Summarizing Complexity in High Dimensional Spaces

Karl Young
University of California, San Francisco
SciPy 2008

The Context

- **Goal** – Obtain sensitive measures for detection and classification of disease state from multimodal medical imaging data

- Example of mining multimodal data (multidimensional feature space) with spatial coordinates (multidimensional index space) – LANDSAT data is another typical example

Some Current Approaches Used In MRI Analysis

- Global Measures
  - Boundary Shift Integral
  - Ventricular Dilatation
  - Regional Volumes, e.g. Hippocampus

- Local Measures
  - Voxel Based Morphometry
  - Tensor Based Morphometry
  - Cortical Thickness

Seek Method That Utilizes Global AND Local Information

Proposal

- Use information theory based complexity estimates to summarize multimodal image data
- Use these complexity estimates as input to learning algorithms
- Provides a nonparametric, multivariate method that utilizes information across a range of scales

Why Complexity Measures ?

- What Is Complexity ?
  - Many strongly interacting components introduce inherent uncertainty into the observation of **complex** (nonlinear) systems like the brain.
- How To Measure It ?
  - Use information theoretic measures that summarize general properties of **complex** (nonlinear) systems like the brain

A Brief (And Highly Biased) History Of Complexity Measures

- Godel, Turing – non-computability a precursor to notions of complexity
- Hartamis, Stearns – space, time associated with algorithms - time = f(length of input)
- Cook, Karp – P =? NP – Notion of hierarchies

A Brief (And Highly Biased) History Of Complexity Measures

- Kolmogorv – Complexity of Algorithm = Entropy
- Chomsky – Hierarchical Classification of Languages Via Complexity of Grammar
- Dynamics – Predictability of Nonlinear System Proportional to Complexity of Grammar Reconstructed From Time Series Data Where:
  - Complexity ~ Information Contained in Distribution Over "Sentences" Determined by "Grammar" (i.e. Dynamics)

A Brief (And Highly Biased) History Of Complexity Measures

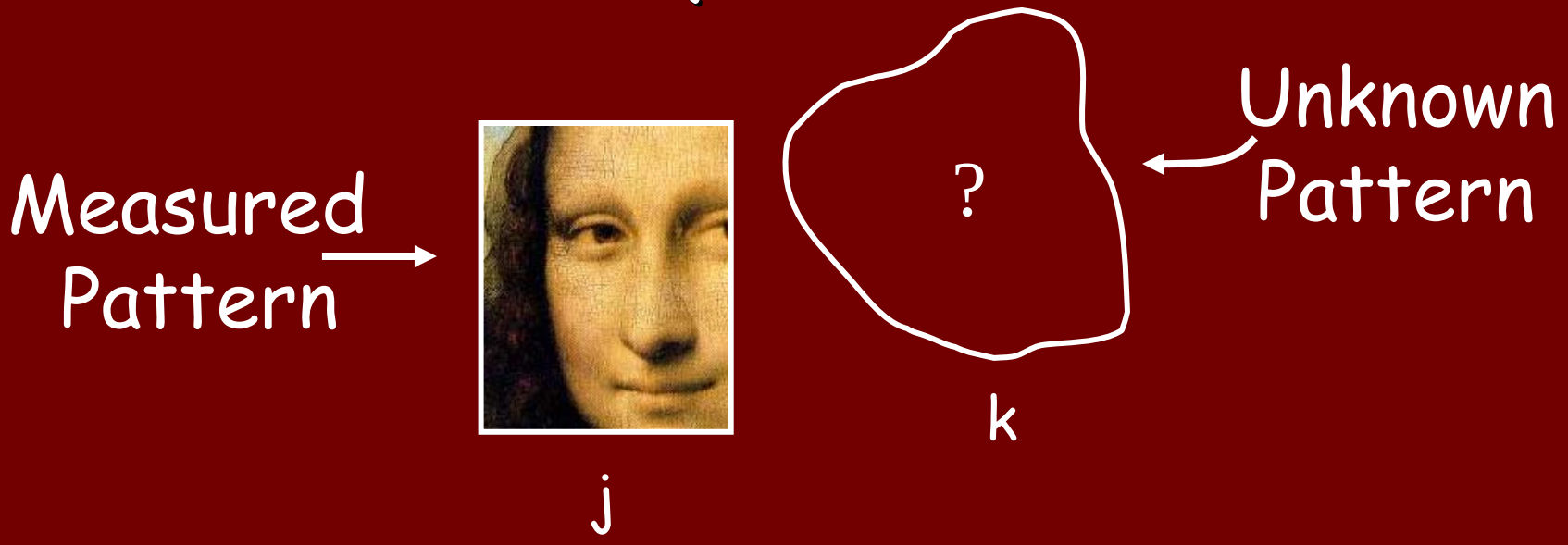
- Early Attempts To Develop Rigorous Measures Of Spatial Complexity Foundered In Ways Similar to:
  - Time Series Analysis -> Spatial Statistics
- Heuristics For Statistical Space-Time Complexity Measures Nonetheless Useful (Hopefully Demonstrated By Current Analysis)

