Magic Pie (mgcpy)

Team Captain: Sambit Panda

Sandhya Ramachandran, Bear Xiong, Richard Guo, Satish Palaniappan, Ananya Swaminathan

Date: 9/12/2018

Sprint 1: Create *mgcpy* package

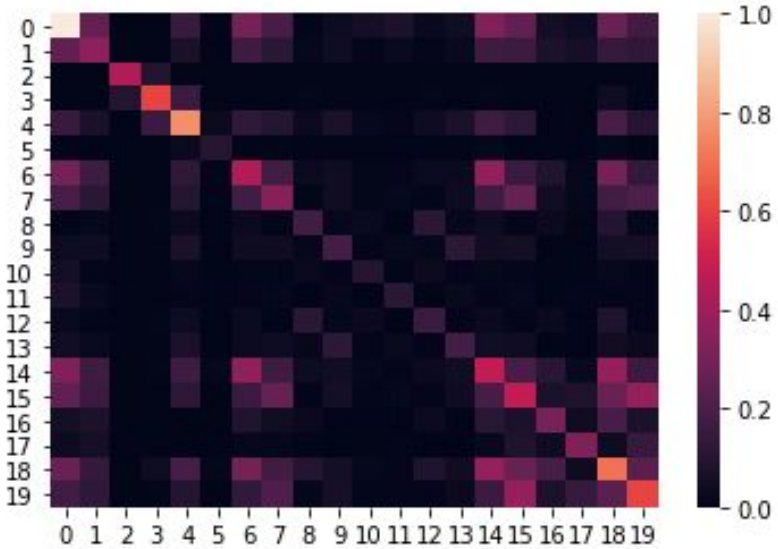
- Task 1: Implement HHG, Pearson/RV/Cca, and Spearman/Kendall into package (Sambit)
- Task 2: Implement MGC into the package (Satish)
- Task 3: Implement MDMR and FastMGC into package (Sandhya)
- Task 4: Implement MCORR, DCORR, and Mantel into package (Bear)
- Task 5: Implement Random Forest independence tests into package (Richard)
- Task 6: Implement 2- sample tests into package (Ananya)

Determine whether a single connectome has linear correlations that can be visualized

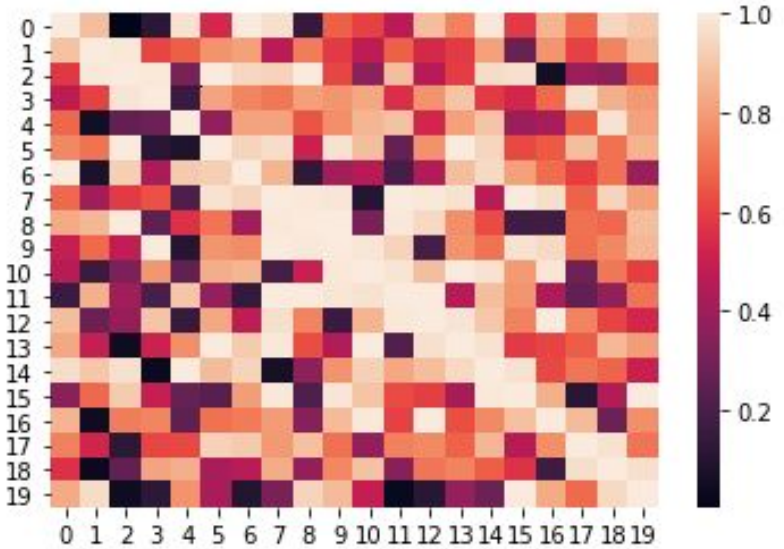
- Patient: sub-NDARAA536PTU_acq-64dir_dwi_JHU_res-1x1x1_measure-spatial-ds.edgelist
- Produced correlation map of part of the edgelist file
- Calculated distance correction of the correlation matrix
- Calculated pearson’s correlation between 2x2 row-wise adjacent blocks of the correlation matrix

