RapidPT: A MATLAB toolbox for fast and efficient Permutation Tests

Felipe Gutierrez Barragan

http://felipegb94.github.io/RapidPT/

The Project - Overview

- **RapidPT** is a MATLAB toolbox for fast, scalable, and reliable permutation testing.

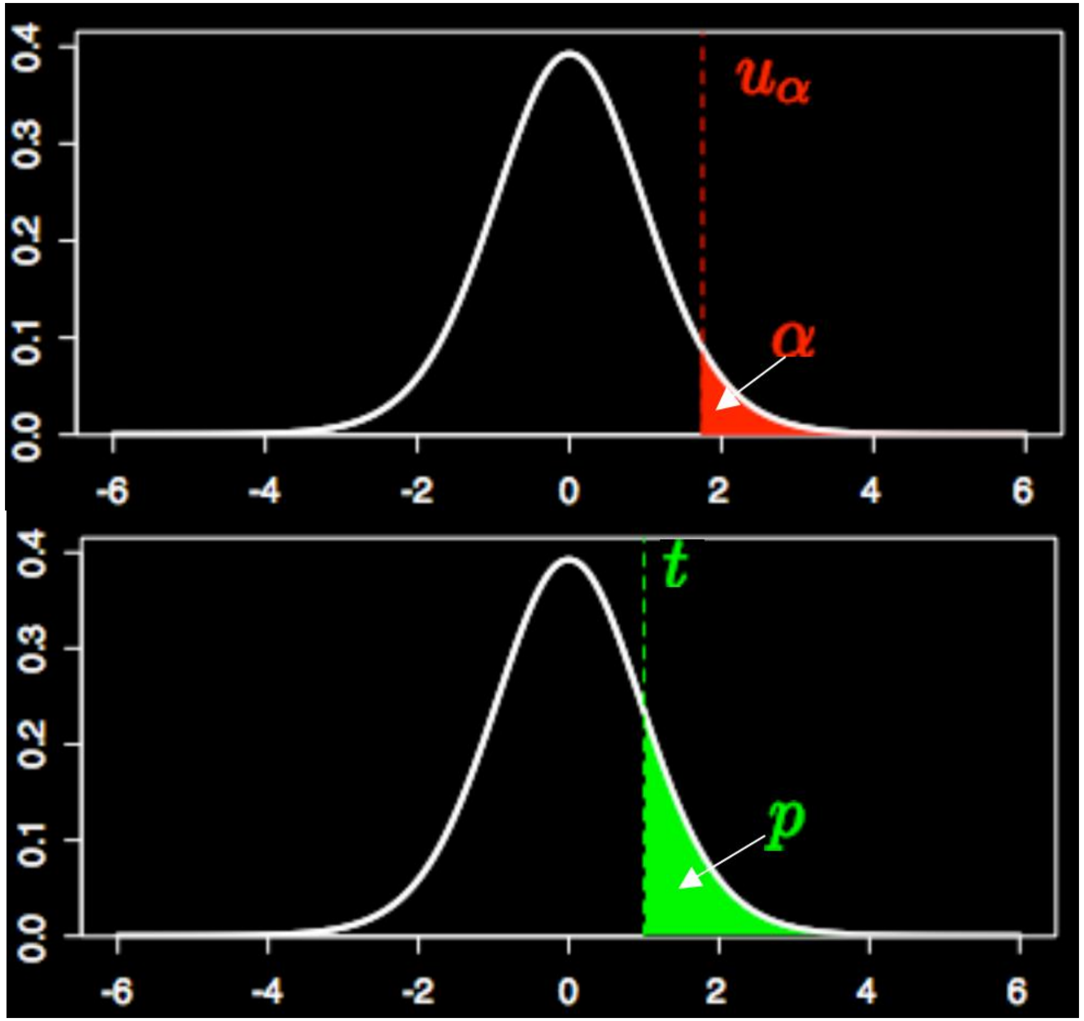
  - **Fast/Scalable:** Speedups of 20x – 80x when compared to a simple permutation testing implementation. 2x – 6x when compared to the state of the art, Statistical NonParametric Mapping (SnPM).
  - **Reliable:** Hundreds of validations runs against SnPM on neuroimaging datasets ranging from 20-400 subjects.
  - **Easy to use:** No setup, unlike hardware-based procedures. Will run fast on any modern machine.

Hypothesis Testing

- **Idea:** Calculate the probability that your claim/hypothesis is true.
- **Question:** Does the data display any interesting activity?
  - In neuroimaging – Voxel-wise differences between rest vs. activation? Healthy vs. Diseased?
- Two Possibilities:
  - Null Hypothesis vs. Alternate Hypothesis
  - **H_0 vs. H_a**

Hypothesis Testing - Procedure

1. Choose appropriate test statistic (t-test, mean difference, etc).
2. State H_0 and H_a
3. Construct the null distribution for the test statistic. This is the distribution of the statistic given that H_0 is true. This can be done analytically in some cases.
4. Compute the t-statistic with the given data.
5. Calculate the probability of observing such a statistic given that H_0 is true (p-value).
6. Accept or reject H_0 .



Hypothesis Testing - Types of Error

Decision	Actual Situation "Truth": H_0 True	Actual Situation "Truth": H_0 False
Do Not Reject H_0	Correct Decision $1 - \alpha$	Incorrect Decision Type II Error β
Rejct H_0	Incorrect Decision Type I Error α	Correct Decision $1 - \beta$

$$\alpha = P(Type\ I\ Error) \quad \beta = P(Type\ II\ Error)$$

Multiple Testing Problem

- **FWER:** Probability of making at least Type I Error (False Positive).

$$P(\text{Making at least 1 error in m tests}) = 1 - (1 - \alpha)$$

$$P(\text{Making at least 1 error in m tests}) = 1 - (1 - \alpha)^m$$

  - For m = 100 → FWER = 0.99.
  - For a functional neuroimaging dataset we would do >100,000 tests. One for every voxel.
- Hence, we want to somehow control for the FWER and keep it under a certain probability α_0.

Controlling FWER

- **Parametric Methods**
  - Assumptions about the data and its distribution.
  - **Bonferroni –** Conservative. Simply set $\alpha_0 = {}^{\alpha}/_{v}$ *(v = number of tests).*
  - **Random Fields Theory –** Estimate number of activation areas, effectively reducing the number of tests.

- **Nonparametric/Resampling Methods**
  - **Permutation Testing –** Empirically estimate the null distribution of the test statistics.
    - Exact control of the FWER.

Permutation Testing

- **Idea:** If the two groups do not differ, then I can permute the group/class labels and end up with approximately same set of t statistics.

- **Procedure:**
  1. Re-label images (permute the labels of the images).
  2. Compute test statistic for each voxel
  3. Repeat N times.

- After the procedure is done we will have the exact null distribution for each voxel, and we can proceed from step 4 of hypothesis testing .

Permutation Testing – Example, Single Test

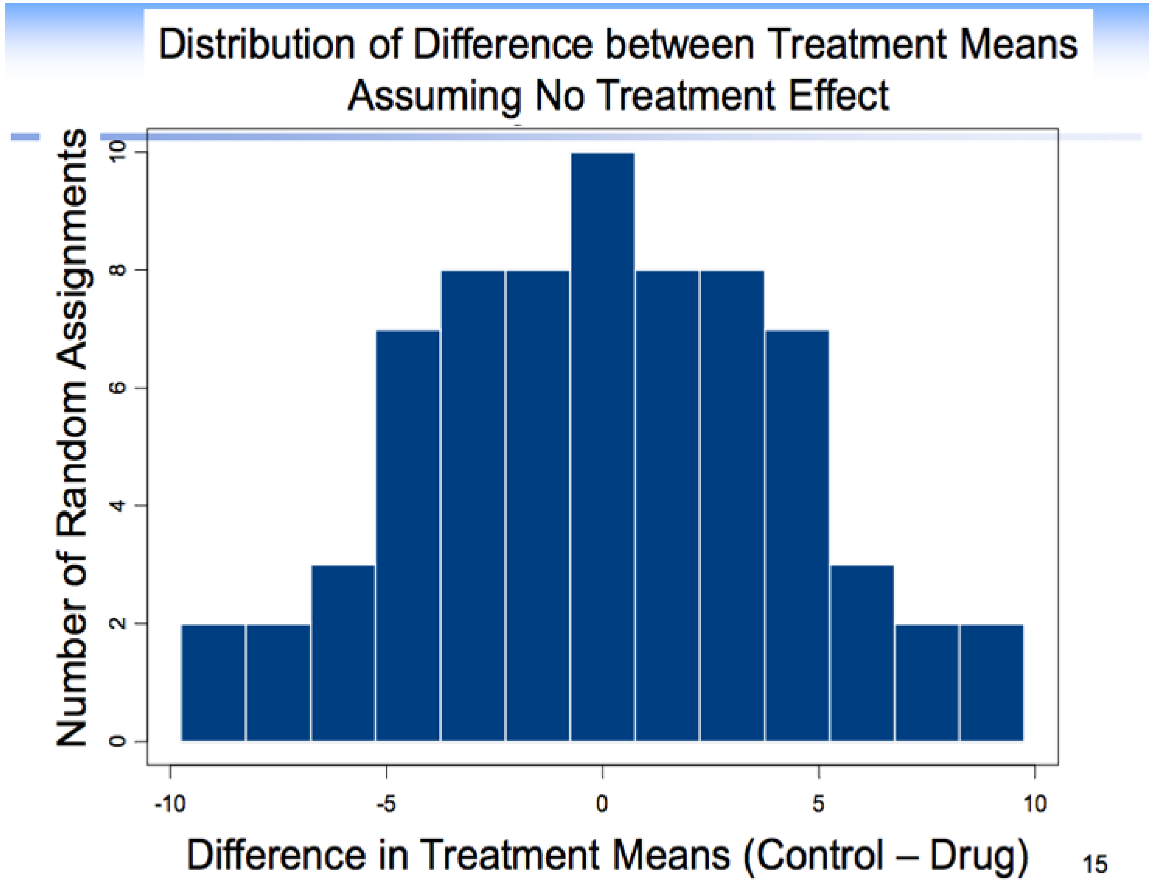
	Control				Drug			
Expression	9	12	14	17	18	21	23	26
Average	13				22			

