CS840a: Machine Learning in Computer Vision

Olga Veksler

Lecture 1

Introduction

Nearest Neighbor

Outline

- Course overview
- Introduction to Machine Learning
- Simplest Machine Learning Technique: Nearest Neighbors

Course Outline

- Prerequisite
  - First-year course in Calculus
  - Introductory Statistics
  - Linear Algebra
  - Some Computer Vision/Image Processing
- Grading
  - Class participation 10%
  - In class paper presentation 30%
  - Final Project Presentation 20%
  - Written project report + code, 40 %
    - Matlab, C/C++, anything else as long as I can run it

Course Outline: Content

- Lecture (1/3 of the time), paper presentation/discussions/video (2/3 of the time)
- Machine Learning Methods (tentatively)
  - Nearest neighbor
  - Linear classifiers
  - Neural nets
  - SVM
  - Boosting
- Applications in Computer Vision
  - Object detection/recognition
  - Segmentation
  - Tracking
  - Inpainting

Course Outline: Textbook

- No required textbook, but recommended
  - "Pattern Classification" by R.O. Duda, P.E. Hart and D.G. Stork, second edition
  - "Machine Learning" by Tom M. Mitchell
- Conference papers, provided

Intro: What is Machine Learning?

- How to write a computer program that automatically improves its performance through experience
- Machine learning is useful when it is too difficult to come up with a program to perform a desired task
- Make computer to learn by showing examples (most frequently with correct answers)
  - "supervised" learning or learning with a teacher
- In practice: computer program (or function) which has a tunable parameters, tune parameters until the desirable behavior on the examples

Different Types of Learning

- Supervised Learning: given training examples of inputs and corresponding outputs, produce the "correct" outputs for new inputs
- Reinforcement Learning (similar to animal learning): an agent takes inputs from the environment, and takes actions that affect the environment. Occasionally, the agent gets a reward or punishment. The goal is to learn to produce action sequences that maximize the expected reward (e.g. driving a robot without bumping into obstacles). Not covered in this course
- Unsupervised Learning: given only inputs as training, find structure in the world: e.g. discover clusters

slide is modified from Y. LeCun

Sketch of Machine Learning *(supervised)*

- Modeling stage:
  - Collect a set of **training** examples with correct answers: (x_1,y_1), (x_2,y_2),…, (x_k,y_k)
    - x_i= features of the example, usually a vector, also called "input"
    - y_i= correct answer for the example, usually a scalar, also called "output"
  - Choose a function f(x,t), where t are the tunable parameters, x is the feature vector, and the function outputs the "correct" answer for training example x
- Training stage:
  - Repeatedly present examples (x_i,y_i) to the function f(x,t), and change parameters t so that f(x,t) gives the correct answer y_i for most examples x_i
- Evaluation stage:
  - Evaluate how well your function f(x,t) is able to predict the answers for examples it hasn't seen so far

Sketch of Machine Learning (supervised)

- None of the stages are easy
- Modeling stage:
  - Which features do we extract from training data (which are usually images in vision). How many features?
- Training stage:
  - Which function f(x,t) do we choose? Has to be expressive enough to model our problem, yet not to complicated to avoid ***overfitting***
  - How do we tweak parameters t to ensure f(x,t) = y for most training samples (x,y) ? This step is usually done by optimization, can be quite expensive.
- Evaluation stage
  - Good performance on the training data does not guarantee good performance on data we haven't seen yet. In fact, no error on training data frequently means that we overfitted to the training data

Application: male or female?

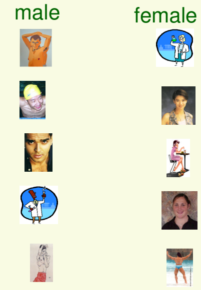
Objects (pictures)

Perfect

PR system

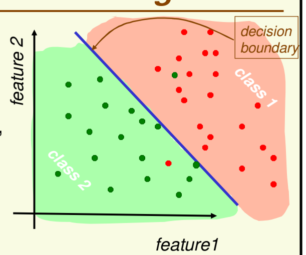
classes

male | female

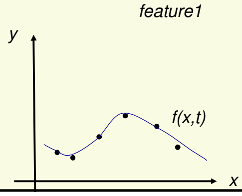


Two types of Machine Learning

1. **Classification** (mostly deal with in this course)
   - outputs y_i are discrete, represent categories (ex.: object categories face, car, etc.)
   - Usually visualize decision regions and decision boundary
   - f(x,t) is usually called classifier

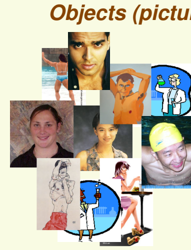


2. **Regression:**
   - outputs y_i are continuous, example: temperature
   - This is also called "curve fitting"



Application: photograph or not?