Specific Complexity Measures Used For Multimodal Image Analysis

- Entropy (H) – measures number, and uniformity of distribution over observed image patterns
- Statistical Complexity (SC) – measures correlations over observed patterns
- Excess Entropy (EE) – measures variation in distribution of patterns as a function of scale

Space/Time Complexity Estimates

- Fundamental Question:



On average how well does measured pattern "at j" help predict unknown pattern "at k" ?

Space/Time Complexity Estimates

- Idea is to learn something about the "grammar" used by the system that generated the "images" (i.e. constraints used by the system in assembling the spatial distribution of multidimensional features)

Image Analysis Proceeds in 4 Stages

I. Choice of appropriate Feature Space (e.g. combination of structural MRI, DTI, MRSI,...)

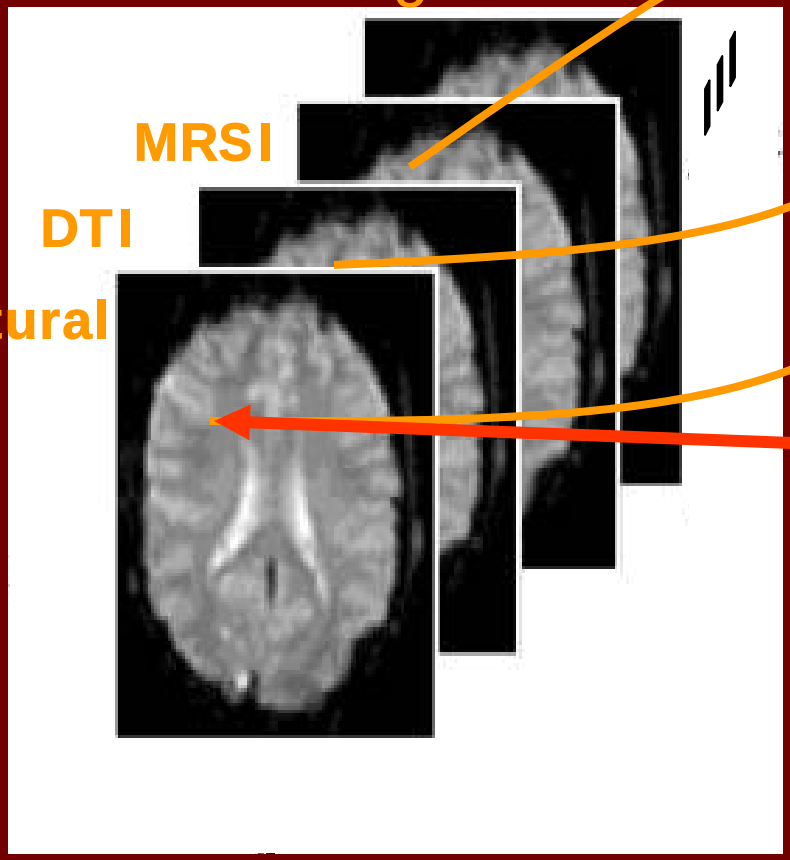
II. Segmentation (Clustering) of Feature Space

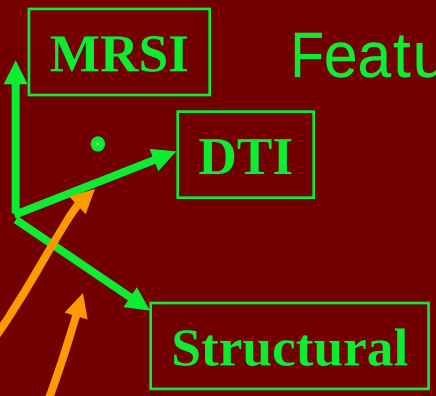
III. Generation of Complexity Estimates From Image of Clustered Values

IV. Classification Based on Complexity Estimates (e.g. supervised or unsupervised)

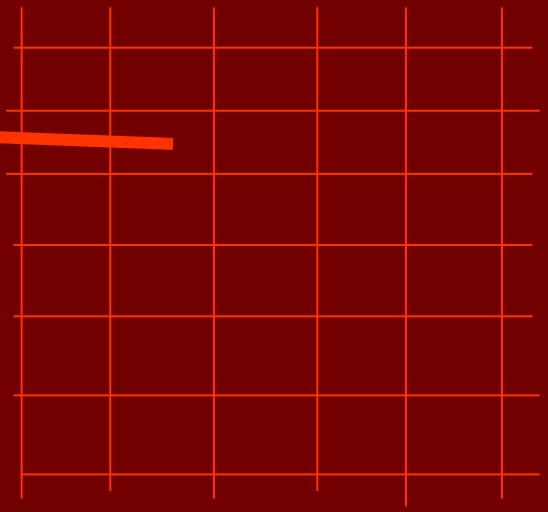
Stage I – Choice of Appropriate Feature Space