Visualization

Normalized Correlation Matrix of Edgelist File



Pearson's Correlation Between Diagonal Elements

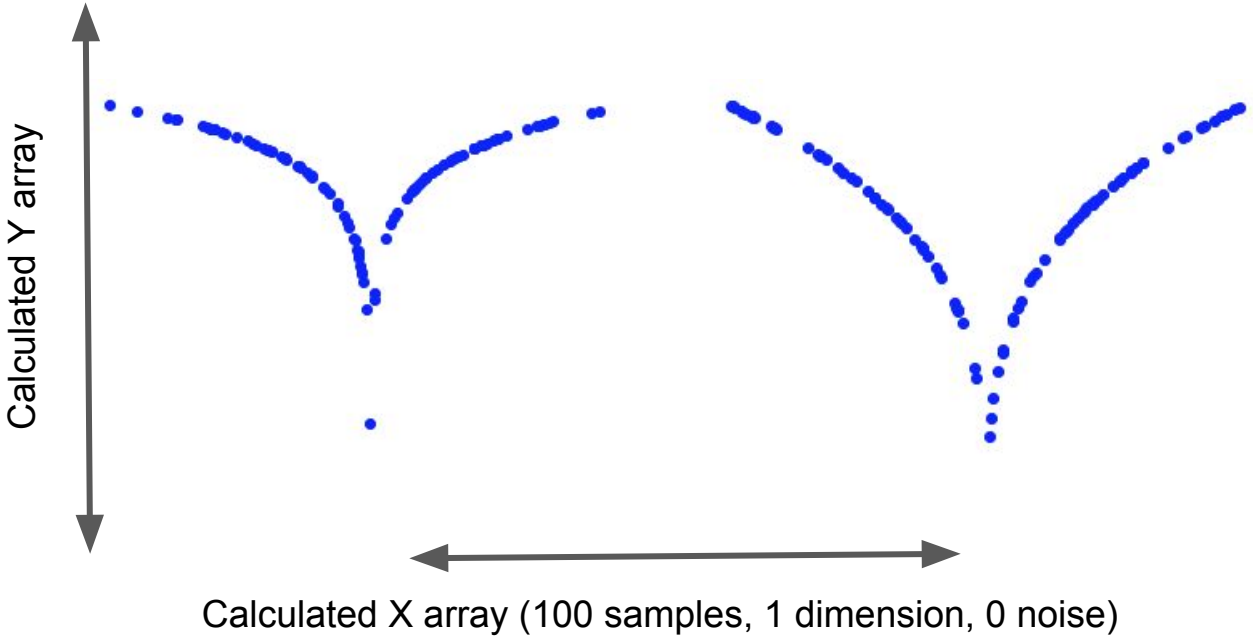


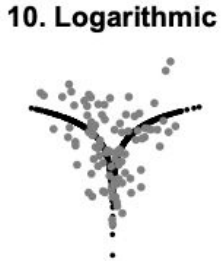
Task 1 (Sambit): Implement HHG, Pearson/RV/Cca, and Spearman/Kendall into package

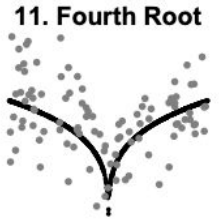
Last Week's Accomplishments:

- Merge simulations into development branch
- Begin creating other simulations mentioned in paper
- Change rv_corr from function to class
- Fixed Bernoulli simulation

Simulation plots







Logarithmic Simulation

4th Root Simulation

Stuff to do this week

- Finish the rest of the simulations in the paper
  - DoD: Plots of the simulations
- Begin creating HHG class
  - DoD: Class uploaded to Github

Task 2 (Satish): Implement MGC into the package

Last Week Accomplishments:

- Implement *MGCSampleStat* module with the *Thresholding*, *Smoothing*, *and mgc.sample* (MGC measure computation) functions from R in Python, with tests and merge PR
- Refactor mgc to extend and implement the *IndependenceTest* abstract class and merge PR

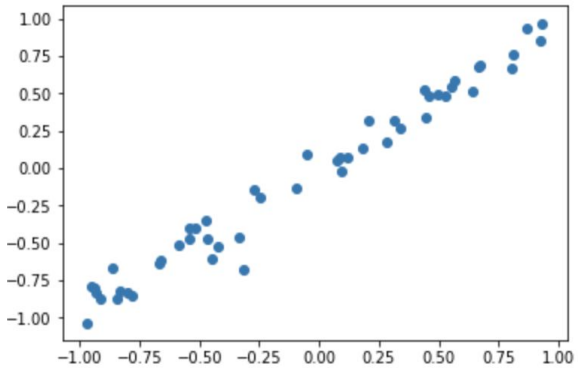
Pseudocode for MGC Test Statistic Computation

