

results_for_manuscript

December 17, 2022

1 Effective Connectivity and Bias Entropy Improve Prediction of Dynamical Regime in Automata Networks

1.1 Supplemental Notebook

1.1.1 Input data:

This notebook reads RBN metadata stored in folders `Results20`, `Results50`, `Results100`, and `Results200`, which are merged and stored in `full_RBN_data.csv` with some preprocessing. Cell Collective network data is also read from the `cc_network_data.csv` file.

The RBN results can be generated using the `PowerLaw_Experiment.py` script. Cell Collective summary data can be generated using the `cc_analysis.py` script.

1.1.2 Analysis and requirements:

All figures from the accompanying manuscript are reproducible by running this notebook. Additional figures are also produced here to supplement the main text. This notebook require `panda`, `numpy`, `sklearn`, `scipy`, and `matplotlib` to be installed.

We begin by importing various libraries and functions we will need for the analysis.

```
[ ]: import pandas as pd
      import numpy as np

      import itertools

      from os import listdir

      from sklearn import metrics
      from sklearn.metrics import RocCurveDisplay
      from scipy import stats, special

      import matplotlib.pyplot as plt
      from matplotlib.patches import Patch
      from matplotlib.lines import Line2D
      from matplotlib.colors import CenteredNorm
      %matplotlib inline
```

Next, we define various helper functions for preprocessing the data we will import.

```
[ ]: def merge_results_frames(df,df2):
    for idx,r in df.iterrows():
        g = r['gamma']
        p = r['bias']
        for col in df.columns:
            if col == 'gamma' or col == 'bias':
                continue
            df.loc[(df['gamma']==g) & (df['bias']==p),col] += df2.
        ↪loc[(df2['gamma']==g) & (df2['bias']==p),col]

def str2list(strarray):
    return [float(x) for x in strarray.strip('[] ').split()]

def df_str2list(df):
    for col in df.columns:
        if col == 'gamma' or col == 'bias':
            continue
        df[col] = df[col].apply(str2list)

def expand_shared_properties_in_dict(df):
    dictall = {}
    for _,row in df.iterrows():
        for col in df.columns:
            if col in ['gamma','bias']:
                rowlist = [row[col]]*len(row['Derrida'])
            else:
                rowlist = row[col]
            if col in dictall:
                dictall[col] += rowlist
            else:
                dictall[col] = rowlist.copy()
    return dictall

def append_extra_columns(dictall):
    dfall = pd.DataFrame(dictall)

    dfall['entropy'] = -(dfall['bias']*np.
    ↪log2(dfall['bias']))-((1-dfall['bias'])*np.log2(1-dfall['bias']))
    dfall['variance'] = dfall['bias']*(1-dfall['bias'])

    dfall['regime']=(dfall['Derrida']>1).astype(int) - (dfall['Derrida']<1).
    ↪astype(int)
    dfall['avgS']=dfall['avgKe']-dfall['avgKc']
    dfall['avgH'] = -dfall['avgP']*np.log2(dfall['avgP'])-(1-dfall['avgP'])*np.
    ↪log2(1-dfall['avgP'])
    dfall['avgV'] = (1-dfall['avgP'])*dfall['avgP']
```

```
    return dfall
```

We now import the RBN data. Note that the variable REGENERATE_DATA determines whether the data will be read from the individual results folder to create a new full_RBN_data.csv file (if True) or read from from a previously assembled full_RBN_data.csv file (if False).

```
[ ]: REGENERATE_DATA=False # takes a couple of minutes to regenerate data frame from individual csv files

results_dir_dict={20:'Results20/',50:'Results50/',100:'Results100/',200:'Results200/'}
if REGENERATE_DATA:
    dfall = pd.DataFrame()
    for N,results_dir in results_dir_dict.items():
        first_in_N = True
        for filename in listdir(results_dir):
            if first_in_N:
                df = pd.read_csv(results_dir+filename)
                df_str2list(df)
                first_in_N = False
            else:
                df2 = pd.read_csv(results_dir+filename)
                df_str2list(df2)
                merge_results_frames(df,df2)
        dictall = expand_shared_properties_in_dict(df)
        dfallN = append_extra_columns(dictall)
        dfallN['N']=N
        dfall = pd.concat([dfall,dfallN],axis=0)
    dfall.to_csv('full_RBN_data.csv')
else:
    dfall=pd.read_csv('full_RBN_data.csv')
Nvals = results_dir_dict.keys()
dfall['gamma'] = dfall['gamma'].round(2) # to fix some entries recorded, e.g., like 2.10000005
dfall['bias'] = dfall['bias'].round(2)
# shuffle data
dfall=dfall.sample(frac=1.0)

dfall
```

```
[ ]:   Unnamed: 0  gamma  bias  Derrida  avgK  medK      avgKe  medKe \
37827      1827  1.5  0.25     1.262  3.400    2.0  2.109341  1.125
83132      11132  1.8  0.05     0.218  2.390    1.0  0.650048  0.000
119595     11595  1.8  0.10     0.372  2.225    1.0  0.814275  0.000
15895      15895  1.9  0.20     0.530  1.900    1.0  0.642839  0.000
58091      22091  2.1  0.10     0.458  1.980    1.0  0.763881  0.000
```

| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
|--------|----------|----------|----------|----------|---------|----------|----------|----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 34465 | 34465 | 2.4 | 0.30 | 0.855 | 1.900 | 1.0 | 1.166268 | 1.000 | | | | | | | | | |
| 16022 | 16022 | 1.9 | 0.25 | 0.437 | 1.700 | 1.0 | 0.674432 | 0.000 | | | | | | | | | |
| 101965 | 29965 | 2.3 | 0.15 | 0.432 | 1.620 | 1.0 | 0.583085 | 0.000 | | | | | | | | | |
| 36720 | 720 | 1.5 | 0.10 | 0.369 | 2.520 | 1.0 | 0.935962 | 0.000 | | | | | | | | | |
| 65708 | 29708 | 2.3 | 0.15 | 0.320 | 1.460 | 1.0 | 0.423904 | 0.000 | | | | | | | | | |
| | | avgKc | medKc | avgP | entropy | variance | regime | avgS | \ | | | | | | | | |
| 37827 | 0.849507 | 0.0 | 0.252906 | 0.811278 | 0.1875 | | 1 | 1.259834 | | | | | | | | | |
| 83132 | 0.429796 | 0.0 | 0.051321 | 0.286397 | 0.0475 | | -1 | 0.220251 | | | | | | | | | |
| 119595 | 0.421050 | 0.0 | 0.099204 | 0.468996 | 0.0900 | | -1 | 0.393225 | | | | | | | | | |
| 15895 | 0.114714 | 0.0 | 0.214286 | 0.721928 | 0.1600 | | -1 | 0.528125 | | | | | | | | | |
| 58091 | 0.312887 | 0.0 | 0.098959 | 0.468996 | 0.0900 | | -1 | 0.450994 | | | | | | | | | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 34465 | 0.324105 | 0.0 | 0.297468 | 0.881291 | 0.2100 | | -1 | 0.842163 | | | | | | | | | |
| 16022 | 0.250018 | 0.0 | 0.246860 | 0.811278 | 0.1875 | | -1 | 0.424414 | | | | | | | | | |
| 101965 | 0.153241 | 0.0 | 0.167824 | 0.609840 | 0.1275 | | -1 | 0.429844 | | | | | | | | | |
| 36720 | 0.545825 | 0.0 | 0.094464 | 0.468996 | 0.0900 | | -1 | 0.390137 | | | | | | | | | |
| 65708 | 0.106248 | 0.0 | 0.137755 | 0.609840 | 0.1275 | | -1 | 0.317656 | | | | | | | | | |
| | | avgH | avgV | N | | | | | | | | | | | | | |
| 37827 | 0.815852 | 0.188945 | 50 | | | | | | | | | | | | | | |
| 83132 | 0.291982 | 0.048687 | 100 | | | | | | | | | | | | | | |
| 119595 | 0.466469 | 0.089363 | 200 | | | | | | | | | | | | | | |
| 15895 | 0.749595 | 0.168367 | 20 | | | | | | | | | | | | | | |
| 58091 | 0.465687 | 0.089166 | 50 | | | | | | | | | | | | | | |
| ... | ... | ... | ... | | | | | | | | | | | | | | |
| 34465 | 0.878174 | 0.208981 | 20 | | | | | | | | | | | | | | |
| 16022 | 0.806263 | 0.185920 | 20 | | | | | | | | | | | | | | |
| 101965 | 0.652703 | 0.139659 | 100 | | | | | | | | | | | | | | |
| 36720 | 0.451196 | 0.085540 | 50 | | | | | | | | | | | | | | |
| 65708 | 0.578329 | 0.118779 | 50 | | | | | | | | | | | | | | |

[144000 rows x 18 columns]

We define a helper function for finding the critical boundary. The thermodynamic critical boundary depends upon the average in-degree. We therefore define a function to calculate this mean (in the thermodynamic limit) from the generating parameter `gamma` (γ) assuming a truncated power-law distribution with exponent γ and cutoff m (here explicitly called `cutoff`).

```
[ ]: def kmean(gamma,cutoff=None):
    if cutoff is None:
        return special.zeta(gamma-1)/special.zeta(gamma)
    else:
        ks = np.arange(1,cutoff+1).astype(float)
        return np.sum(ks**(1-gamma)) / np.sum(ks**(-gamma))
```

We explore how finite-size effects influence the critical boundary. The figures produced here show the fraction of networks for each sampled point in parameter space that have Derrida coefficients (δ) greater than 1 (chaotic, in red) or less than 1 (ordered, in blue). Bold outlined points have at least a 15 – 85% split between ordered and chaotic networks. These allow us to visualize a “fuzzy” critical boundary for various sizes of networks.

```
[ ]: biases = np.arange(0.05,0.5,0.05).round(2)
gammas = np.arange(1.5,2.5,0.1).round(2)
fs=32
fsa=24
fsl=20
for N in Nvals:
    fig, ax = plt.subplots(figsize=(10, 10),facecolor='white')

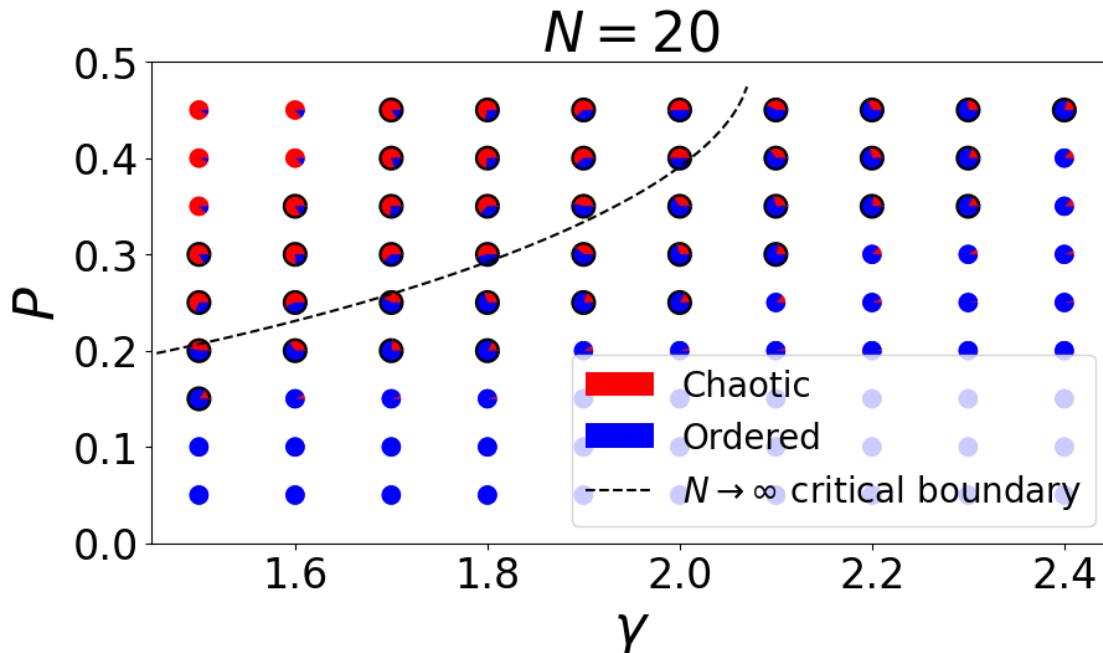
    xvals = np.arange(1.4, 2.6, 0.005)
    ax.plot(xvals,[0.5-0.5*np.sqrt(1-2/kmean(xg,cutoff=15)) for xg in
    ↪xvals], 'k--')

    for g,p in itertools.product(gammas,biases):
        g=g.round(2)
        p=p.round(2)
        ones = (dfall[(dfall['N'] == N) & (dfall['gamma'] == g) &
        ↪(dfall['bias'] == p)]['Derrida']>1).mean()
        ax.pie([ones, 1-ones], center=(g, p), radius=0.01, colors=['r', 'b'],
        ↪frame=True)
        if ones > 0.85 or ones < 0.15:
            ax.pie([ones, 1-ones], center=(g, p), radius=0.01, colors=['r',
            ↪'b'], frame=True)
        else:
            ax.pie([ones, 1-ones], center=(g, p), radius=0.013, colors=['k',
            ↪'k'], frame=True)
            ax.pie([ones, 1-ones], center=(g, p), radius=0.01, colors=['r',
            ↪'b'], frame=True)
        ax.set_xlim(1.45,2.45)
        ax.set_ylim(0,0.5)

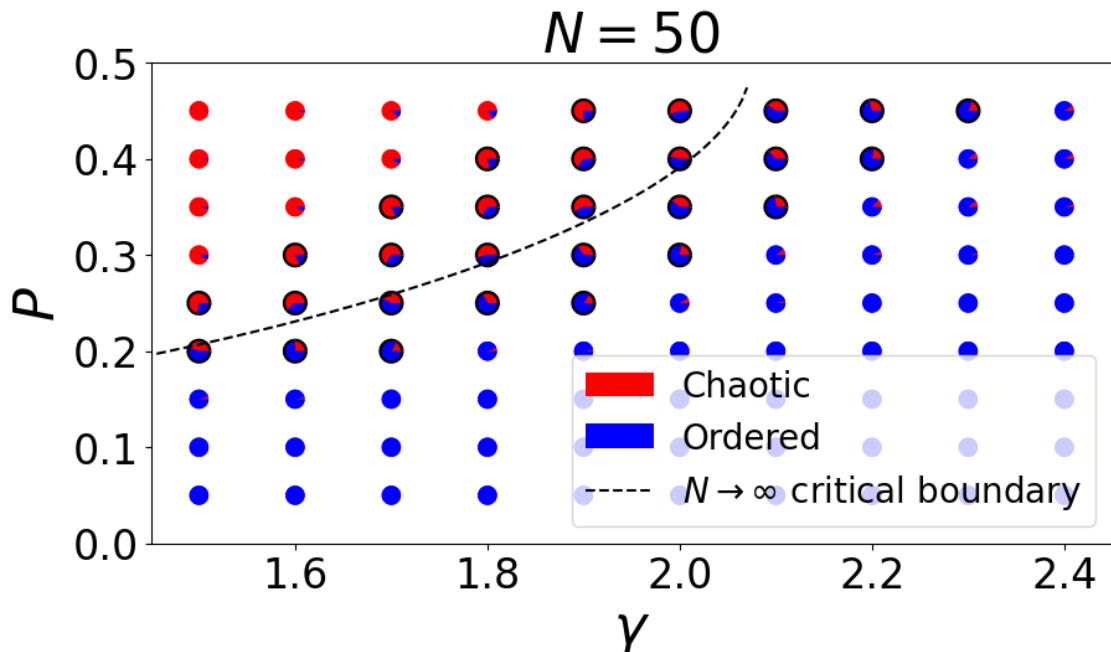
    legend_elements = [Patch(facecolor='r', label='Chaotic'),
    ↪Patch(facecolor='b', label='Ordered'),
    ↪Line2D([0],[0],color='k',linestyle='--',label=r'$N\rightarrow\infty$',
    ↪critical boundary)]
    plt.title(f'$N={N}$',fontsize=fs)
    plt.xlabel('$\gamma$',fontsize=fs)
    plt.ylabel('$P$',fontsize=fs)
    ax.tick_params(axis='both', which='major', labelsize=fsa)
    ax.legend(handles=legend_elements, loc='lower right',fontsize=fsl)
    plt.savefig(f'figures/PieChartFigure_{N}.pdf',bbox_inches='tight')
```

```
plt.savefig(f'figures/PieChartFigure_{N}.png',bbox_inches='tight')
plt.show()
```

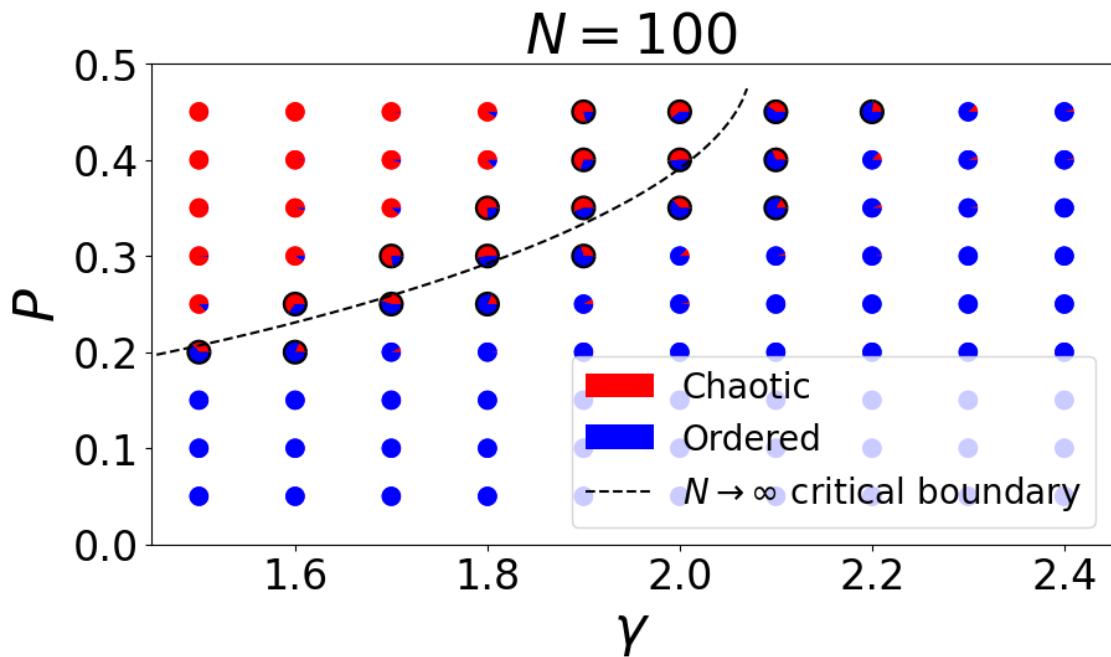
C:\Users\jcroz\AppData\Local\Temp\ipykernel_7764\1460200497.py:10:
RuntimeWarning: invalid value encountered in sqrt
 ax.plot(xvals,[0.5-0.5*np.sqrt(1-2/kmean(xg,cutoff=15)) for xg in
xvals],'k--')