Co-registered, warped, interpolated/smoothed images





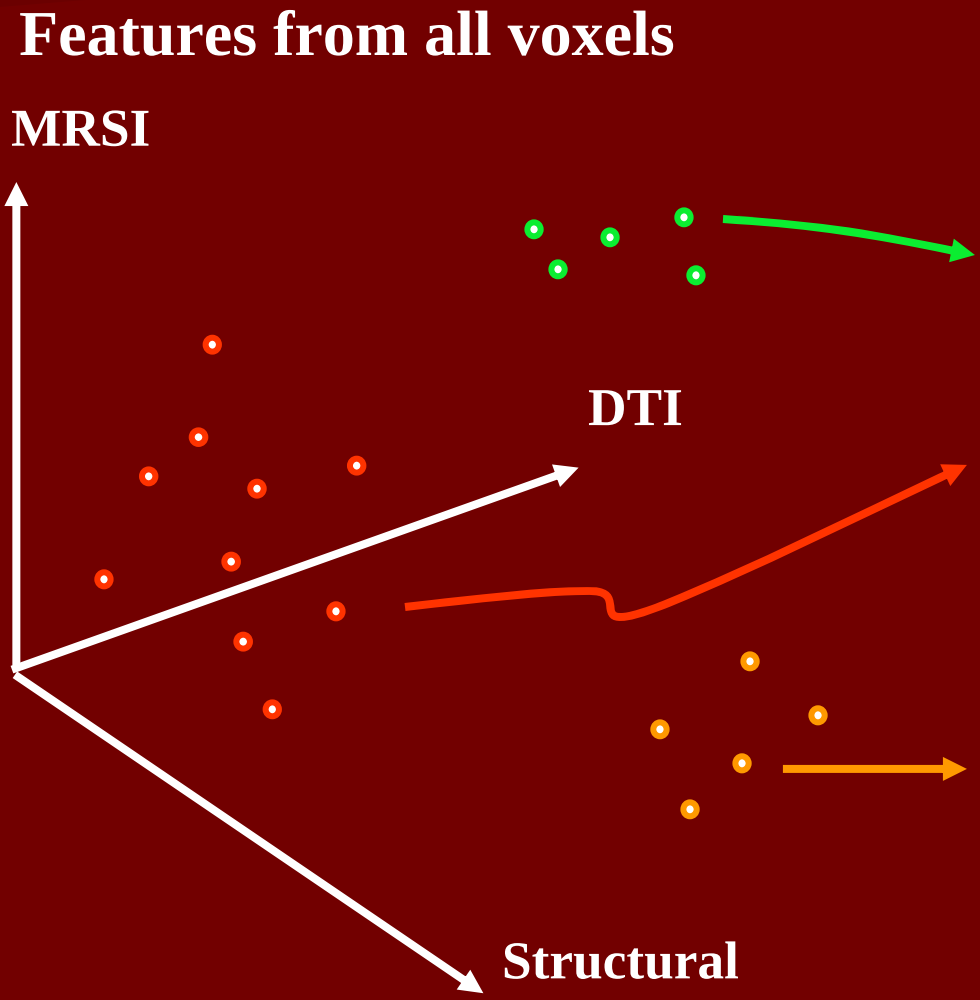
Voxel Grid

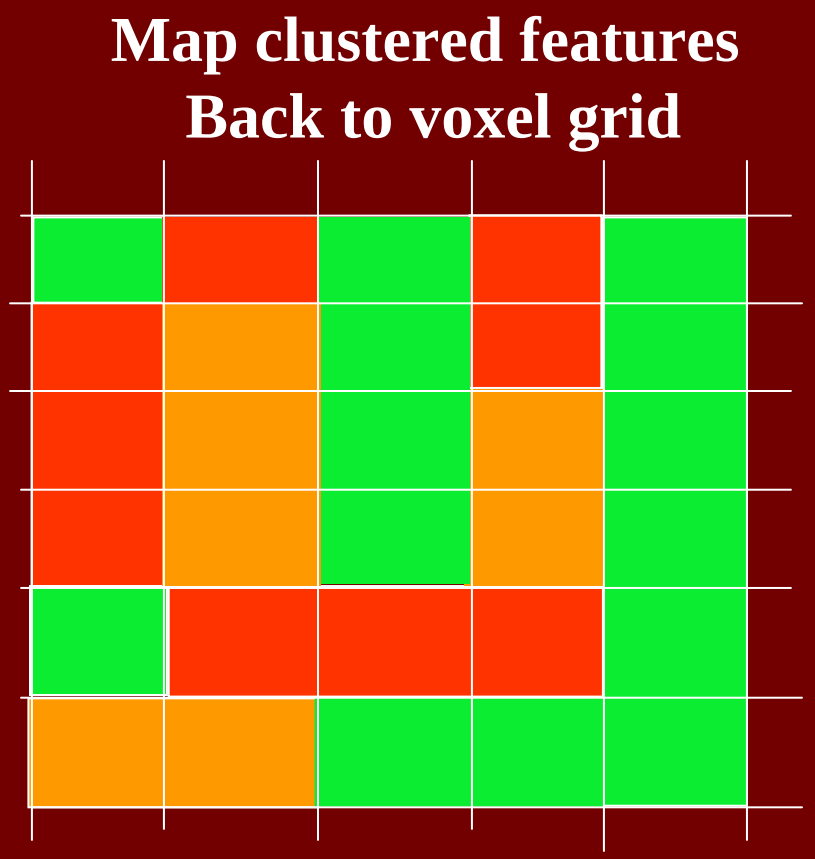


Stage I – Generate Feature Space

```
...
import numpy as N
...
def make_FeatureSpace(images,mask,pathlength):
...
    # Make a dictionary indexed by start point, and direction
    pathdic = {}
    # define a "ruler" of ones of length totpath to slide across mask
    ruler = N.ones([l],N.int)
...
    elif dims == 2:
        for i in range(mask.shape[0]):
            for j in range(mask.shape[1] - pathlength + 1):
                temp = ruler*mask[i,j:j+l]
                if len(temp) == len(N.compress(temp != 0,temp)):
                    if pathlength > 1:
                        pathdic[(i,j,2)] = pathnum
                        pathdic[(i,j,-2)] = pathnum+1
                        pathnum += 2
                    for p in range(len(images)):
                        for q in range(pathlength):
                            newpt1[0,p*l+q] = images[p][i,j+q]
                            if pathlength > 1:
                                newpt2[0,p*pathlength+q] = images[p][i,j+pathlength-q-1]
                    if (not N.array(featsp).any()):
...
                    else:
                        if pathlength > 1:
                            featsp = N.concatenate((featsp,newpt1,newpt2),0)
                        else:
                            featsp = N.concatenate((featsp,newpt1),0)
...
    return featsp,pathdic
```

Stage II - Segment Feature Space (I.e. Find Clusters) and Map Cluster Values Back to Voxel Grid





Stage II - Segment Feature Space

```
...
from rpy import *
...
def call_Pam(datmat,type,clusforce,maxtry):
    if type == 'data':
        dist = False # data matrix
    else:
        dist = True # distance matrix
    r.library('cluster')
    if clusforce:  # force number of clusters = clusforce
        pamclus = r.pam(datmat,clusforce,dist)
        sfin = pamclus["clustering"]
        maxclasses = clusforce
        smax =  pamclus["silinfo"]["avg.width"]
    else:
        if maxtry:
            maxtr = maxtry
        else:
            maxtr = datmat.shape[0]
        maxcl = 0  (smax = 0.0, sfin = 0, stmp = 0)
        for i in range(2,maxtr):
            pamclus = r.pam(datmat,i,dist)
            sil = pamclus["clustering"]
            stmp = sil
            silly =  pamclus["silinfo"]["avg.width"]
            if silly > smax:
                clusfinal = pamclus
                sfin = stmp
                smax = silly
                maxclasses = i
    return sfin,maxclasses,smax
```

Stage III - Generation of Complexity Estimates

- Generate joint distribution by parsing labeled image
- Calculate complexity measures from joint distribution

Stage III – Generate Histogram (Joint Density Estimate)