1. **[test_stat, optimal_scale] = mgc_sample(X,Y)**
2. **Inputs:** X and Y are [n*d] data matrices
3. **Outputs:** mgc test statistic and optimal scale
4. **Algorithm**
    - **Step 1:** Compute all local correlations
    - **Step 2:** Thresholding
        - Compute parametric and non-parametric threshold using beta approx. and data adaptive methods
        - Take max of both as the threshold
        - Threshold the local correlation matrix
        - Find the largest connected component of significant correlations
    - **Step 3:** Smoothing
        - Default test statistic at maximal scale
        - If connected region's area is sufficiently large
            - find the scale within the significant connected region that maximizes the local correlation
        - Return test statistic and optimal scale

```
local_corr:
array([[0.35, 0.4 , 0.2 , 0.2 ],
       [0.35, 0.  , 0.2 , 0.2 ],
       [0.1 , 0.  , 0.9 , 0.5 ],
       [0.1 , 0.  , 0.8 , 0.4 ]])

threshold: 0.21

significant_local_corr:
array([[ True,  True, False, False],
       [ True, False, False, False],
       [False, False,  True,  True],
       [False, False,  True,  True]])

label_significant_local_corr:
array([[1, 1, 0, 0],
       [1, 0, 0, 0],
       [0, 0, 2, 2],
       [0, 0, 2, 2]])

max_significant_local_corr:
array([[0, 0,   0,   0],
       [0, 0,   0,   0],
       [0, 0, 0.9, 0.5],
       [0, 0, 0.8, 0.4]])

test_statistic = 0.9
optimal_scale = [3, 3]
```

Comparison with results from R (notebook)

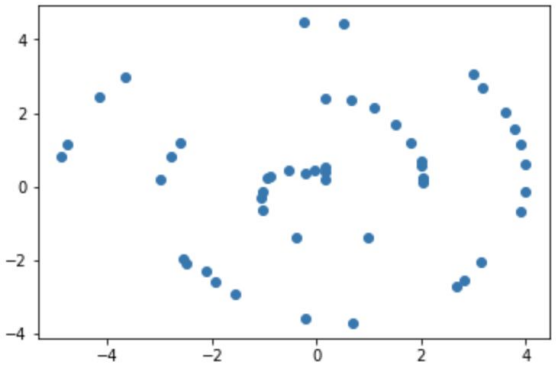
```
'''
Linear data. Generated using, simulations from R MGC.
data = mgc.sims.spiral(50, 1, 0.1) # (samples, features, noise)
'''
```




```
'''
Results from R:
> result = mgc.sample(data$X, data$Y)
> result$statMGC
[1] 0.9674665
> result$optimalScale
$x
[1] 50
$y
[1] 50
'''
```

```
MGC stats from Python:
mgc_test_statistic: 0.9674665210658532
optimal_scale: [50, 50]
```

```
'''
Non-linear (spiral data). Generated using, simulations from R MGC
data = mgc.sims.spiral(50, 1, 0.1) # (samples, features, noise)
'''
```




```
'''
Results from R:
> result = mgc.sample(data$X, data$Y)
> result$statMGC
[1] 0.1935397
> result$optimalScale
$x
[1] 2
$y
[1] 7
'''
```

```
MGC stats from Python:
mgc_test_statistic: 0.19353972252036264
optimal_scale: [2, 7]
```

Stuff to do this week

- Extract Bear's permutation test from DCorr into a common utility and use that to compute p-value in MGC.
- Run Python MGC on higher dimensional data and report the performance and accuracy, comparing to the R version.
- Write a master function in main.py that can call the different independence tests.

Task 3 (Sandhya): Implement MDMR and FastMGC into package

Last Week's Accomplishments:

- Rewrite code to automatically test all columns of X data: load data and run, functions
- Contact author again, ask about testing multiple columns at a time

```
Reloaded modules: mdmrpy
Column = 1
<class 'int'>
Fperm Statistic = [-13.70298802] P-value = [0.00990099]
Column = 2
<class 'int'>
Fperm Statistic = [-22.6870867] P-value = [0.00990099]
Column = 3
<class 'int'>
Fperm Statistic = [-2.70117626] P-value = [0.00990099]
```

```
columns = 1
#permutations = 100
print ("Column =", columns)

[a,b] = mdmr(D,X,columns, permutations)
print ("Fperm Statistic =", a, "P-value =", b)
```

3X

----------VS----------

```
Reloaded modules: mdmrpy
<class 'int'>
<class 'int'>
<class 'int'>
Column = 1
F_Perm = -13.702988017797464
P-Value = 0.009900990099009901
Column = 2
F_Perm = -22.687086699585606
P-Value = 0.009900990099009901
Column = 3
F_Perm = -2.7011762606456204
P-Value = 0.009900990099009901
```

```
38 results = mdmr(D,X)
39 for i in range(0,results.shape[0]):
40     print("Column =", int(results[i,0]))
41     print("F_Perm =", results[i,1])
42     print("P-Value =", results[i,2])
```

Next Week:

- Read MDMR paper on analytic p-values: paper
- Make changes to code for columns input as author (Anibal) suggests
- Reach: Write pseudocode or code for analytic p-values
- Discuss incorporation of finished MDMR code (with permutation p-values) with team members more experienced with github