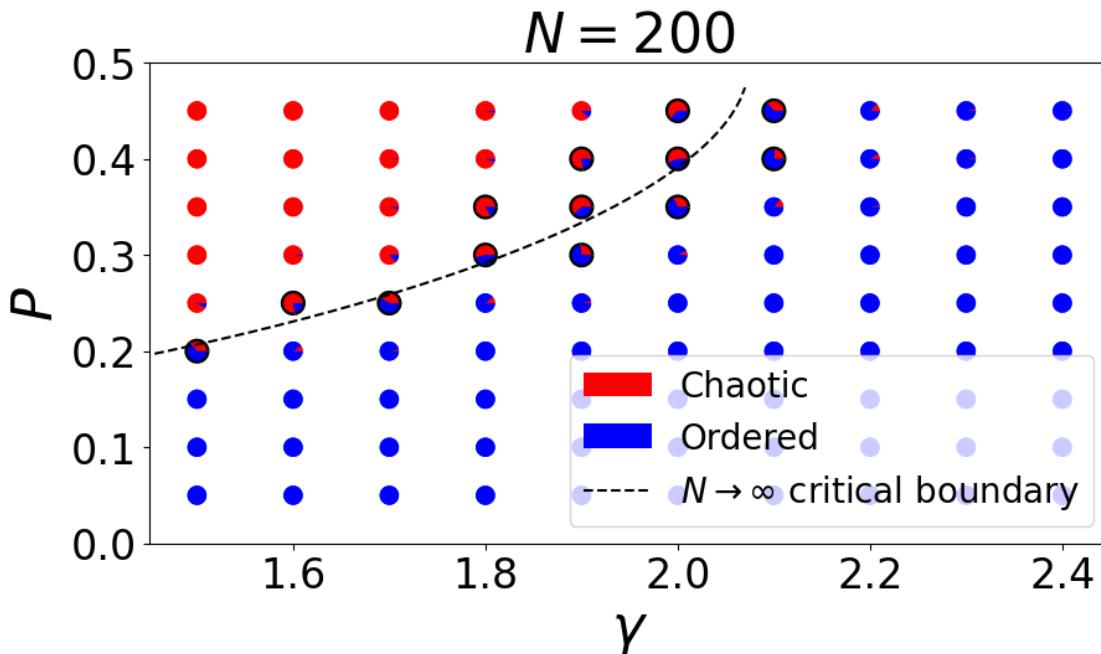
C:\Users\jcroz\AppData\Local\Temp\ipykernel_7764\1460200497.py:10:
RuntimeWarning: invalid value encountered in sqrt
 ax.plot(xvals,[0.5-0.5*np.sqrt(1-2/kmean(xg,cutoff=15)) for xg in
xvals],'k--')



```
C:\Users\jcroz\AppData\Local\Temp\ipykernel_7764\1460200497.py:10:
RuntimeWarning: invalid value encountered in sqrt
    ax.plot(xvals,[0.5-0.5*np.sqrt(1-2/kmean(xg,cutoff=15)) for xg in
xvals],'k--')
```



```
C:\Users\jcroz\AppData\Local\Temp\ipykernel_7764\1460200497.py:10:
RuntimeWarning: invalid value encountered in sqrt
    ax.plot(xvals,[0.5-0.5*np.sqrt(1-2/kmean(xg,cutoff=15)) for xg in
xvals],'k--')
```



As expected, the fuzzy boundary becomes sharper as the size of the networks increases, and the boundary appears to be converging to the theoretical prediction.

We consider various ways to fit the critical boundary. We begin by defining the functional forms we will attempt to fit to and associated helper functions.

```
[ ]: from scipy.optimize import curve_fit
def powerlaw(x,c,m,b):
    return c*x**m + b

def linear_law(x, m, b):
    return m*x + b

def critical_kappa(rho,c,m,b):
    return (1/rho)*((1-b)/c)**(1/m)

def critical_kappa_lin(rho,m,b):
    return (1-b)/(m*rho)

def mse(x,y,f):
    return np.nanmean((f(x)-y)**2)
```

We fit power law functions to the critical boundary in RBNs. As described in the main text, we fit $\sigma^2 k$, $\sigma^2 k_e$, and Hk_e to δ using a power law function for various sizes of network.

```
[ ]: fs = 36
fsl = 28
fsa = 28
for N in Nvals:
    dfN = dfall[dfall['N']==N]

    fig, ax = plt.subplots(1,3,figsize=(30, 10),facecolor='white')
    cmap = plt.cm.get_cmap('coolwarm')#.reversed()
    x=dfN['avgK']*dfN['avgV']
    y=dfN['Derrida']
    popt, pcov = curve_fit(powerlaw,x[np.isfinite(x)],dfN['Derrida'][np.
    ↪isfinite(x)])
    c, m, b = popt
    xfit=np.arange(x.min(),x.max(),0.01)
    yfit=powerlaw(xfit,c,m,b)
    gof = mse(x,dfN['Derrida'],lambda xi: powerlaw(xi,c,m,b))
    mstr='{' + str(np.round(m,2)) + '}'
    label = f'${np.round(c,2)}(\sigma^2 k)^{mstr}{np.round(b,2):+} $\n$MSE={np.
    ↪round(gof,5)}$'
    ax[0].plot(xfit,yfit,'k--', label=label)
    sc=ax[0].scatter(x, y, s=1,
                      c=dfN['Derrida'], cmap=cmap,
                      norm=CenteredNorm(vcenter=1,halfrange=0.5))
    ax[0].set_xlabel("$\sigma^2 k$", fontsize=fs)
    ax[0].set_ylabel("$\delta$ ", fontsize=fs)
    ax[0].tick_params(axis='both', which='major', labelsize=fsa)
    ax[0].legend(fontsize=fsl, loc = 'lower right')
    ax[0].set_ylim(0,3)
    ax[0].set_xlim(0,1.5)

    x=dfN['avgKe']*dfN['avgV']
    popt, pcov = curve_fit(powerlaw,x[np.isfinite(x)],dfN['Derrida'][np.
    ↪isfinite(x)])
    c, m, b = popt
    xfit=np.arange(x.min(),x.max(),0.01)
    yfit=powerlaw(xfit,c,m,b)
    gof = mse(x,dfN['Derrida'],lambda xi: powerlaw(xi,c,m,b))
    mstr='{' + str(np.round(m,2)) + '}'
    label = f'${np.round(c,2)}(\sigma^2 k_e)^{mstr}{np.round(b,2):+} $\n$MSE={np.
    ↪round(gof,5)}$'
    ax[1].plot(xfit,yfit,'k--', label=label)
    sc=ax[1].scatter(x, y, s=1,
                      c=dfN['Derrida'], cmap=cmap,
                      norm=CenteredNorm(vcenter=1,halfrange=0.5))
```

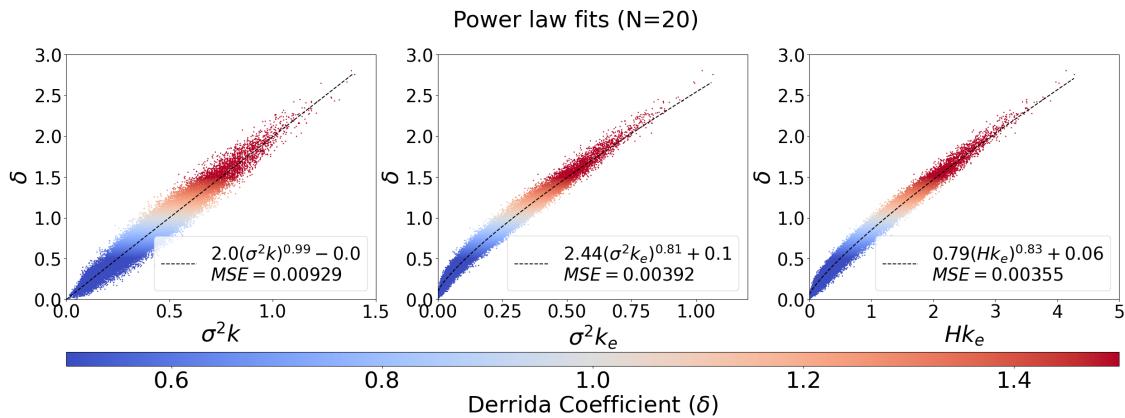
```

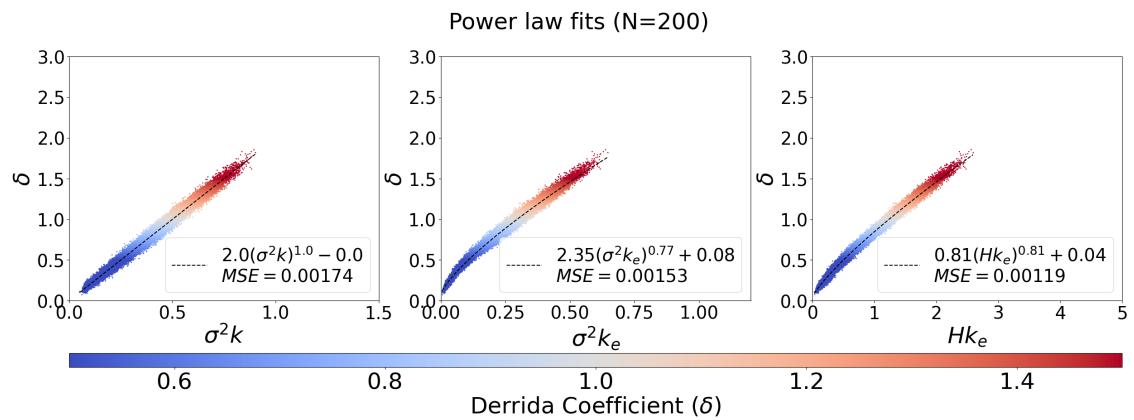
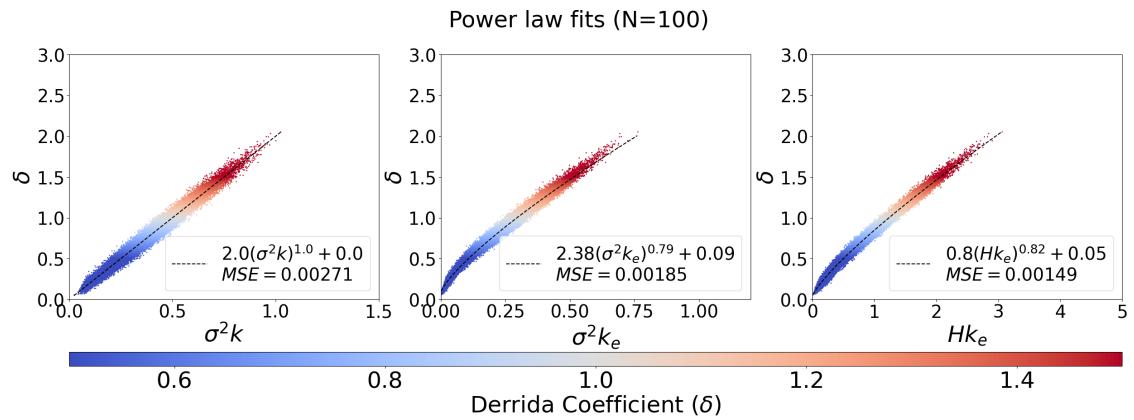
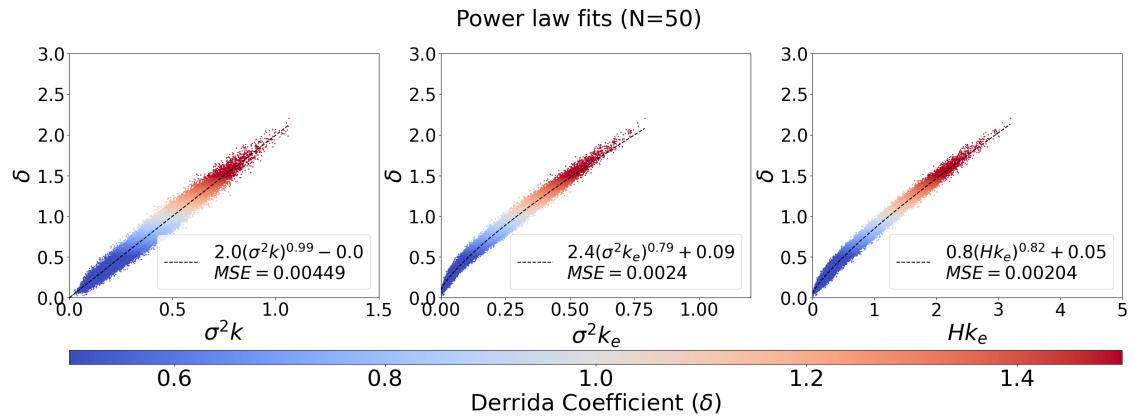
ax[1].set_xlabel("$\sigma^2 k_e$", fontsize=fs)
ax[1].set_ylabel("$\delta$", fontsize=fs)
ax[1].tick_params(axis='both', which='major', labelsize=fs)
ax[1].legend(fontsize=fsl, loc = 'lower right')
ax[1].set_ylim(0,3)
ax[1].set_xlim(0,1.2)

x=dfN['avgKe']*dfN['avgH']
popt, pcov = curve_fit(powerlaw,x[np.isfinite(x)],dfN['Derrida'][np.
isfinite(x)])
c, m, b = popt
xfit=np.arange(x.min(),x.max(),0.01)
yfit=powerlaw(xfit,c,m,b)
gof = mse(x,dfN['Derrida'],lambda xi: powerlaw(xi,c,m,b))
mstr='{'+str(np.round(m,2))+'}'
label = f'${np.round(c,2)}{(Hk_e)^{mstr}}{np.round(b,2)}:$\n$MSE={np.
round(gof,5)}$'
ax[2].plot(xfit,yfit,'k--',label=label)
sc=ax[2].scatter(x, y, s=1,
                  c=dfN['Derrida'], cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5))
ax[2].set_xlabel("Hk_e", fontsize=fs)
ax[2].set_ylabel("$\delta$", fontsize=fs)
ax[2].tick_params(axis='both', which='major', labelsize=fs)
ax[2].legend(fontsize=fsl, loc = 'lower right')
ax[2].set_ylim(0,3)
ax[2].set_xlim(0,5)

cbar=fig.colorbar(sc,ax=ax,location='bottom',aspect=75)
cbar.ax.tick_params(axis='both',labelsize=fs)
cbar.set_label('Derrida Coefficient ($\delta$)', fontsize=fs)
plt.suptitle(f'Power law fits (N={N})', fontsize=fs)
# plt.savefig(f'figures/PowerLawFits_RBN_{N}.pdf',bbox_inches='tight')
plt.savefig(f'figures/PowerLawFits_RBN_{N}.png',bbox_inches='tight')
plt.show()

```





Note a small but consistent improvement in the mean squared error when moving from left to right

in the above figures.

