

Evaluate / Select Model

Assess Model

- 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 set by statisticians)
- For large data (say a few thousands), 80-10-10 is usually fine.

Tuning hyperparameters (e.g., k in k-NN)

Validation set method:

- For each hyperparameter $h \in H$
  - Train $\hat{f}$ on train set with h
  - Compute the error rate of $\hat{f}$ on validation set
- Choose the best performing hyperparameter h^*
- Use h^* to retrain the final model $\hat{f}^*$ with both train and validation set.
- Finally, evaluate $\hat{f}^*$ on test set to estimate its future performance.

hyperparameter: parameters of the model that are not trained automatically by ML algorithms. (e.g., k in k-NN)

parameters: those that are trained automatically (e.g., tree structures in decision tree)

Pro tip

- Do not use arithmetic grids; use geometric grids.

Don’t	k = 1, 3, 5, 7, 9, …
Do	k = 1, 2, 4, 8, 16, …

Downside: How much do we trust the validation set?

Tuning hyperparameters

K-fold cross validation

- Randomly partition train set S into K disjoint sets; call them $\mathrm{fold}_1, \dots, \mathrm{fold}_K$
- For each hyperparameter $h \in \{1, \dots, H\}$
  - For each $k \in \{1, \dots, K\}$
    - train $\hat{f}_k^h$ with $S \setminus \mathrm{fold}_\mathrm{k}$
    - measure error rate $e_{h,k}$ of $\hat{f}_k^h$ on fold_k
  - Compute the average error of the above: $\widehat{err}^h = \frac{1}{K} \sum_{k=1}^{K} e_{h,k}$
- Choose $\hat{h} = \arg\min_h \widehat{err}^h$
- Train $\hat{f}^*$ using S (all the training points) with hyperparameter $\hat{\boldsymbol{h}}$
- Finally, evaluate $\hat{f}^*$ on test set to estimate its future performance.

K=10 is standard, but K=5 is okay, too

Leave one-out = m-fold cross validation (m: train set size)
$\Longrightarrow$ When (1) the dataset is small (2) ML algorithm's retraining time complexity is low (e.g., kNN)

Cross Validation using numpy.random.permutation

```
permidx = np.random.permutation(12)
```

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

```
idx = np.array([(i % 5) for i in np.arange(12)])
```

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

```
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 X (n by d array; n data points) and Y (length-n array)

- train set: X[folds_except[0],:], Y[folds_except[0]]
- validation set: X[folds[0],:], Y[folds[0]]

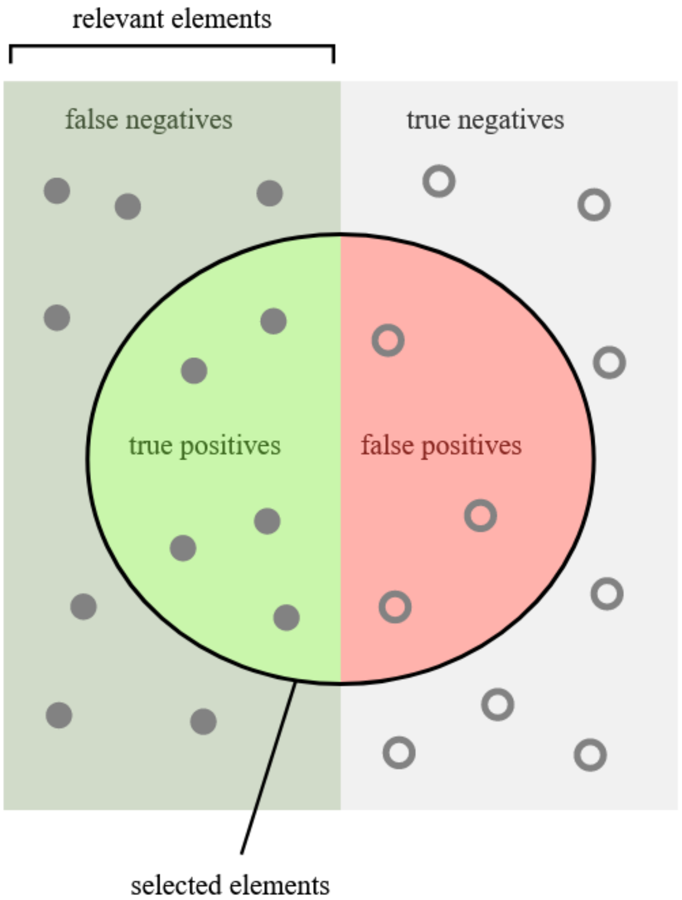
Stratification

- Issue: Say we have few positive labels (=imbalanced class)
  The error rates in CV can be unstable.

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

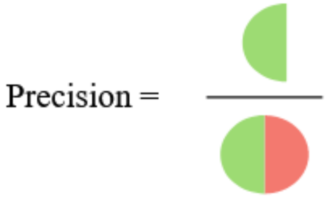
- E.g., |S|=100. 20 positive/80 negative. K=10
  - Pool positive data points, randomly shuffle them; place 2 data points for each fold.
  - Perform the same with negative data points.

Evaluating Classifiers



For binary classifiers we evaluate a couple standard metrics,

How many selected items are relevant?



$$\frac{\text{True Positives}}{\text{True Positives} \quad + \text{False Positives}}$$

How many relevant items are selected?

$$\frac{\text{True Positives}}{\text{True Positives} \quad + \text{False Negatives}}$$

Evaluating Classifiers

Tuning with precision vs. recall can be tricky, so we use F1 score,

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}$$

- This is the *harmonic mean* of precision and recall
  - min(x,y) <= harmonic_mean(x,y) <= geometric_mean(x,y) <= arithmetic_mean(x,y) <= max(x,y)

$$\frac{1}{\frac{1}{2}(\frac{1}{x} + \frac{1}{y})} \qquad \sqrt{xy} \qquad \frac{1}{2}(x + y)$$

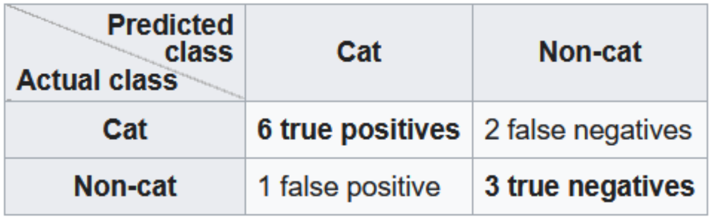
- Can be very sensitive to *class imbalance* (num. positives vs negative)

- Gives equal importance to precision and recall – F1 may not be best when you care about one more than the other (e.g., in medical tests we care about recall)

Confusion Matrix

Suppose our classifier distinguishes between cats and non-cats.

We can make the following table called **confusion matrix**:



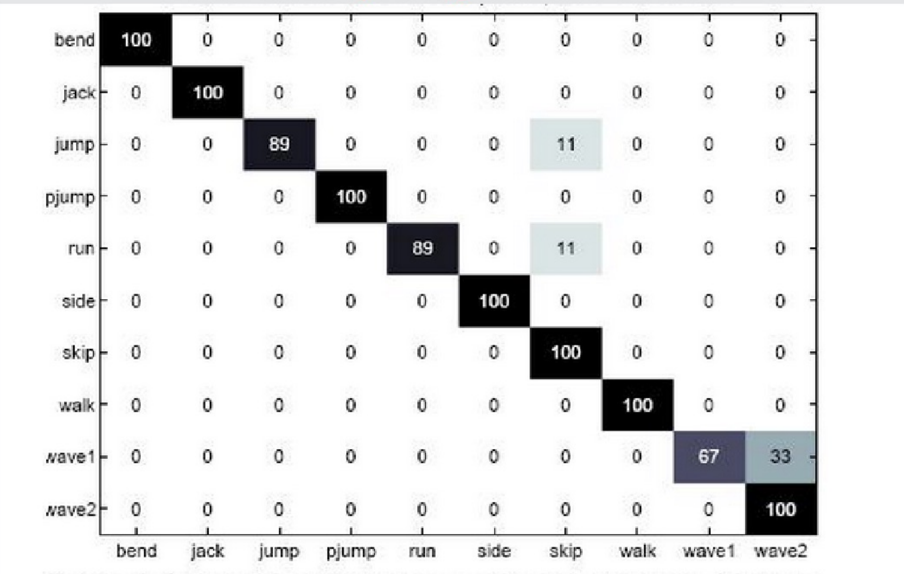
Predicted class Actual class	Cat	Non-cat
Cat	**6 true positives**	2 false negatives
Non-cat	1 false positive	**3 true negatives**

It tells us if classifier is biased towards certain mistakes (False Positives, False Neg.)

Good for investigating opportunities to improve the classifier.

Confusion Matrix

task: activity recognition from video



actual class

	bend	jack	jump	pjump	run	side	skip	walk	wave1	wave2
bend	100	0	0	0	0	0	0	0	0	0
jack	0	100	0	0	0	0	0	0	0	0
jump	0	0	89	0	0	0	11	0	0	0
pjump	0	0	0	100	0	0	0	0	0	0
run	0	0	0	0	89	0	11	0	0	0
side	0	0	0	0	0	100	0	0	0	0
skip	0	0	0	0	0	0	100	0	0	0
walk	0	0	0	0	0	0	0	100	0	0
wave1	0	0	0	0	0	0	0	0	67	33
wave2	0	0	0	0	0	0	0	0	0	100

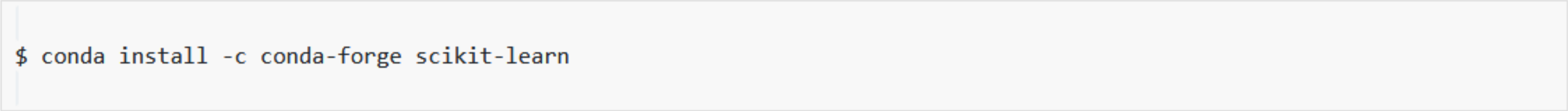
predicted class

figure from vision.jhu.edu

Don’t just stare at the overall error rate! Let’s investigate what errors it is making.

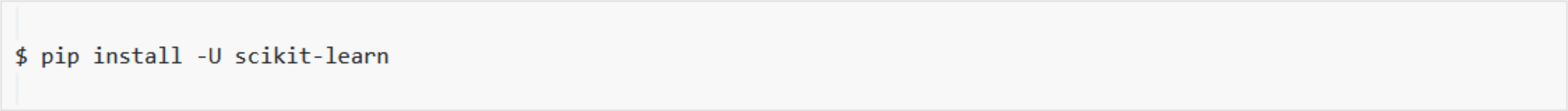
Scikit-Learn

Python library for machine learning. Install using Anaconda:



```
$ conda install -c conda-forge scikit-learn
```

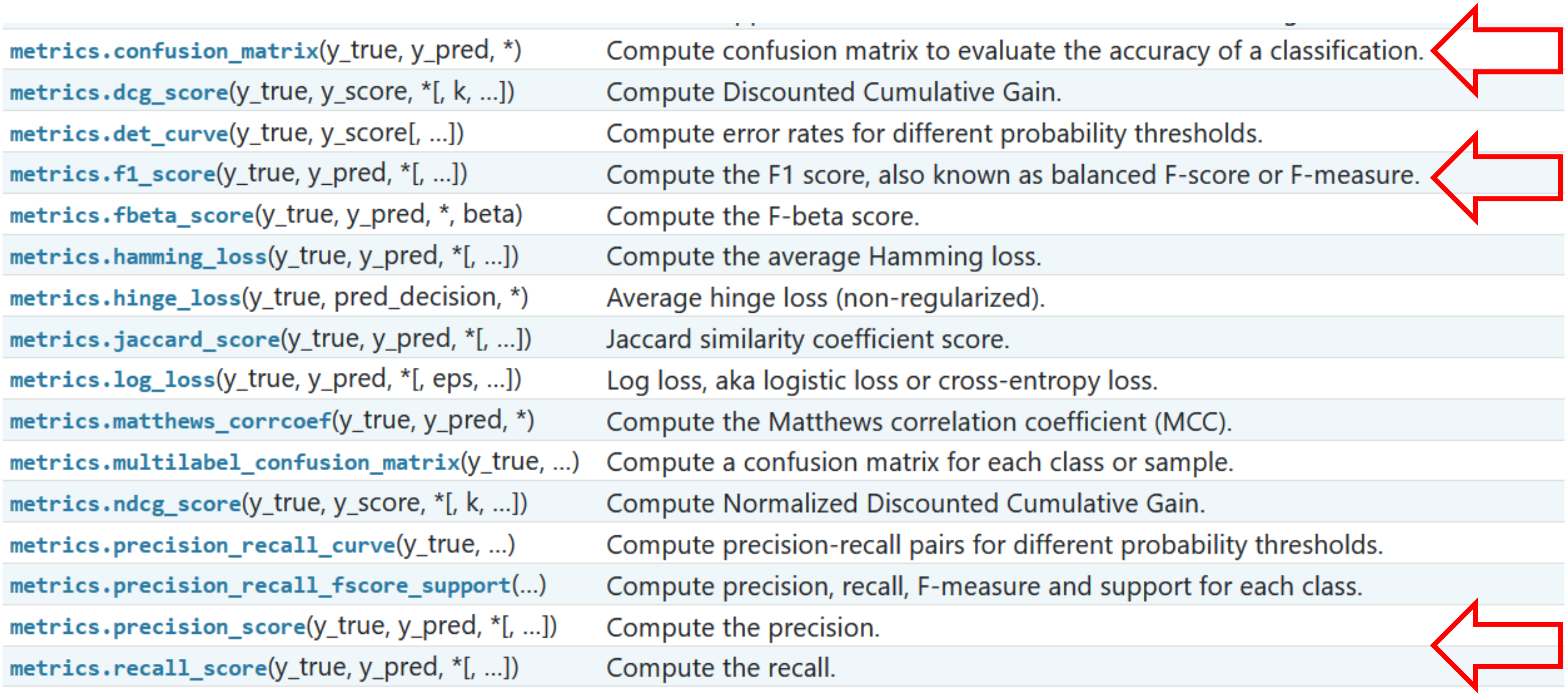
Or using PyPi:



```
$ pip install -U scikit-learn
```

Evaluation in Scikit-Learn

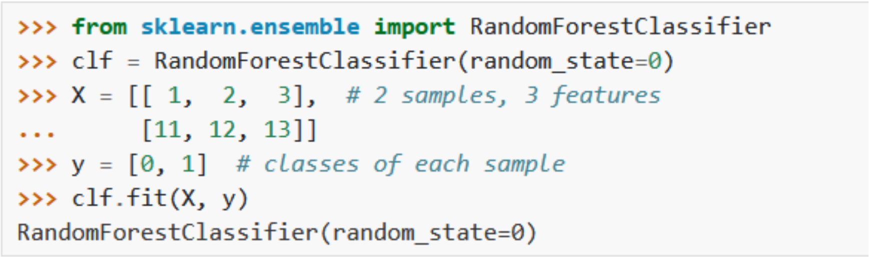
Evaluation functions live in `metrics`



metrics.confusion_matrix(y_true, y_pred, *)	Compute confusion matrix to evaluate the accuracy of a classification.
metrics.dcg_score(y_true, y_score, *[, k, ...])	Compute Discounted Cumulative Gain.
metrics.det_curve(y_true, y_score[, ...])	Compute error rates for different probability thresholds.
metrics.f1_score(y_true, y_pred, *[, ...])	Compute the F1 score, also known as balanced F-score or F-measure.
metrics.fbeta_score(y_true, y_pred, *, beta)	Compute the F-beta score.
metrics.hamming_loss(y_true, y_pred, *[, ...])	Compute the average Hamming loss.
metrics.hinge_loss(y_true, pred_decision, *)	Average hinge loss (non-regularized).
metrics.jaccard_score(y_true, y_pred, *[, ...])	Jaccard similarity coefficient score.
metrics.log_loss(y_true, y_pred, *[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
metrics.matthews_corrcoef(y_true, y_pred, *)	Compute the Matthews correlation coefficient (MCC).
metrics.multilabel_confusion_matrix(y_true, ...)	Compute a confusion matrix for each class or sample.
metrics.ndcg_score(y_true, y_score, *[, k, ...])	Compute Normalized Discounted Cumulative Gain.
metrics.precision_recall_curve(y_true, ...)	Compute precision-recall pairs for different probability thresholds.
metrics.precision_recall_fscore_support(...)	Compute precision, recall, F-measure and support for each class.
metrics.precision_score(y_true, y_pred, *[, ...])	Compute the precision.
metrics.recall_score(y_true, y_pred, *[, ...])	Compute the recall.

Scikit-Learn

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



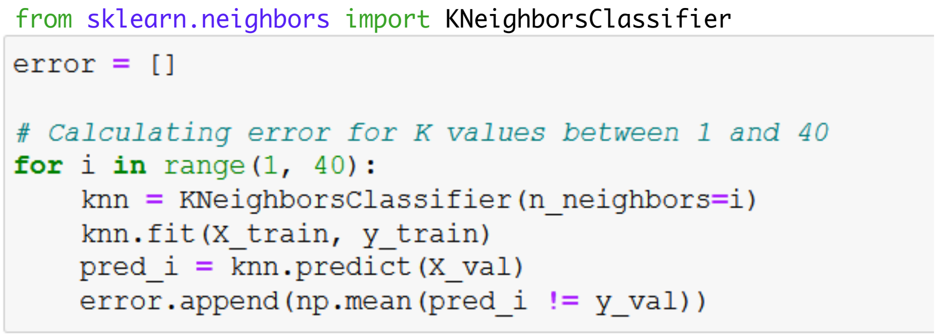
```
>>> from sklearn.ensemble import RandomForestClassifier
>>> clf = RandomForestClassifier(random_state=0)
>>> X = [[ 1,  2,  3],  # 2 samples, 3 features
...      [11, 12, 13]]
>>> y = [0, 1]  # classes of each sample
>>> clf.fit(X, y)
RandomForestClassifier(random_state=0)
```

`fit()` Generally accepts 2 inputs

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

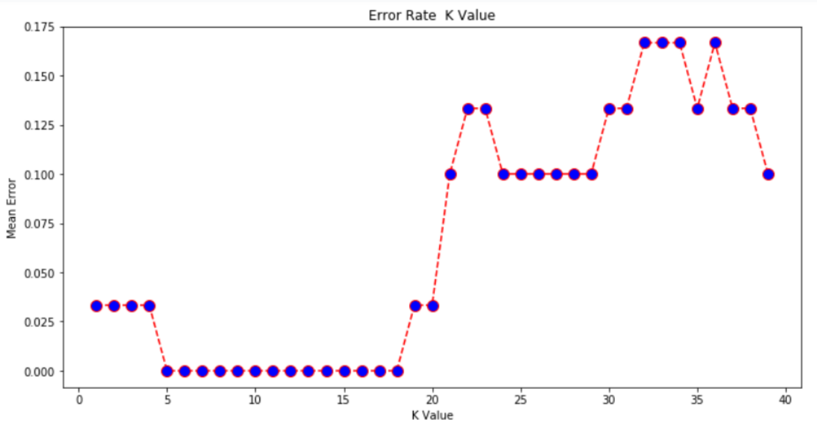
k-Nearest Neighbors

Train / evaluate the KNN classifier for each value K,



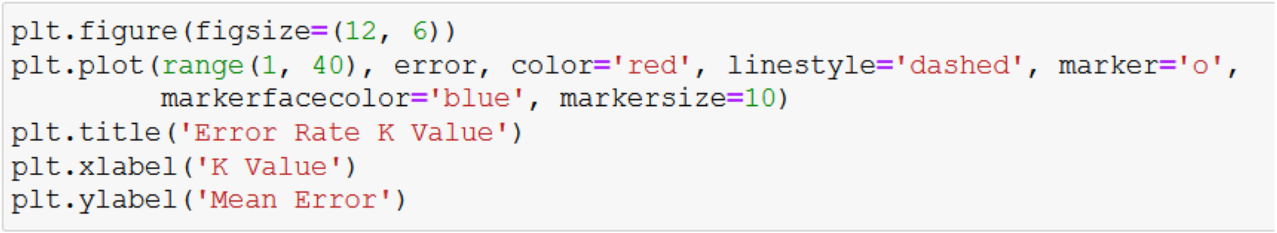
```
from sklearn.neighbors import KNeighborsClassifier
error = []

