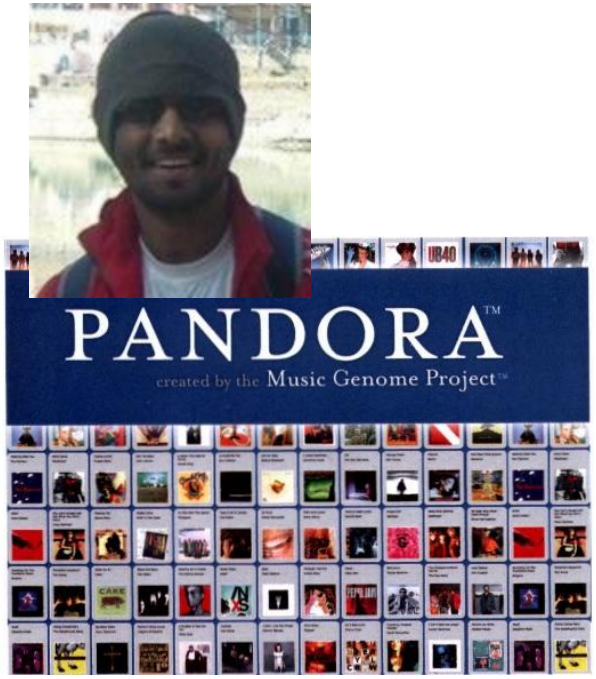
January 29
CSE 6242/CS-4803-DAVA

Classification

How to predict a discrete variable?

http://poloclub.gatech.edu/cse6242



How will I rate "Chopin's 5th Symphony"?

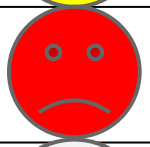
Songs	Label
Some nights	
Skyfall	
Comfortably numb	
We are young	
…	…
…	…
Chopin's 5th	???

Classification

What tools do you need for classification?

1. Data $S = \{(x_i, y_i)\}_{i = 1,...,n}$
   - x_i represents each example with d attributes
   - y_i represents the label of each example
2. Classification **model** $f_{(a,b,c,....)}$ with some parameters *a, b, c,...*
   - a model/function maps examples to labels
3. Loss function *L(y, f(x))*
   - how to penalize mistakes

Features $x_i = (x_{i1}, \ldots, x_{id})$

X	Y	Artist	Len.	...	F_d
Some nights		Fun	4:23		...
Skyfall		Adele	4:00		...
Comf. numb		Pink Fl.	6:13		...
We are young		Fun	3:50		...
...	...	...	...		...
...	...	...	...		...
Chopin's 5th	??	Chopin	5:32		...

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Training a classifier

Q: How do you learn appropriate values for *a, b, c, ...* such that

- (Part I) $y_i = f_{(a,b,c,\ldots)}(x_i),\ i = 1, \ldots, n$
  - Low/no error on the training set
- (Part II) $y = f_{(a,b,c,\ldots)}(x),$ for any new x
  - Low/no error on future queries (songs)

Possible A: Minimize $\sum_{i=1}^{n} L(y_i, f_{(a,b,c,..)}(x_i))$ with respect to *a, b, c,...*