Task 4: Implement mcorr, dcorr, mantel (Bear)

Last Week

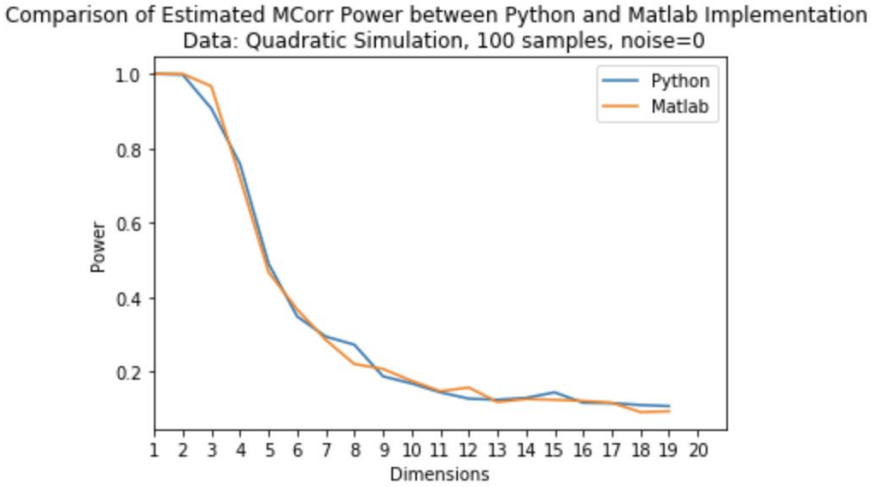
- Compute power
  - DoD: verify results against matlab code CorrIndTestDim.m
- Permutation test for p-value of dcorr, mantel
  - DoD: verify results against matlab code DCorPermutationTest.m

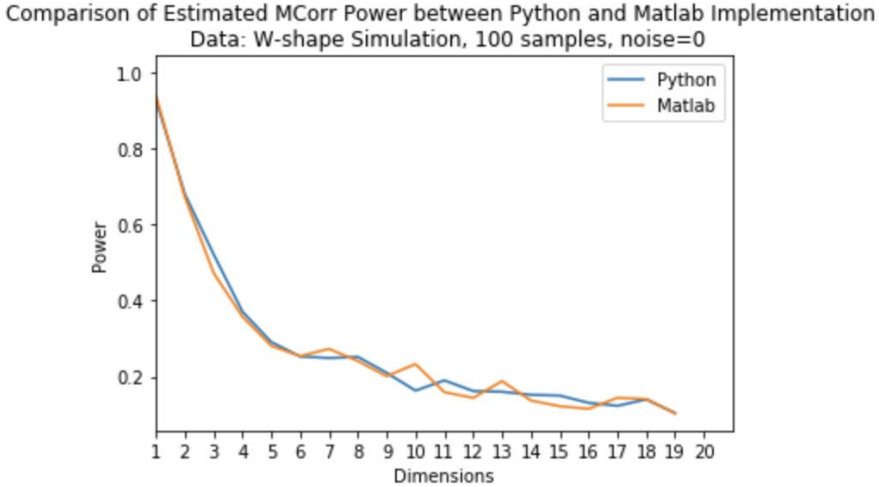
Power (code & test)

Algorithm 1 ComputePower

Input: Simulation to sample from, independence testing procedure, number of repeats, type I error level α

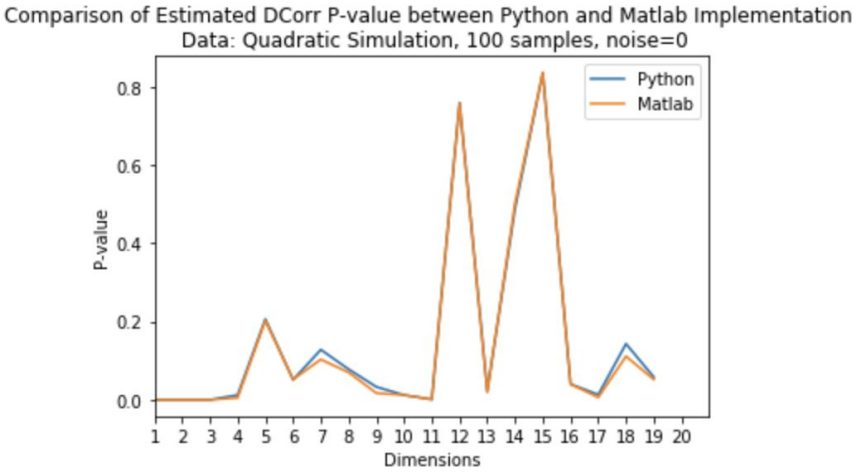
for r in range($NumRepeats$) **do**
    $X, Y \leftarrow$ Sample from Simulation
    $Y_{permuted} \leftarrow$ Permutation(Y)
    $T_{null}[r] \leftarrow$ Test Statistic($X, Y_{permuted}$)
    $T_{alt}[r] \leftarrow$ Test Statistic(X, Y)
end for
$C \leftarrow SortDescending(T_{null})[\alpha \times NumRepeats]$
return $count(T_{alt} \geq C)/NumRepeats$