We repeat the procedure using a partial linear fit. The values of δ for the partial fit are chosen so that the data are approximately linear in all cases and for agreement with values calculated for empirical models, which will be analyzed later.

```
[ ]: fs = 36
fsl = 28
fsa = 28
for N in Nvals:
    dfN = dfall[dfall['N']==N]
    dselect = (1.5>dfN['Derrida']) & (dfN['Derrida']>0.5)

    fig, ax = plt.subplots(1,3,figsize=(30, 10),facecolor='white')
    cmap = plt.cm.get_cmap('coolwarm').reversed()
    x=dfN['avgK']*dfN['avgV'] / (dselect)
    y=dfN['Derrida']
    popt, pcov = curve_fit(linear_law,x[np.isfinite(x)],dfN['Derrida'][np.
    isfinite(x)])
    x=dfN['avgK']*dfN['avgV']
    m, b = popt
    xfit=np.arange(x.min(),x.max(),0.01)
    yfit=linear_law(xfit,m,b)
    gof = mse(x[dselect],dfN['Derrida'][dselect],lambda xi: linear_law(xi,m,b))
    label = f'${np.round(m,2)}(\sigma^2_k){np.round(b,2)}$' + '\n$MSE={np.
    round(gof,5)}$'
    ax[0].plot(xfit,yfit,'k--', label=label)
    sc=ax[0].scatter(x, y, s=1,
                      c=dfN['Derrida'], cmap=cmap,
                      norm=CenteredNorm(vcenter=1,halfrange=0.5))
    ax[0].set_xlabel("$\sigma^2_k$", fontsize=fs)
    ax[0].set_ylabel("$\delta$ ", fontsize=fs)
    ax[0].tick_params(axis='both', which='major', labelsize=fsa)
    ax[0].legend(fontsize=fsl, loc = 'lower right')
    ax[0].set_ylim(0,3)
    ax[0].set_xlim(0,1.5)

    x=dfN['avgKe']*dfN['avgV'] / (dselect)
    popt, pcov = curve_fit(linear_law,x[np.isfinite(x)],dfN['Derrida'][np.
    isfinite(x)])
    x=dfN['avgKe']*dfN['avgV']
    m, b = popt
    xfit=np.arange(x.min(),x.max(),0.01)
    yfit=linear_law(xfit,m,b)
    gof = mse(x[dselect],dfN['Derrida'][dselect],lambda xi: linear_law(xi,m,b))
    label = f'${np.round(m,2)}(\sigma^2_{k_e}){np.round(b,2)}$' + '\n$MSE={np.
    round(gof,5)}$'
```

```

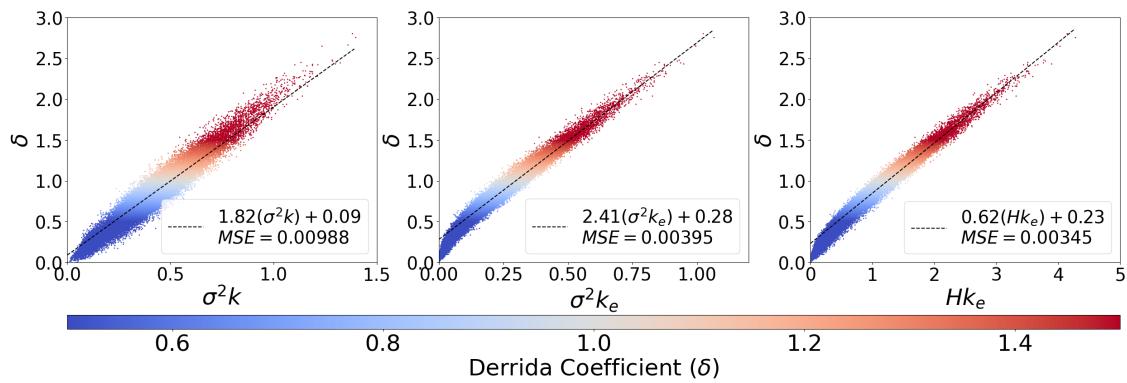
ax[1].plot(xfit,yfit,'k--', label=label)
sc=ax[1].scatter(x, y, s=1,
                  c=dfN['Derrida'], cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5))
ax[1].set_xlabel("$\sigma^2 k_e$", fontsize=fs)
ax[1].set_ylabel("$\delta$ ", fontsize=fs)
ax[1].tick_params(axis='both', which='major', labelsize=fs)
ax[1].legend(fontsize=fsl, loc = 'lower right')
ax[1].set_ylim(0,3)
ax[1].set_xlim(0,1.2)

x=dfN['avgKe']*dfN['avgH']
x=dfN['avgKe']*dfN['avgH'] / (dselect)
popt, pcov = curve_fit(linear_law,x[np.isfinite(x)],dfN['Derrida'][np.
↪isfinite(x)])
x=dfN['avgKe']*dfN['avgH']
m, b = popt
xfit=np.arange(x.min(),x.max(),0.01)
yfit=linear_law(xfit,m,b)
gof = mse(x[dselect],dfN['Derrida'][dselect],lambda xi: linear_law(xi,m,b))
label = f'${np.round(m,2)}$(Hk_e)${np.round(b,2)}$ MSE=[np.round(gof,5)]$'
ax[2].plot(xfit,yfit,'k--',label=label)
sc=ax[2].scatter(x, y, s=1,
                  c=dfN['Derrida'], cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5))
ax[2].set_xlabel("Hk_e", fontsize=fs)
ax[2].set_ylabel("$\delta$ ", fontsize=fs)
ax[2].tick_params(axis='both', which='major', labelsize=fs)
ax[2].legend(fontsize=fsl, loc = 'lower right')
ax[2].set_ylim(0,3)
ax[2].set_xlim(0,5)

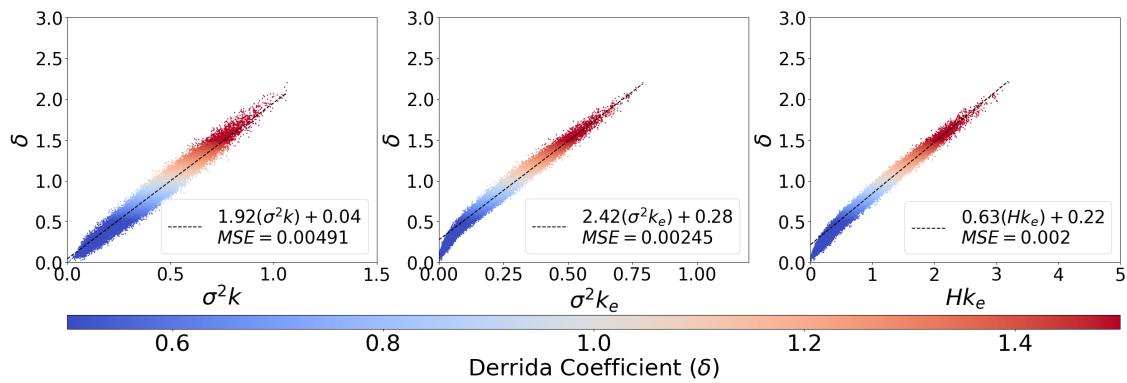
cbar=fig.colorbar(sc,ax=ax,location='bottom',aspect=75)
cbar.ax.tick_params(axis='both',labelsize=fs)
cbar.set_label('Derrida Coefficient ($\delta$)', fontsize=fs)
plt.suptitle(f'Linear fits ({N=}), $1.5>\delta > 0.5$ only', fontsize=fs)
# plt.savefig(f'figures/LinearFits_RBN_{N}.pdf',bbox_inches='tight')
plt.savefig(f'figures/LinearFits_RBN_{N}.png',bbox_inches='tight')
plt.show()

```

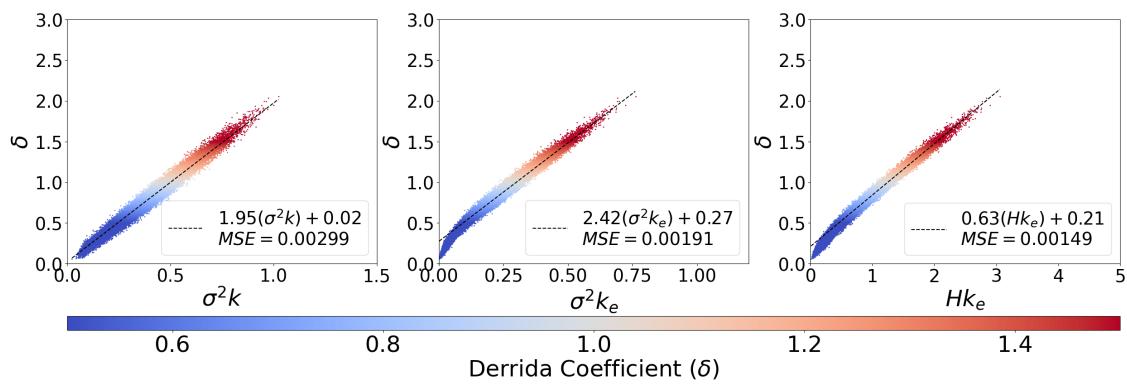
Linear fits ($N=20$), $1.5 > \delta > 0.5$ only

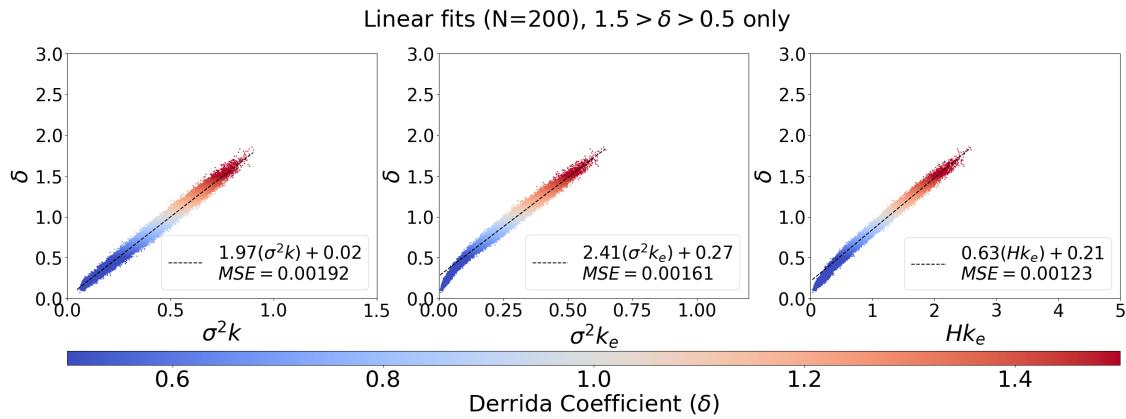


Linear fits ($N=50$), $1.5 > \delta > 0.5$ only



Linear fits ($N=100$), $1.5 > \delta > 0.5$ only





Again, there is a small but consistent improvement in mean squared error when moving from the left to the right.

We show the power law fits in the connectivity/bias-spread planes.

```
[ ]: fs = 36
fsa = 28
for N in Nvals:
    dfN = dfall[dfall['N']==N]

    fig, ax = plt.subplots(1,3,figsize=(30, 10),facecolor='white')
    cmap = plt.cm.get_cmap('coolwarm').reversed()
    x=dfN['avgK']*dfN['avgV']
    y=dfN['Derrida']

    popt, pcov = curve_fit(powerlaw,x[np.isfinite(x)],dfN['Derrida'][np.isfinite(x)])
    c, m, b = popt
    xfit=np.arange(0.005,0.255,0.005)
    yfit=critical_kappa(xfit,c,m,b)
    label = 'estimated critical boundary'
    ax[0].plot(xfit,yfit,'k--', label=label)
    sc=ax[0].scatter(dfN['avgV'], dfN['avgK'], s=1,
                      c=dfN['Derrida'], cmap=cmap,
                      norm=CenteredNorm(vcenter=1,halfrange=0.5))
    ax[0].set_xlabel("$\sigma^2$ k",fontsize=fs)
    ax[0].set_ylabel("$k$",fontsize=fs)
    ax[0].tick_params(axis='both', which='major', labelsize=fsa)
    #ax[0].legend(fontsize=fs, loc = 'lower right')
    ax[0].set_xlim(0,0.25)
    ax[0].set_ylim(1,5)

    x=dfN['avgKe']*dfN['avgV']
```

```

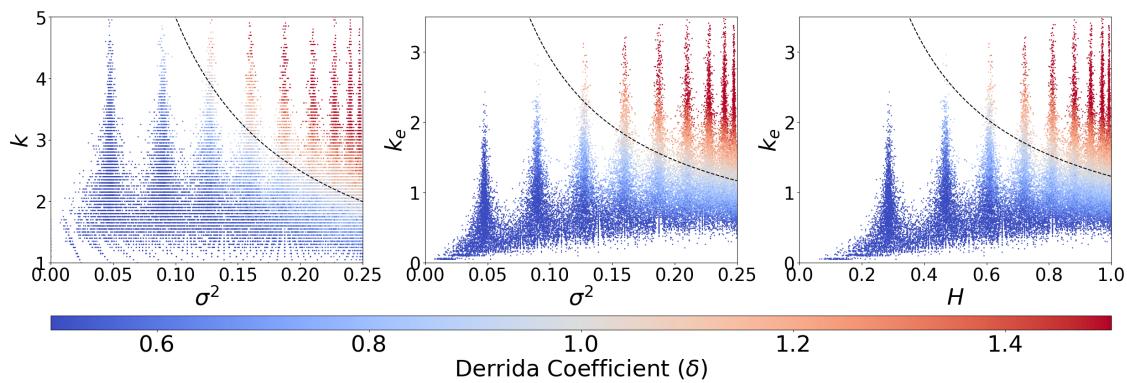
popt, pcov = curve_fit(powerlaw,x[np.isfinite(x)],dfN['Derrida'][np.
˓→isfinite(x)])
c, m, b = popt
xfit=np.arange(0.01,0.255,0.005)
yfit=critical_kappa(xfit,c,m,b)
label = 'estimated critical boundary'
ax[1].plot(xfit,yfit,'k--', label=label)
sc=ax[1].scatter(dfN['avgV'], dfN['avgKe'], s=1,
                  c=dfN['Derrida'], cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5))
ax[1].set_xlabel("$\sigma^2$", fontsize=fs)
ax[1].set_ylabel("k_e", fontsize=fs)
ax[1].tick_params(axis='both', which='major', labelsize=fs)
#ax[1].legend(fontsize=fs, loc = 'lower right')
ax[1].set_ylim(0,3.5)
ax[1].set_xlim(0,0.25)

x=dfN['avgKe']*dfN['avgH']
popt, pcov = curve_fit(powerlaw,x[np.isfinite(x)],dfN['Derrida'][np.
˓→isfinite(x)])
c, m, b = popt
xfit=np.arange(x.min(),x.max(),0.01)
xfit=np.arange(0.01,1.01,0.01)
yfit=critical_kappa(xfit,c,m,b)
label = 'estimated critical boundary'
ax[2].plot(xfit,yfit,'k--',label=label)
sc=ax[2].scatter(dfN['avgH'], dfN['avgKe'], s=1,
                  c=dfN['Derrida'], cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5))
ax[2].set_xlabel("H", fontsize=fs)
ax[2].set_ylabel("k_e", fontsize=fs)
ax[2].tick_params(axis='both', which='major', labelsize=fs)
#ax[2].legend(fontsize=fs, loc = 'lower right')
ax[2].set_ylim(0,3.5)
ax[2].set_xlim(0,1)

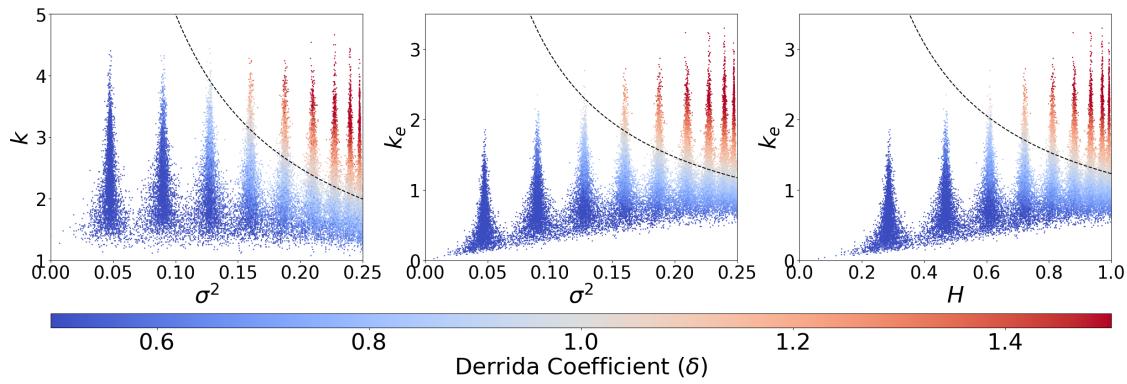
cbar=fig.colorbar(sc,ax=ax,location='bottom',aspect=75)
cbar.ax.tick_params(axis='both',labelsize=fs)
cbar.set_label('Derrida Coefficient ($\delta$)',fontsize=fs)
plt.suptitle(f'Power law fits ({N})',fontsize=fs)
# plt.savefig(f'figures/PowerLawFitsAlt_RBN_{N}.pdf',bbox_inches='tight')
plt.savefig(f'figures/PowerLawFitsAlt_RBN_{N}.png',bbox_inches='tight')
plt.show()

```

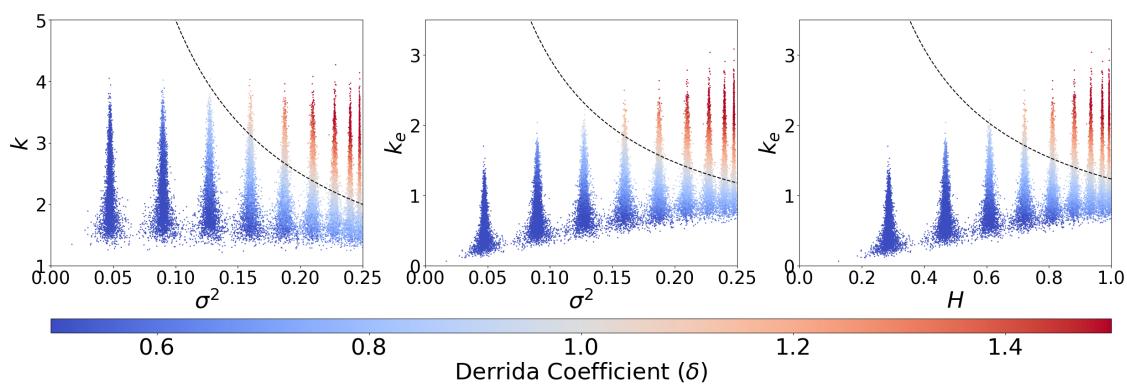
Power law fits ($N=20$)

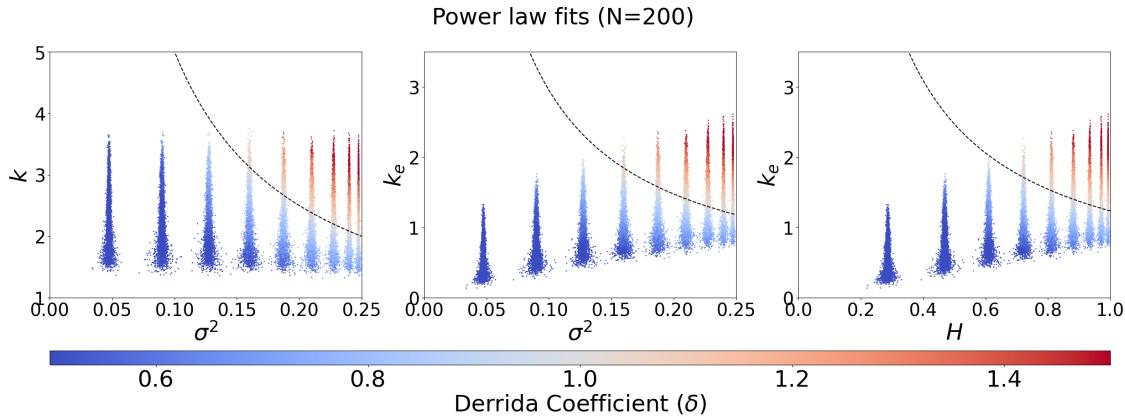


Power law fits ($N=50$)



Power law fits ($N=100$)