Objects (pictures)

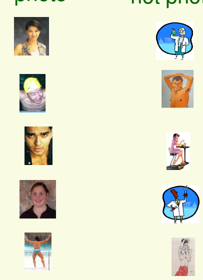


Perfect

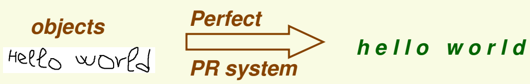
PR system

classes

photo | not photo



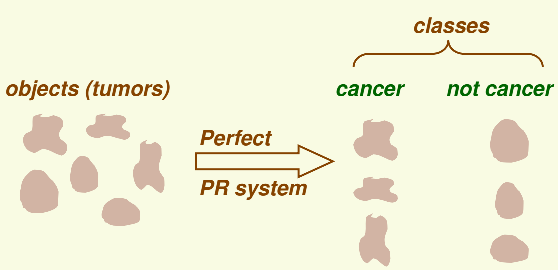
Application: Character Recognition



- In this case, the classes are all possible characters: **a**, **b**, **c**,…., **z**

13

Application: Medical diagnostics



14

How to design a Classification system?

- Collect data and classify by hand

- Preprocess by segmenting fish from background

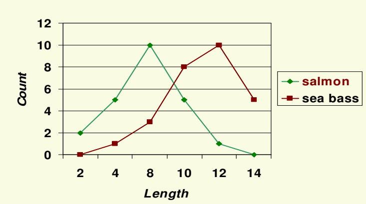
- Extract possibly discriminating features
  - length, lightness,width,number of fins,etc.
- Classifier design
  - Choose model
  - Train classifier on part of collected data (training data)
- Test classifier on the rest of collected data (test data) i.e. the data not used for training
  - Should classify new data (new fish images) well

15

Classifier design

- Notice salmon tends to be shorter than sea bass
- Use *fish length* as the discriminating feature
- Count number of bass and salmon of each length

	2	4	8	10	12	14
bass	0	1	3	8	10	5
salmon	2	5	10	5	1	0



16

Fish length as discriminating feature

- Find the best length ***L*** threshold

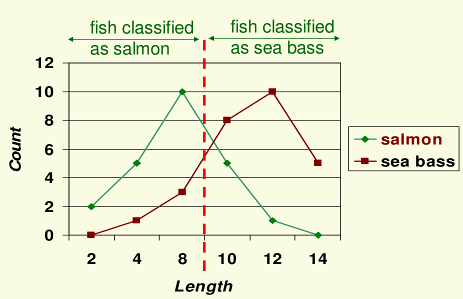
fish length < L	***fish length > L***
classify as salmon	***classify as sea bass***

- For example, at ***L*** = 5, misclassified:
  - 1 sea bass
  - 16 salmon

	2	4	8	10	12	14
bass	0	1	3	8	10	5
salmon	2	5	10	5	1	0

- Classification error (total error): $\frac{17}{50}$ = ***34%***

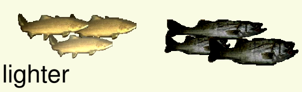
Fish Length as discriminating feature



- After searching through all possible thresholds ***L***, the best ***L***= 9, and still 20% of fish is misclassified

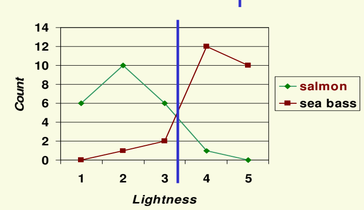
Next Step

- Lesson learned:
  - Length is a poor feature alone!
- What to do?
  - Try another feature
  - Salmon tends to be lighter
  - Try average fish lightness