# 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)
    error.append(np.mean(pred_i != y_val))
```

↑ vector operation!



in practice: use geometric grid like 1,2,4,8,…

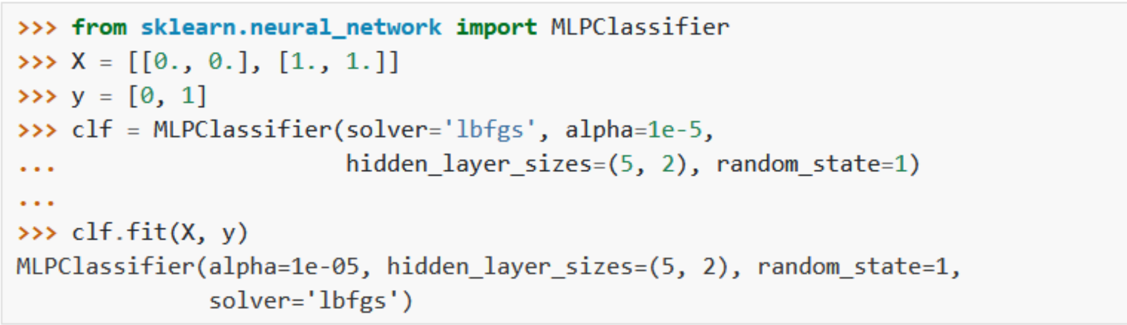
Print error:



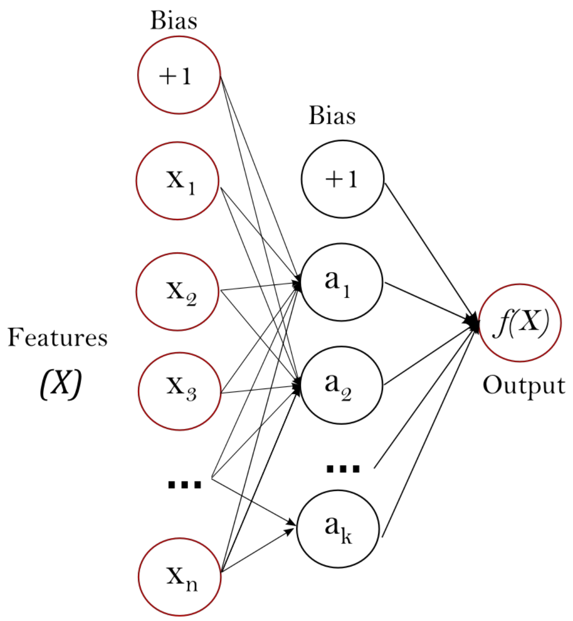
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Scikit-Learn

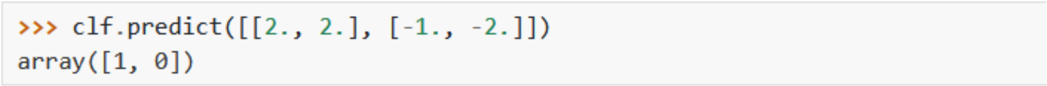
Can fit Neural Networks as well, for example a *multilayer perceptron* (MLP) for classification,



```
>>> from sklearn.neural_network import MLPClassifier
>>> X = [[0., 0.], [1., 1.]]
>>> y = [0, 1]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(5, 2), random_state=1)
...
>>> clf.fit(X, y)
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,
              solver='lbfgs')
```



Now do some prediction on new data…

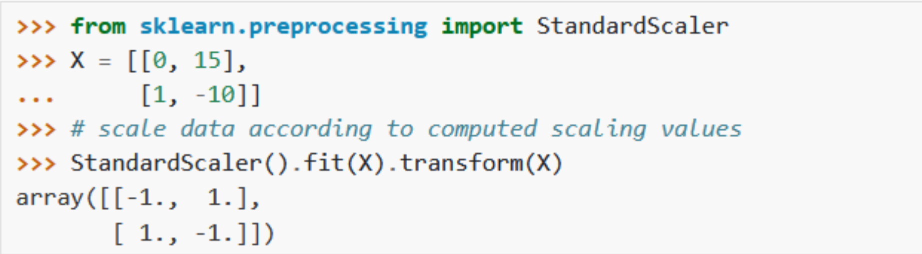