8 choose 4 = 70 possible permutations

Rearrangement of data

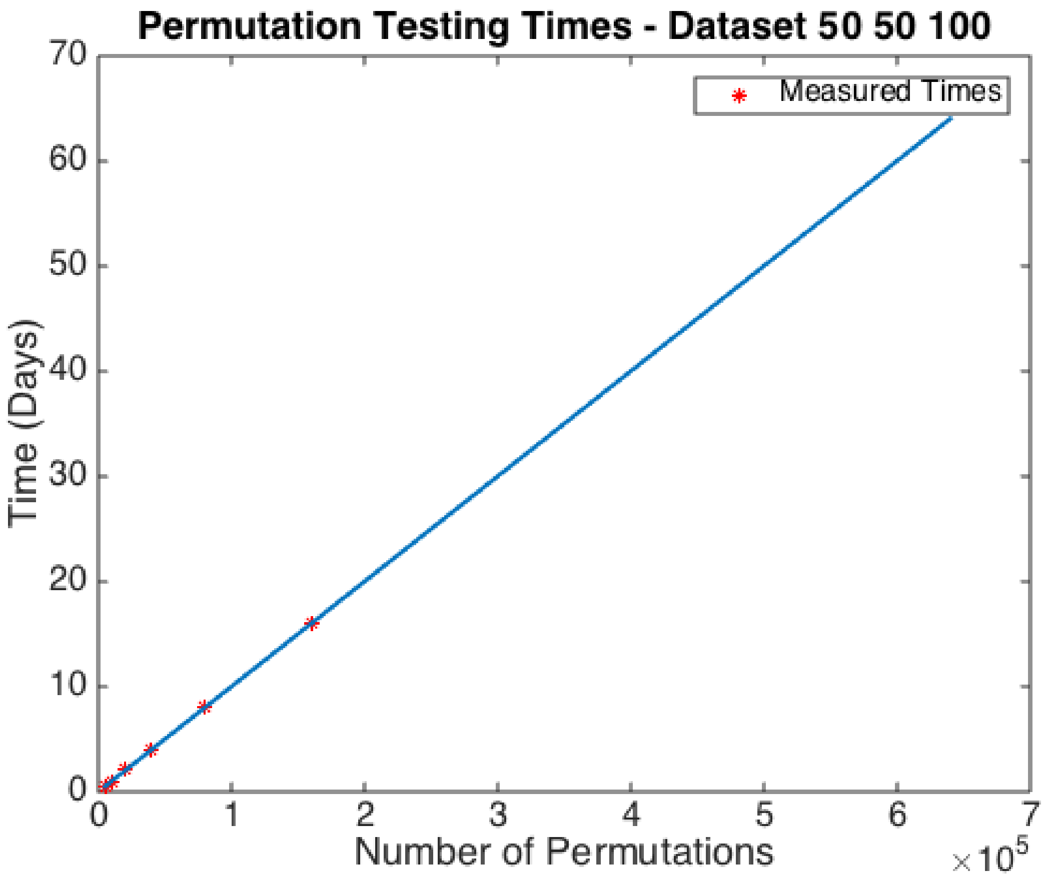
Random Assignment	Control				Drug				Difference in Averages
1	9	12	14	17	18	21	23	26	9.0
2	9	12	14	18	17	21	23	26	8.5
3	9	12	14	21	17	18	23	26	7.0
4	9	12	14	23	17	18	21	26	6.0
5	9	12	14	26	17	18	21	23	4.5
6	9	12	17	18	14	21	23	26	7.0
7	9	12	17	21	14	18	23	26	5.5
8	9	12	17	23	14	18	21	26	4.5
9	9	12	17	26	14	18	21	23	3.0
10	9	12	18	21	14	17	23	26	5.0
11	9	12	18	23	14	17	21	26	4.0
12	9	12	18	26	14	17	21	23	2.5
13	9	12	21	23	14	17	18	26	2.5
14	9	12	21	26	14	17	18	23	1.0
15	9	12	23	26	14	17	18	21	0.0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
69	18	21	23	26	9	12	14	17	-8.5
70	18	21	23	26	9	12	14	17	-9.0

14



15

Computational Issues with Permutation Tests



Memory Issues

- Permutation Testing Matrix is very large
  - Let v be the number of voxels (around 5e5)
  - Let T be the number of permutations (10,000)
  - Let P be the permutation testing matrix (v x T)
  - Memory for that matrix is around **40 gigabytes**
    - *Note* – MATLAB crashes at some point when dealing with matrices that big.

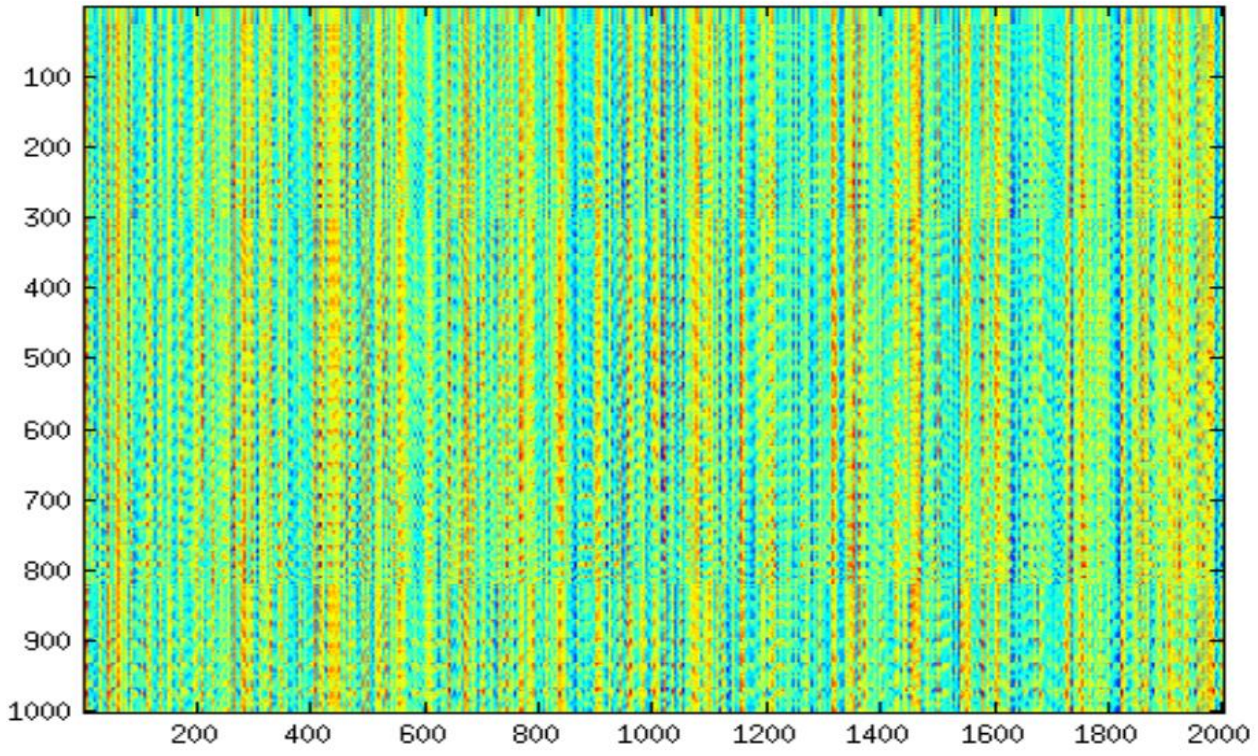
Motivation: We need a lot of permutations…