Fish lightness as discriminating feature

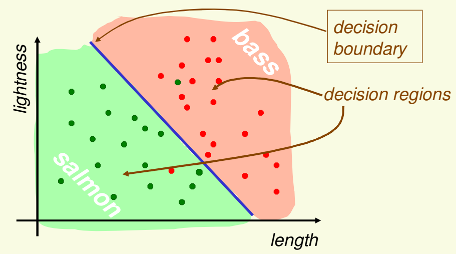
	1	2	3	4	5
bass	0	1	2	10	12
salmon	6	10	6	1	0



- Now fish are well separated at lightness threshold of 3.5 with classification error of 8%

Can do even better by feature combining

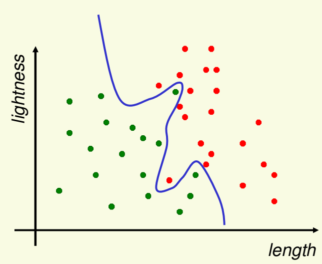
- Use both *length* and *lightness* features
- Feature vector [*length*,*lightness*]



- Classification error 4%

21

Better decision boundary

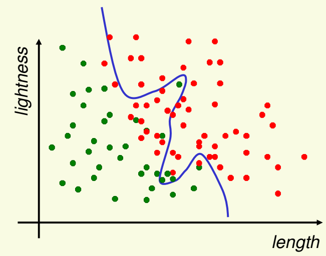


- Ideal decision boundary, 0% classification error

22

Test Classifier on New Data

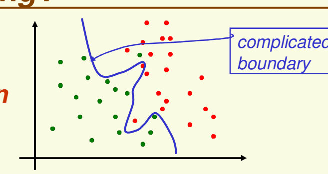
- Classifier should perform well on new data
- Test "ideal" classifier on new data: 25% error



23

What Went Wrong?

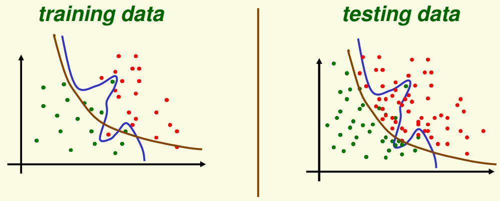
- Poor ***generalization***



- Complicated boundaries do not generalize well to the new data, they are too "tuned" to the particular training data, rather than some true model which will separate salmon from sea bass well.
  - This is called overfitting the data

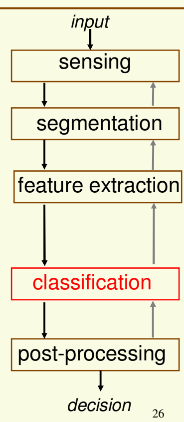
24

Generalization



- Simpler decision boundary does not perform ideally on the training data but generalizes better on new data
- Favor simpler classifiers
  - William of Occam (1284-1347): “entities are not to be multiplied without necessity”

System Structure



domain dependent

camera, microphones, medical imaging devices, etc.

Patterns should be well separated and should not overlap.

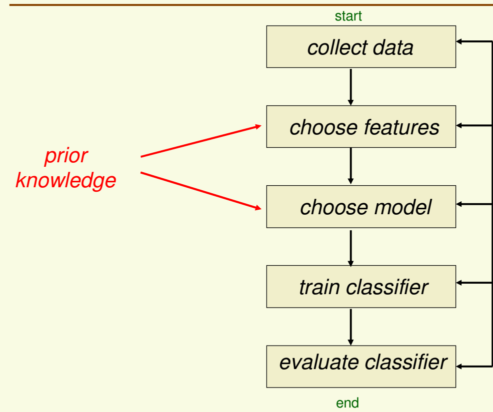
Extract discriminating features. Good features make the work of classifier easy.

Use features to assign the object to a category. Better classifier makes feature extraction easier.
Our main topic in this course

Exploit context (input depending information) to improve system performance

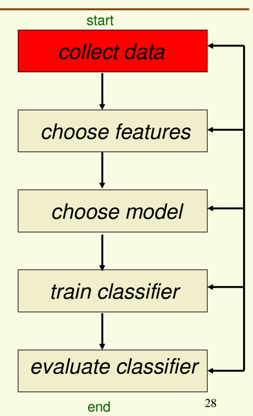
Tne cat ➡ ***The cat***

How to design a classification system?