```
>>> clf.predict([[2., 2.], [-1., -2.]])
array([1, 0])
```

Neural nets for regression too:
`sklearn.neural_network.MLPRegressor`

Preprocessing : Z-Score

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



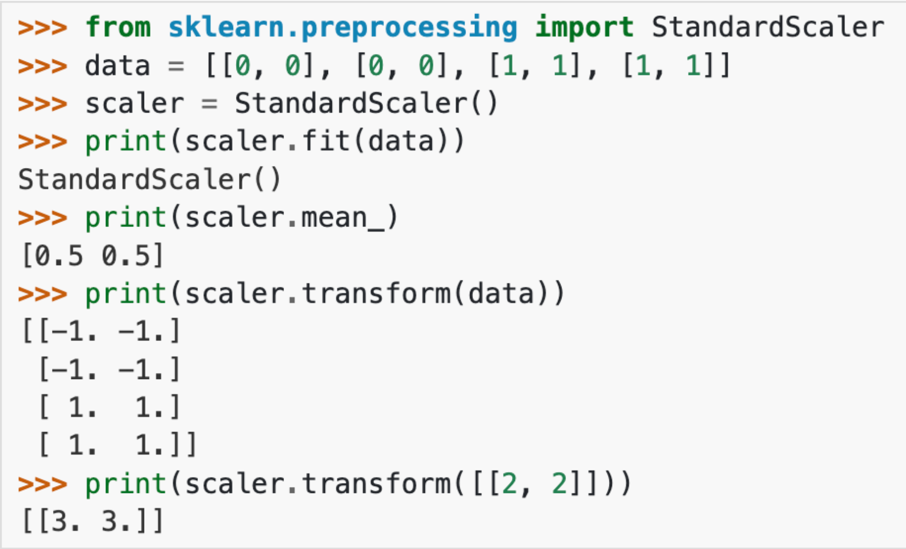
```
>>> from sklearn.preprocessing import StandardScaler
>>> X = [[0, 15],
...      [1, -10]]
>>> # scale data according to computed scaling values
>>> StandardScaler().fit(X).transform(X)
array([[-1.,  1.],
       [ 1., -1.]])
```

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

Example use this to do standardization in k-NN.

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

Preprocessing : Z-Score



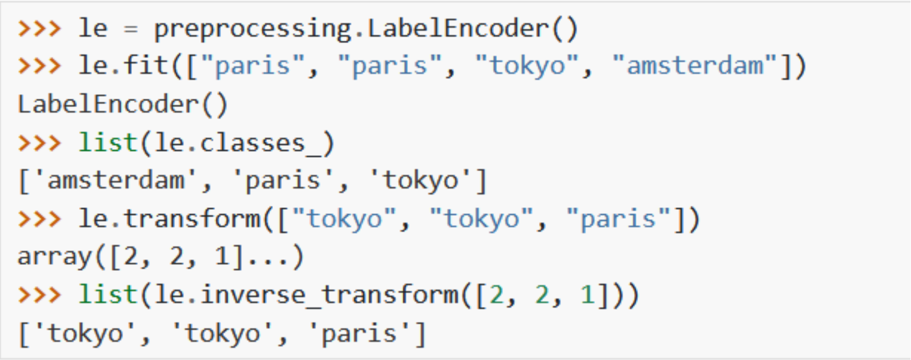
```
>>> from sklearn.preprocessing import StandardScaler
>>> data = [[0, 0], [0, 0], [1, 1], [1, 1]]
>>> scaler = StandardScaler()
>>> print(scaler.fit(data))
StandardScaler()
>>> print(scaler.mean_)
[0.5 0.5]
>>> print(scaler.transform(data))
[[-1. -1.]
 [-1. -1.]
 [ 1.  1.]
 [ 1.  1.]]
>>> print(scaler.transform([[2, 2]]))
[[3. 3.]]
```

- From k-NN: We learned Standardization
- [Definition] **Standardization**
  - For each feature f, compute $\mu_f = \frac{1}{m}\sum_{i=1}^{m} x_f^{(i)}$, $\sigma_f = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(x_f^{(i)} - \mu_f\right)^2}$
  - Then, transform the data by $\forall f \in \{1,\dots,d\}, \forall i \in \{1,\dots,m\},\ x_f^{(i)} \leftarrow \frac{x_f^{(i)} - \mu_f}{\sigma_f}$

- Be sure to keep the “standardize” function and apply it to the test points.
  - Save $\{(\mu_f, \sigma_f)\}_{f=1}^{d}$
  - For test point x^*, apply $x_f^* \leftarrow \frac{x_f^* - \mu_f}{\sigma_f}$, $\forall f$

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()
>>> 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']
```

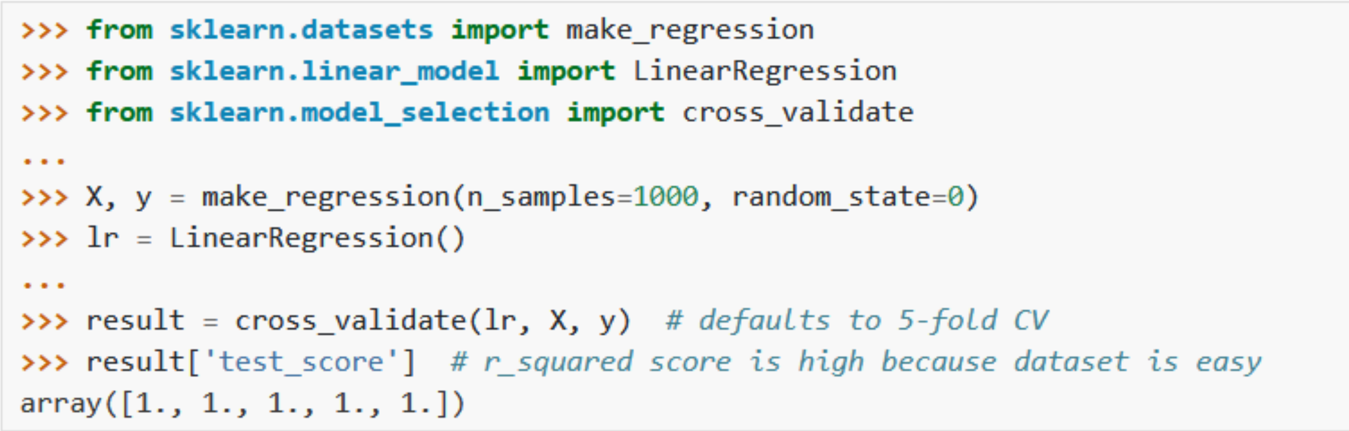
`LabelEncoder` transforms these into integer values, e.g. for categorical distributions

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

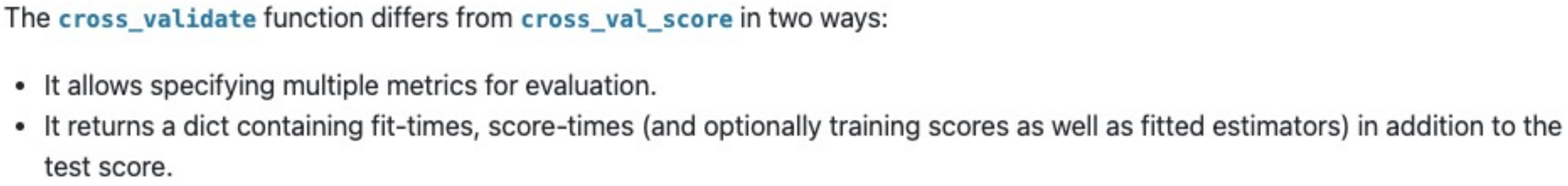
Cross-Validation

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.])
```

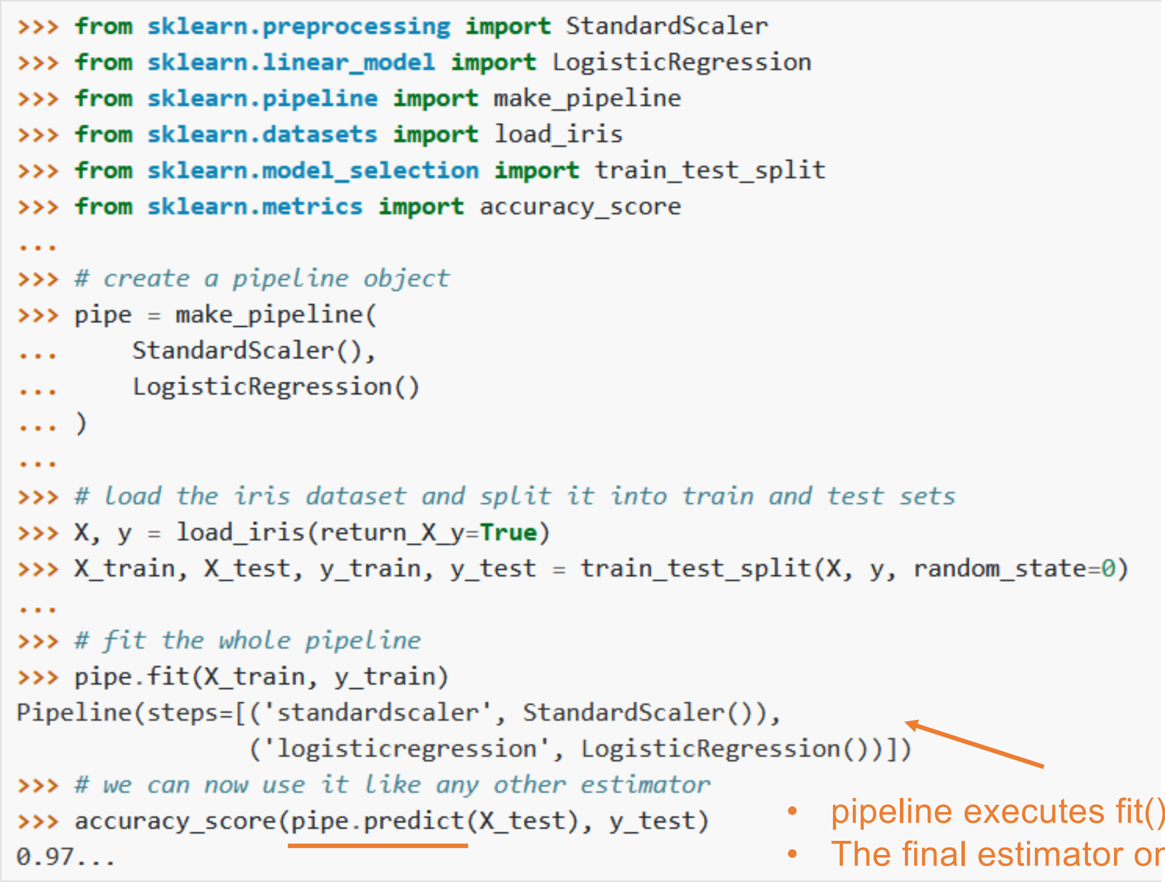
- `sklearn.model_selection`
  - Many split functions: K-fold, leave-one-out, etc.



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.

Pipeline