- 1 test
  - 10,000 permutations → The smallest possible p-value is $\frac{1}{10{,}000}$.
  - Therefore the uncertainty near alpha = 0.01 is ±0.0001 OR ±1%
- 2 tests
  - 10,000 permutations → The smallest possible p-value is $\frac{1}{10{,}000}$ in each test.
  - Now we have 2 opportunities for a type I error
  - Therefore the uncertainty near alpha 0.01 in the worst case is ±0.0002 or 2%
- 500,000 tests
  - 10,000 permutations → The smallest possible p-value is $\frac{1}{10{,}000}$ in each test.
  - Therefore the uncertainty in the worst case scenario is 100%

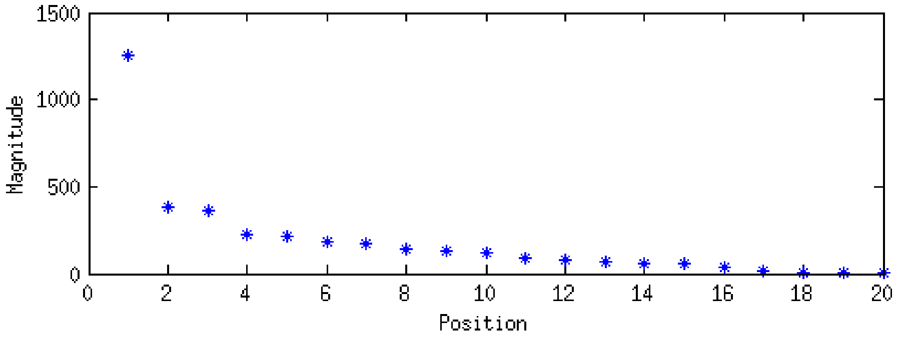
Idea: Look at the structure of *P*

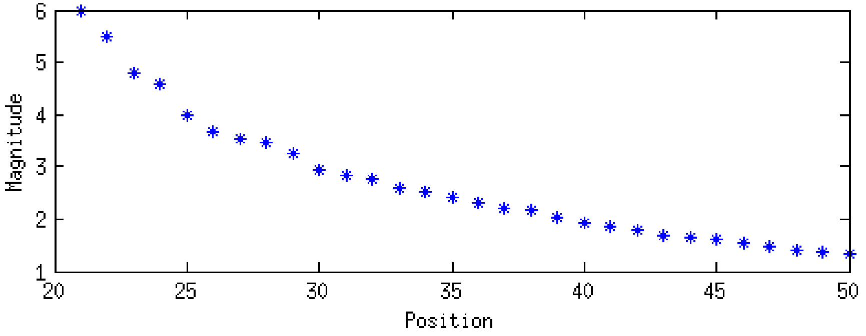
- *P* is "highly structured" – A combination of low-rank signal and high-rank residual
- Example: MRI data 100 healthy vs. non-healthy. v = 1,000, T = 2,000

Permutation Testing Matrix *P*



Singular values of *P*





Core of RapidPT

- Many columns in P look similar to othe columns as well as many rows look similar to other rows.
- If we compute a small number of entries of P we should be able to estimate the rest of it.
- Mathematically,

$$P = UW + S$$

  - U is the low-rank basis of P.
  - W is the coefficient matrix
  - S is a high-rank random residual (noise).
- How many entries? In our experiments, subsampling <1% was enough

RapidPT Algorithm

Algorithm 3 Efficient Permutation Testing Algorithm

procedure RapidPT(data, nGroup1, numPermutations, numTrainingSamples, samplingRate)
▷ Training
$P\_exact \leftarrow PermTest(data, nGroup1, numTrainingSamples)$
$U \leftarrow EstimateBasis(P\_exact)$
$W \leftarrow EstimateCoeffMat(U, P\_exact)$
$S \leftarrow ApproximateNoise(U, P\_exact)$
▷ Recovery
$maxnull \leftarrow zeros(1, numPermutations)$
$subset \leftarrow zeros(1, samplingRate * numVoxels)$
for $i \leftarrow 1, numPermutations$ **do**
$subset \leftarrow randsample(1 : numVoxels, samplingRate * numVoxels)$
$p \leftarrow PermTest(Data(:, subset), nGroup1, 1)$
$w \leftarrow EstimateCoeffMat(U(:, subU), p)$
$s \leftarrow EstimateNoise(U(:, subU), p)$
$p \leftarrow U * w + s$
$maxnull(1, i) \leftarrow max(p)$
end for
return $maxnull$ ▷ Maximum Null Distribution
end procedure