Next, we examine confusion matrices describing the ability of these critical boundaries to discriminate between dynamical regimes

```
[ ]: fs = 24
for N in Nvals:
    dfN = dfall[dfall['N']==N]

    fig, ax = plt.subplots(2, 2, figsize=(10, 10), sharey='row', sharex='col', facecolor='white')
    cmap = plt.cm.get_cmap('coolwarm').reversed()

    #x=dfN['avgS']
    x=dfN['Derrida']
    y=dfN['Derrida']
    popt, pcov = curve_fit(powerlaw,x[np.isfinite(x)],dfN['Derrida'][np.isfinite(x)])
    c, m, b = popt
    yfit=powerlaw(x,1,1,0)
    truth = (y>1).astype(int) - (y<1).astype(int)
    preds = (yfit>1).astype(int)-(yfit<1).astype(int)
    confusion_matrix = metrics.confusion_matrix(truth,preds)
    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=['Ordered', 'Critical', 'Chaotic'])
    cm_display.plot(ax=ax[0,0])
    cm_display.im_.colorbar.remove()
    ax[0,0].set_title('Sensitivity')

    x=dfN['avgK']*dfN['avgV']
    y=dfN['Derrida']
    popt, pcov = curve_fit(powerlaw,x[np.isfinite(x)],dfN['Derrida'][np.isfinite(x)])
    c, m, b = popt
    yfit=powerlaw(x,c,m,b)
```

```

truth = (y>1).astype(int) - (y<1).astype(int)
preds = (yfit>1).astype(int)-(yfit<1).astype(int)
confusion_matrix = metrics.confusion_matrix(truth,preds)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = ['Ordered', 'Critical', 'Chaotic'])
cm_display.plot(ax=ax[0,1])
cm_display.im_.colorbar.remove()
mstr='{' + str(np.round(m,2)) + '}'
label = f'${np.round(c,2)}(\sigma^2 k)^{mstr}+{np.round(b,2)}$'
ax[0,1].set_title(label)

x=dfN['avgKe']*dfN['avgV']
y=dfN['Derrida']
popt, pcov = curve_fit(powerlaw,x[np.isfinite(x)],dfN['Derrida'][np.isfinite(x)])
c, m, b = popt
yfit=powerlaw(x,c,m,b)
truth = (y>1).astype(int) - (y<1).astype(int)
preds = (yfit>1).astype(int)-(yfit<1).astype(int)
confusion_matrix = metrics.confusion_matrix(truth,preds)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = ['Ordered', 'Critical', 'Chaotic'])
cm_display.plot(ax=ax[1,0])
cm_display.im_.colorbar.remove()
mstr='{' + str(np.round(m,2)) + '}'
label = f'${np.round(c,2)}(\sigma^2 k_e)^{mstr}+{np.round(b,2)}$'
ax[1,0].set_title(label)

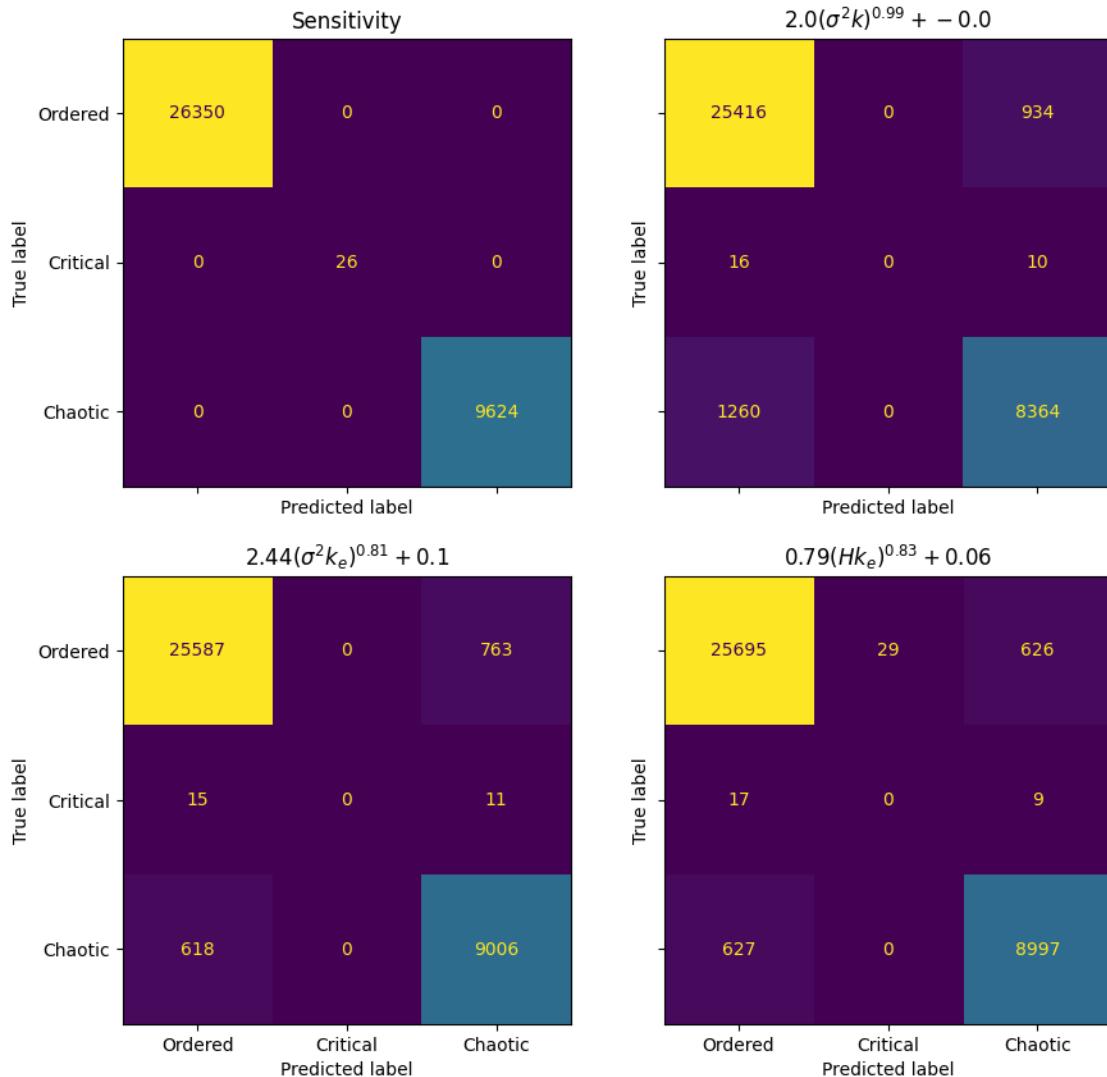
x=dfN['avgKe']*dfN['avgH']
y=dfN['Derrida']
popt, pcov = curve_fit(powerlaw,x[np.isfinite(x)],dfN['Derrida'][np.isfinite(x)])
c, m, b = popt
yfit=powerlaw(x,c,m,b)
truth = (y>1).astype(int) - (y<1).astype(int)
preds = (yfit>1).astype(int)-(yfit<1).astype(int)
confusion_matrix = metrics.confusion_matrix(truth,preds)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = ['Ordered', 'Critical', 'Chaotic'])
cm_display.plot(ax=ax[1,1])
cm_display.im_.colorbar.remove()
mstr='{' + str(np.round(m,2)) + '}'
label = f'${np.round(c,2)}(H k_e)^{mstr}+{np.round(b,2)}$'
ax[1,1].set_title(label)

fig.suptitle(f'Criticality predictions ({N=})', fontsize=fs)
plt.savefig(f'figures/PowerLawConfusion_RBN_{N}.pdf', bbox_inches='tight')

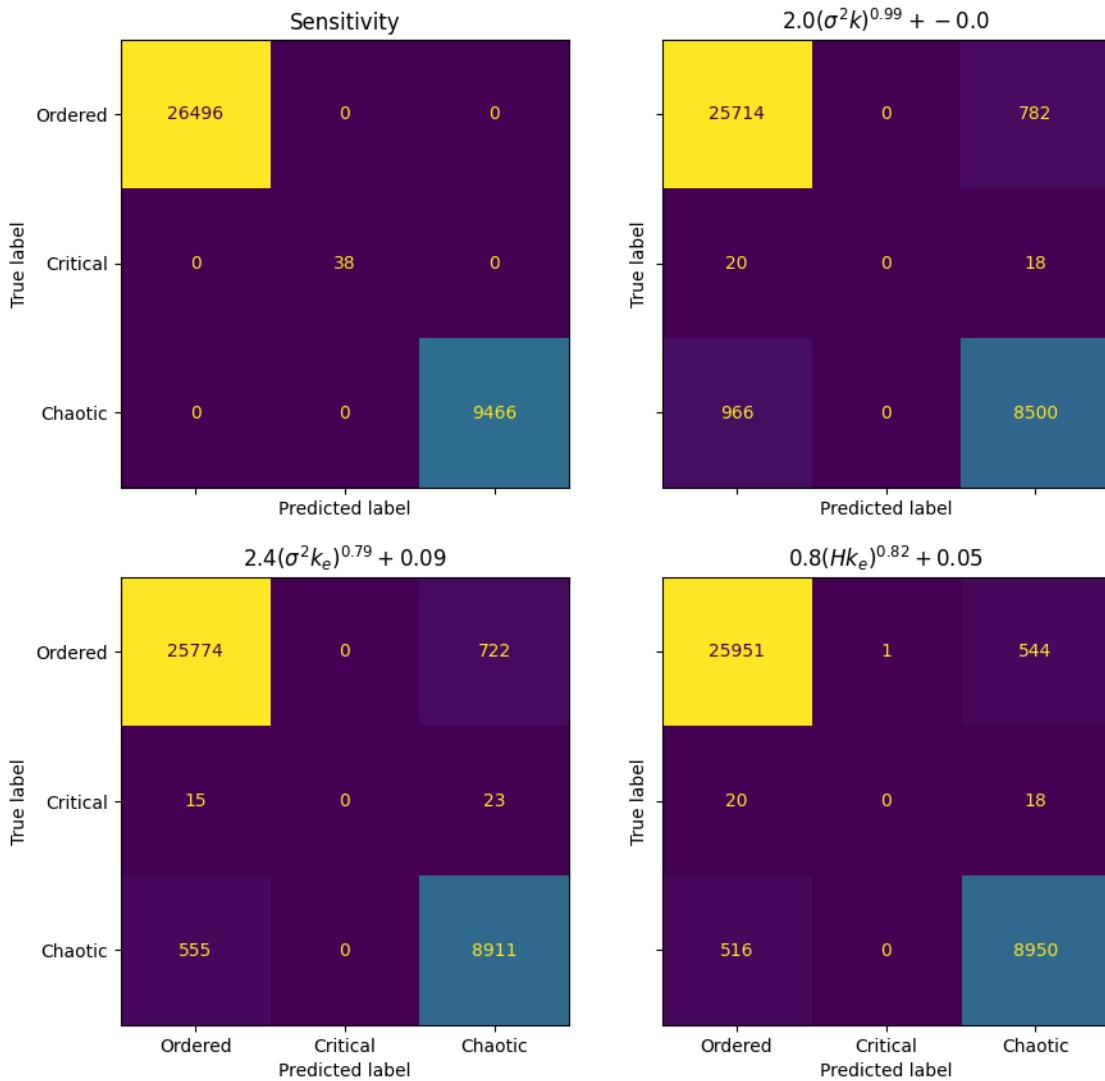
```

```
plt.savefig(f'figures/PowerLawConfusion_RBN_{N}.png',bbox_inches='tight')
plt.show()
```

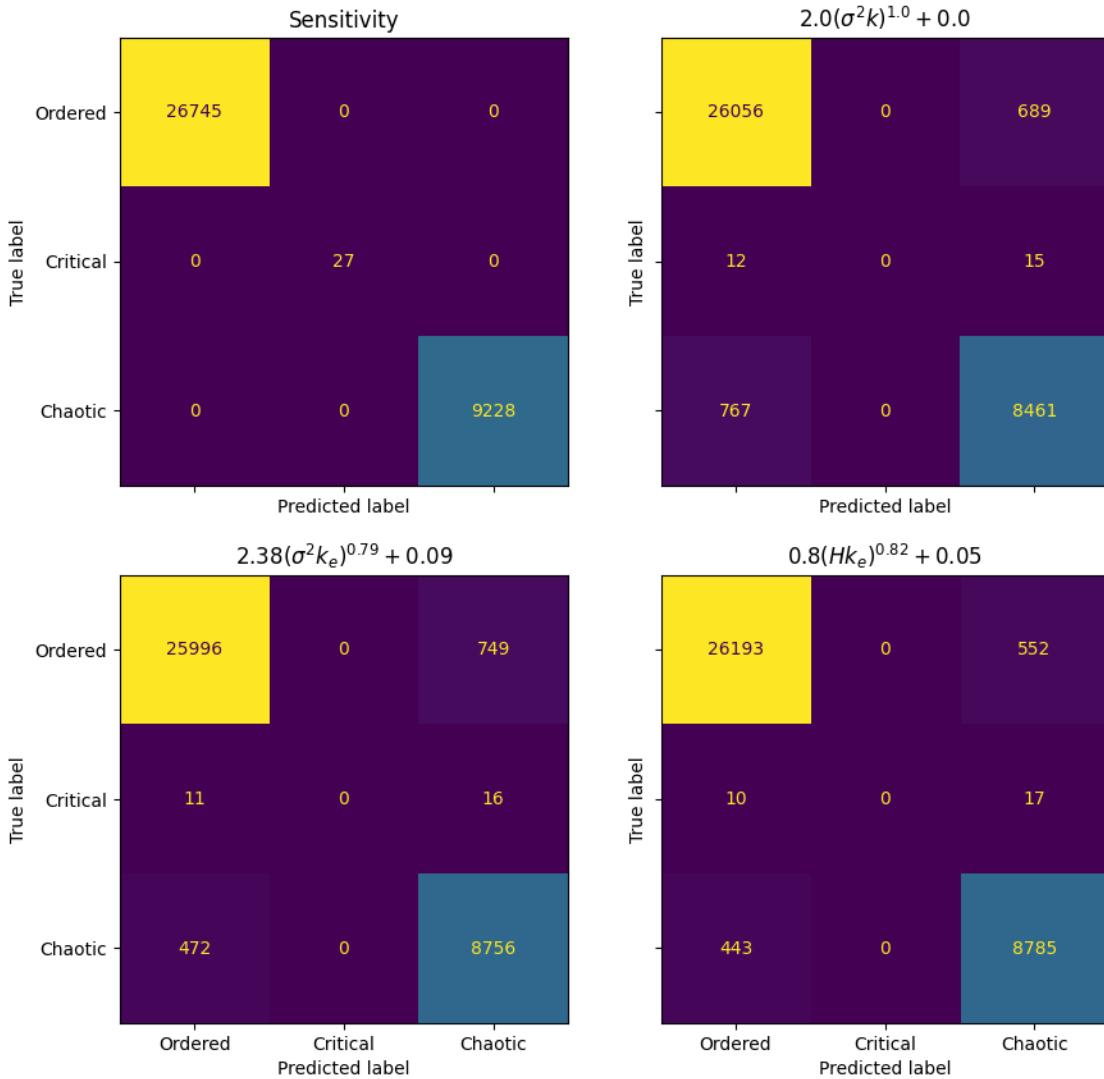
Criticality predictions (N=20)



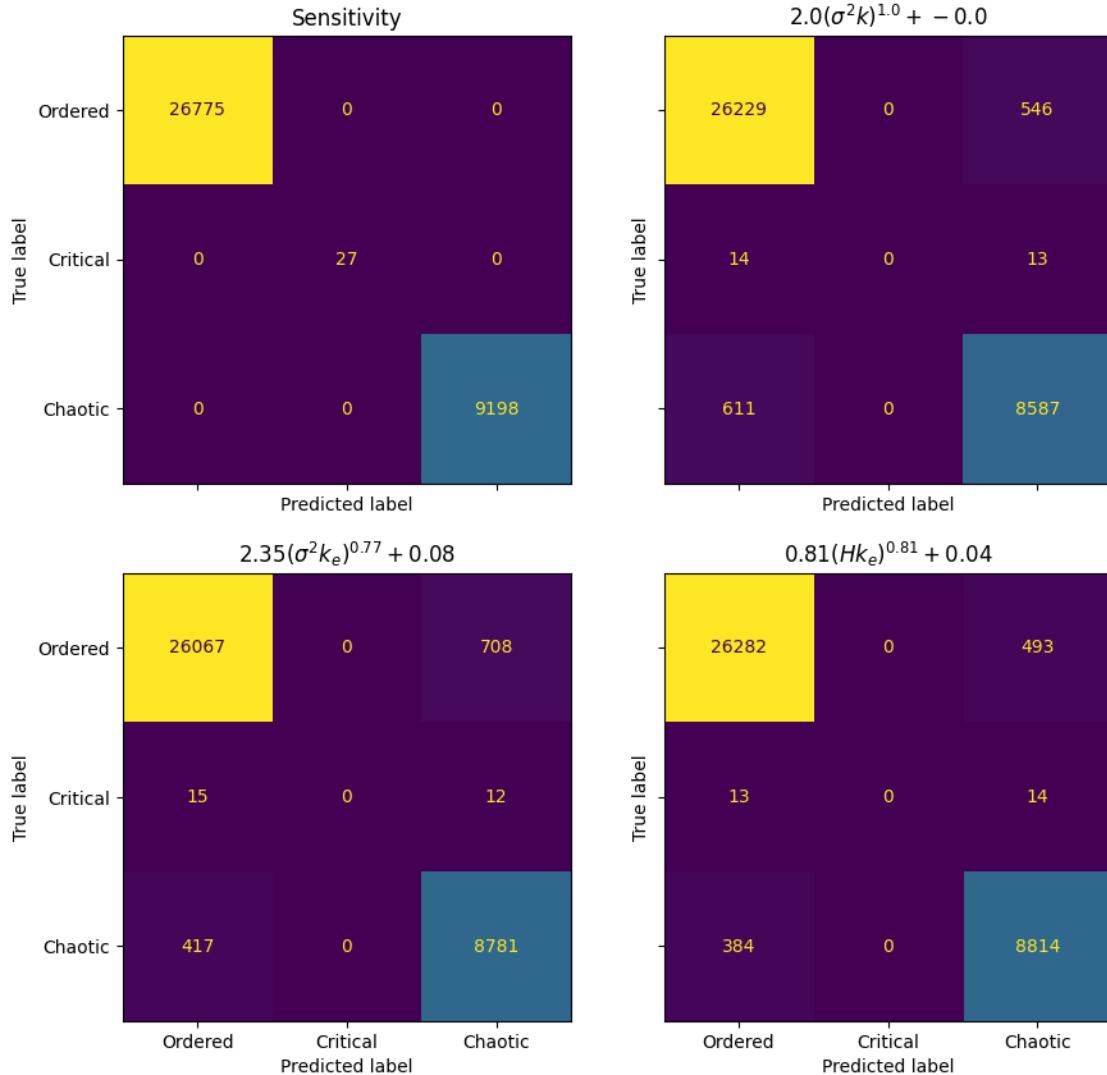
Criticality predictions (N=50)