```
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
...
>>> # create a pipeline object
>>> pipe = make_pipeline(
...     StandardScaler(),
...     LogisticRegression()
... )
...
>>> # load the iris dataset and split it into train and test sets
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
...
>>> # fit the whole pipeline
>>> pipe.fit(X_train, y_train)
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
>>> # we can now use it like any other estimator
>>> accuracy_score(pipe.predict(X_test), y_test)
0.97...
```

ML workflows can be complicated. Chain tasks into a *pipeline*…

Example Standardizes data and fits logistic regression classifier

Nice `train_test_split` helper function

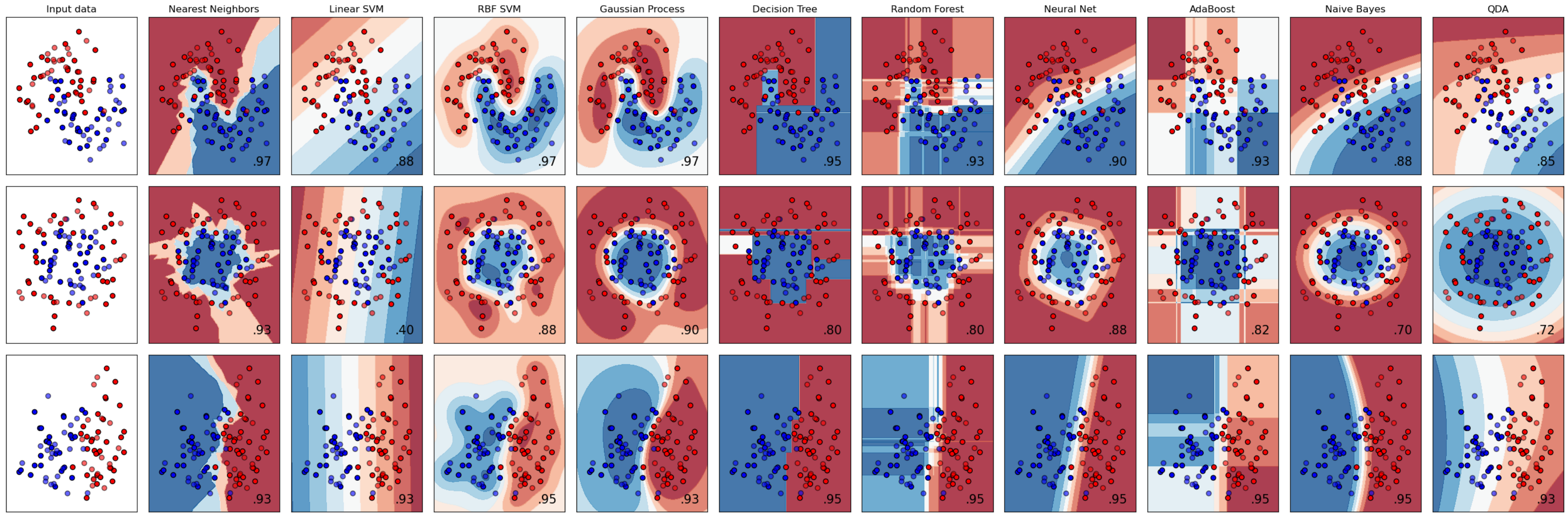
(default: 0.75 - 0.25 split)

- pipeline executes fit()/transform() functions in sequence!
- The final estimator only needs to implement fit.

↑ calls predict() in the last object in the pipeline

Scikit-Learn

Easily try out *all* the classifiers…

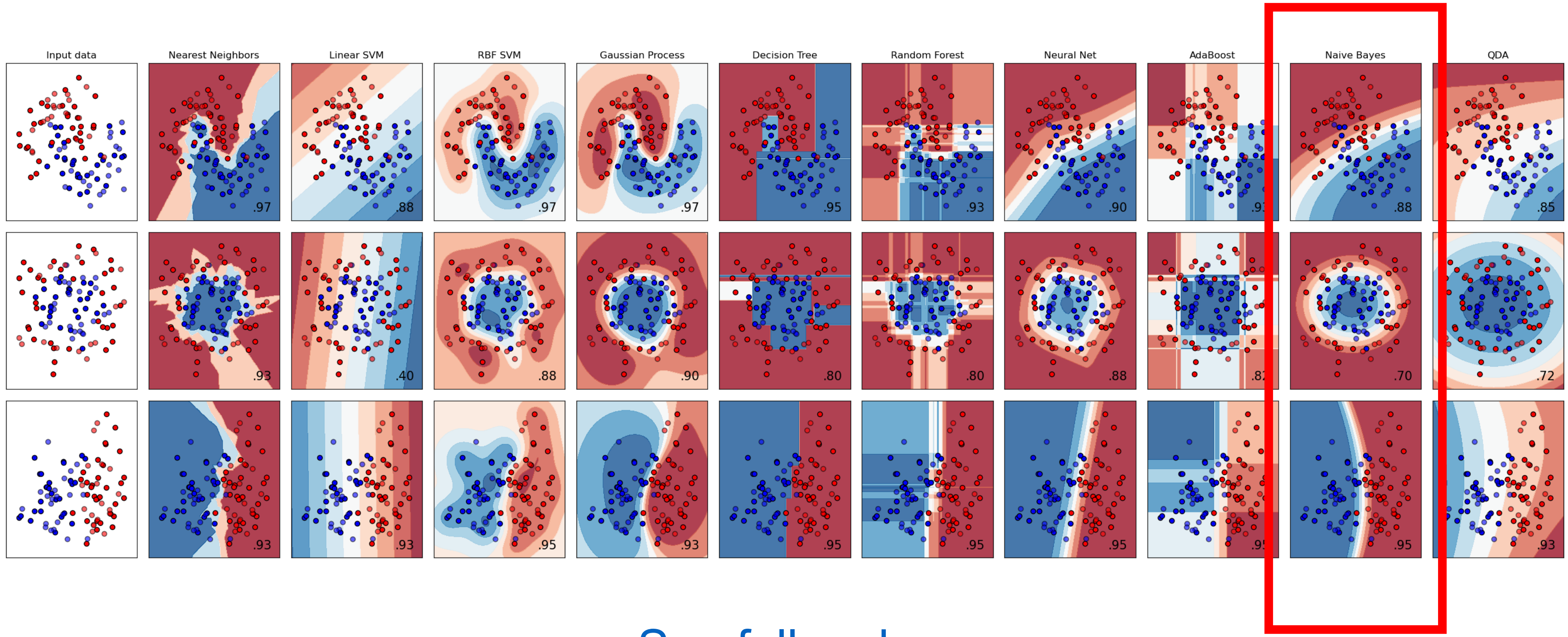


See full code.

Scikit-Learn

Easily try out *all* the classifiers…

Naïve Bayes



See full code.

Naïve Bayes Classifier

Naïve Bayes Overview

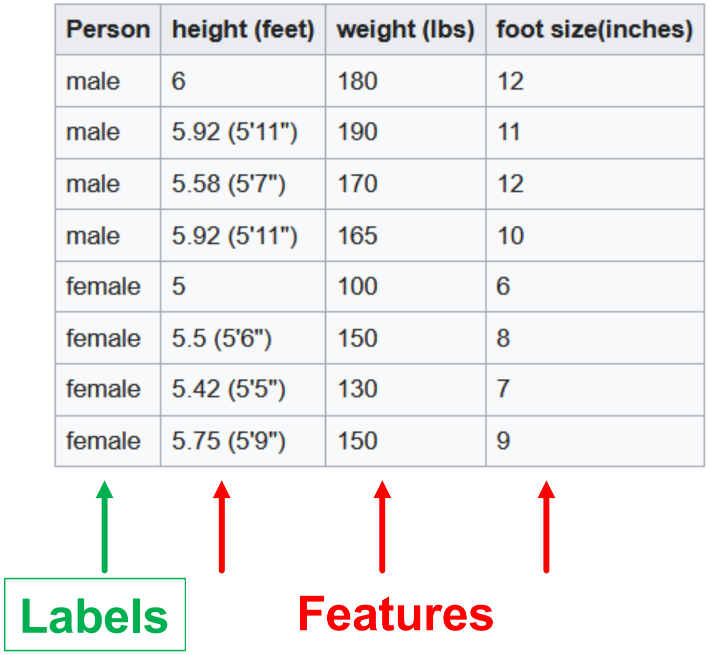
<u>Heads Up</u> This section will return to some math as we go in depth. There will be lots of abstraction. However, much of it is review of MLE that you already know with a new application (Naïve Bayes Classification)

1) **Ask questions if you are lost**
2) **Prerequisite/ details like derivative will not be included in your evaluation**

- Introduction to Naïve Bayes Classifier
- Maximum Likelihood Estimation

Probabilistic Approach to ML

Training Data:



Person	height (feet)	weight (lbs)	foot size(inches)
male	6	180	12
male	5.92 (5'11")	190	11
male	5.58 (5'7")	170	12
male	5.92 (5'11")	165	10
female	5	100	6
female	5.5 (5'6")	150	8
female	5.42 (5'5")	130	7
female	5.75 (5'9")	150	9

Labels

Features

Task: Observe features $x_1, \dots, x_D$ and predict class label $y \in \{1, \dots, C\}$

Model: Assume that the feature x and its label y follows certain type of distribution $\mathcal{D}$ with parameter θ.

$$(x, y) \overset{\text{i.i.d.}}{\sim} \mathcal{D}_\theta$$

Training Algorithm: Estimate θ

(if D is gaussian, then θ is the mean & var)

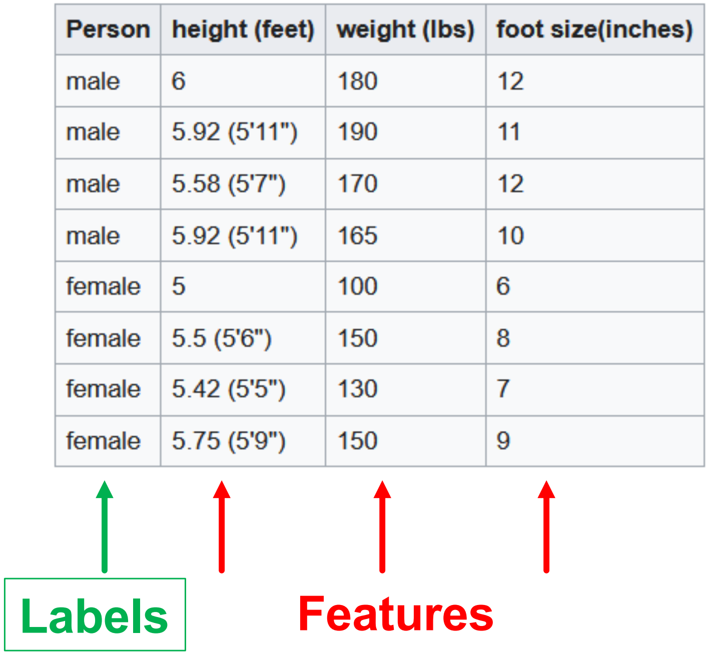
To classify: Compute

$$\hat{y} = \arg \max_{c \in \{1,\dots,C\}} p(y = c \mid x; \hat{\theta})$$

what comes after semicolon is the parameter of the distribution
In this case, think of $\mathcal{D}_{\hat{\theta}}$

Naïve Bayes is a Specific Probabilistic Approach

Training Data:



Person	height (feet)	weight (lbs)	foot size(inches)
male	6	180	12
male	5.92 (5'11")	190	11
male	5.58 (5'7")	170	12
male	5.92 (5'11")	165	10
female	5	100	6
female	5.5 (5'6")	150	8
female	5.42 (5'5")	130	7
female	5.75 (5'9")	150	9

Labels
Features

Task: Observe features $x_1, \ldots, x_D$ and predict class label $y \in \{1, \ldots, C\}$

Naïve Bayes Model: Treat features as *conditionally independent* given class label,

$$p(x,y) = p(y)p(x|y) = p(y)\prod_{d=1}^{D} p(x_d \mid y)$$

build individual models for these

To classify a given instance x: Bayes rule!

$$p(y=c \mid x) = \frac{p(y=c)p(x \mid y=c)}{p(x)}$$

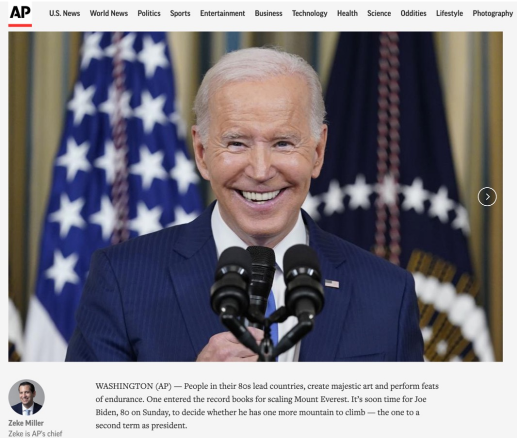
P(y=c): Class prior dist.

P(x|y=c): class-conditional dist.

Naïve Bayes Classifier

Features are typically not independent!

Example 1 If a recent news article contains word “Biden” it is much more likely to contain the word “Joe”.



WASHINGTON (AP) — People in their 80s lead countries, create majestic art and perform feats of endurance. One entered the record books for scaling Mount Everest. It’s soon time for Joe Biden, 80 on Sunday, to decide whether he has one more mountain to climb — the one to a second term as president.

Example 2 If the flower <u>petal *width*</u> is very large then the <u>petal *length*</u> is also likely to be large.

Source: Matt Gormley

Naïve Bayes: The Key Feature