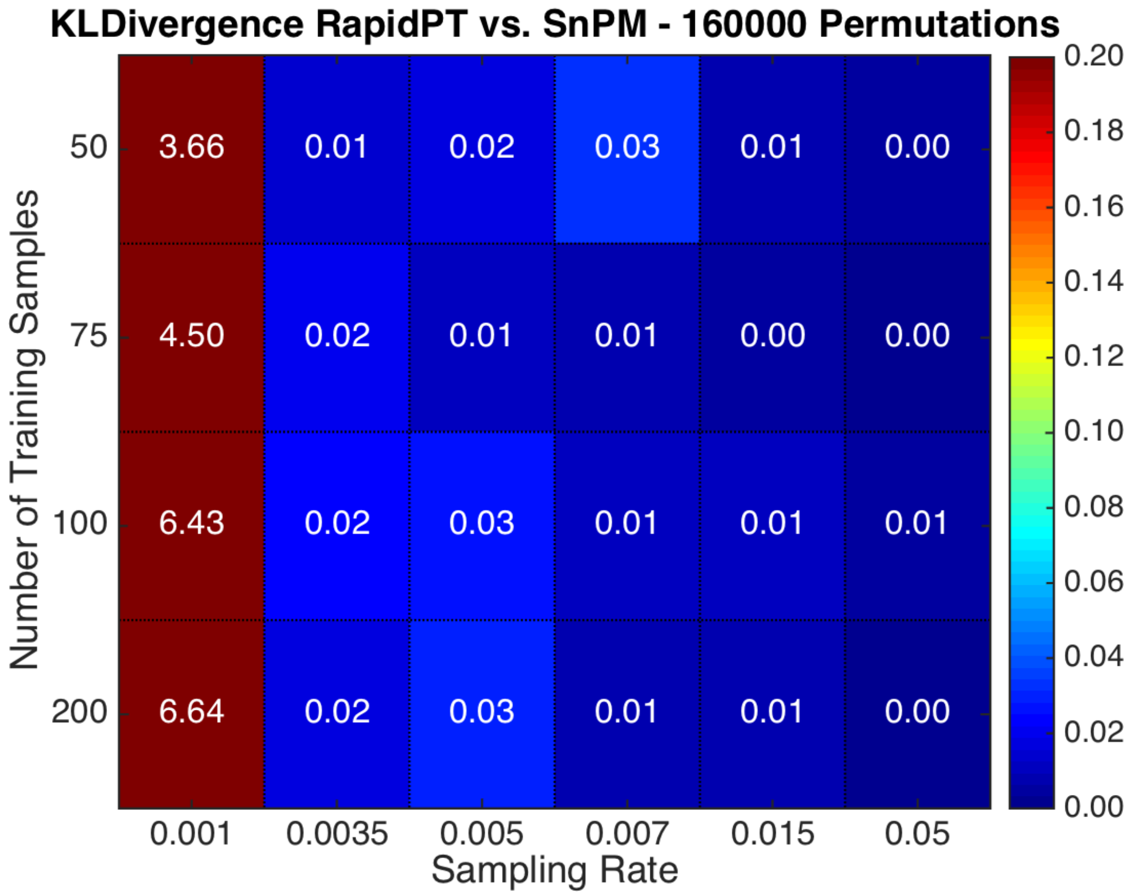
Evaluations Setup

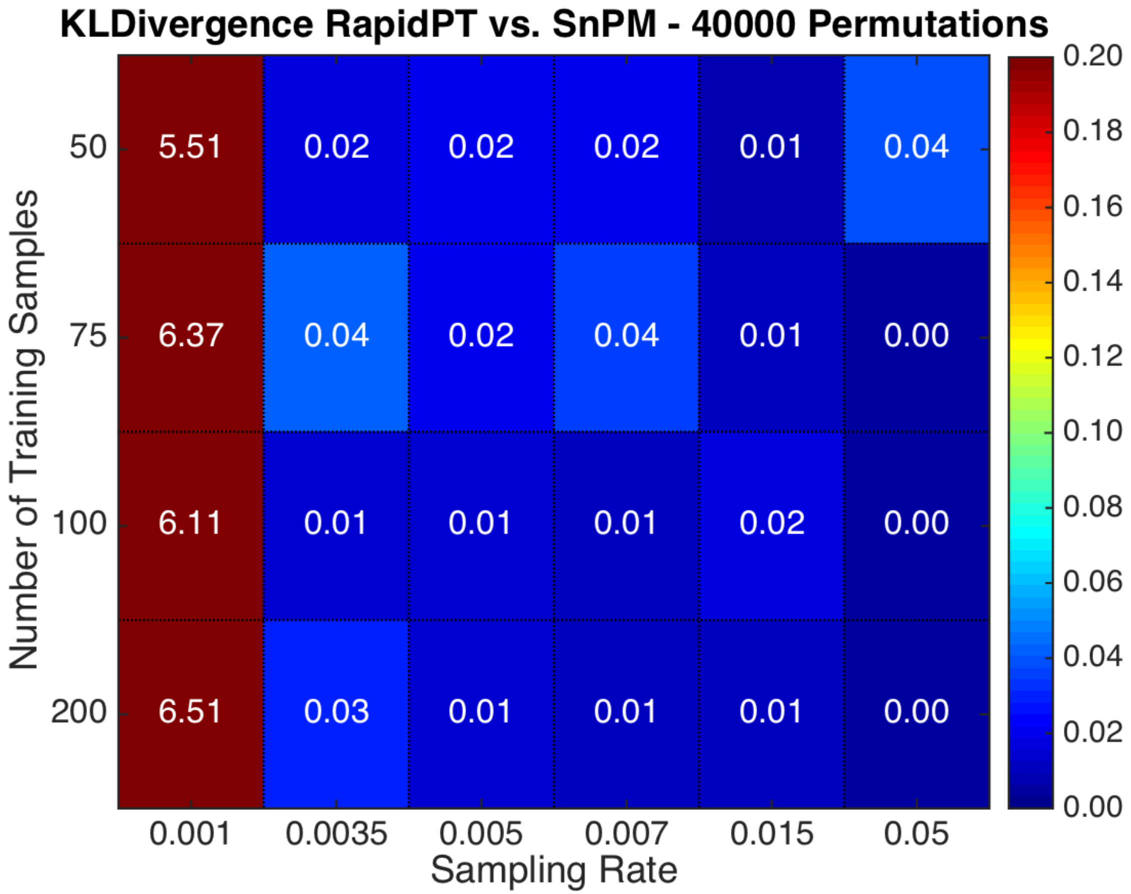
- **Data:** T1 MRIs from ADNI2 are used.
  - 601 subjects (259 AD and 342 CN)
  - SPM preprocessing is applied.
  - GM images with approx. 500,000+ voxels are extracted.
  - Multiple combinations of dataset sizes
- **Experiments:** Can we recover the Maxnull distribution?
  - Stability of hyperparameters – Sub-sampling rates, and training samples
  - Computational Speedups (RapidPT vs. SnPM, RapidPT vs. NaivePT)

Recovered MaxNull Distribution Accuracy

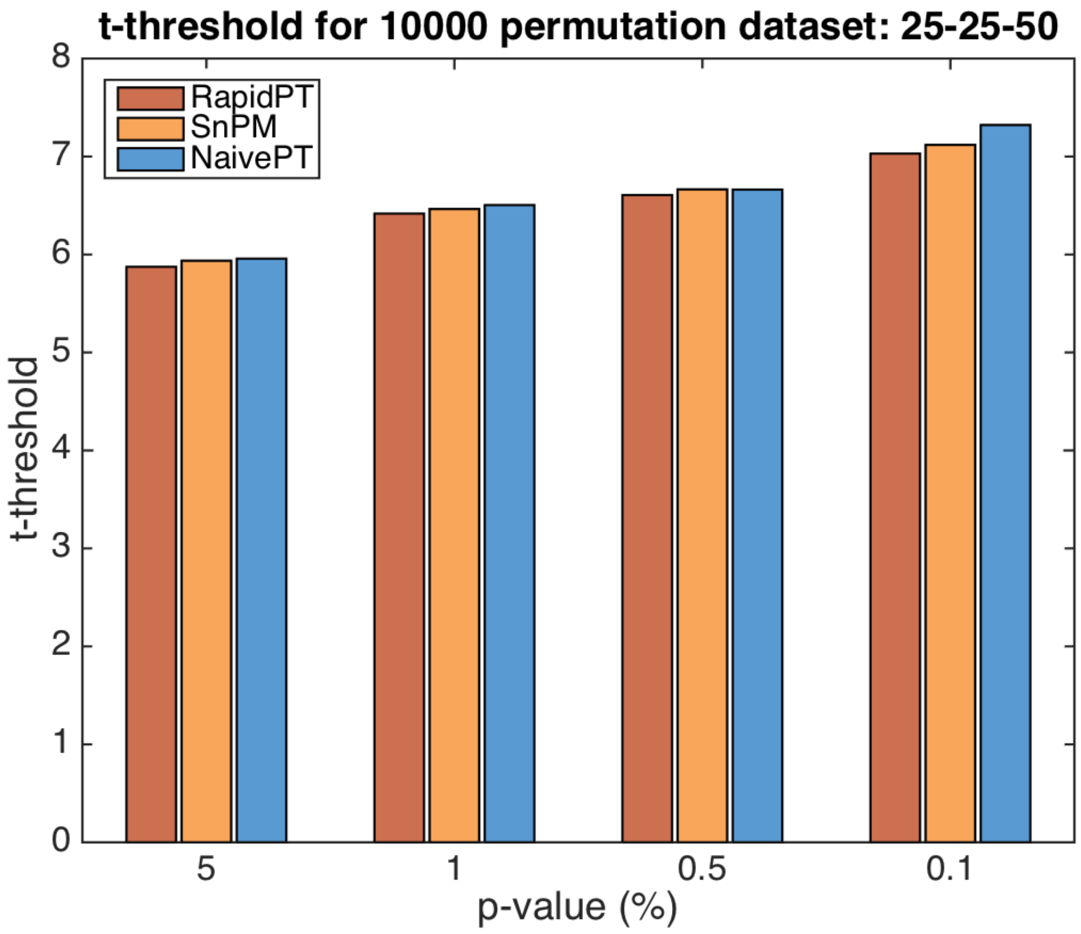
KLDivergence: Measure of the difference between two distributions