Classification loss function

Most common loss: **0-1 loss function**

$$L_{0-1}(y, f(x)) = \mathbb{I}(y \neq f(x))$$

More general loss functions are defined by a *m x m* cost matrix *C* such that

$$L(y, f(x)) = C_{ab}$$

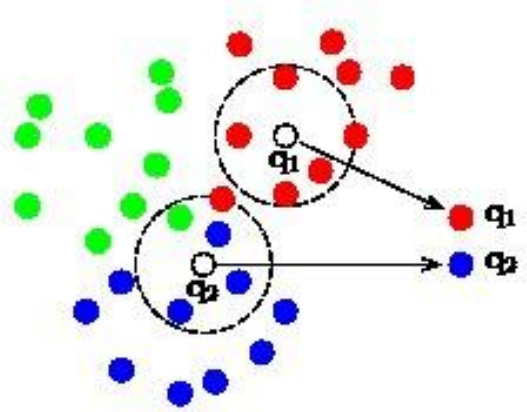
where $y = a$ and $f(x) = b$

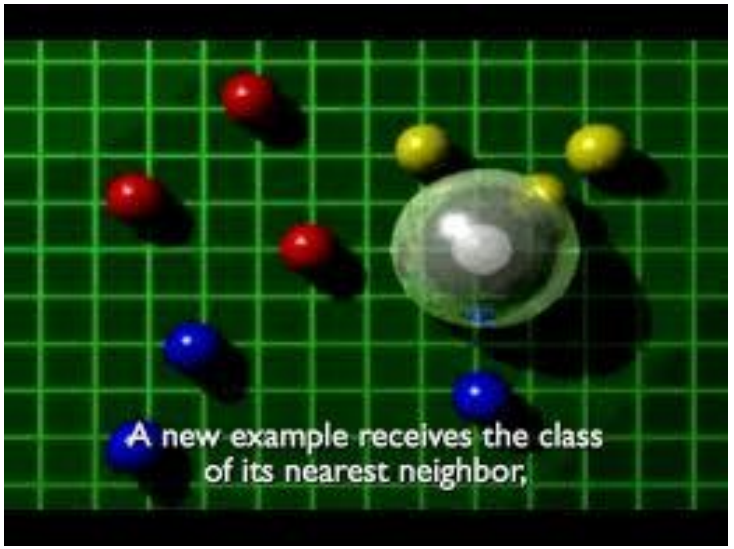
Class	T0	T1
P0	0	C_{10}
P1	C_{01}	0

T0 (true class 0), **T1** (true class 1)

P0 (predicted class 0), **P1** (predicted class 1)

k-Nearest-Neighbor Classifier





The classifier:

f(x) = majority label of the *k* nearest neighbors (NN) of *x*

Model parameters:

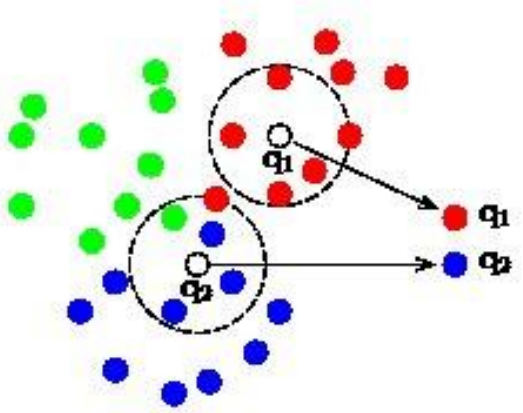
- number of neighbors *k*
- distance function *d(.,.)*

k-Nearest-Neighbor Classifier

If *k* and *d(.,.)* are fixed

Things to learn: ?

How to learn them: ?



If *d(.,.)* is fixed, but you can change *k*

Things to learn: ?

How to learn them: ?

$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$

k-Nearest-Neighbor Classifier

If *k* and *d(.,.)* are fixed

Things to learn: Nothing

How to learn them: N/A

If *d(.,.)* is fixed, but you can change *k*

Things to learn: Nothing

How to learn them: N/A

Selecting *k*: Try different values of *k* on some hold-out set

Cross-validation

Find the best performing *k*

1. Hold out a part of the training data
2. Train on the rest of the data
3. Evaluate the test error on the holdout set
4. Do this multiple times and pick the *k* with best performance
   - with respect to the error (on hold-out set) averaged over all hold-out sets

Cross-validation: Holdout sets

Leave-one-out cross-validation (LOO-CV)

- hold out sets of size 1

K-fold cross-validation

- hold sets of size *(n / K)*

Learning vs. Cross-validation

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

k-Nearest-Neighbor Classifier

If *k* is fixed, but you can change *d(.,.)*

Things to learn: ?

How to learn them: ?

Cross-validation: ?

Possible distance functions:

- Euclidean distance: $\|x_i - x_j\|_2 = \sqrt{(x_i - x_j)^\top (x_i - x_j)}$
- Manhattan distance: $\|x_i - x_j\|_1 = \sum_{l=1}^{d} |x_{il} - x_{jl}|$
- Mahalanobis distance: $\|x_i - x_j\|_\Sigma = \sqrt{(x_i - x_j)^\top \Sigma (x_i - x_j)}$

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

k-Nearest-Neighbor Classifier

If *k* is fixed, but you can change *d(.,.)*

Things to learn: distance function *d(.,.)*

How to learn them: optimization

Cross-validation: any regularizer you have on your distance function

Final words on k-NN classifier

- Advantages
  - No learning (unless you are learning the distance functions)
  - quite powerful in practice (and has theoretical guarantees as well)
- Caveats
  - Computationally expensive as test time

Reading material:

- ESL book, Chapter 13.3
  http://www-stat.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf
- Le Song's slides on kNN classifier
  http://www.cc.gatech.edu/~lsong/teaching/CSE6704/lecture2.pdf

Points about cross-validation

Requires (lot of) extra computation, but gives you information about expected test error

LOO-CV:

- Advantages
  - Unbiased estimate of test error (especially for small n)
  - Low variance
- Caveats
  - Extremely time consuming -- generally only practical for small data sets