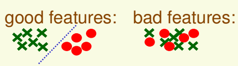
Design Cycle cont.

- Collect Data
  - Can be quite costly
  - How do we know when we have collected an adequately representative set of testing and training examples?

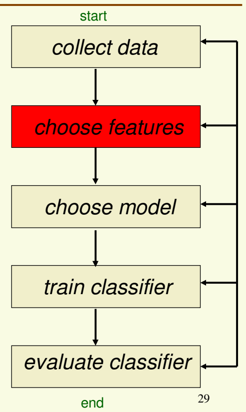


Design Cycle cont.

- Choose features
  - Should be discriminating, i.e. similar for objects in the same category, different for objects in different categories:



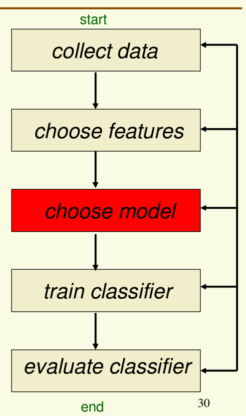
  - Prior knowledge plays a great role (domain dependent)
  - Should be easy to extract
  - Insensitive to noise and irrelevant transformations



29

Design Cycle cont.

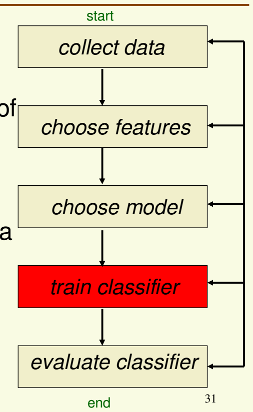
- Choose model
  - What type of classifier to use?
  - When should we try to reject one model and try another one?
  - What is the best classifier for the problem?



30

Design Cycle cont.

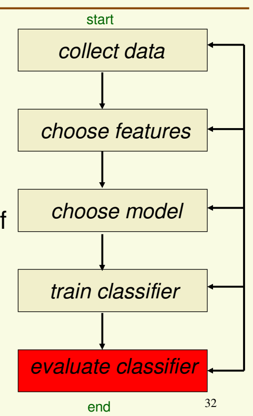
- Train classifier
  - Process of using data to determine the parameters of classifier
  - Change parameters of the chosen model so that the model fits the collected data
  - Many different procedures for training classifiers
  - Main scope of the course



31

Design Cycle cont.

- Evaluate Classifier
  - measure system performance
  - Identify the need for improvements in system components
  - How to adjust complexity of the model to avoid over-fitting? Any principled methods to do this?
  - Trade-off between computational complexity and performance



32

Learning is NOT Memorization

- rote learning is easy: just memorize all the training examples and their corresponding outputs
- When a new input comes in, compare it to all the memorized samples, and produce the output associated with the matching sample
- PROBLEM: in general, new inputs are different from training samples
- The ability to produce correct outputs or behavior on previously unseen inputs is called GENERALIZAITION
- Rote learning is memorization without generalization
- The big question of Learning Theory (and practice): how to get good generalization with a limited number of examples

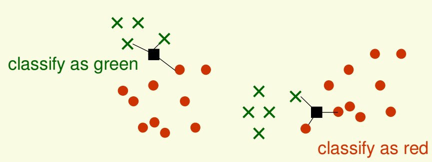
slide is modified from Y. LeCun

kNN: How Well Does it Work?

- kNN rule is certainly simple and intuitive, but does it work?
- Assume we have an unlimited number of samples
- Theoretically, the best possible error rate is the Bayes rate ***E****
  - Bayes error rate is the best error rate a classifier can have, but we do not study it in this course
- Nearest-neighbor rule leads to an error rate greater than ***E****
- But even for ***k*** =1, as $n \rightarrow \infty$, it can be shown that nearest neighbor rule error rate is smaller than ***2E****
- As we increase ***k***, the upper bound on the error gets better and better, that is the error rate (as $n \rightarrow \infty$) for the ***kNN*** rule is smaller than ***cE****, with smaller ***c*** for larger ***k***
- If we have a lot of samples, the kNN rule will do very well !