Simplifying Assumption: "*Class conditional"* distribution assumes features are conditionally independent given class

$$p(x \mid y) = \prod_{d=1}^{D} p(x_d \mid y)$$

- "Naïve" as in general features are likely to be dependent.
- Every feature can have a different class-conditional distribution

Doesn't capture correlation among features. But why would it be a good idea?

- Easy computation**:** For C classes and D features only $O(CD)$ parameters
- Prevents overfitting
- Simplicity

Naïve Bayes Classifier: Class prior parameters

For the class prior distribution, take categorical distribution. (recall: extension of Bernoulli)

$$y \sim \text{Categorical}(\pi), \qquad \pi \in \mathbb{R}^C, \pi_c \geq 0, \sum_c \pi_c = 1$$

$\Rightarrow p(y = c) = \pi_c$

$\Rightarrow$ *C parameters for the ‘class prior distribution’*

Naïve Bayes Classifier: likelihood parameters

For real-valued features we can use Normal distribution:

$$p(x \mid y = c) = \prod_{d=1}^{D} \mathcal{N}(x_d \mid \underbrace{\mu_{cd}, \sigma_{cd}^2}_{\text{Parameters of featured for class } c})$$

Q: how many parameters?

A: 2 CD

For binary features $x_d \in \{0,1\}$ can use Bernoulli distributions:

$$p(x \mid y = c) = \prod_{d=1}^{D} \text{Bernoulli}(x_d \mid \underset{\uparrow}{\theta_{cd}})$$

Q: how many parameters?

"Coin bias" for d$^{\text{th}}$ feature and class c

- K-valued discrete features: use Categorical.
- Can mix-and-match, e.g. some discrete, some continuous features

$$p(x \mid y = c) = \prod_{d=1}^{D\prime} \text{Bernoulli}(x_d \mid \theta_{cd}) \prod_{d=D^{\prime}+1}^{D} \mathcal{N}(x_d \mid \mu_{cd}, \sigma_{cd}^2)$$

Naïve Bayes Classifier: Example

- When class prior distribution, class-conditional distributions are all Bernoulli

Feature Binary vectors of length K

$\mathbf{x} \in \{0,1\}^K$

e.g.) x= (Smoke?, Military?, Religion?)

y= (resident=0, foreigner=1)

Assumption

$Y \sim \text{Bernoulli}(\phi)$

$X_k \sim \text{Bernoulli}(\theta_{k,Y}) \ \forall k \in \{1,\ldots,K\}$

Model:

$$\begin{aligned} p_{\phi,\boldsymbol{\theta}}(\boldsymbol{x},y) &= p_{\phi,\boldsymbol{\theta}}(x_1,\ldots,x_K,y) \\ &= p_\phi(y)\prod_{k=1}^{K} p_{\boldsymbol{\theta}_k}(x_k|y) \\ &= (\phi)^y(1-\phi)^{(1-y)}\prod_{k=1}^{K}(\theta_{k,y})^{x_k}(1-\theta_{k,y})^{(1-x_k)} \end{aligned}$$

Classify: After training (fix ϕ and $\theta_{k,y}$),
if x=(1,0,1),
substitute x1=1, x2=0, x3=1
and compare p(x,0) and p(x,1)
if p(x,1)>p(x,0), then
output (prediction) is 1.

Naïve Bayes Model : Maximum Likelihood

Fitting the model requires learning all parameters…

$$p(x \mid y = c) = p(y = c; \pi) \prod_{d=1}^{D} p(x_d \mid \theta_{cd})$$

Class Prior Parameters **Likelihood Parameters**

Given training data $\mathcal{D} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$ maximize the likelihood function,

$$\theta^{\text{MLE}} = \arg\max_{\pi,\theta} \log p(\mathcal{D}; \pi, \theta)$$

Naïve Bayes Model : Maximum Likelihood

Fitting the model requires learning all parameters…

$$p(x \mid C_\ell) = p(C_\ell; \pi) \prod_{d=1}^{D} p(x_d \mid \theta_{d\ell})$$

Prior Parameters

Likelihood Parameters

Let’s review maximum likelihood estimation…

Given training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ maximize the likelihood function,

$$\theta^{\text{MLE}} = \arg\max_{\pi,\theta} \log p(\mathcal{D}; \pi, \theta)$$

Substitute general form of Naïve Bayes distribution and simplify…

Maximum Likelihood

Maximum Likelihood Estimator (MLE) as the name suggests, maximizes the likelihood function.

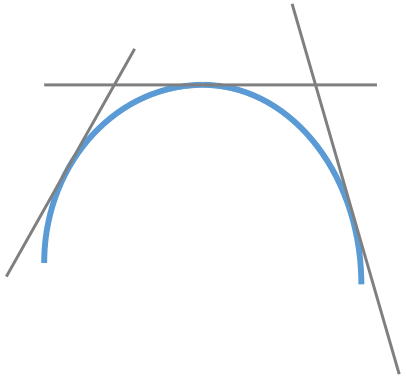
$$\hat{\theta}^{\text{MLE}} = \operatorname*{arg\,max}_{\theta} \mathcal{L}_N(\theta) = \operatorname*{arg\,max}_{\theta} \prod_{i=1}^{N} p(x^{(i)}, y^{(i)}; \theta)$$

Question How do we find the MLE?

1. closed-form
2. iterative methods

Finding the maximum/maximizer of a function

Example: Suppose $f(\theta) = -a\theta^2 + b\theta + c$ with $a > 0$

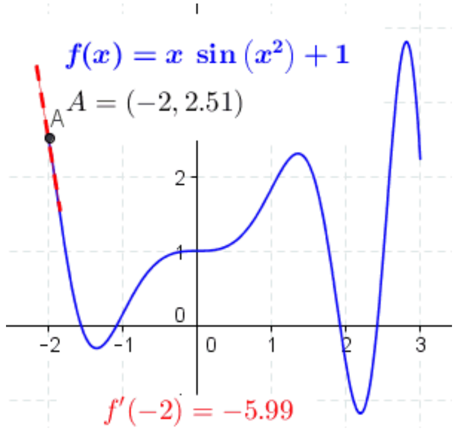


It is a quadratic function.
=> finding the 'flat' point suffices

Compute the gradient and set it equal to 0