Points about cross-validation

K-fold CV:

- Advantages
  - More efficient than LOO-CV
- Caveats
  - *K* needs to be large for low variance
  - Too small *K* leads to **under-use** of data, leading to higher bias
- Usually accepted value ***K* = 10**

Reading material:

- ESL book, Chapter 7.10
  http://www-stat.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf
- Le Song's slides on CV
  http://www.cc.gatech.edu/~lsong/teaching/CSE6704/lecture13-cv.pdf

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Bayes classifier

In a Bayes classifier,

$$f(x) = \arg\max_y P(Y = y | X = x)$$

By Bayes' rule

$$P(Y|X) = P(Y)\, P(X|Y) / P(X)$$

Classification can be done as

$$f(x) = \arg\max_y P(Y = y)\, P(X = x | Y = y)$$

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Bayes classifier

$f(x) = \arg\max_y P(Y = y)\, P(X = x | Y = y)$

Say you have a tool to learn any probability *P()* given some observations:

Things to learn: ?

How to learn them: ?

Cross-validation: ?

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Bayes classifier

$f(x) = arg\ max_y\ P(Y = y)\ P(X = x|Y = y)$

Say you have a tool to learn any probability *P()* given some observations:

Things to learn: $P(Y = y)$, $P(X|Y = y)$ for every class *y*

How to learn them: Using the tool

Cross-validation: None usually

$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$

Estimating the probability

P(Y = y) are the class weights and can be approximated from the training set

What about *P(X|Y = y)* ?

- Assume $X|Y = y \sim \mathcal{N}(\mu, \Sigma)$
  - Maximum-likelihood
- Estimate *P(X|Y = y)* with no assumptions
  - Kernel-density estimation

Generally a hard task if *d* is large!

$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$

Naive-Bayes classifier (NBC)

X is d-dimensional $(X_1,\ldots,X_d)$

How to learn $P(X|Y = y)$ for all classes?

The "naive" assumption:

$P(X|Y) = P(X_1|Y)*P(X_2|Y)*\ldots*P(X_d|Y)$

(Usual) further assumption:

$P(X_i|Y)$ is a known type of probability density/mass function

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Commonly chosen function

$$X_i | Y = j \sim \mathcal{N}(\mu_{ij}, \sigma_{ij})$$

Things to learn: ?

How to learn them: ?

Cross-validation: ?

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Commonly chosen function

$$X_i | Y = j \sim \mathcal{N}(\mu_{ij}, \sigma_{ij})$$

Things to learn: $\mu_{ij}, \sigma_{ij} \; \forall i, j$

How to learn them: Maximizing the log-likelihood of the observed data

Cross-validation: None
(unless you add some regularization to the log-likelihood to get *penalized log-likelihood*)