k-Nearest Neighbors

- find k closest neighbors
- Classify unknown point with the most common class



- How to choose ***k***?
- A good "rule of thumb" is $k = \sqrt{n}$, where ***n*** is the number of samples
  - Interesting theoretical properties
- In practice, ***k*** = 1 is often used
- Can find the best ***k*** through cross-validation, to be studied later

1NN: Voronoi Cells

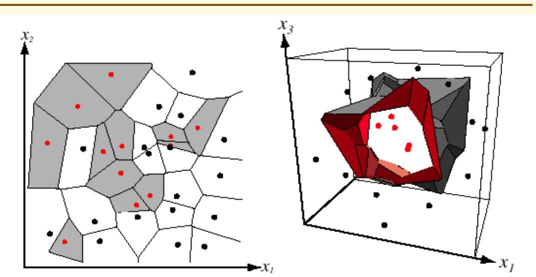
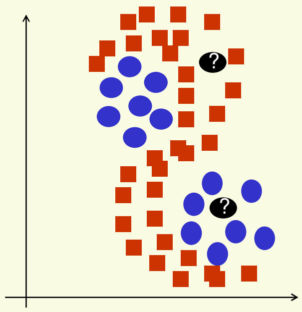


FIGURE 4.13. In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

kNN: Multi-Modal Distributions

- Most parametric distributions would not work for this 2 class classification problem:
- Nearest neighbors will do reasonably well, provided we have a lot of samples



37

kNN: How to Choose k?

- In theory, when the infinite number of samples is available, the larger the ***k***, the better is classification (error rate gets closer to the optimal Bayes error rate)
- But the caveat is that all ***k*** neighbors have to be close to ***x***
  - Possible when infinite # samples available
  - Impossible in practice since # samples is finite

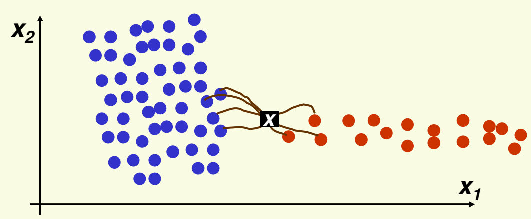
38

kNN: How to Choose k?

- In practice
  1. ***k*** should be large so that error rate is minimized
     - ***k*** too small will lead to noisy decision boundaries
  2. ***k*** should be small enough so that only nearby samples are included
     - ***k*** too large will lead to over-smoothed boundaries
- Balancing ***1*** and ***2*** is not trivial
  - This is a recurrent issue, need to smooth data, but not too much

39

kNN: How to Choose k?



- For ***k*** = 1, …,7 point ***x*** gets classified correctly
  - red class
- For larger ***k*** classification of ***x*** is wrong
  - blue class

40

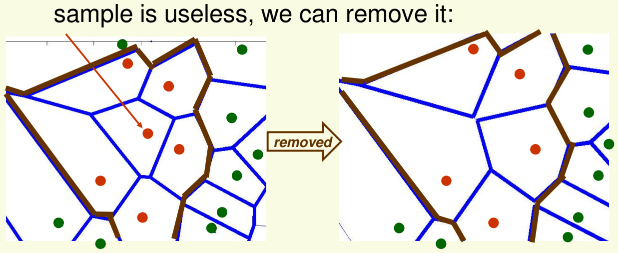
kNN: Computational Complexity

- Basic ***kNN*** algorithm stores all examples. Suppose we have ***n*** examples each of dimension ***d***
  - $O(d)$ to compute distance to one example
  - $O(nd)$ to find one nearest neighbor
  - $O(knd)$ to find k closest examples examples
  - Thus complexity is $O(knd)$
- This is prohibitively expensive for large number of samples
- But we need large number of samples for ***kNN*** to work well!

41

Reducing Complexity: Editing 1NN

- If all voronoi neighbors have the same class, a sample is useless, we can remove it:

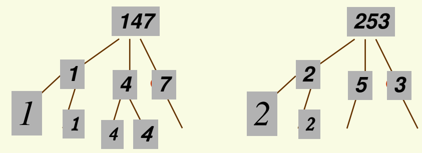


- Number of samples decreases
- We are guaranteed that the decision boundaries stay the same

42

Reducing Complexity: kNN prototypes

- Explore similarities between samples to represent data as search trees of *prototypes*