Criticality predictions (N=100)



Criticality predictions (N=200)



1.2 We now begin analysis of the Cell Collective models.

We begin by importing the data and defining the bias entropy and bias variance.

```
[ ]: dfcc = pd.read_csv('cc_network_data.csv')
dfcc['avgV_est']=dfcc['biasUnweighted']*(1-dfcc['biasUnweighted'])
dfcc['avgH_est']=-dfcc['biasUnweighted']*np.
    ↪log2(dfcc['biasUnweighted'])-(1-dfcc['biasUnweighted'])*np.
    ↪log2(1-dfcc['biasUnweighted'])
```

We will need a helper function for optimizing the critical boundary.

```
[ ]: def optimize_cut(x,truth,method='Cohen kappa'):
    best_score = 0
    best_cut = 0
    for cut in sorted(x):
        cpreds = (x > cut)
        tp = np.sum(cpreds & truth)
        fp = np.sum(cpreds & ~truth)
        tn = np.sum(~cpreds & ~truth)
        fn = np.sum(~cpreds & truth)

        if method == 'MCC':
            denom = np.sqrt((tn+fn)*(fp+tp)*(tn+fp)*(fn+tp))
            if denom == 0:
                continue
            score = (tn*tp-fp*fn)/denom

        elif method == 'Accuracy':
            score = (tp+tn)/(tp+tn+fn+fp)

        elif method == 'Cohen kappa':
            denom = (tp+fp)*(fp+tn)+(tp+fn)*(fn+tn)
            if denom == 0:
                continue
            score = 2*(tp*tn-fn*fp)/denom
        else:
            raise ValueError
        if score > best_score:
            best_score = score
            best_cut = cut
            preds = np.copy(cpreds)
    return preds, best_cut, best_score
```

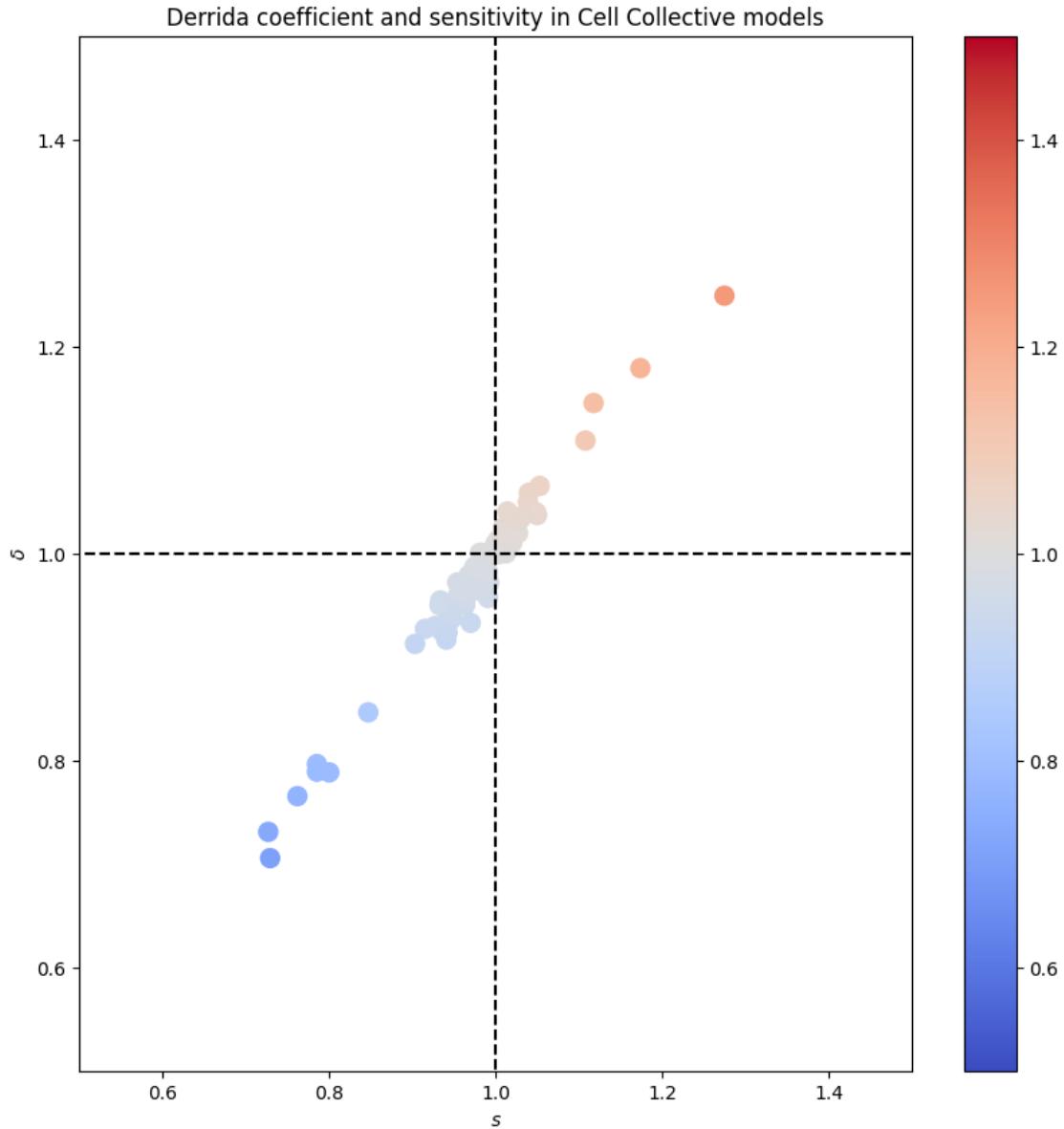
We examine some summary statistics of the Cell Collective networks.

```
[ ]: print('average number of nodes:', dfcc['Nnodes'].mean())
print('Derrida coefficient quartiles:', np.percentile(dfcc['dc'], [0, 25, 50, 75, 100]))
print('Derrida coefficient mean and standard deviation:', dfcc['dc'].mean(), dfcc['dc'].std())
```

```
average number of nodes: 46.78378378378378
Derrida coefficient quartiles: [0.7055  0.9470625 0.9793125 1.0119375 1.249]
]
Derrida coefficient mean and standard deviation: 0.9736300675675675 ,
0.08681282869300079
```

We highlight that sensitivity and the Derrida coefficient are in good agreement in these models.

```
[ ]: fig, ax = plt.subplots(figsize=(10, 10), facecolor='white')
ax.set_facecolor('white')
varx = 's'
vary = 'dc'
cmap = plt.cm.get_cmap('coolwarm')#.reversed()
sc=ax.scatter(dfcc[varx],dfcc[vary],s=100,c=dfcc['dc'],cmap=cmap,
              norm=CenteredNorm(vcenter=1,halfrange=0.5),alpha=1.0)
ax.vlines(1.0,0,2,label='$s=1$',color='k',linestyles='--')
ax.hlines(1.0,0,2,label='$\delta=1$',color='k',linestyles='--')
ax.set_xlim(0.5,1.5)
ax.set_ylim(0.5,1.5)
plt.colorbar(sc)
plt.xlabel('$s$')
plt.ylabel('$\delta$')
plt.title('Derrida coefficient and sensitivity in Cell Collective models')
#plt.legend()
plt.savefig(f'figures/DerridaSensitivity_CC.pdf',bbox_inches='tight')
plt.savefig(f'figures/DerridaSensitivity_CC.png',bbox_inches='tight')
plt.show()
```



We show the dependence of δ and s on the connectivity-spread products considered earlier. We highlight a critical region between dotted lines. This region is centered on $\delta = 1$ and has width equal to the width of the IQR of the δ distribution.

```
[ ]: fs = 24
fig, ax = plt.subplots(1,3,figsize=(30, 10),facecolor='white',sharey='row')
cmap = plt.cm.get_cmap('coolwarm')

dd=np.subtract(*np.percentile(dfcc['dc'], [75, 25]))
```

```

sc=ax[0].
    ↪scatter(dfcc['avgV_est']*dfcc['k'],dfcc['dc'],s=100,c=dfcc['s'],cmap=cmap,
            norm=CenteredNorm(vcenter=1,halfrange=0.5),alpha=1.0)
ax[0].set_ylim(0.5,1.5)
ax[0].set_xlim(0,1.5)
#ax[0].legend(loc='upper left',fontsize=fs)
ax[0].set_xlabel('$\sigma^2 k$',fontsize=fs)
ax[0].set_ylabel('$\delta$',fontsize=fs)
ax[0].tick_params(axis='both', which='major', labelsize=fs)
ax[0].hlines([1-dd,1+dd],xmin=0,xmax=1.5,colors='k',linestyles='--')

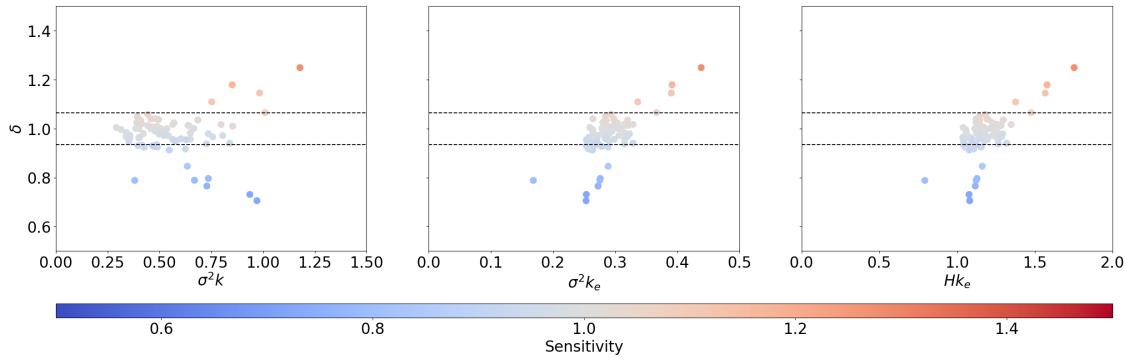
sc=ax[1].
    ↪scatter(dfcc['avgV_est']*dfcc['ke'],dfcc['dc'],s=100,c=dfcc['s'],cmap=cmap,
            norm=CenteredNorm(vcenter=1,halfrange=0.5),alpha=1.0)
ax[1].set_ylim(0.5,1.5)
ax[1].set_xlim(0,0.5)
#ax[1].legend(loc='upper left',fontsize=fs)
ax[1].set_xlabel('$\sigma^2 k_e$',fontsize=fs)
#ax[1].set_ylabel('$\delta$',fontsize=fs)
ax[1].tick_params(axis='both', which='major', labelsize=fs)
ax[1].hlines([1-dd,1+dd],xmin=0,xmax=0.5,colors='k',linestyles='--')

sc=ax[2].
    ↪scatter(dfcc['avgH_est']*dfcc['ke'],dfcc['dc'],s=100,c=dfcc['s'],cmap=cmap,
            norm=CenteredNorm(vcenter=1,halfrange=0.5),alpha=1.0)
ax[2].set_ylim(0.5,1.5)
ax[2].set_xlim(0,2)
#ax[2].legend(loc='upper left',fontsize=fs)
ax[2].set_xlabel('$H k_e$',fontsize=fs)
#ax[2].set_ylabel('$\delta$',fontsize=fs)
ax[2].tick_params(axis='both', which='major', labelsize=fs)
ax[2].hlines([1-dd,1+dd],xmin=0,xmax=2,colors='k',linestyles='--')

cbar=fig.colorbar(sc,ax=ax,location='bottom',aspect=75)
cbar.ax.tick_params(axis='both',labelsize=fs)
cbar.set_label('Sensitivity',fontsize=fs)

plt.savefig(f'figures/DerridaConnectivitySpread_CC_from_bias.
    ↪pdf',bbox_inches='tight')
plt.savefig(f'figures/DerridaConnectivitySpread_CC_from_bias.
    ↪png',bbox_inches='tight')
plt.show()

```



Next, we attempt to find the optimal critical boundary according to various measures.

```
[ ]: fs = 24
fig, ax = plt.subplots(3,3, figsize=(10, 10), sharey='row', sharex='col', facecolor='white')
truth = (dfcc['dc'] > 1)
for row, method in enumerate(['MCC', 'Accuracy', 'Cohen kappa']):

    x = dfcc['k']*dfcc['avgV_est']
    preds,cut,score=optimize_cut(x,truth,method=method)
    confusion_matrix = metrics.confusion_matrix(truth, preds)
    cm_display = metrics.ConfusionMatrixDisplay(
        confusion_matrix=confusion_matrix, display_labels=['$\delta \leq 1$', '$\delta > 1$'])
    cm_display.plot(ax=ax[row,0])
    cm_display.im_.colorbar.remove()
    mstr = '{'+str(np.round(m, 2))+'}'
    label = f'$\sigma^2 k={np.round(cut,3)}$, {method}={np.round(score,2)}'
    ax[row,0].set_title(label)

    x = dfcc['ke']*dfcc['avgV_est']
    preds,cut,score=optimize_cut(x,truth,method=method)
    confusion_matrix = metrics.confusion_matrix(truth, preds)
    cm_display = metrics.ConfusionMatrixDisplay(
        confusion_matrix=confusion_matrix, display_labels=['$\delta \leq 1$', '$\delta > 1$'])
    cm_display.plot(ax=ax[row,1])
    cm_display.im_.colorbar.remove()
    mstr = '{'+str(np.round(m, 2))+'}'
    label = f'$\sigma^2 k_e={np.round(cut,3)}$, {method}={np.round(score,2)}'
    ax[row,1].set_title(label)

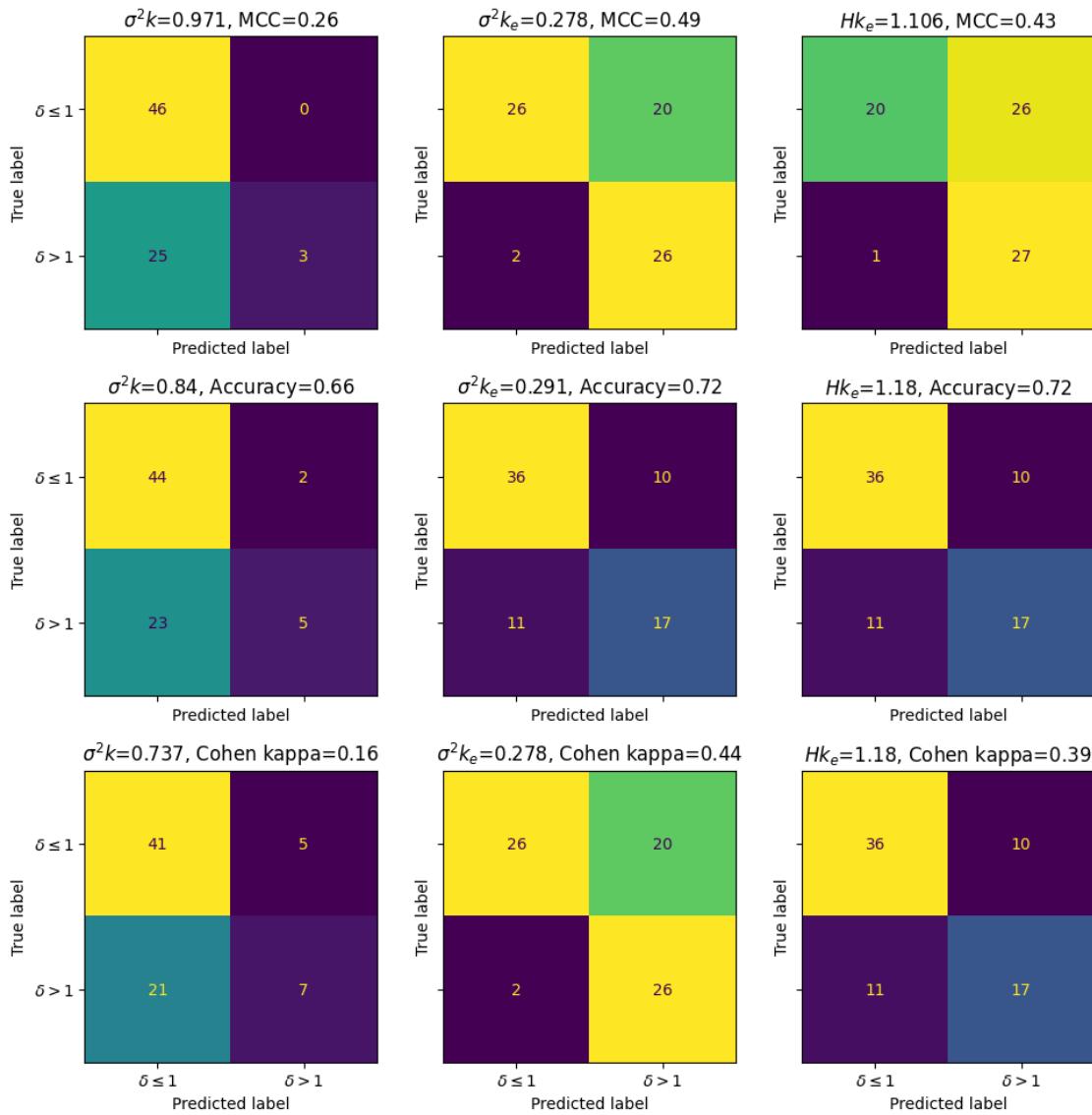
    x = dfcc['ke']*dfcc['avgH_est']
    preds,cut,score=optimize_cut(x,truth,method=method)
```

```

confusion_matrix = metrics.confusion_matrix(truth, preds)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=['$\delta \leq 1$', '$\delta > 1$'])
cm_display.plot(ax=ax[row,2])
cm_display.im_.colorbar.remove()
mstr = '{'+str(np.round(m, 2))+'}'
label = f'$H_{k_e}={np.round(cut,3)}$, {method}={np.round(score,2)}'
ax[row,2].set_title(label)

fig.tight_layout()
plt.savefig(f'figures/ConfusionOptimized_CC_from_bias.png',bbox_inches='tight')
plt.savefig(f'figures/ConfusionOptimized_CC_from_bias.pdf',bbox_inches='tight')
plt.show()