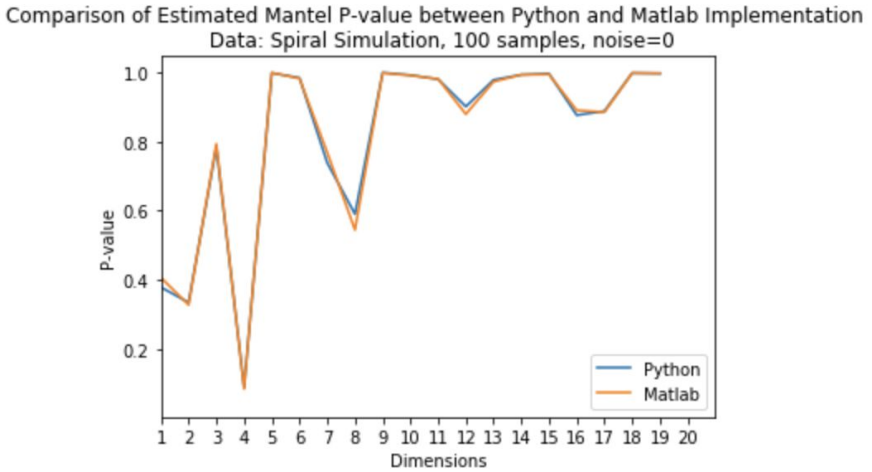




Permutation test for dcorr/mantel p-value (python code, matlab code, test)

```
# estimate the null by a permutation test
test_stats_null = np.zeros(repeats)
for rep in range(repeats):
    permuted_y = np.random.permutation(self.data_matrix_Y)
    test_stats_null[rep] = self.test_statistic(data_matrix_X=self.data_matrix_X, data_matrix_Y=permuted_y)
# p-value is the probability of observing more extreme test statistic under the null
return np.where(test_stats_null >= test_stat)[0].shape[0] / repeats
```





Next Week

- Complete testing for the dcorr module, merge code & unit tests into development branch
  - DoD: Merged PR
- Start graph independence test?
  - DoD: Discuss w/ jovo

Task 5 (Richard): Random Forest Independence Test

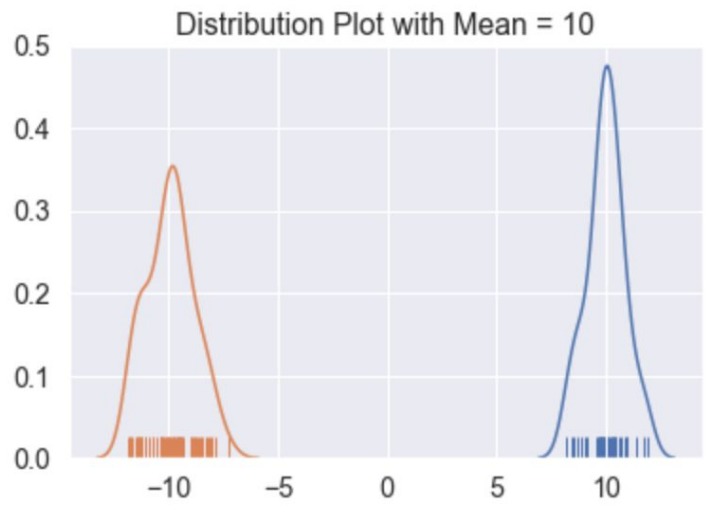
Last Week's Accomplishments:

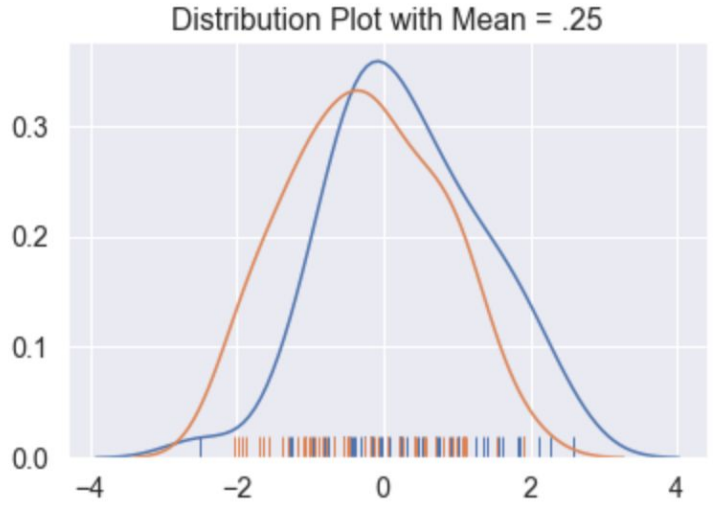
- Clean up jupyter notebooks to show initial results of algorithm
  - DoD: Link to very clean jupyter notebook
- Showing convergence of random forest estimator in simple case (LINK)