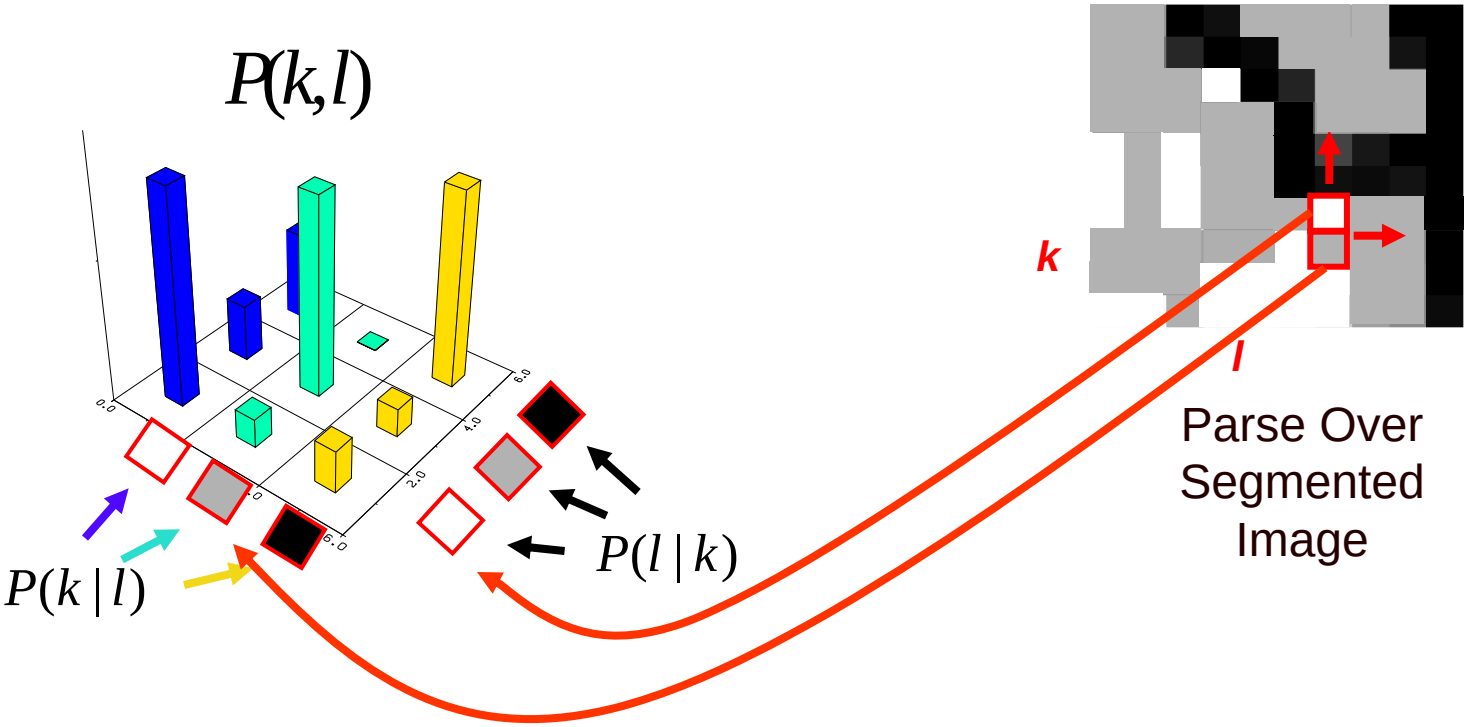
```
def make_Histo(classim,mask,numsym,totpath):
    histo = N.zeros([pow(numsym,totpath)],N.int32)
    histbins = len(histo)
    # define a "ruler" of ones of length totpath to slide across mask
    ruler = N.ones([totpath],N.int)
    # precalculate the histogram multipliers
    formult = N.ones([totpath],N.int)
    bakmult = N.ones([totpath],N.int)
...
    for i in range(totpath):
        formult[i] = pow(numsym,(totpath-i-1))
        bakmult[i] = pow(numsym,i)
...
    # want symbols to go from 0 to numsym -1
    myclasses = classim - 1
    paths = 0
    dims = len(mask.shape)
...
    elif dims == 2:
        for i in range(mask.shape[0]):
            for j in range(mask.shape[1] - totpath + 1):
                temp = ruler*mask[i,j:j+totpath]
                if len(temp) == len(N.compress(temp != 0,temp)):
                    histo[dot(formult,myclasses[i,j:j+totpath])] += 1
                    histo[dot(bakmult,myclasses[i,j:j+totpath])] += 1
                    paths += 2
...
    if paths < 2*histbins: # recursively call until you have enough data for the bins - note
        newlength = totpath-1
        totpath,paths,histo = make_Histo(classim,mask,numsym,newlength)

    return totpath,paths,histo
```

Generation Of H and SC From Segmented MRI

Entropy: $H = \sum_{k,l} P(k,l) \log_2 P(k,l)$

Statistical Complexity: $SC = -\sum_{\blacksquare\blacksquare\blacksquare} \left(\sum_{l} P(k \mid l) \right) \log_2 \left(\sum_{l} P(k \mid l) \right)$



Stage IV – Classification Based on Complexity Estimates

- E.g. feed complexity estimates to a supervised or unsuprevised learning algorithm (LDA, SVM, Bayes net, Random Forest,...)