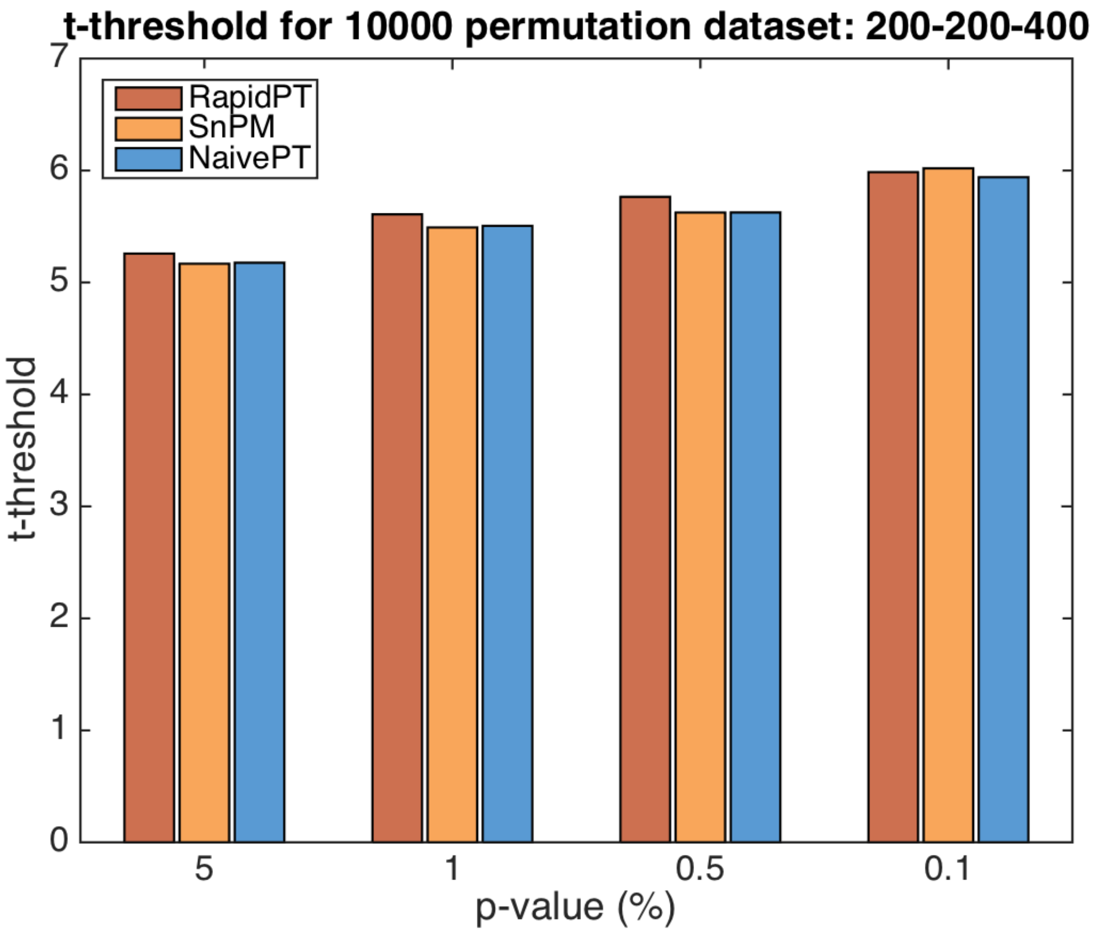
Dataset: 100 subject, 50 AD 50 CN



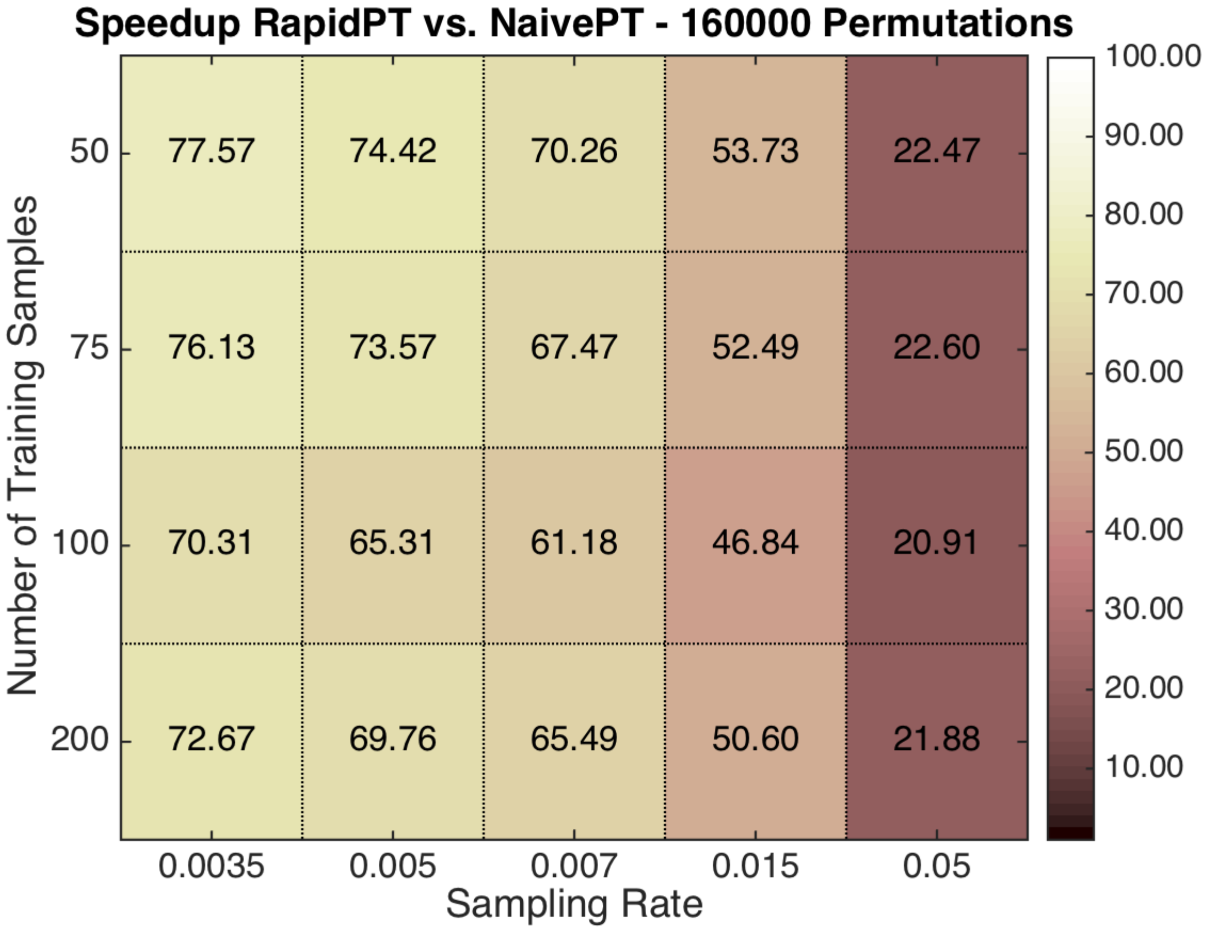


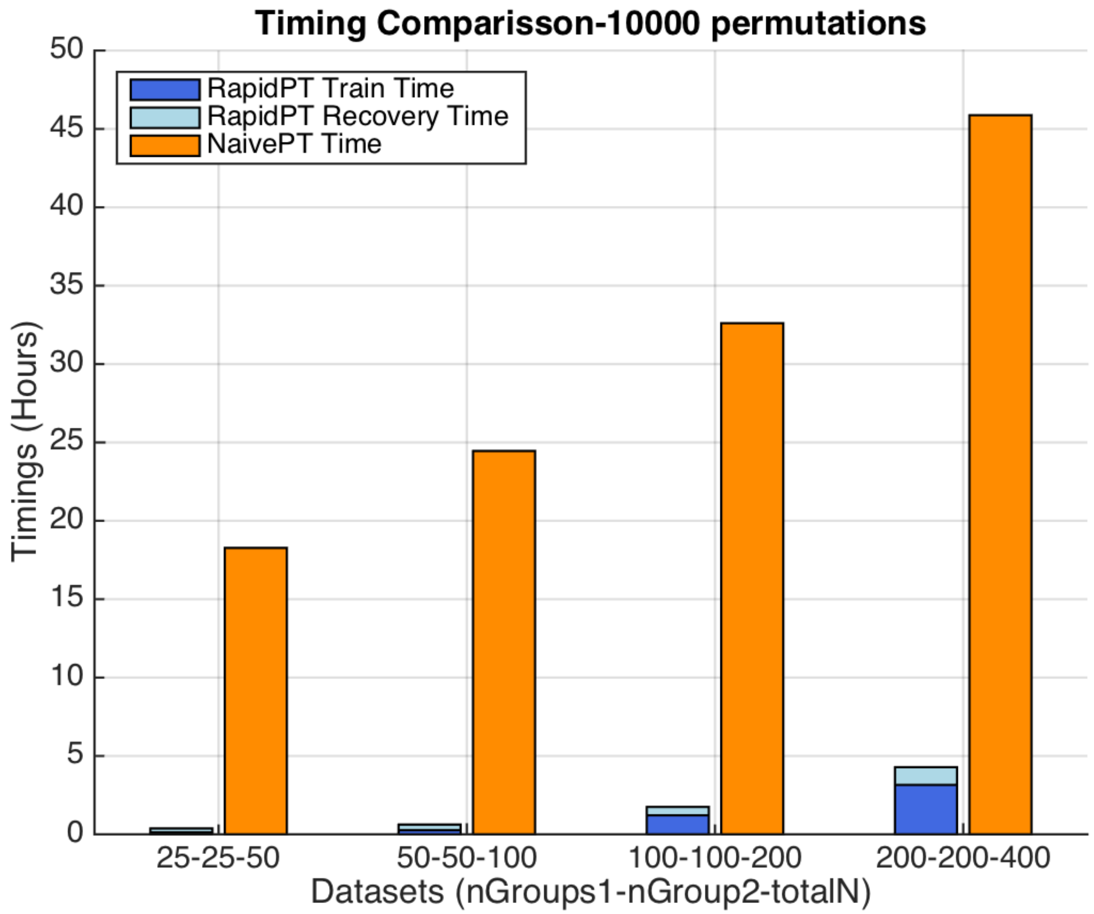
Calculated T-Threshold for a Given P-Value