- Advantages: Complexity decreases
- Disadvantages:
  - finding good search tree is not trivial
  - will not necessarily find the closest neighbor, and thus **not** guaranteed that the decision boundaries stay the same

43

kNN: Selection of Distance

- So far we assumed we use Euclidian Distance to find the nearest neighbor:

$$D(a,b) = \sqrt{\sum_k (a_k - b_k)^2}$$

- However some features (dimensions) may be much more discriminative than other features (dimensions)
- Euclidean distance treats each feature as equally important

44

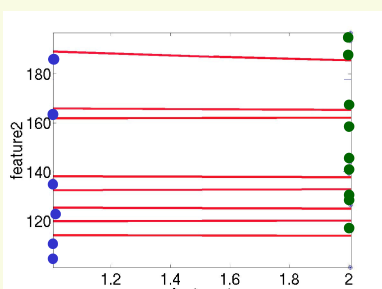
kNN: Selection of Distance

- Extreme Example
  - feature 1 gives the correct class: 1 or 2
  - feature 2 gives irrelevant number from 100 to 200
- Suppose we have to find the class of x=[1 100] and we have 2 samples [1 150] and [2 110]

$$D\left(\begin{bmatrix}1\\100\end{bmatrix},\begin{bmatrix}1\\150\end{bmatrix}\right)=\sqrt{(1-1)^2+(100-150)^2}=50 \qquad D\left(\begin{bmatrix}1\\100\end{bmatrix},\begin{bmatrix}2\\110\end{bmatrix}\right)=\sqrt{(1-2)^2+(100-110)^2}=10.5$$

- x = [1 100] is misclassified!
- The denser the samples, the less of the problem
  - But we rarely have samples dense enough

kNN: Extreme Example of Distance Selection

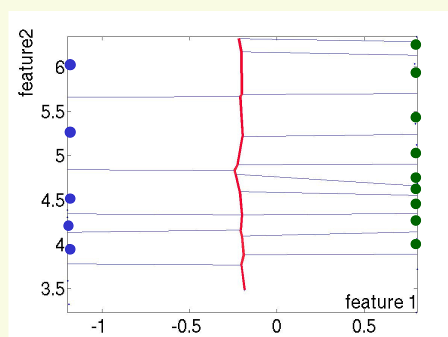


- decision boundaries for blue and green classes are in red
- These boundaries are really bad because
  - feature 1 is discriminative, but it's scale is small
  - feature 2 gives no class information (noise) but its scale is large

kNN: Selection of Distance

- Notice the 2 features are on different scales:
  - feature 1 takes values between 1 or 2
  - feature 2 takes values between 100 to 200
- We could normalize each feature to be between of mean 0 and variance 1
- If $\boldsymbol{X}$ is a random variable of mean μ and variance σ^2, then $(\boldsymbol{X} - \mu)/\sigma$ has mean 0 and variance 1
- Thus for each feature vector $\boldsymbol{x}_i$, compute its sample mean and variance, and let the new feature be $[\boldsymbol{x}_i - \text{mean}(\boldsymbol{x}_i)]/\text{sqrt}[\text{var}(\boldsymbol{x}_i)]$
- Let's do it in the previous example

kNN: Normalized Features



- The decision boundary (in red) is very good now!

kNN: Selection of Distance

- However in high dimensions if there are a lot of irrelevant features, normalization will not help

$$D(a,b) = \sqrt{\sum_k (a_k - b_k)^2} = \sqrt{\underbrace{\sum_i (a_i - b_i)^2}_{\text{discriminative feature}} + \underbrace{\sum_j (a_j - b_j)^2}_{\text{noisy features}}}$$

- If the number of discriminative features is smaller than the number of noisy features, Euclidean distance is dominated by noise

kNN: Feature Weighting

- Scale each feature by its importance for classification

$$D(a,b) = \sqrt{\sum_k w_k (a_k - b_k)^2}$$

- Can learn the weights w_k from the validation data
  - Increase/decrease weights until classification improves

kNN Summary

- Advantages
  - Can be applied to the data from any distribution
  - Very simple and intuitive
  - Good classification if the number of samples is large enough
- Disadvantages
  - Choosing best ***k*** may be difficult
  - Computationally heavy, but improvements possible
  - Need large number of samples for accuracy
    - Can never fix this without assuming parametric distribution