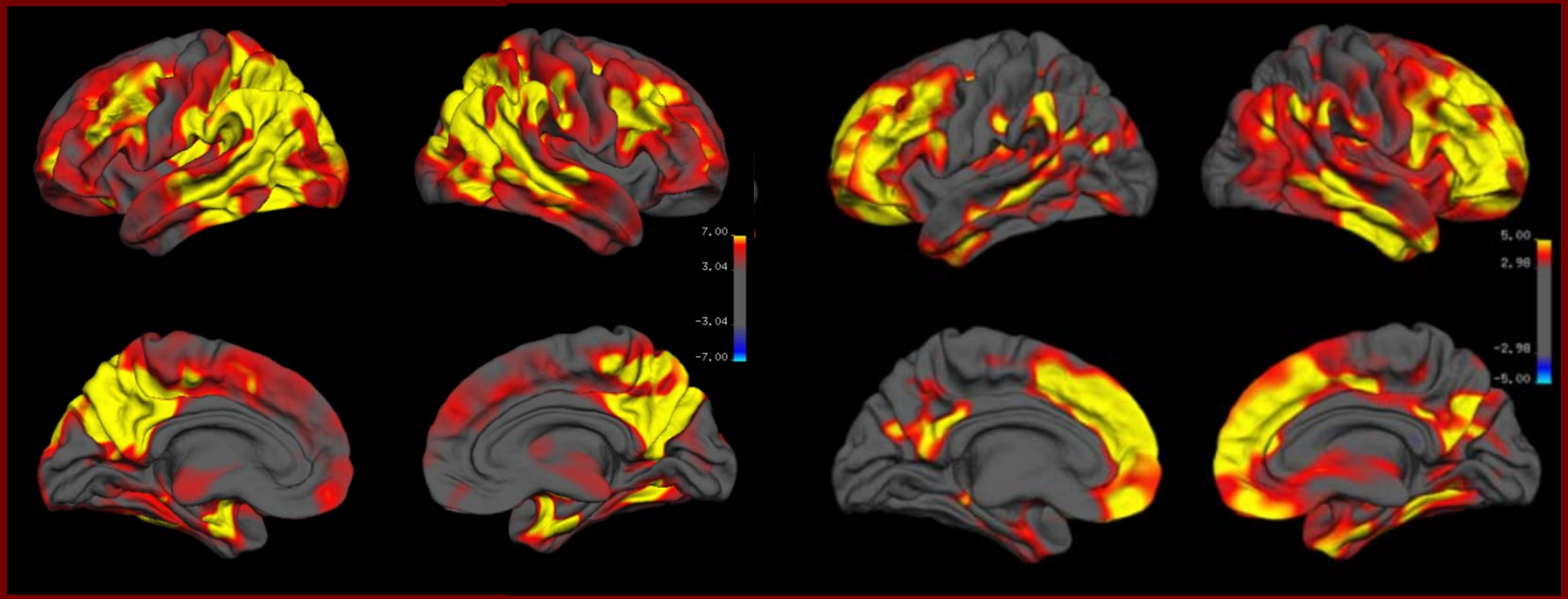
Classification of Alzheimer's Disease and Frontotemporal Dementia

- Compare use of MRI measures
  - tissue volumes
  - cortical thickness
  - complexity estimates

Previous Findings by Cortical Thickness

Cortical Thinning in AD vs Control

Cortical Thinning in FTD vs Control



Classification of Alzheimer's Disease and Frontotemporal Dementia

- 23 Cognitively Normal subjects
- 24 Patients Alzheimer's Disease
- 19 with Frontotemporal Dementia