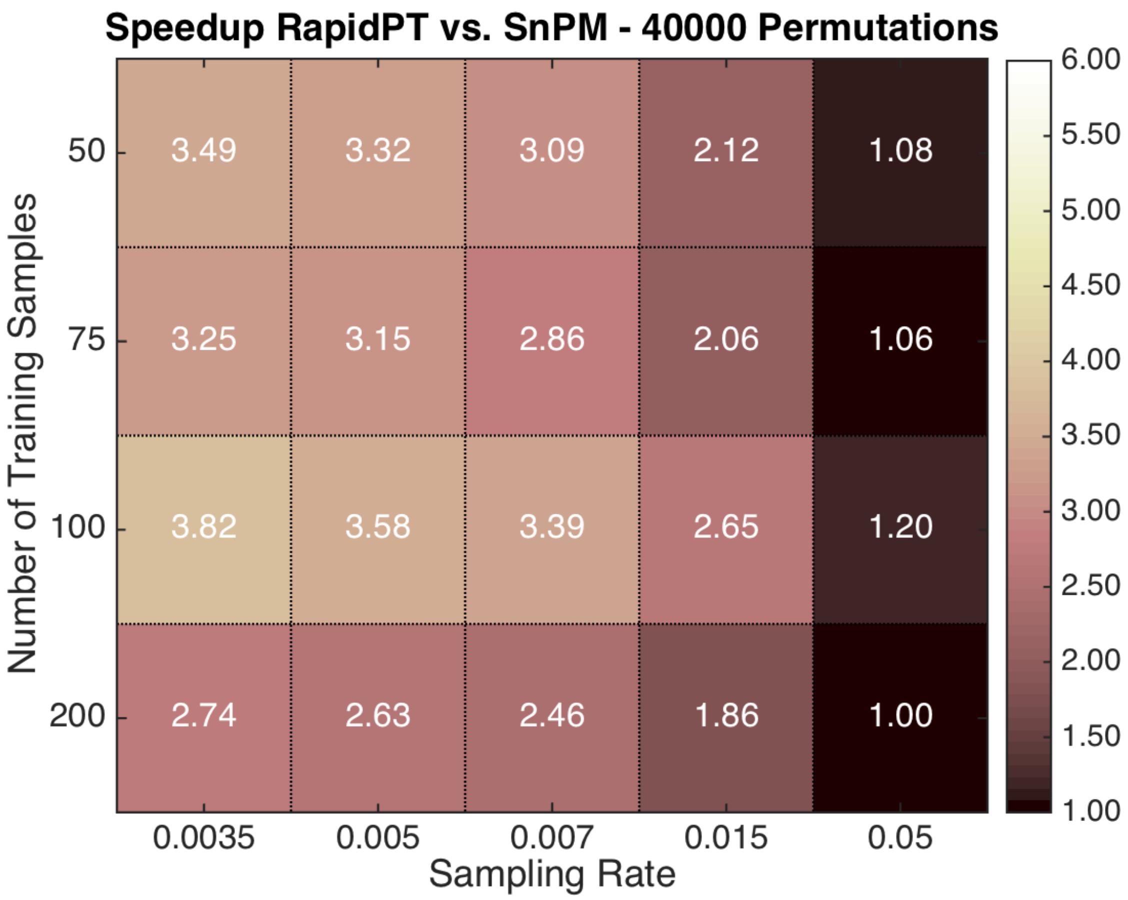
Speedup RapidPT vs. NaivePT

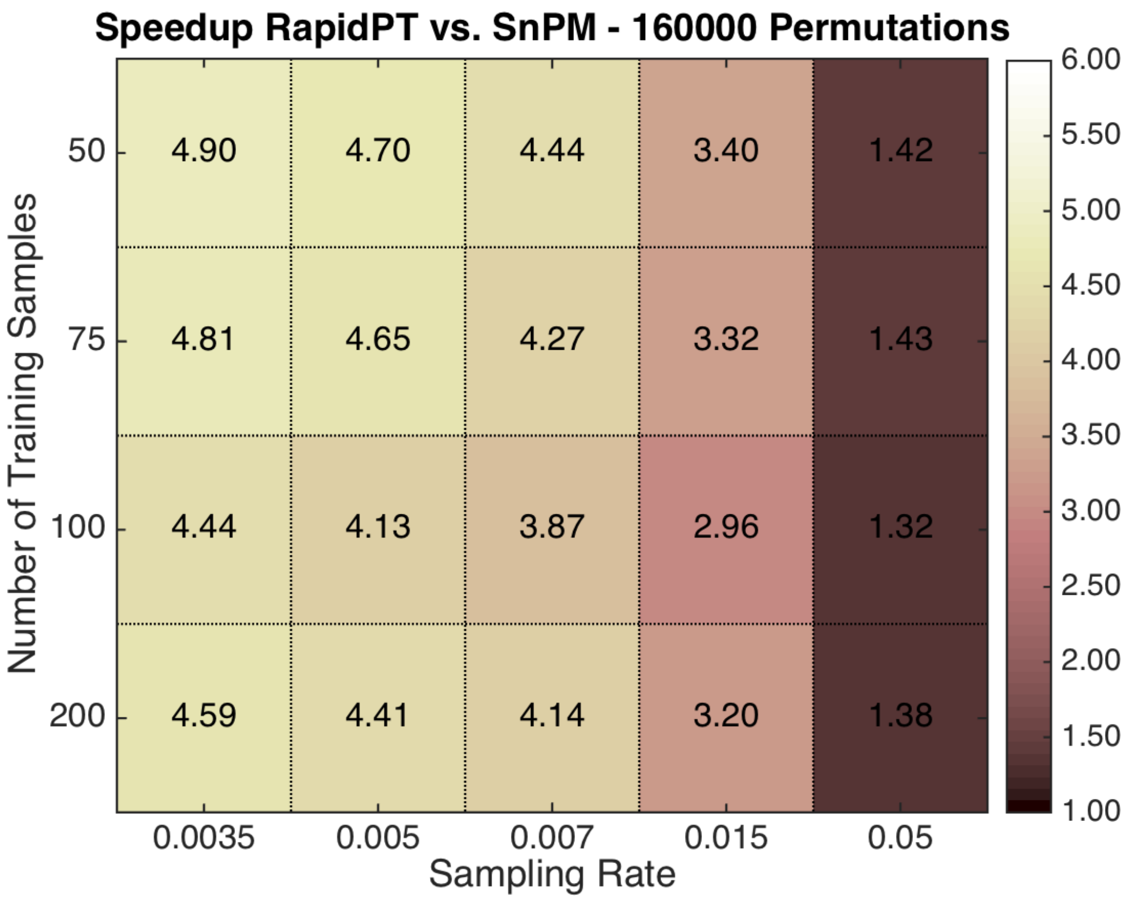




Speedup RapidPT vs. SnPM

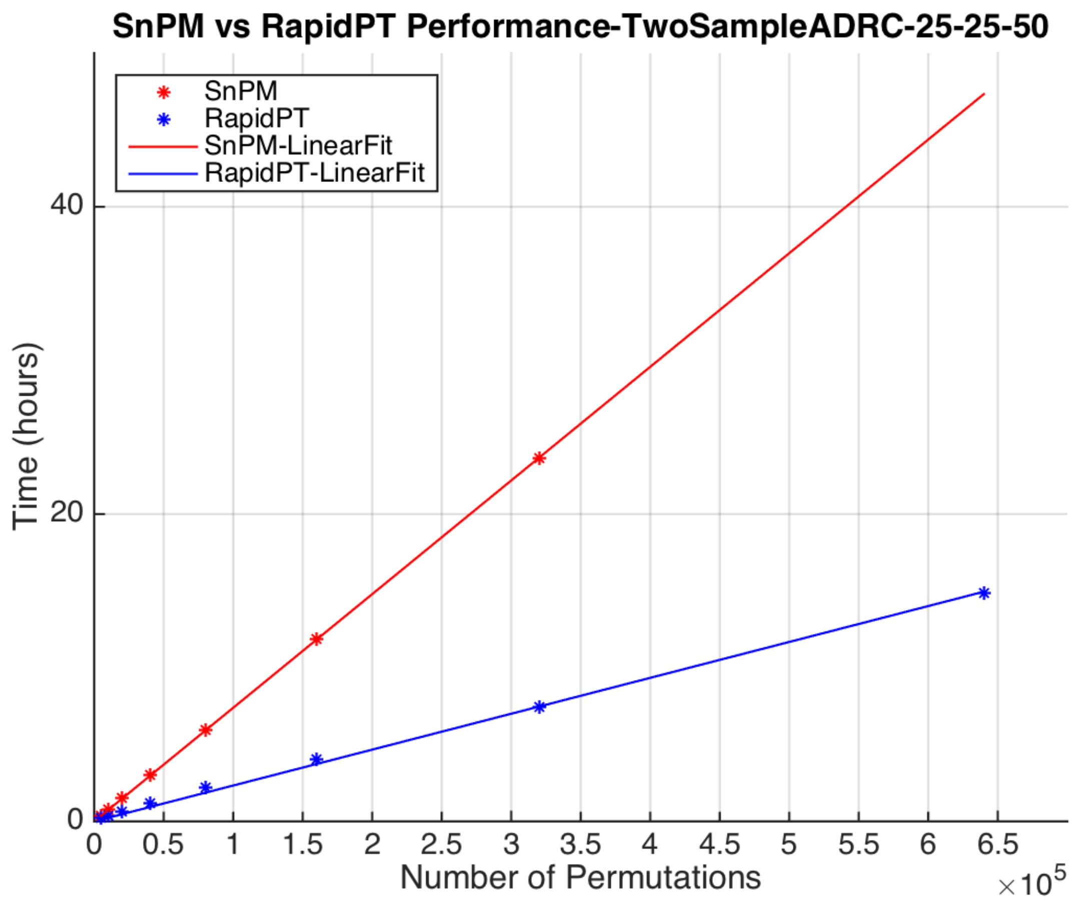
Dataset used: 100 subjects, 50 AD 50 CN





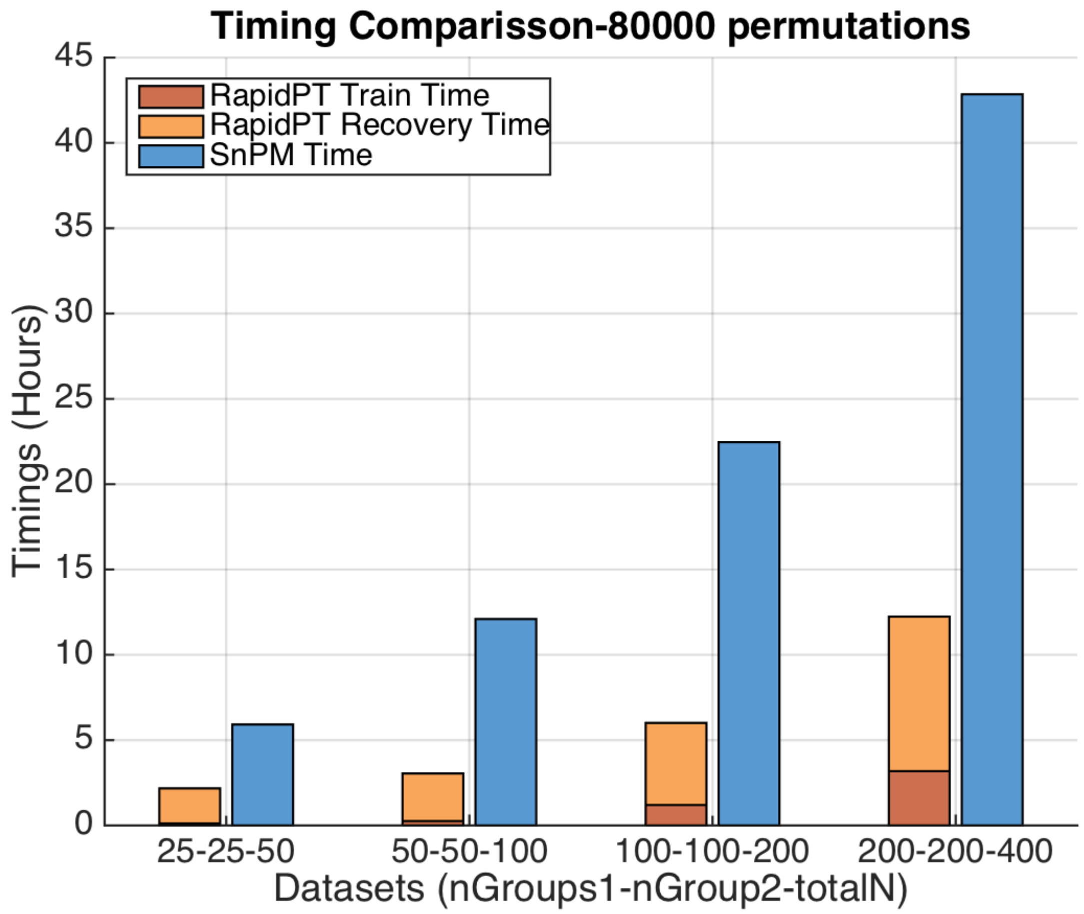
Scaling analysis RapidPT vs. SnPM

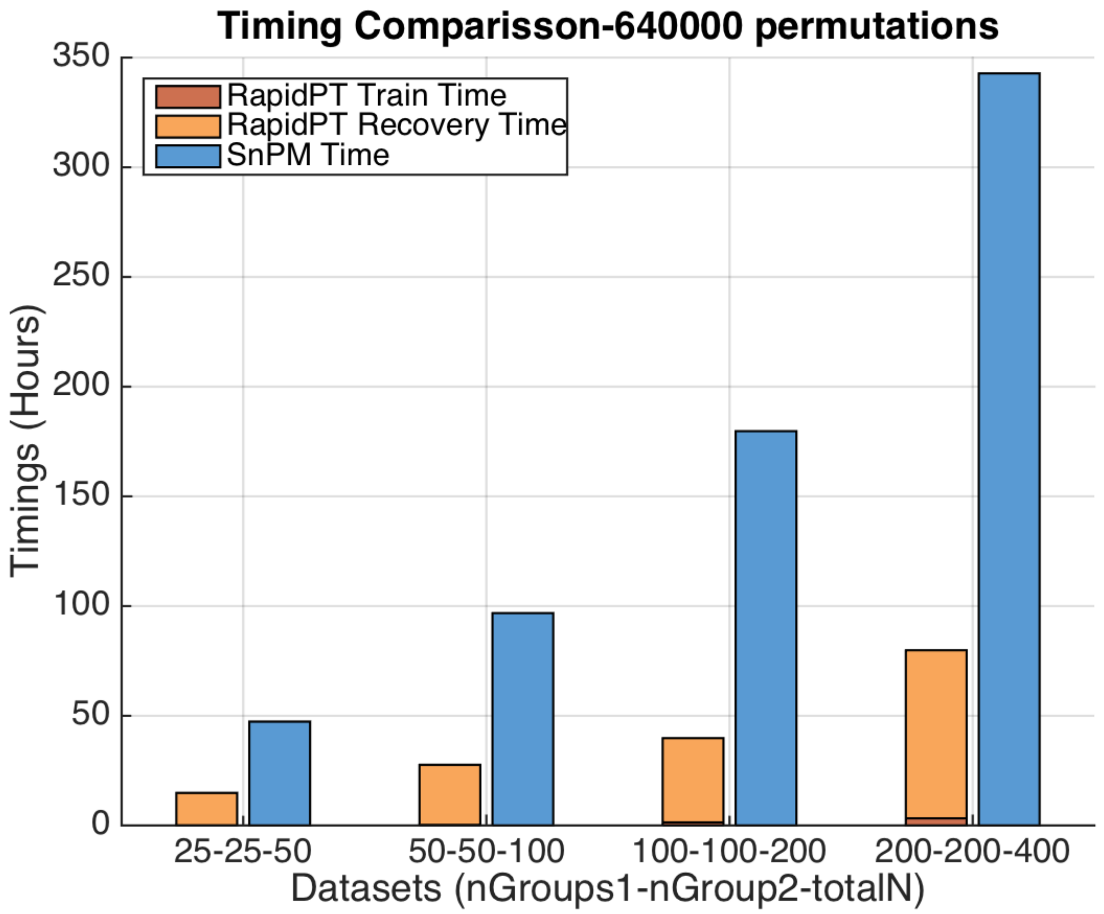
Putting the speedups into context…



Timings RapidPT vs. SnPM

Putting the speedups into context…





Discussion

- RapidPT achieves state of the art performance
  - It is able to recover the Maximum null distribution to a high degree of accuracy. Good control for FWER.
  - Fast – A few hours vs. days, and a day or two vs. a week or two..
  - Scalable – As the number of permutations increases and the dataset size increases the speedups against SnPM and NaivePT improve…
  - Easy to use????