$$f'(\theta) = -2a\theta + b \qquad \Rightarrow \qquad \theta = \frac{b}{2a}$$

Closed form!



(gradient illustration)

Q: Does this trick work for other functions?
⇒ Yes, for **concave** functions!
⇒ Roughly speaking, functions that curves down only, never upwards

Finding the maximum/maximizer of a function

What if there is no closed form solution?

Example: $f(\theta) = \frac{1}{2}x(ax - 2\log(x) + 2)$

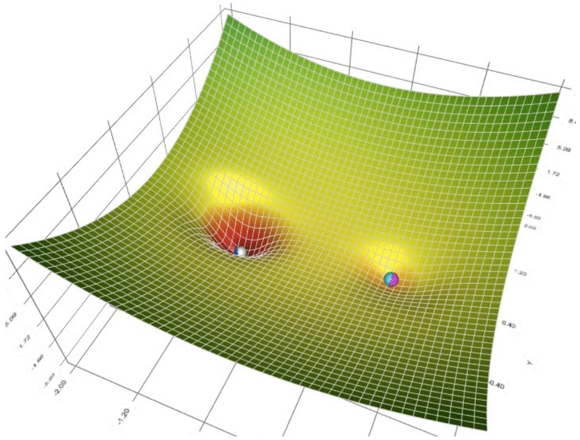
$$f'(\theta) = ax - \log(x)$$

No known closed form for $ax = \log(x)$

Iterative methods:

- Gradient ascent (or *descent* if you are minimizing)
- Newton's method
- Etc. (beyond the scope of our class)

Iterative methods for **concave** functions
=> Will find the global maximum
for **nonconcave**,
=> usually find a local maximum but could also get stuck at *stationary point*.



Gradient 'descent'

Naïve Bayes Model : Maximum Likelihood

Fitting the model requires learning all parameters…

$$p(x \mid C_\ell) = p(C_\ell; \pi) \prod_{d=1}^{D} p(x_d \mid \theta_{d\ell})$$

Prior Parameters

Likelihood Parameters

…OK, back to Naïve Bayes

Given training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ maximize the likelihood function,

$$\theta^{\mathrm{MLE}} = \arg\max_{\pi,\theta} \log p(\mathcal{D}; \pi, \theta)$$

Substitute general form of Naïve Bayes distribution and simplify…

Naïve Bayes Model : Maximum Likelihood

Fitting the model requires learning all parameters…

$$p(x \mid y = c) = p(y = c; \pi) \prod_{d=1}^{D} p(x_d \mid \theta_{cd})$$

Prior Parameters

Likelihood Parameters

Given training data $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ maximize the likelihood function,

$$\theta^{\text{MLE}} = \arg\max_{\pi,\theta} \log p(\mathcal{D}; \pi, \theta)$$

Naïve Bayes Model : Maximum Likelihood

$$\theta^{\mathrm{MLE}} = \arg\max_{\pi,\theta} \log p(\mathcal{D};\pi,\theta)$$

$(\mathcal{D} \coloneqq \{(x^{(i)}, y^{(i)})\}_{i=1}^{m})$

Since data are iid

$$= \arg\max_{\pi,\theta} \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)};\pi,\theta)$$

log(ab) = log a + log b

$$= \arg\max_{\pi,\theta} \sum_{i=1}^{m} \log p(x^{(i)}, y^{(i)};\pi,\theta)$$

Conditional probability +
Naïve Bayes assumption

$$= \arg\max_{\pi,\theta} \sum_{i=1}^{m} \log p(y^{(i)};\pi) + \sum_{i=1}^{m}\sum_{d=1}^{D} \log p\left(x_d^{(i)} \middle| y^{(i)};\theta_{y^{(i)}d}\right)$$

θ_{cd}: parameter for feature d for class c

Find zero-gradient if concave, or gradient-based optimization otherwise

Example: Naïve Bayes with Bernoulli Features

Analogy:

Roll a biased C-sided die (can be unfair)

Y =

Flip D biased coins head prb

X_1 | Y

θ_{Y1}

X_2 | Y

θ_{Y2}

... **...**

X_D | Y

θ_{YD}

y	x_1	x_2		...		x_D
5	0	1	1	0	1	0
2	1	0	0	0	0	0
3	1	1	1	1	0	0
...				...		
4	0	0	1	1	0	1

Adapted from: Matt Gormley

Example: Naïve Bayes with Bernoulli Features

Let each feature follow a Bernoulli distribution then the model is…

$$y \sim \text{Categorical}(\pi) \qquad x_d \mid y = c \ \sim \ \text{Bernoulli}(\theta_{cd})$$

The Naïve Bayes joint distribution is then:

$$p(\mathcal{D}; \pi, \theta) = \prod_{i=1}^{m} \left(p(y^{(i)}; \pi) \prod_{d} p\left(x_d^{(i)}; \theta_{y^{(i)}d} \right) \right)$$

Write down log-likelihood and optimize…

Bernoulli Naïve Bayes MLE

Let $m_c \coloneqq \sum_{i=1}^{m} \mathbf{I}\{y^{(i)} = c\}$ be number of training examples in class c then,

$$\log p(\mathcal{D}; \pi, \theta) = \sum_{c=1}^{C} m_c \log \pi_c + \sum_{c=1}^{C} \sum_{i:y^{(i)}=c} \sum_{d=1}^{D} \log p\left(x_d^{(i)}; \theta_{cd}\right)$$

Log-likelihood function is concave in all parameters so…

1. Take derivatives with respect to π and θ separately.
2. Set derivatives to zero and solve

$$\hat{\pi}_c = \frac{m_c}{m}$$

Fraction of training examples from class c

$$\hat{\theta}_{cd} = \frac{m_{cd}}{m_c}$$

Number of “heads” in training set from class c

$$m_{cd} = \sum_{i=1}^{m} I\{y^{(i)} = c, x_d^{(i)} = 1\}$$

Naïve Bayes in Scikit-learn

Scikit-learn has separate classes each feature type