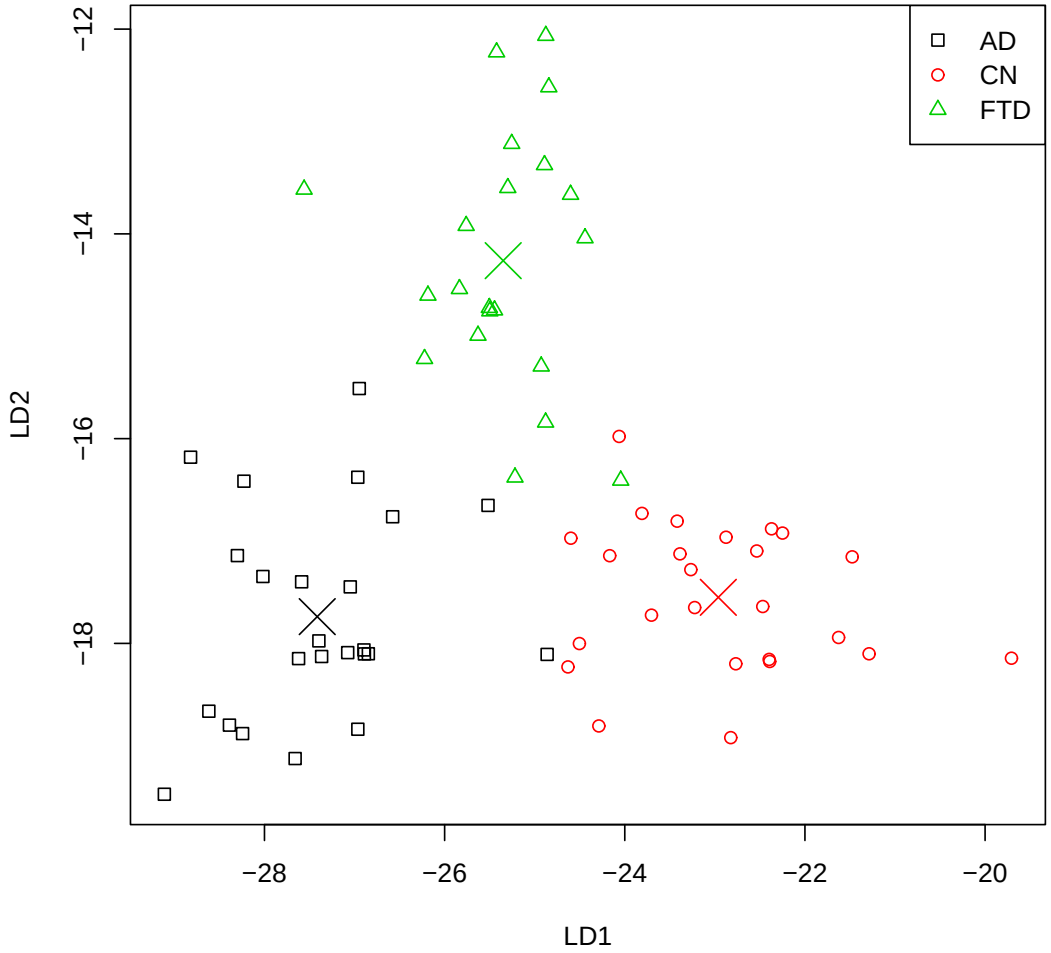
- Regional Complexity Estimates
  - From Spatially Normalized, Tissue Segmented T1 Images
  - Classification Accuracy Determined Using:
    - Support Vector Machine
    - 10 Fold Cross Validation
- Tissue Volume and Cortical Thickness Estimates
  - Obtained Using FreeSurfer
  - Classification Accuracy Determined Using:
    - Logistic Regression
    - Leave One Out Cross Validation