```



We will record the values that give rise to the greatest accuracy.

```
[ ]: acut_KV_fb = 0.84
      acut_KeV_fb = 0.291
      acut_KeH_fb = 1.18
```

We now compare these optimal boundaries to the ones obtained from the RBN ensemble.

```
[ ]: fig, ax = plt.subplots(1,3,figsize=(30, 10),facecolor='white')
bounds = ([0,0,-10],[10,10,10]) #c,m,b
fs = 24
cmap = plt.cm.get_cmap('coolwarm')
```

```

varx = 'avgV_est'
vary = 'k'

sc=ax[0].scatter(dfcc[varx],dfcc[vary],s=100,c=dfcc['dc'],cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5),alpha=1.0)
px = np.arange(0.01,1,0.01)
py=acut_KV_fb/px
py2=critical_kappa(px,2,1,0)
ax[0].plot(px,py2,'--k',label='Fit from RBNs (N=50)')
ax[0].plot(px,py,'-k',label='Most accurate for CC')
ax[0].set_ylim(1,7)
ax[0].set_xlim(0.125,0.25)
ax[0].legend(loc='upper left',fontsize=fs)
ax[0].set_xlabel('$\sigma^2$',fontsize=fs)
ax[0].set_ylabel('$k$',fontsize=fs)
ax[0].tick_params(axis='both', which='major', labelsize=fs)
ax[0].set_title('$\sigma^2$ vs $\delta$ critical boundary',fontsize=fs)

varx = 'avgV_est'
vary = 'ke'

sc=ax[1].scatter(dfcc[varx],dfcc[vary],s=100,c=dfcc['dc'],cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5),alpha=1.0)
px = np.arange(0.01,1,0.01)
py=acut_KeV_fb/px
py2=critical_kappa(px,2.4,0.8,0.09)
ax[1].plot(px,py2,'--k',label='Fit from RBNs (N=50)')
ax[1].plot(px,py,'-k',label='Most accurate for CC')
ax[1].set_ylim(1,2)
ax[1].set_xlim(0.125,0.25)
ax[1].legend(loc='upper left',fontsize=fs)
ax[1].set_xlabel('$\sigma^2$',fontsize=fs)
ax[1].set_ylabel('$k_e$',fontsize=fs)
ax[1].tick_params(axis='both', which='major', labelsize=fs)
ax[1].set_title('$\sigma^2$ vs $\delta$ critical boundary',fontsize=fs)

varx = 'avgH_est'
vary = 'ke'

sc=ax[2].scatter(dfcc[varx],dfcc[vary],s=100,c=dfcc['dc'],cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5),alpha=1.0,label=None)
px = np.arange(0.01,1.01,0.01)
py=acut_KeH_fb/px
py2=critical_kappa(px,0.8,0.83,0.05)
ax[2].plot(px,py2,'--k',label='Fit from RBNs (N=50)')
ax[2].plot(px,py,'-k',label='Most accurate for CC')

```

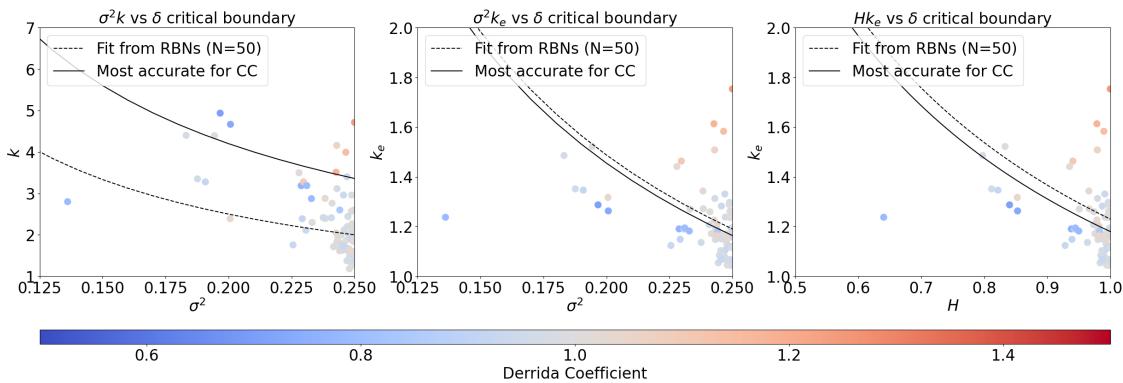
```

ax[2].set_ylim(1,2)
ax[2].set_xlim(0.5,1)
ax[2].legend(loc='upper left', fontsize=fs)
ax[2].set_xlabel('$H$', fontsize=fs)
ax[2].set_ylabel('$k_e$', fontsize=fs)
ax[2].tick_params(axis='both', which='major', labelsize=fs)
ax[2].set_title('$H k_e$ vs $\delta$ critical boundary', fontsize=fs)

cbar=fig.colorbar(sc, ax=ax, location='bottom', aspect=75)
cbar.ax.tick_params(axis='both', labelsize=fs)
cbar.set_label('Derrida Coefficient', fontsize=fs)

plt.savefig(f'figures/PowerLawFits_CC_from_bias.pdf', bbox_inches='tight')
plt.savefig(f'figures/PowerLawFits_CC_from_bias.png', bbox_inches='tight')
plt.show()

```



We now consider an alternate averaging scheme for the Cell Collective models. The theory of RBNs considers σ^2 as computed from the overall bias of the network, rather than computed from the average of each node's output variance. This traditional approach gives rise to the classical results in the theory. However, the second, less-conventional approach we propose here performs better in the Cell Collective (especially when paired with effective connectivity), as we will demonstrate. We use the prime to denote that the parameters are computed using this alternate averaging scheme.

```

[ ]: fs = 36
fsa = 24
fig, ax = plt.subplots(1,3, figsize=(30, 10), facecolor='white', sharey='row')
cmap = plt.cm.get_cmap('coolwarm')

dd=np.subtract(*np.percentile(dfcc['dc'], [75, 25]))
sc=ax[0].scatter(dfcc['avgV']*dfcc['k'], dfcc['dc'], s=100, c=dfcc['s'], cmap=cmap,
                  norm=CenteredNorm(vcenter=1, halfrange=0.5), alpha=1.0)
ax[0].set_ylimit(0.5,1.5)
ax[0].set_xlim(0,1.5)

```

```

ax[0].set_xlabel('$\sigma^2 \prime k$', fontsize=fs)
ax[0].set_ylabel('$\delta$', fontsize=fs)
ax[0].tick_params(axis='both', which='major', labelsize=fsa)
ax[0].hlines([1-dd, 1+dd], xmin=0, xmax=1.5, colors='k', linestyles='--')

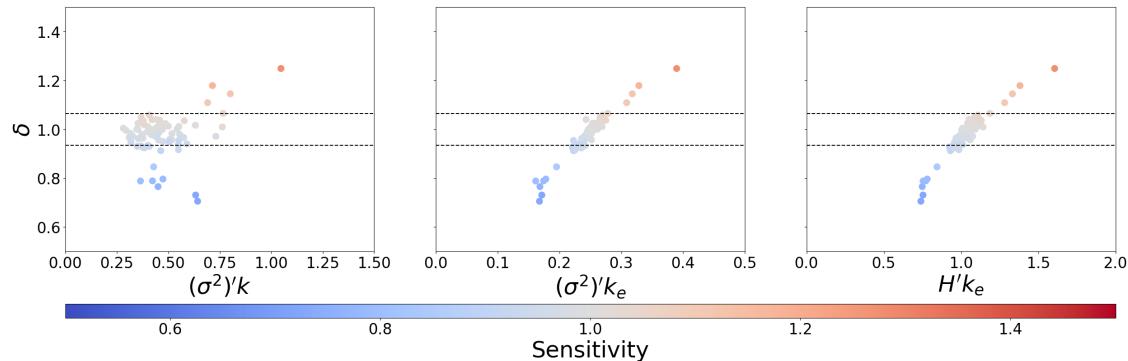
sc=ax[1].scatter(dfcc['avgV']*dfcc['ke'], dfcc['dc'], s=100, c=dfcc['s'], cmap=cmap,
                  norm=CenteredNorm(vcenter=1, halfrange=0.5), alpha=1.0)
ax[1].set_ylim(0.5, 1.5)
ax[1].set_xlim(0, 0.5)
ax[1].set_xlabel('$\sigma^2 \prime k_e$', fontsize=fs)
ax[1].tick_params(axis='both', which='major', labelsize=fsa)
ax[1].hlines([1-dd, 1+dd], xmin=0, xmax=0.5, colors='k', linestyles='--')

sc=ax[2].scatter(dfcc['avgH']*dfcc['ke'], dfcc['dc'], s=100, c=dfcc['s'], cmap=cmap,
                  norm=CenteredNorm(vcenter=1, halfrange=0.5), alpha=1.0)
ax[2].set_ylim(0.5, 1.5)
ax[2].set_xlim(0, 2)
ax[2].set_xlabel('$H \prime k_e$', fontsize=fs)
ax[2].tick_params(axis='both', which='major', labelsize=fsa)
ax[2].hlines([1-dd, 1+dd], xmin=0, xmax=2, colors='k', linestyles='--')

cbar=fig.colorbar(sc, ax=ax, location='bottom', aspect=75)
cbar.ax.tick_params(axis='both', labelsize=fsa)
cbar.set_label('Sensitivity', fontsize=fs)

plt.savefig(f'figures/DerridaConnectivitySpread_CC.pdf', bbox_inches='tight')
plt.savefig(f'figures/DerridaConnectivitySpread_CC.png', bbox_inches='tight')
plt.show()

```



The first indication that the alternate averaging scheme is more appropriate in this setting is the qualitatively improved correlation relative to the previously explored averaging scheme.

We now optimize the critical boundaries using the parameters above.

```
[ ]: fs = 24
import matplotlib as mpl
mpl.rcParams["font.size"] = 20
tfs = 12
fig, ax = plt.subplots(3,3, figsize=(10, 10), sharey='row', sharex='col', facecolor='white')
truth = (dfcc['dc'] > 1)
for row, method in enumerate(['MCC', 'Accuracy', 'Cohen kappa']):

    x = dfcc['k']*dfcc['avgV']
    preds,cut,score=optimize_cut(x,truth,method=method)
    confusion_matrix = metrics.confusion_matrix(truth, preds)
    cm_display = metrics.ConfusionMatrixDisplay(
        confusion_matrix=confusion_matrix, display_labels=['$\delta \leq 1$', '$\delta > 1$'])
    cm_display.plot(ax=ax[row,0])
    cm_display.im_.colorbar.remove()
    mstr = '{'+str(np.round(m, 2))+'}'
    label = f'${(\sigma^2)^\prime k}={np.round(cut,3)}$, {method}={np.round(score,2)}'
    ax[row,0].set_title(label, fontsize=tfs)

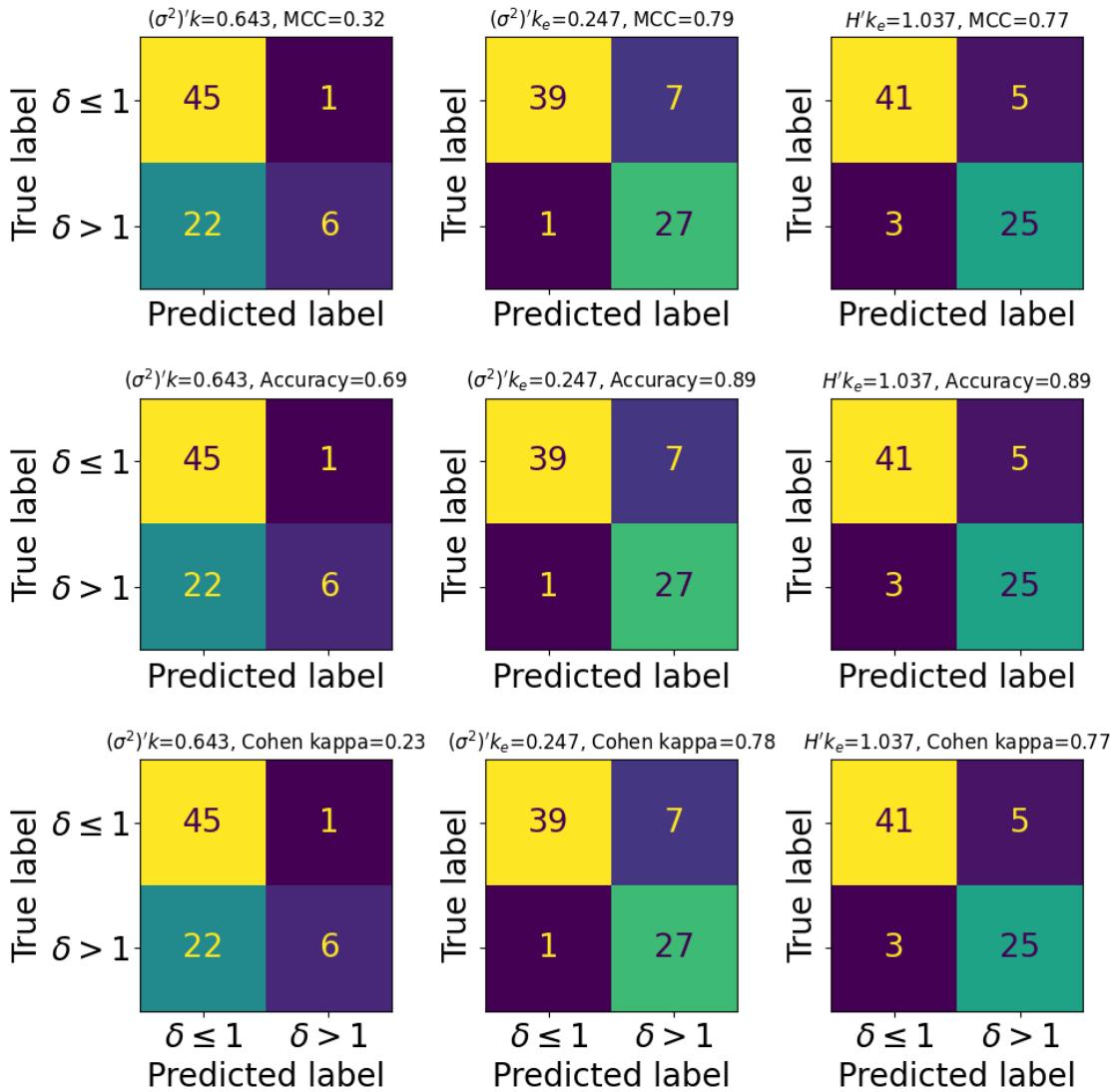
    x = dfcc['ke']*dfcc['avgV']
    preds,cut,score=optimize_cut(x,truth,method=method)
    confusion_matrix = metrics.confusion_matrix(truth, preds)
    cm_display = metrics.ConfusionMatrixDisplay(
        confusion_matrix=confusion_matrix, display_labels=['$\delta \leq 1$', '$\delta > 1$'])
    cm_display.plot(ax=ax[row,1])
    cm_display.im_.colorbar.remove()
    mstr = '{'+str(np.round(m, 2))+'}'
    label = f'${(\sigma^2)^\prime k_e}={np.round(cut,3)}$, {method}={np.round(score,2)}'
    ax[row,1].set_title(label, fontsize=tfs)

    x = dfcc['ke']*dfcc['avgH']
    preds,cut,score=optimize_cut(x,truth,method=method)
    confusion_matrix = metrics.confusion_matrix(truth, preds)
    cm_display = metrics.ConfusionMatrixDisplay(
        confusion_matrix=confusion_matrix, display_labels=['$\delta \leq 1$', '$\delta > 1$'])
    cm_display.plot(ax=ax[row,2])
    cm_display.im_.colorbar.remove()
    mstr = '{'+str(np.round(m, 2))+'}'
    label = f'${H^\prime k_e}={np.round(cut,3)}$, {method}={np.round(score,2)}'
    ax[row,2].set_title(label, fontsize=tfs)
```

```

fig.tight_layout()
plt.savefig(f'figures/ConfusionOptimized_CC_all.png',bbox_inches='tight')
plt.savefig(f'figures/ConfusionOptimized_CC_all.pdf',bbox_inches='tight')
plt.show()

```



We record the most accurate thresholds (which also optimize all other metrics considered, in this case).

```

[ ]: acut_KV = 0.643
      acut_KeV = 0.247
      acut_KeH = 1.037