Does our random forest estimate converge to the true conditional entropy?

Experiment Setup:

1. Let Y ~ Bernoulli(½) (Y = -1 or 1)
2. Let X ~ Normal(y*mu, 1)

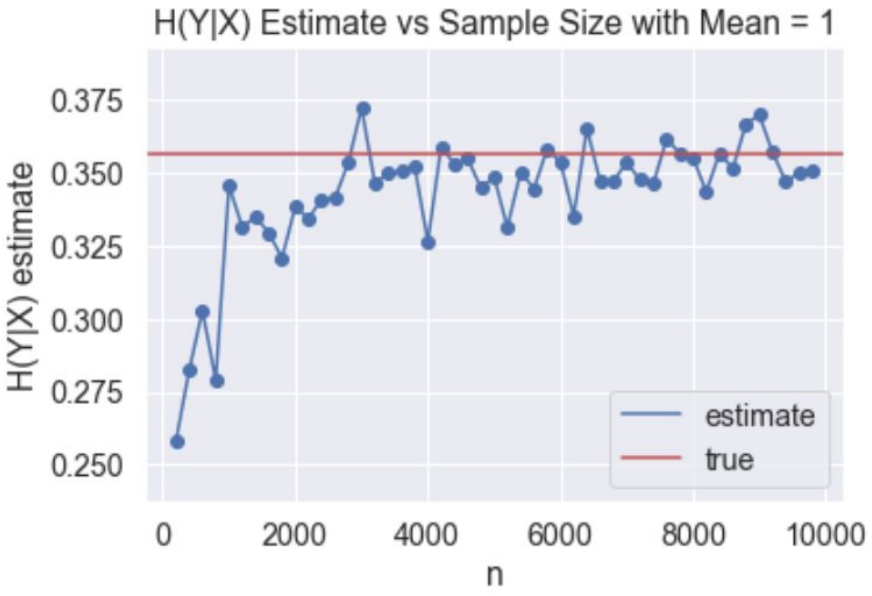


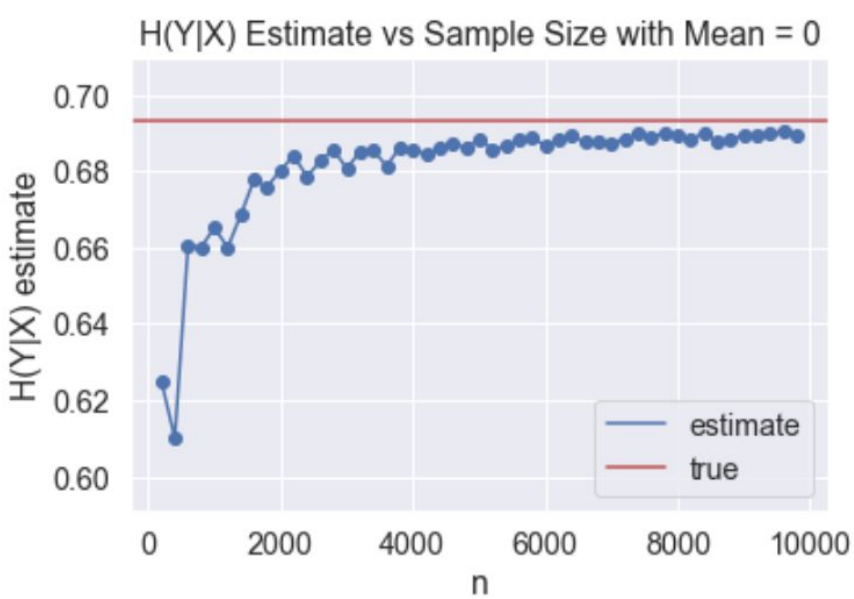


We want to ask these two questions. Given n (X_i, Y_i) samples:

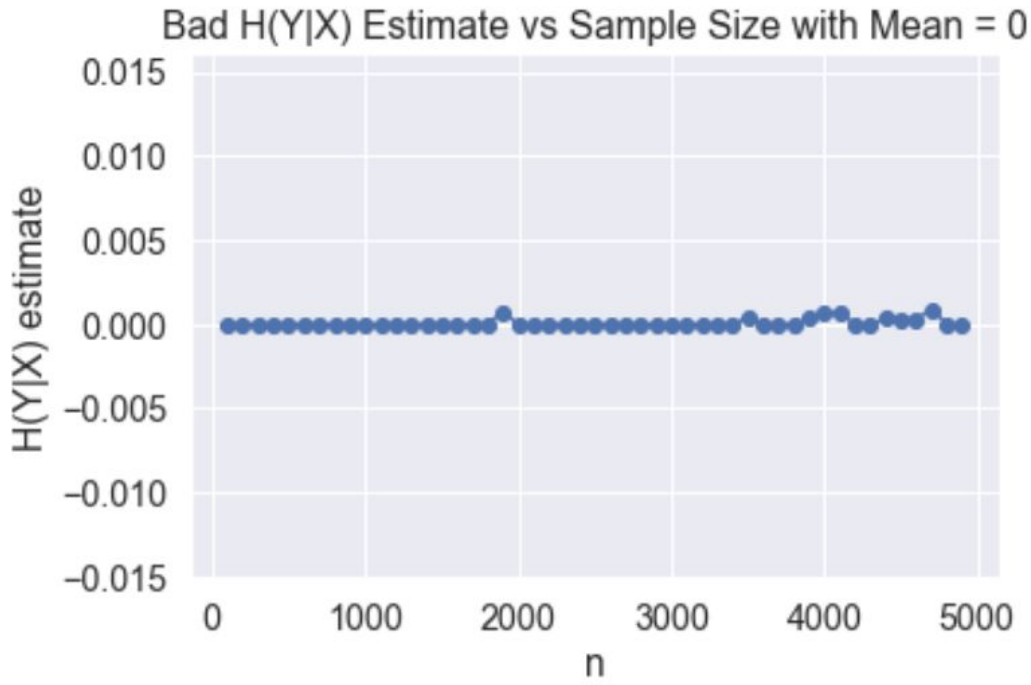
1. Does our estimate converge to true H(Y|X) as n -> \infty?
2. Does our estimate converge to 0 as mu -> \infty?

Does our estimate converge to true H(Y|X) as n -> infinity?

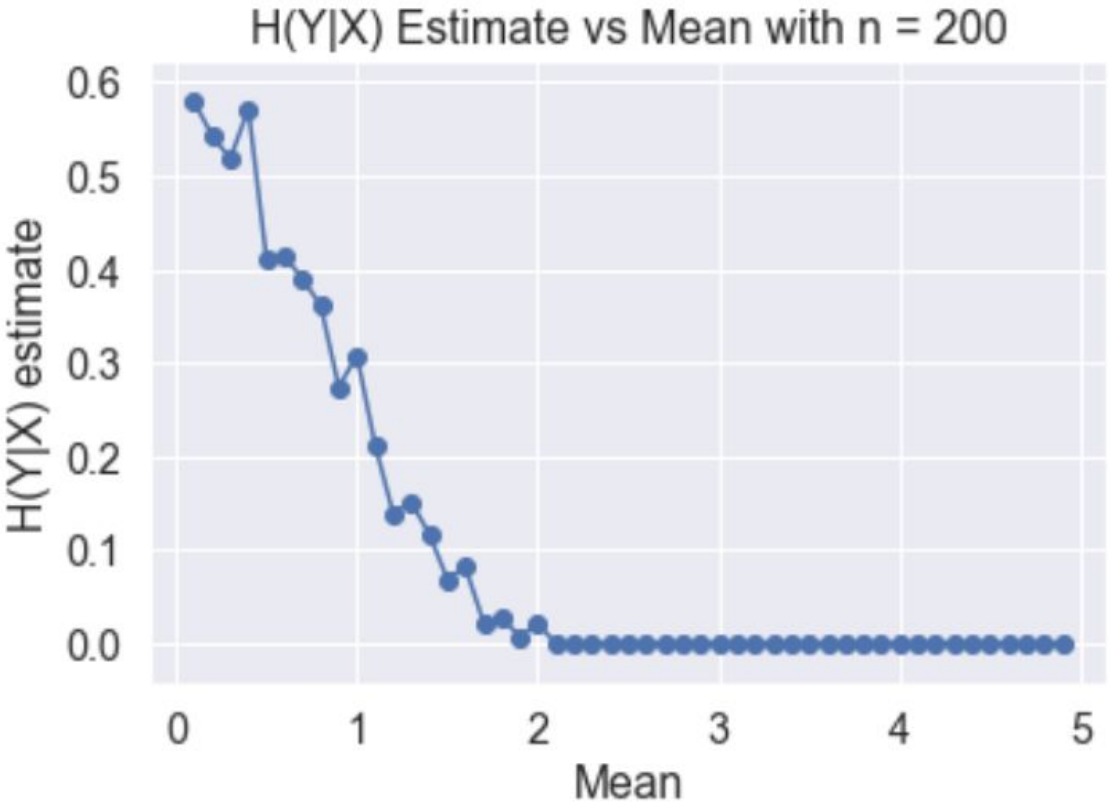




When our trees overfit, bad things happen!