```
sklearn.naive_bayes.GaussianNB
```

Real-valued features

```
sklearn.naive_bayes.MultinomialNB
```

Discrete K-valued feature counts (e.g. multiple die rolls, word count for an article)

```
sklearn.naive_bayes.BernoulliNB
```

Binary features (e.g. coinflip)

```
sklearn.naive_bayes.CategoricalNB
```

Discrete K-valued features (e.g. single die roll)

For large training data that don't fit in memory use Scikit-learn's out-of-core learning

https://scikit-learn.org/stable/modules/naive_bayes.html

Bernoulli Naïve Bayes MLE

$$\hat{\pi}_c = \frac{m_c}{m}$$

Fraction of training examples from class c

$$\hat{\theta}_{cd} = \frac{m_{cd}}{m_c}$$

Number of "heads" in training set from class c

What if there are *no* examples of class c in the training set?

$$\hat{\pi}_c = 0$$

Model will never learn to guess class c

What if all data points i in class c has $x_d^{(i)} = 0$ in the training set?

$$\hat{\theta}_{cd} = 0$$

Model will assign 0 likelihood for test data with $x_d = 1$ for class c (i.e., $p(x|y = c)$).

What does it imply on $p(y = c|x)$?

0!

Training data needs to see *every possible outcome for each feature*

Any ideas how we can fix this problem?

Fixing Bernoulli MLE

We could add a small constant to prevent zero probabilities…

$\hat{\pi}_c \propto m_c + \alpha$ $\qquad$ $\hat{\theta}_{cj} \propto m_{cj} + \beta$ $\qquad$ $\alpha, \beta > 0$

Pseudocounts
add-α Smoothing
Laplace smoothing
....

Coincides with so-called *Maximum a Posteriori (MAP)* estimate! (as opposed to MLE)

Bayesian approach: Place a *prior* distribution over the parameter π *and* $\{\theta_{cd}\}$ and then compute the *posterior* mean.

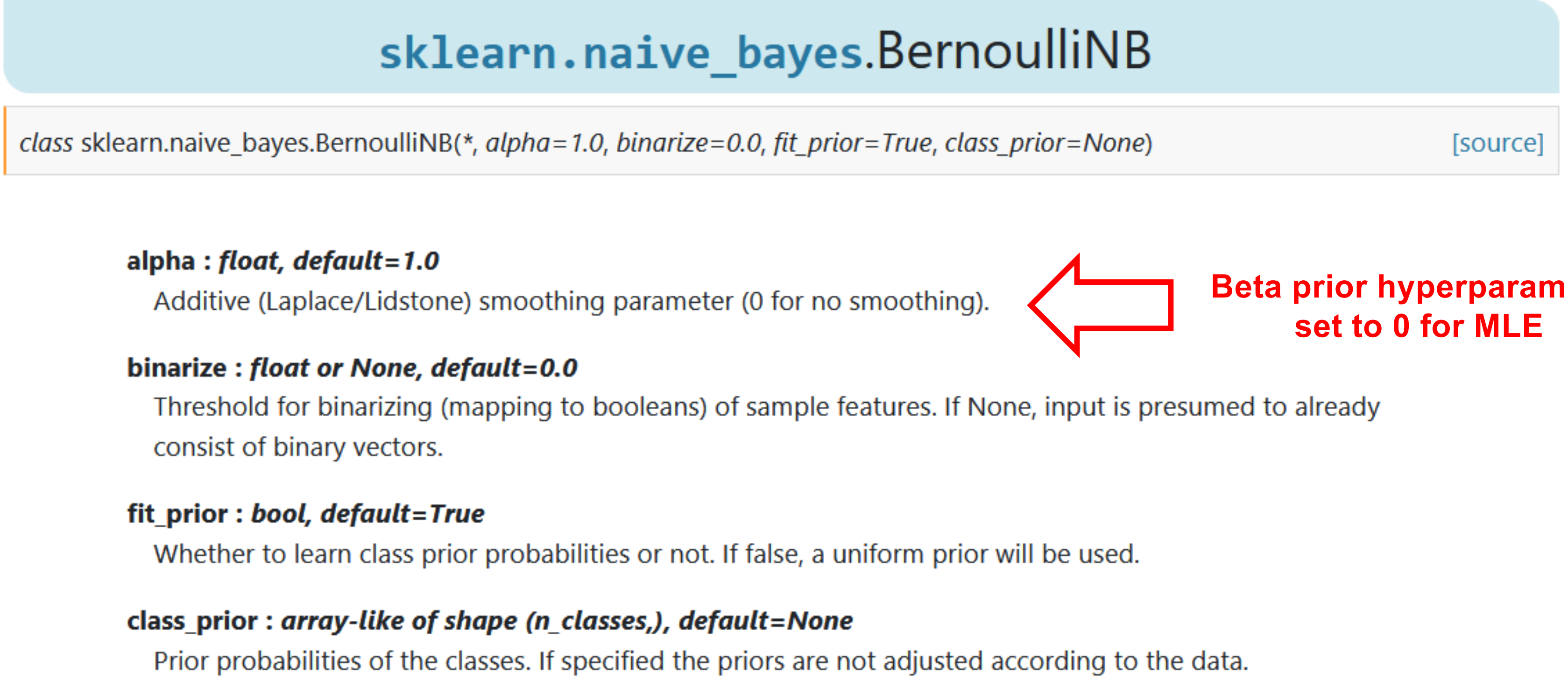
E.g., assume: $\pi \sim \text{Dirichlet}(\alpha_1, \dots, \alpha_C)$ and $y^{(1)}, \dots, y^{(n)} \sim \text{Bernoulli}(\pi)$

(Theorem) $p\left(\pi \mid y^{(1)}, \dots, y^{(n)}\right) = \text{Dirichlet}(m_1 + \alpha_1, \dots, m_C + \alpha_C)$

It follows that $\mathbf{E}\left[\pi \mid y^{(1)}, \dots, y^{(m)}\right] \propto m_c + \alpha_c$

typical choice: set $\boldsymbol{\alpha = \beta = 1}$

Bernoulli Naïve Bayes in Scikit-learn



sklearn.naive_bayes.BernoulliNB

class sklearn.naive_bayes.BernoulliNB(*, *alpha=1.0*, *binarize=0.0*, *fit_prior=True*, *class_prior=None*) [source]

alpha : *float, default=1.0*

Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

binarize : *float or None, default=0.0*

Threshold for binarizing (mapping to booleans) of sample features. If None, input is presumed to already consist of binary vectors.

fit_prior : *bool, default=True*

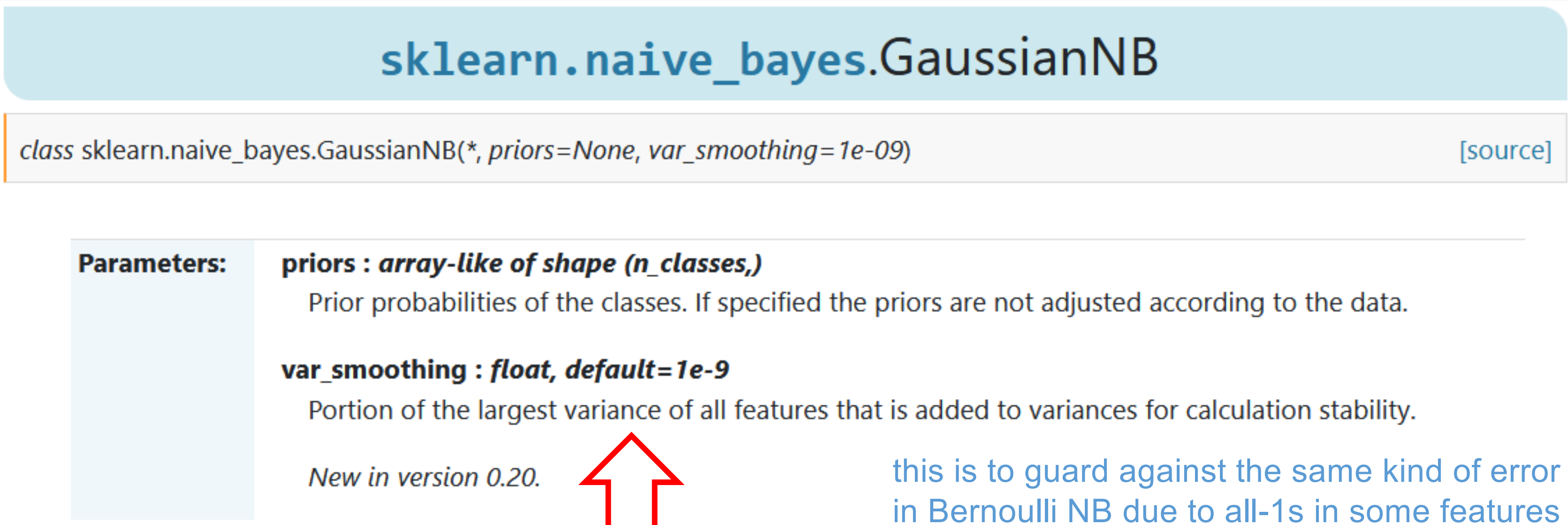
Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

class_prior : *array-like of shape (n_classes,), default=None*

Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

Beta prior hyperparameter set to 0 for MLE

Gaussian Naïve Bayes in Scikit-learn



sklearn.naive_bayes.GaussianNB

class sklearn.naive_bayes.GaussianNB(*, *priors=None*, *var_smoothing=1e-09*) [source]

Parameters:

priors : *array-like of shape (n_classes,)*

Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

var_smoothing : *float, default=1e-9*

Portion of the largest variance of all features that is added to variances for calculation stability.

New in version 0.20.

this is to guard against the same kind of error as in Bernoulli NB due to all-1s in some features for a particular class.

Bayesian prior on class-conditional variances
MLE if set to 0