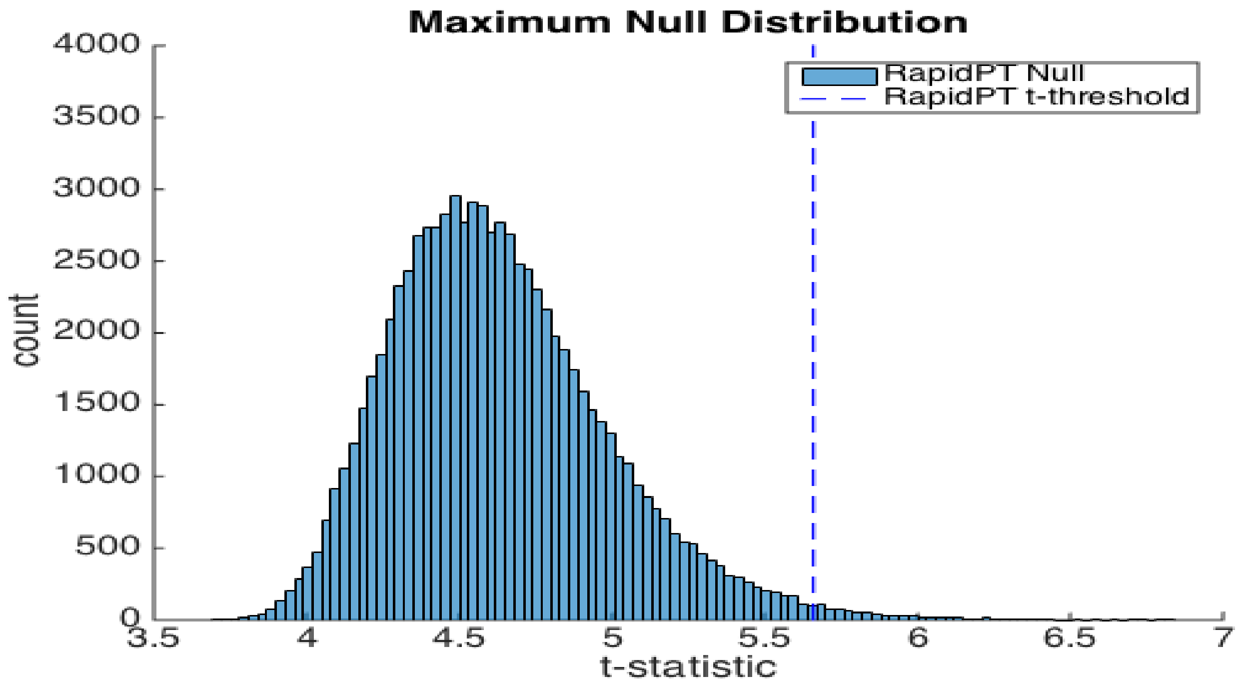
RapidPT - Usage

Do pre-processing of .nii images for each subject and construct an NxV data matrix.

```
% Addpath RapidPT Repository Path (current working dir in this case)
RapidPTLibraryPath = '.';
addpath(RapidPTLibraryPath);

% Load input data and input labels
dataPath = '~/PermTest/data/ADRC/TwoSample/ADRC_50_25_25.mat';
% dataPath = '~/PermTest/data/face/Data_face.mat';
load(dataPath);
% N subjects, V voxels (or statistics)
[N,V] = size(Data);
numPermutations = 40000;
nGroup1 = 25; % You should what is the size of one of your groups prior.
% Set write to 1 if you want the matrices used to recover the permutation matrix.
% Setting this to 1 will make outputs a very large variable, but may be
% useful in certain cases.
write = 0;

[outputs, timings] = TwoSampleRapidPT(Data, numPermutations, nGroup1, write, RapidPTLibraryPath);

save(strcat('outputs/outputs_TwoSampleFace_',num2str(numPermutations),'.mat'),'outputs');
save(strcat('timings/timings_TwoSampleFace_',num2str(numPermutations),'.mat'),'timings');
```

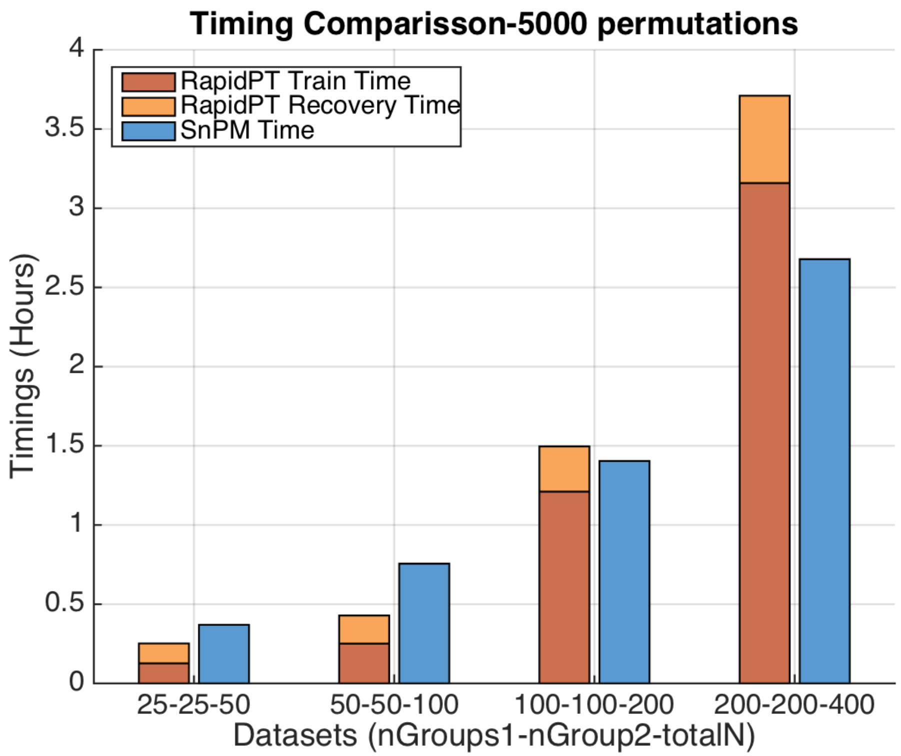
RapidPT – Postprocess Example

```
% Get the outputs struct you obtained from RapidPT
load('~/PermTest/outputs/TwoSample_ADRC_200_200_400/rapidpt/outputs_80000_0
alpha = 0.01; % Significance level of 1 percent
tThresh_RapidPT = prctile(outputs.maxT, 100 - (100*alpha));
% Get the data
load('~/PermTest/data/ADRC/TwoSample/ADRC_400_200_200.mat');
[h,p,ci,stats]=ttest2(Data(1:200,:),Data(201:400,:),0.05,'both','unequal');
SampleMaxT = max(stats.tstat);
```



Warnings for RapidPT and Permutation Testing

- Small datasets and few permutations
  - A couple minutes vs one minute
  - Large uncertainty due to the smallest possible p-value being large.
    - 10 subjects → 10 choose 5 = 252
    - Smallest possible p-val = 1/252.



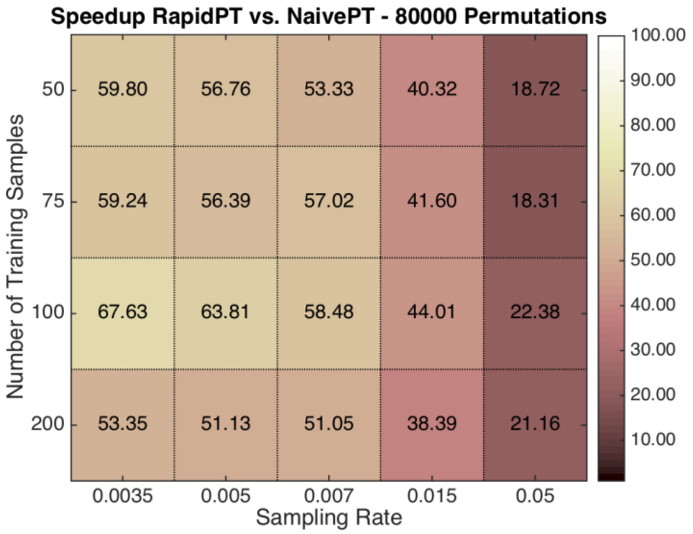
Acknoledgements and Website

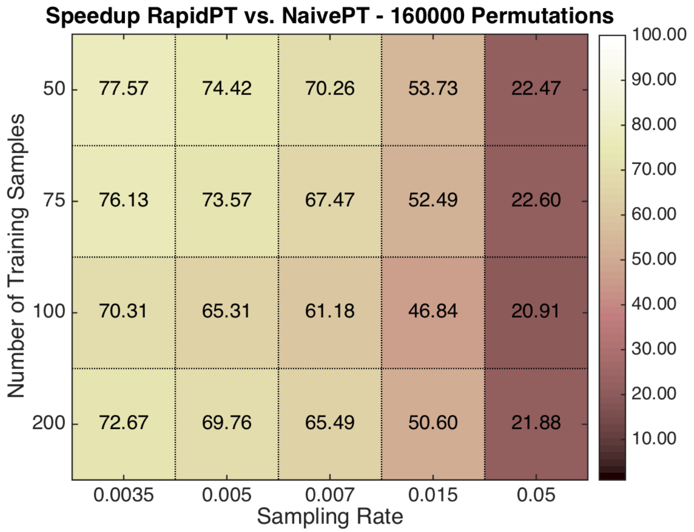
- Vikas Singh
- Vamsi Ithapu
- ADNI and ADRC

- Repository and project website:
  - https://github.com/felipegb94/RapidPT (Repository)
  - http://felipegb94.github.io/RapidPT/ (Website)

More Motivation: Implementation

- Implementation can easily be made inneficient
  - Inneficient data accesses
  - Not take advantage of MATLAB's optimized features





Algorithm 1 Traditional Two-Sample Permutation Testing. In this implementation ttest2 refers to the two-sample t-test operation described in equation 1.

procedure TwoSampleNaivePT($\mathbf{X}, r, N_1, N_2$)
    for $i \leftarrow 1, r$ **do**
        $shuffleRows(\mathbf{X})$
        $\mathbf{X}_1 \leftarrow \mathbf{X}(1 : N_1, :)$
        $\mathbf{X}_2 \leftarrow \mathbf{X}(N_1 + 1 : end, :)$
        $\mathbf{t} \leftarrow ttest2(\mathbf{X}_1, \mathbf{X}_2)$
        $\mathbf{T}(i, :) \leftarrow \mathbf{t}$
    end for ▷ Mean and Variance Calculation
end procedure