Does our estimate converge to 0 as mu -> infinity?



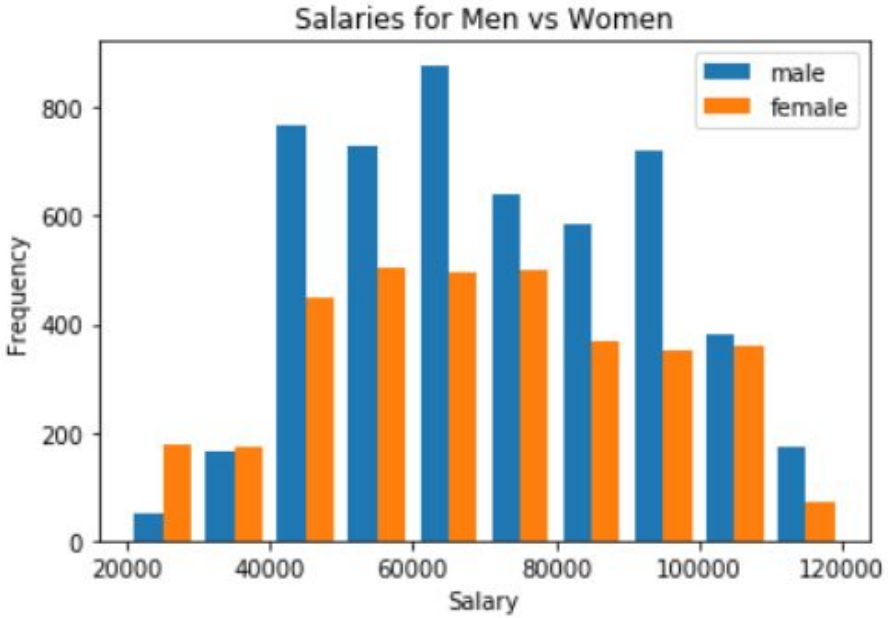
Next Week:

- Talk to jovo about how to prevent random forest from overfitting
- Talk to jovo about new interesting tests to run

Task 6 (Ananya): Implement 2-sample tests into package

- Normalize kernel matrix
  - DoD: Verify equality between ENERGY and MMD holds after normalization
- Run ENERGY in R
  - DoD: Make sure R-ENERGY and P-ENERGY output the same test statistic
- Run MMD in R
  - DoD: Make sure R-MMD and P-MMD output the same test statistic

Looking at the data



Employee salaries in Montgomery, MD (2017) (https://catalog.data.gov/dataset/employee-salaries-2017)

Code for histogram here

Kernel matrix must be normalized before using MMD

- Use k(xi,xj) = 1 - d(xi,xj)/(max d(xi,xj)) to get kernel matrix from distance matrix
-

Theorem 3. When the kernel is bijective of the metric, sample Hsic equals sample Dcov up to scaling by the maximum distance and kernel entries:

$$\mathtt{Dcov}_N(\mathbf{X},\mathbf{Y}) / \max_{i,j=1,\dots,n}(\mathbf{D}^{\mathbf{X}}_{ij}) \max_{i,j=1,\dots,n}(\mathbf{D}^{\mathbf{Y}}_{ij}) = \mathtt{Hsic}_N(\mathbf{X},\mathbf{Y}) / \max_{i,j=1,\dots,n}(\hat{\mathbf{K}}^{\mathbf{X}}_{ij}) \max_{i,j=1,\dots,n}(\hat{\mathbf{K}}^{\mathbf{Y}}_{ij}), \quad (8)$$

and similarly for bijective induced metric. They have the same normalized sample statistic as described by Equation 1, and the same p-value via permutation test.

ENERGY and MMD output the same test statistic after normalization (yay)

Code for ENERGY here
Test statistic: 0.000163

Code for MMD here
Test statistic: 0.000163

R-ENERGY and P-ENERGY do not output the same test statistic

Code for R-ENERGY here

```
> eqdist.e(x,sizes,distance=FALSE,method=c("original"))
E-statistic
   929626.7
```