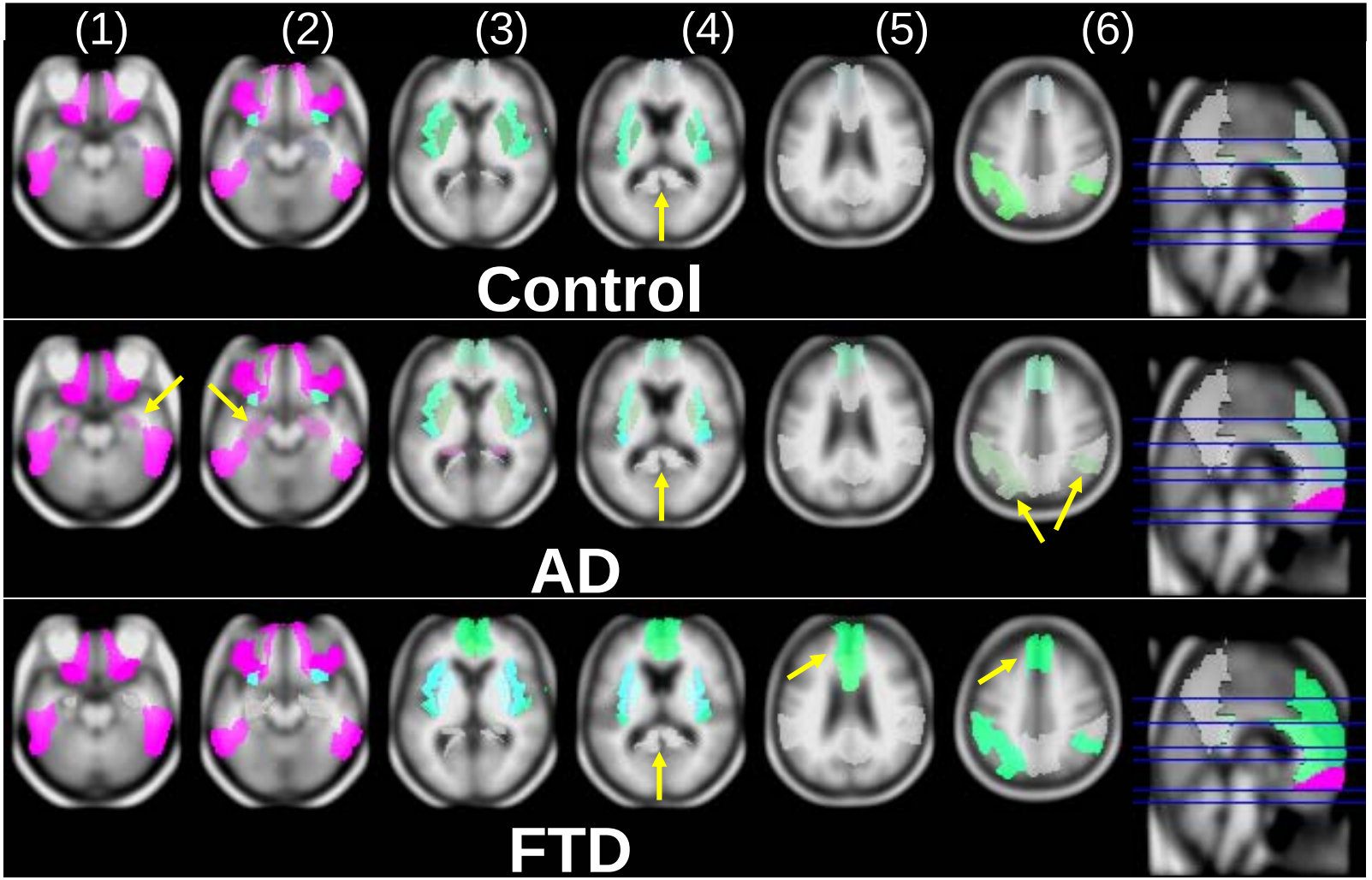
2 Class Classification Accuracy

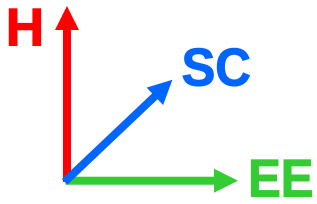
Measure \ Groups	AD vs. CN (%)	FTD vs. CN (%)	AD vs. FTD (%)
Parietal Lobe Volume (FreeSurfer)	95 ± 4	81 ± 7	85 ± 6
Parietal Lobe Thickness (FreeSurfer)	96 ± 3	82 ± 6	86 ± 6
Multi-region Complexity	91.6 ± 0.8	86.6 ± 0.7	90.2 ± 0.8

3 Class LDA Results

13 Regions




(1)
(2)
(3)
(4)
(5)
(6)
Control
AD
FTD


H
SC
EE

Conclusion

- Complexity measures provided greater power for classification between AD and FTD than current cortical thickness and volume measuring algorithms
- A scalable complexity measure (EE) in combination with H, SC provides additional information that is not easily discernable using other methods.

SciPy

- Could Not Have Performed Analysis Without SciPy (or perhaps Matlab, IDL but we know better than that...) And I'm Eternally Grateful
- But As Usual People Get Greedy So...

SciPy Wish List

- What about Rpy ?
  - Should scipy.stats try to replace R for pythonistas or should there be more/some integration between scipy.stats and rpy (or neither) ?
  - I could meet current needs for this project with better medical image I/O (NiPy, NDimage ?) and availability of wider range of clustering algorithms as well as tools for comparison of clustering algorithms (both of which I currently use R/Rpy for)

SciPy Wish List

- More Efficient Generation of Joint Desnity Estimate (Histogram Construction)
  - Some will notice similarity to building a “co-occurence matrix”; Stefan has a nice ctypes package, glcom, but need to add more general template parsing (on my to do list) – too specialized to include in NDimage ?

SciPy Wish List

- Better, More Transparent, Parallelization
  - Much of the parallelization for this project is trivial though even with pypar (nice API to a useful, truncated subset of MPI functionality) coding isn't always trivial (see, e.g. notes from Brian's talks)
  - Plan is to go to IPython1 (initially still using message passing model via mpi4py)
  - Nice to have increased community support to help Brian and Fernando promote broader use of IPython1 (and hence more code examples to steal...)

SciPy Wish List

- It would be nice to be more standalone re. integrated testing of various algorithms for the full pipeline including “preprocessing”, e.g. image registration, warping, interpolation, ...
  - Bite the bullet and incorporate ITK ?
  - NDImage ?
  - NiPy ?

Acknowledgements

- Financial support in part by:
  - Research Applications Grant from the Department of Veterans Affairs
  - NIH (NIA) grant PO1AG19724
  - NIH (ADRC) grant P50-AG023501