Further simplification of NBC

- Every class has the same variance

$$\sigma_{ij} = \sigma_j \; \forall i$$

- Every dimension has the same variance

$$\sigma_{ij} = \sigma_i \; \forall j$$

- Every class and dimension has the same variance

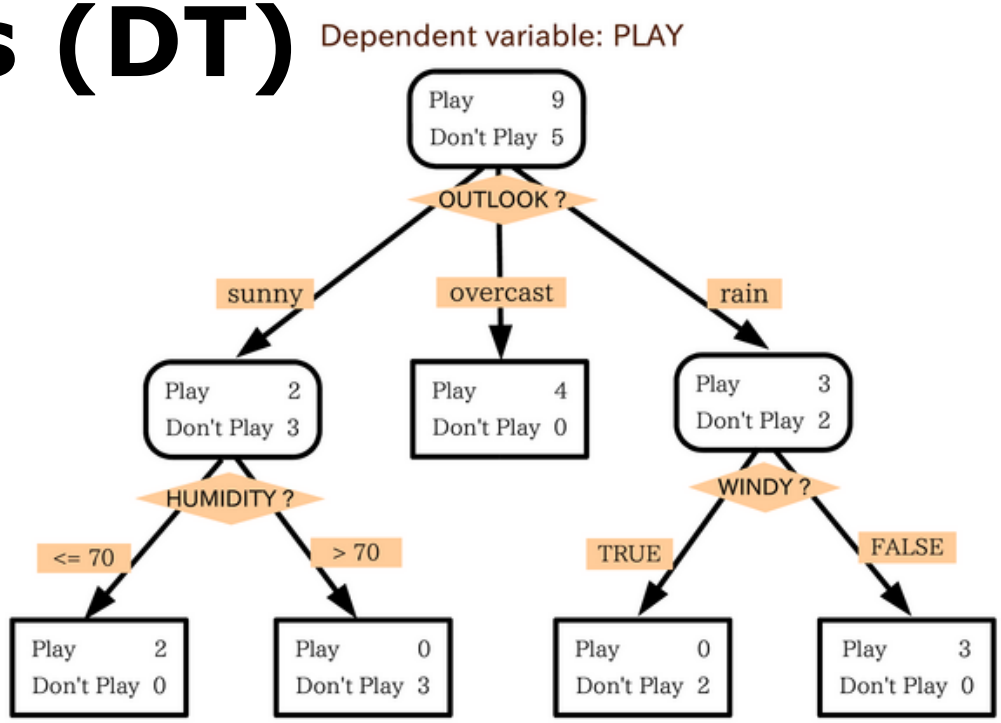
$$\sigma_{ij} = \sigma \; \forall i,j$$

Final words of NBC

- Advantages
  - Extremely simple -- efficient training
  - Not many tuning parameters
  - ** Works quite well for real datasets
  - Parallelizable
    - Each classes estimation can be done separately
- Caveats
  - Invalid when the assumptions do not hold
- Reading material
  - Le Song's slides
    http://www.cc.gatech.edu/~lsong/teaching/CSE6704/lecture3.pdf

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Decision trees (DT)



The classifier:

$f_T(x)$ is the majority class in the leaf in the tree T containing x

Model parameters: The tree structure and size

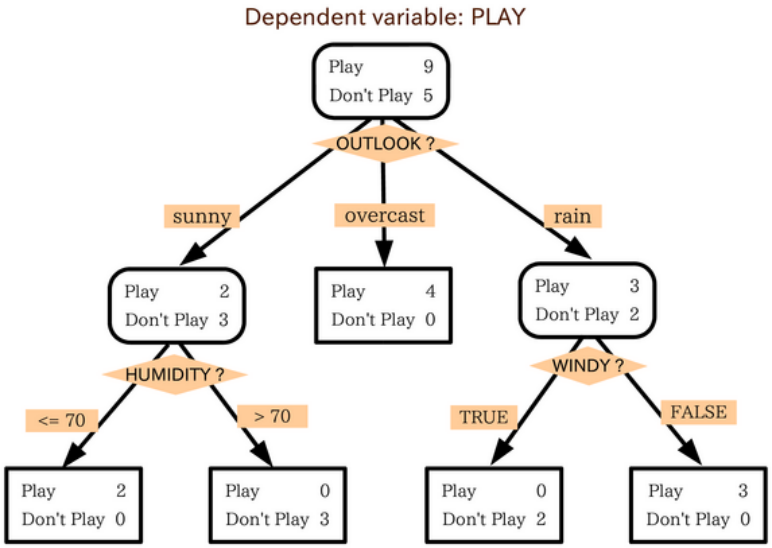
$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Decision trees

Things to learn: ?

How to learn them: ?

Cross-validation: ?



$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Decision trees

Things to learn: the tree structure

How to learn them: (greedily) minimize the overall classification loss

Cross-validation: finding the best sized tree with *K*-fold cross-validation

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Learning the tree structure

Pieces:

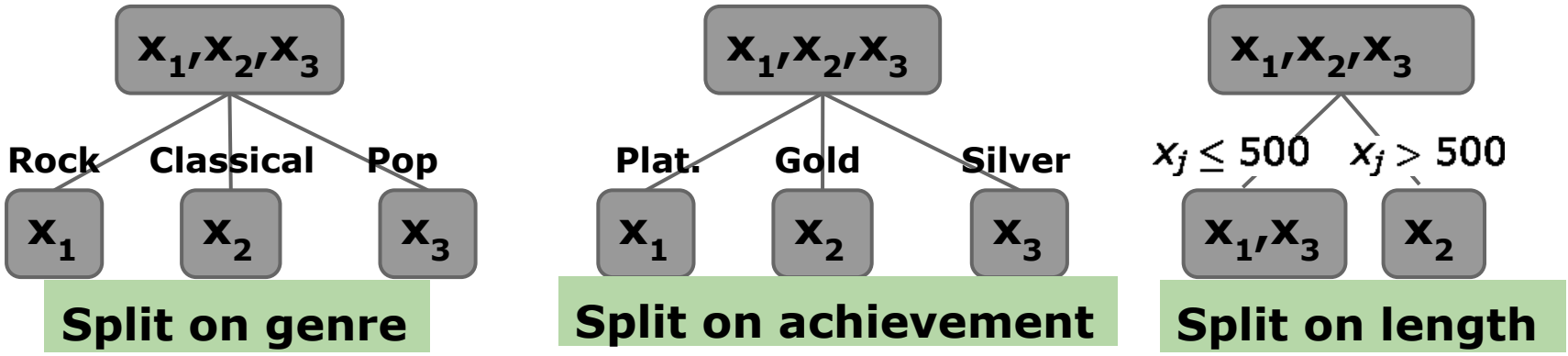
1. best split on the chosen attribute
2. best attribute to split on
3. when to stop splitting
4. cross-validation