```

Because these metrics are all optimized by the same boundary, we can condense the above figure

for more convenient presentation.

```
[ ]: fs = 14
mpl.rcParams["font.size"] = 20
fig, ax = plt.subplots(1,3, figsize=(10, 10), sharey='row', sharex='col', facecolor='white')
truth = (dfcc['dc'] > 1)

x = dfcc['k']*dfcc['avgV']
preds,cut,score=optimize_cut(x,truth,method='Accuracy')
confusion_matrix = metrics.confusion_matrix(truth, preds)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=[ f'${(\sigma^2)^\prime} \leq ${np.round(cut,2)}', f'${(\sigma^2)^\prime} k > ${np.round(cut,2)}'])#['$\delta \leq 1$', '$\delta > 1$'])
cm_display.plot(ax=ax[0])
cm_display.im_.colorbar.remove()
mstr = '{'+str(np.round(m, 2))+'}'
label = f'${(\sigma^2)^\prime} k = {np.round(cut,3)}'
#ax[0].set_title(label, fontsize=fs)

x = dfcc['ke']*dfcc['avgV']
preds,cut,score=optimize_cut(x,truth,method='Accuracy')
confusion_matrix = metrics.confusion_matrix(truth, preds)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=[ f'${(\sigma^2)^\prime} k_e \leq ${np.round(cut,2)}', f'${(\sigma^2)^\prime} k_e > ${np.round(cut,2)}'])#['$\delta \leq 1$', '$\delta > 1$'][['$\delta \leq 1$', '$\delta > 1$'],)
cm_display.plot(ax=ax[1])
cm_display.im_.colorbar.remove()
mstr = '{'+str(np.round(m, 2))+'}'
label = f'${(\sigma^2)^\prime} k_e = {np.round(cut,3)}'
#ax[1].set_title(label, fontsize=fs)

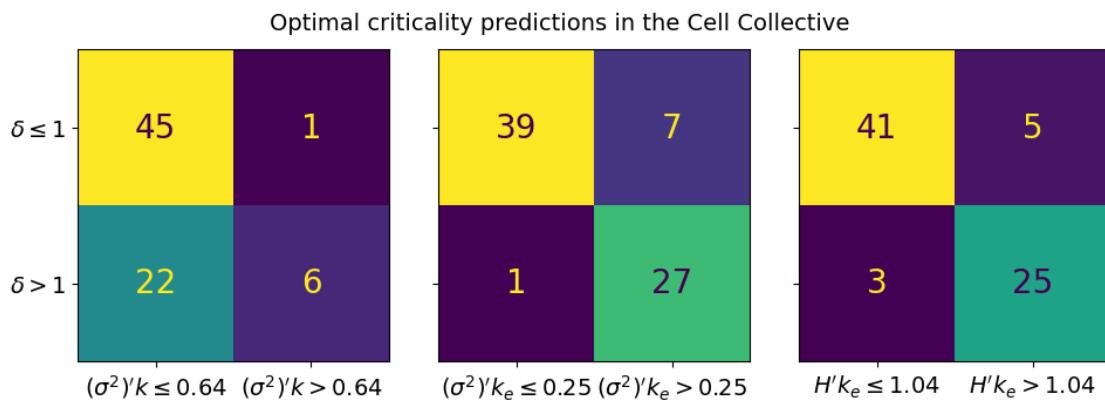
x = dfcc['ke']*dfcc['avgH']
preds,cut,score=optimize_cut(x,truth,method='Accuracy')
confusion_matrix = metrics.confusion_matrix(truth, preds)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=[ f'$H^\prime k_e \leq ${np.round(cut,2)}', f'$H^\prime k_e > ${np.round(cut,2)}'])#['$\delta \leq 1$', '$\delta > 1$'][['$\delta \leq 1$', '$\delta > 1$'],)
cm_display.plot(ax=ax[2])
cm_display.im_.colorbar.remove()
mstr = '{'+str(np.round(m, 2))+'}'
label = f'$H^\prime k_e = {np.round(cut,3)}'
#ax[2].set_title(label, fontsize=fs)
```

```

# fix font sizes
for axis in ax:
    axis.tick_params(axis='both', which='major', labelsize=fs)
    axis.set_xlabel('')
    axis.set_ylabel('')
#axis.xaxis.set_ticklabels(['Below', 'Above']);
axis.yaxis.set_ticklabels(['$\delta \leq 1$', '$\delta > 1$']);

fig.suptitle(f'Optimal criticality predictions in the Cell Collective', fontsize=fs)
fig.tight_layout()
fig.subplots_adjust(top=1.6)
plt.savefig(f'figures/ConfusionOptimized_CC.png', bbox_inches='tight')
plt.savefig(f'figures/ConfusionOptimized_CC.pdf', bbox_inches='tight')
plt.show()

```



We now plot the critical boundaries we have obtained via the optimization above.

```

[ ]: fig, ax = plt.subplots(1,3,figsize=(30, 10),facecolor='white')
bounds = ([0,0,-10],[10,10,10]) #c,m,b
fs = 36
fsa = 24
cmap = plt.cm.get_cmap('coolwarm')

varx = 'avgV'
vary = 'k'

sc=ax[0].scatter(dfcc[varx],dfcc[vary],s=100,c=dfcc['dc'],cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5),alpha=1.0)
px = np.arange(0.01,1,0.01)
py=acut_KV/px

```

```

ax[0].plot(px,py,'-k',label='Most accurate for CC')
ax[0].set_ylim(1,7)
ax[0].set_xlim(0.125,0.25)
#ax[0].legend(loc='upper left',fontsize=fs)
ax[0].set_xlabel('$(\sigma^2)^{\prime}$',fontsize=fs)
ax[0].set_ylabel('$k$',fontsize=fs)
ax[0].tick_params(axis='both', which='major', labelsize=fs)
ax[0].set_title('$(\sigma^2)^{\prime} k$ vs $\delta$ critical boundary',fontsize=fs)

varx = 'avgV'
vary = 'ke'

sc=ax[1].scatter(dfcc[varx],dfcc[vary],s=100,c=dfcc['dc'],cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5),alpha=1.0)
px = np.arange(0.01,1,0.01)
py=acut_KeV/px
ax[1].plot(px,py,'-k',label='Most accurate for CC')
ax[1].set_ylim(1,2)
ax[1].set_xlim(0.125,0.25)
#ax[1].legend(loc='upper left',fontsize=fs)
ax[1].set_xlabel('$(\sigma^2)^{\prime}$',fontsize=fs)
ax[1].set_ylabel('$k_e$',fontsize=fs)
ax[1].tick_params(axis='both', which='major', labelsize=fs)
ax[1].set_title('$(\sigma^2)^{\prime} k_e$ vs $\delta$ critical boundary',fontsize=fs)

varx = 'avgH'
vary = 'ke'

sc=ax[2].scatter(dfcc[varx],dfcc[vary],s=100,c=dfcc['dc'],cmap=cmap,
                  norm=CenteredNorm(vcenter=1,halfrange=0.5),alpha=1.0,label=None)
px = np.arange(0.01,1.01,0.01)
py=acut_KeH/px
ax[2].plot(px,py,'-k',label='Most accurate for CC')
ax[2].set_ylim(1,2)
ax[2].set_xlim(0.5,1)
#ax[2].legend(loc='upper left',fontsize=fs)
ax[2].set_xlabel('$H^{\prime}$',fontsize=fs)
ax[2].set_ylabel('$k_e$',fontsize=fs)
ax[2].tick_params(axis='both', which='major', labelsize=fs)
ax[2].set_title('$H^{\prime} k_e$ vs $\delta$ critical boundary',fontsize=fs)

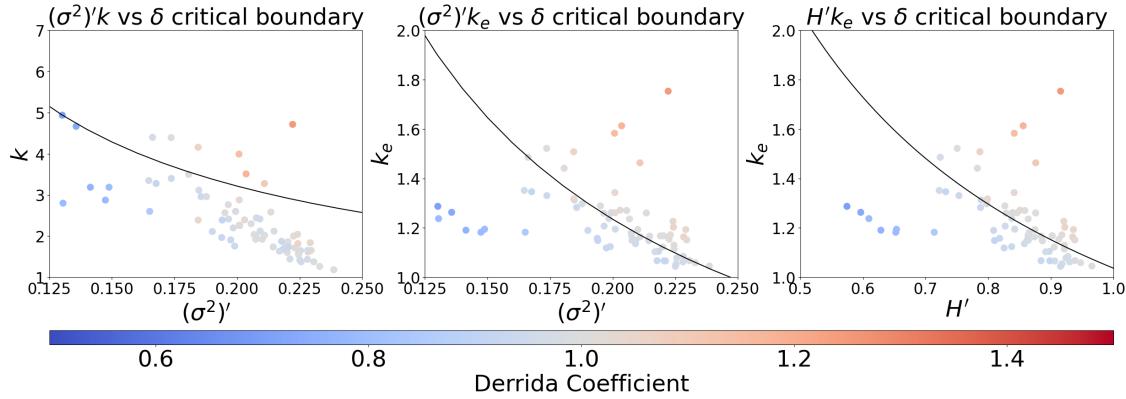
cbar=fig.colorbar(sc,ax=ax,location='bottom',aspect=75)
cbar.ax.tick_params(axis='both',labelsize=fs)
cbar.set_label('Derrida Coefficient',fontsize=fs)

```

```

plt.savefig(f'figures/PowerLawFits_CC.pdf',bbox_inches='tight')
plt.savefig(f'figures/PowerLawFits_CC.png',bbox_inches='tight')
plt.show()

```



We now examine how well the critical regions predicted by the fits agree with that obtained from δ directly. The critical region here is taken to be the range of δ values (predicted by the connectivity-spread products or measured) that is centered at $\delta = 1$ and has width equal to the IQR of the δ distribution (predicted or measured). We use the optimal boundaries obtained earlier.

```

[ ]: fs = 20
mpl.rcParams["font.size"] = 20
dev_frac = 2
critical_range = np.subtract(*np.percentile(dfcc['dc'], [75, 25]))/
    ↪dev_frac#dfcc['dc'].std()/dev_frac
truth = (dfcc['dc'] > (1+critical_range)).astype(int) - \
    (dfcc['dc'] < (1-critical_range)).astype(int)

fig, ax = plt.subplots(1, 3, figsize=
    15, 10), sharey='row', sharex='col', facecolor='white')

x = dfcc['k']*dfcc['avgV']
yfit = x/acut_KV
critical_range = np.subtract(*np.percentile(yfit, [75, 25]))/dev_frac#yfit.
    ↪std()/dev_frac
preds = (yfit > (1+critical_range)).astype(int) - \
    (yfit < (1-critical_range)).astype(int)
confusion_matrix = metrics.confusion_matrix(truth, preds)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=['Ordered', 'Critical', ↪
    'Chaotic'])
cm_display.plot(ax=ax[0])

```

```

cm_display.im_.colorbar.remove()
mstr = '{'+str(np.round(m, 2))+'}'
label = f'${(\sigma^2)^\prime}$ boundary, fit from CC'
#ax[0].set_title(label, fontsize=fs)
ax[0].set_xlabel(f'Dynamical regime from ${(\sigma^2)^\prime}$', fontsize=fs)

x = dfcc['ke']*dfcc['avgV']
yfit = x/acut_KeV
critical_range = np.subtract(*np.percentile(yfit, [75, 25]))/dev_frac#yfit.
    ↵std()/dev_frac
preds = (yfit > (1+critical_range)).astype(int) - \
    (yfit < (1-critical_range)).astype(int)
confusion_matrix = metrics.confusion_matrix(truth, preds)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=['Ordered', 'Critical', ↵
    ↵'Chaotic'])
cm_display.plot(ax=ax[1])
cm_display.im_.colorbar.remove()
mstr = '{'+str(np.round(m, 2))+'}'
label = f'${(\sigma^2)^\prime k_e}$ boundary, fit from CC'
#ax[1].set_title(label, fontsize=fs)
ax[1].set_xlabel(f'Dynamical regime from ${(\sigma^2)^\prime k_e}$', fontsize=fs)

x = dfcc['ke']*dfcc['avgH']
yfit = x/acut_KeH
critical_range = np.subtract(*np.percentile(yfit, [75, 25]))/dev_frac#yfit.
    ↵std()/dev_frac
preds = (yfit > (1+critical_range)).astype(int) - \
    (yfit < (1-critical_range)).astype(int)
confusion_matrix = metrics.confusion_matrix(truth, preds)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=['Ordered', 'Critical', ↵
    ↵'Chaotic'])
cm_display.plot(ax=ax[2])
cm_display.im_.colorbar.remove()
mstr = '{'+str(np.round(m, 2))+'}'
label = f'$H^\prime k_e$ boundary, fit from CC'
#ax[2].set_title(label, fontsize=fs)
ax[2].set_xlabel(f'Dynamical regime from $H^\prime k_e$', fontsize=fs)

# fix font sizes
for axis in ax:
    axis.tick_params(axis='both', which='major', labelsize=fs)
    #axis.set_xlabel('');
    axis.set_ylabel('');
    #axis.xaxis.set_ticklabels(['Below', 'Above']);
    axis.yaxis.set_ticklabels(['Ordered', 'Critical', 'Chaotic']);

```

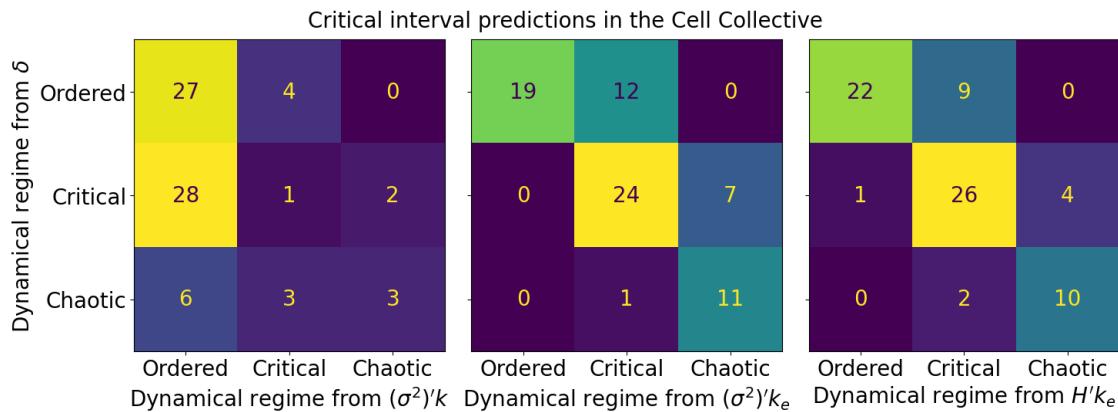
```

ax[0].set_ylabel('Dynamical regime from $\delta$', fontsize=fs)

fig.suptitle(f'Critical interval predictions in the CellCollective', fontsize=fs)
fig.tight_layout()
fig.subplots_adjust(top=1.45)

plt.savefig(f'figures/ConfusionIQR_CC.pdf', bbox_inches='tight')
plt.savefig(f'figures/ConfusionIQR_CC.png', bbox_inches='tight')
plt.show()

```



We also explore the effect of varying the threshold for the critical boundary for the various measures. We begin by constructing the ROCs.

```

[ ]: fs = 24
fig, ax = plt.subplots(1,1, figsize=(10, 10), sharey='row', sharex='col', facecolor='white')
truth = (dfcc['dc']>1)

x = dfcc['k']*dfcc['avgV']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
fpr = fp/(tn+fp)
tpr = tp/(tp+fn)
AUC=-np.sum(tpr[0:-1]*np.diff(fpr)).round(3)
ax.plot(fpr,tpr,'-.',c='orange',label=f'${(\sigma^2)'}k$, {AUC=}')

x = dfcc['ke']*dfcc['avgV']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])

```

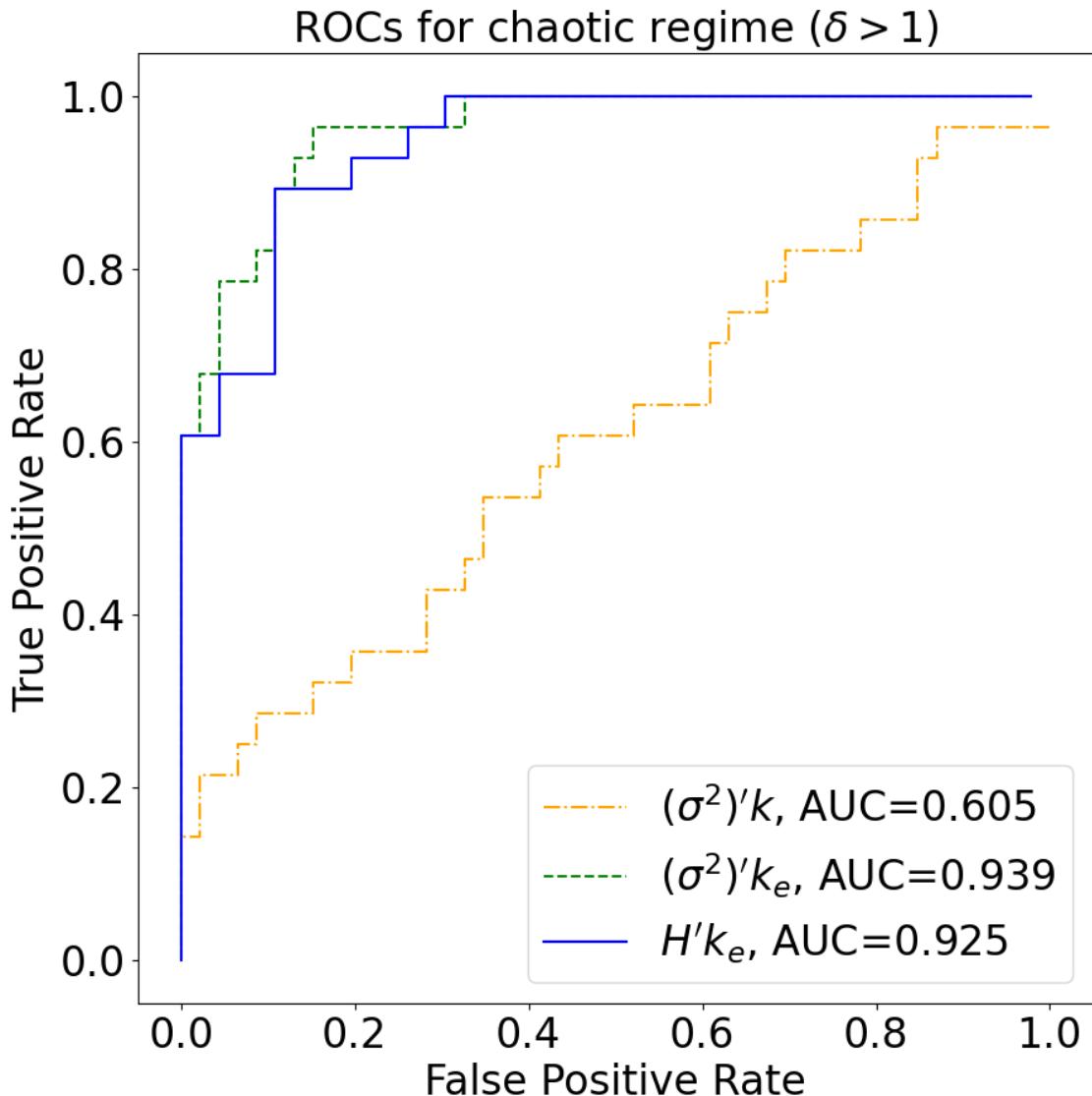
```

tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
fpr = fp/(tn+fp)
tpr = tp/(tp+fn)
AUC=-np.sum(tpr[0:-1]*np.diff(fpr)).round(3)
ax.plot(fpr,tpr,'--',c='green',label=f'${(\sigma^2)^{\prime}}_{\text{e}}$, {AUC=}')

x = dfcc['ke']*dfcc['avgH']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
fpr = fp/(tn+fp)
tpr = tp/(tp+fn)
AUC=-np.sum(tpr[0:-1]*np.diff(fpr)).round(3)
ax.plot(fpr,tpr,'-',c='blue',label=f'${H^{\prime}}_{\text{e}}$, {AUC=}')

ax.set_ylabel('True Positive Rate', fontsize=fs)
ax.set_xlabel('False Positive Rate', fontsize=fs)
ax.legend(fontsize=fs)
ax.tick_params(axis='both', which='major', labelsize=fs)
ax.set_title('ROC for chaotic regime ($\delta>1$)', fontsize=fs)
plt.savefig(f'figures/ROC_DC.pdf', bbox_inches='tight')
plt.savefig(f'figures/ROC_DC.png', bbox_inches='tight')
plt.show()