$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$

Choosing the split

Split types for a selected attribute *j*:

1. Categorical attribute (e.g. `genre′)
   x_{1j} = *Rock*, x_{2j} = *Classical*, x_{3j} = *Pop*
2. Ordinal attribute (e.g. `achievement′)
   x_{1j}=*Gold*, x_{2j}=*Platinum*, x_{3j}=*Silver*
3. Continuous attribute (e.g. song length)
   x_{1j} = *235*, x_{2j} = *543*, x_{3j} = *378*



$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Choosing the split

At a node T for a given attribute d, select a split s as following:

$$min_s\ loss(T_L) + loss(T_R)$$

where $loss(T)$ is the loss at node T

Node loss functions:

- Total loss: $\sum_{x_i \in T} L(y_i, f_T(x_i))$
- Cross-entropy: $-\sum_{c \in T} p_{cT} \log p_{cT}$
  where p_{cT} is the proportion of class c in node T

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Choosing the attribute

Choice of attribute:

1. Attribute providing the maximum improvement in training loss

$$\min_{j \in \{1,\ldots,d\}} \min_{s(j)} loss(T_L) + loss(T_R)$$

2. Attribute with maximum information gain

$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$

When to stop splitting?

1. Homogenous node (all points in the node belong to the same class OR all points in the node have the same attributes)
2. Node size less than some threshold
3. Further splits provide no improvement in training loss
   ($loss(T) <= loss(T_L) + loss(T_R)$)
4. If the class distribution is independent of the attribute values -- information gain is zero for every attribute

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Controlling tree size

In most cases, you can drive training error to zero (how? is that good?)

What is wrong with really deep trees?

- Really high "variance"

What can be done to control this?

- *Regularize* the tree complexity
  - Penalize complex models and prefers simpler models (why?)

Look at Le Song's slides on the decomposition of error in bias and variance of the estimator http://www.cc.gatech.edu/~lsong/teaching/CSE6704/lecture13-cv.pdf

$$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$$

Regularization

"Regularized training" minimizes

$$\sum_{i=1}^{n} L(y_i, f_{(a,b,c,\ldots)}(x_i)) + C * M(f_{(a,b,c,\ldots)})$$

where *M()* denotes complexity of a function, and *C* is called the "regularization parameter"

Cross-validate for *C* selected from a discrete set $\{C_1, \ldots, C_m\}$

- Compute CV error for each value of C_j
- Select C_j with lowest CV error

$x_i = (x_{i1}, \ldots, x_{id}); y_i = \{1, \ldots, m\}$

Regularization in DT

Cost-complexity pruning:

$M(f_T)$ = # of leaves in T

Let $S(T)$ denote the set of leaves L in the subtree T. Then the regularized cost of the subtree rooted at node T:

$$RegLoss(T) = \sum_{L \in S(T)} loss(L) + C * card(S(T))$$

If $RegLoss(T) \geq loss(T) + C * 1$

replace the subtree with T as a leaf

Cross-validation

Cross-validation steps:

- For each value in the set $\{C_1,\ldots,C_N\}$
  1. Train on the non-holdout set and regularize with C_j
  2. Compute error on holdout set
  3. Pick C_j with the lowest average error on the holdout sets
  4. Prune the tree on the whole training set with the chosen C_j

Final words on decision trees

- Advantages
  - Easy to implement
  - Interpretable
  - Very fast test time
  - Can work seamlessly with mixed attributes
  - ** Works quite well in practice
- Caveats
  - Can be too simplistic
  - Training can be very expensive
  - Cross-validation is hard (and plain annoying)

Final words on decision trees

Reading material:

- ESL book, Chapter 9.2
  http://www-stat.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf
- Le Song's slides
  http://www.cc.gatech.edu/~lsong/teaching/CSE6704/lecture6.pdf

Method	**Coding**	**Training time**	**Cross validation**	**Testing time**	**Accuracy**
kNN classifier		None	Can be slow	Slow	??
Naive Bayes classifier		Fast	None	Fast	??
Decision trees		Slow	Very slow	Very fast	??