```



We now construct the PRCs.

```
[ ]: fs = 24
fig, ax = plt.subplots(1,1, figsize=(10, 10), sharey='row', sharex='col', facecolor='white')
truth = (dfcc['dc']>1)

x = dfcc['k']*dfcc['avgV']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
prec = tp/(tp+fp)
```

```

reca = tp/(tp+fn)
AUC=-np.sum(prec[0:-1]*np.diff(reca)).round(3)
ax.plot(reca,prec,'-.',c='orange',label=f'${(\sigma^2)^\prime k}$, {AUC=}'')

x = dfcc['ke']*dfcc['avgV']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
prec = tp/(tp+fp)
reca = tp/(tp+fn)
AUC=-np.nanmean(prec[0:-1]*np.diff(reca)).round(3)
ax.plot(reca,prec,'--',c='green',label=f'${(\sigma^2)^\prime k_e}$, {AUC=}'')

x = dfcc['ke']*dfcc['avgH']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
prec = tp/(tp+fp)
reca = tp/(tp+fn)
AUC=-np.nanmean(prec[0:-1]*np.diff(reca)).round(3)
ax.plot(reca,prec,'-',c='blue',label=f'$H^\prime k_e$, {AUC=}'')

noskill=np.sum(truth)/len(truth)
ax.hlines(noskill,0,1,color='k',linestyle='--',label="no skill classifier")

ax.set_xlim(0,1)
ax.set_ylim(noskill-0.05,1.05)

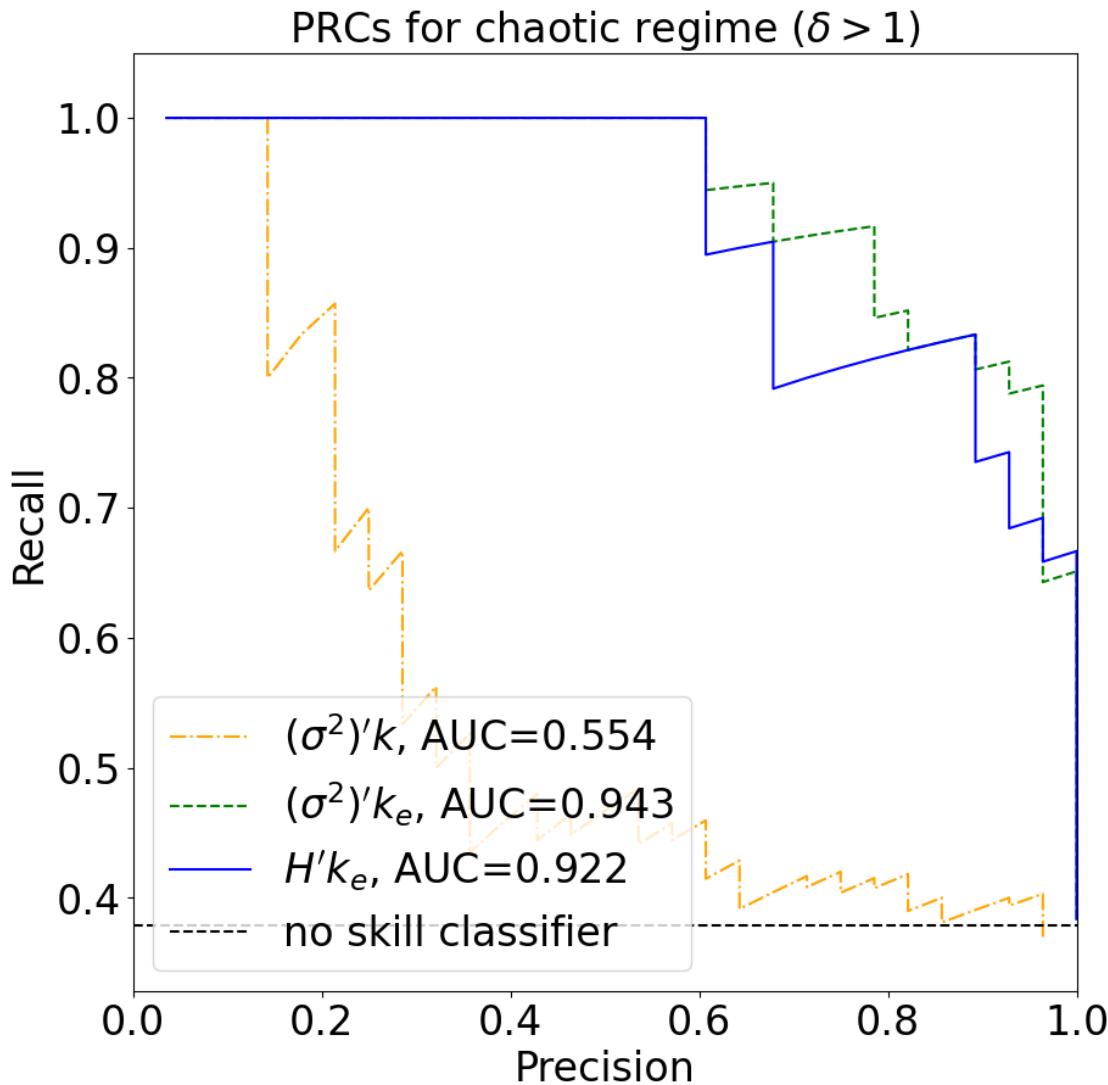
ax.set_ylabel('Recall', fontsize=fs)
ax.set_xlabel('Precision', fontsize=fs)
ax.legend(fontsize=fs, loc='lower left')
ax.tick_params(axis='both', which='major', labelsize=fs)
ax.set_title('PRCs for chaotic regime ($\delta>1$)', fontsize=fs)
plt.savefig(f'figures/PRC_DC.pdf', bbox_inches='tight')
plt.savefig(f'figures/PRC_DC.png', bbox_inches='tight')
plt.show()

```

```

C:\Users\jcroz\AppData\Local\Temp\ipykernel_7764\3623365655.py:11:
RuntimeWarning: invalid value encountered in divide
    prec = tp/(tp+fp)
C:\Users\jcroz\AppData\Local\Temp\ipykernel_7764\3623365655.py:21:
RuntimeWarning: invalid value encountered in divide
    prec = tp/(tp+fp)
C:\Users\jcroz\AppData\Local\Temp\ipykernel_7764\3623365655.py:31:
RuntimeWarning: invalid value encountered in divide
    prec = tp/(tp+fp)

```



We also consider the ability of these parameters to predict the dynamical regime in terms of sensitivity.

```
[ ]: fs = 24

truth = (dfcc['s'] > 1)
fig, ax = plt.subplots(3,3, figsize=(10, 10), sharey='row', sharex='col', facecolor='white')

for row,method in enumerate(['MCC','Accuracy','Cohen kappa']):

    x = dfcc['k']*dfcc['avgV']
    preds,cut, score=optimize_cut(x,truth,method=method)
```

```

confusion_matrix = metrics.confusion_matrix(truth, preds)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=['$s\leq 1$', '$s>1$'])
cm_display.plot(ax=ax[row,0])
cm_display.im_.colorbar.remove()
mstr = '{'+str(np.round(m, 2))+'}'
label = f'$\sigma^2 \prime k$={np.round(cut,3)}, {method}={np.
round(score,2)}'
ax[row,0].set_title(label)

x = dfcc['ke']*dfcc['avgV']
preds,cut,score=optimize_cut(x,truth,method=method)
confusion_matrix = metrics.confusion_matrix(truth, preds)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=['$s\leq 1$', '$s>1$'])
cm_display.plot(ax=ax[row,1])
cm_display.im_.colorbar.remove()
mstr = '{'+str(np.round(m, 2))+'}'
label = f'$\sigma^2 \prime k_e$={np.round(cut,3)}, {method}={np.
round(score,2)}'
ax[row,1].set_title(label)

x = dfcc['ke']*dfcc['avgH']
preds,cut,score=optimize_cut(x,truth,method=method)
confusion_matrix = metrics.confusion_matrix(truth, preds)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=['$s\leq 1$', '$s>1$'])
cm_display.plot(ax=ax[row,2])
cm_display.im_.colorbar.remove()
mstr = '{'+str(np.round(m, 2))+'}'
label = f'$H \prime k_e$={np.round(cut,3)}, {method}={np.round(score,2)}'
ax[row,2].set_title(label)

fig.tight_layout()
plt.savefig(f'figures/SensitivityConfusionOptimized_CC.pdf',bbox_inches='tight')
plt.savefig(f'figures/SensitivityConfusionOptimized_CC.png',bbox_inches='tight')
plt.show()

```

$(\sigma^2)'k=0.643$, MCC ≈ 0.88 , $H'k_e=0.91$, MCC $=0.87$

| True label | $s \leq 1$ | $s > 1$ |
|-----------------|------------|---------|
| Predicted label | 50 17 | 1 6 |
| $s \leq 1$ | 50 | 1 |
| $s > 1$ | 17 | 6 |

| True label | $s \leq 1$ | $s > 1$ |
|-----------------|------------|---------|
| Predicted label | 49 1 | 2 22 |
| $s \leq 1$ | 49 | 2 |
| $s > 1$ | 1 | 22 |

| True label | $s \leq 1$ | $s > 1$ |
|-----------------|------------|---------|
| Predicted label | 50 3 | 1 20 |
| $s \leq 1$ | 50 | 1 |
| $s > 1$ | 3 | 20 |

$(\sigma^2)'k=0.643$, Accuracy ≈ 0.752 , Accuracy $H'k_e=0.958$, Accuracy $=0.95$

| True label | $s \leq 1$ | $s > 1$ |
|-----------------|------------|---------|
| Predicted label | 50 17 | 1 6 |
| $s \leq 1$ | 50 | 1 |
| $s > 1$ | 17 | 6 |

| True label | $s \leq 1$ | $s > 1$ |
|-----------------|------------|---------|
| Predicted label | 49 1 | 2 22 |
| $s \leq 1$ | 49 | 2 |
| $s > 1$ | 1 | 22 |

| True label | $s \leq 1$ | $s > 1$ |
|-----------------|------------|---------|
| Predicted label | 50 3 | 1 20 |
| $s \leq 1$ | 50 | 1 |
| $s > 1$ | 3 | 20 |

$(\sigma^2)'k=0.575$, Cohen kappa ≈ 0.523 , Cohen kappa $H'k_e=0.58$, Cohen kappa $=0.91$

| True label | $s \leq 1$ | $s > 1$ |
|-----------------|------------|---------|
| Predicted label | 47 15 | 4 8 |
| $s \leq 1$ | 47 | 4 |
| $s > 1$ | 15 | 8 |

| True label | $s \leq 1$ | $s > 1$ |
|-----------------|------------|---------|
| Predicted label | 49 1 | 2 22 |
| $s \leq 1$ | 49 | 2 |
| $s > 1$ | 1 | 22 |

| True label | $s \leq 1$ | $s > 1$ |
|-----------------|------------|---------|
| Predicted label | 50 3 | 1 20 |
| $s \leq 1$ | 50 | 1 |
| $s > 1$ | 3 | 20 |

Notably, $(\sigma^2)'k_e$ and $H'k_e$ show very good agreement with the sensitivity. ##### We now build the ROCs for sensitivity prediction.

```
[ ]: fs = 24
fig, ax = plt.subplots(1,1, figsize=(10, 10), sharey='row', sharex='col', facecolor='white')
truth = (dfcc['s']>1)

x = dfcc['k']*dfcc['avgV']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
fpr = fp/(tn+fp)
tpr = tp/(tp+fn)
AUC=-np.sum(tpr[0:-1]*np.diff(fpr)).round(3)
ax.plot(fpr,tpr,'-.',c='orange',label=f'${(\sigma^2)'}k$, {AUC=}')

x = dfcc['ke']*dfcc['avgV']
```

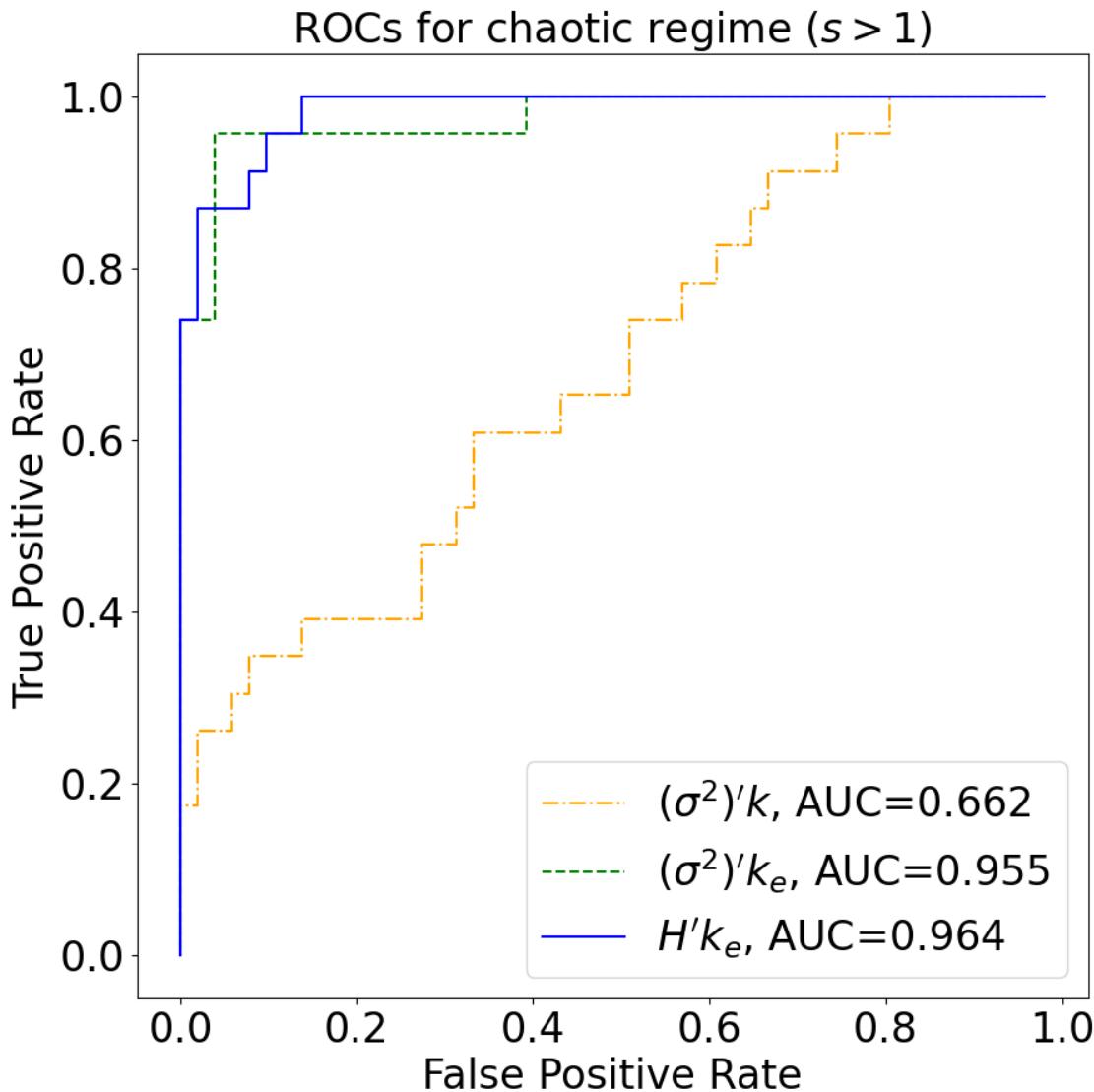
```

fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
fpr = fp/(tn+fp)
tpr = tp/(tp+fn)
AUC=-np.sum(tpr[0:-1]*np.diff(fpr)).round(3)
ax.plot(fpr,tpr,'--',c='green',label=f'${(\sigma^2)^{\prime} k_e}$, {AUC=}')

x = dfcc['ke']*dfcc['avgH']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
fpr = fp/(tn+fp)
tpr = tp/(tp+fn)
AUC=-np.sum(tpr[0:-1]*np.diff(fpr)).round(3)
ax.plot(fpr,tpr,'-',c='blue',label=f'$H^{\prime} k_e$', {AUC=})

ax.set_ylabel('True Positive Rate', fontsize=fs)
ax.set_xlabel('False Positive Rate', fontsize=fs)
ax.legend(fontsize=fs)
ax.tick_params(axis='both', which='major', labelsize=fs)
ax.set_title('ROC for chaotic regime ($s>1$)', fontsize=fs)
plt.savefig(f'figures/ROC_S.pdf',bbox_inches='tight')
plt.savefig(f'figures/ROC_S.png',bbox_inches='tight')
plt.show()

```



We now build the PRCs for sensitivity prediction.

```
[ ]: fs = 24
fig, ax = plt.subplots(1,1, figsize=(10, 10), sharey='row', sharex='col', facecolor='white')
truth = (dfcc['s']>1)

x = dfcc['k']*dfcc['avgV']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
prec = tp/(tp+fp)
```

```

reca = tp/(tp+fn)
AUC=-np.sum(prec[0:-1]*np.diff(reca)).round(3)
ax.plot(reca,prec,'-.',c='orange',label=f'$\sigma^2 \prime k$, {AUC=})'

x = dfcc['ke']*dfcc['avgV']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
prec = tp/(tp+fp)
reca = tp/(tp+fn)
AUC=-np.nanmean(prec[0:-1]*np.diff(reca)).round(3)
ax.plot(reca,prec,'--',c='green',label=f'$\sigma^2 \prime k_e$, {AUC=})'

x = dfcc['ke']*dfcc['avgH']
fp = np.array([np.sum((x > cut) & ~truth) for cut in sorted(x)])
tp = np.array([np.sum((x > cut) & truth) for cut in sorted(x)])
tn = np.array([np.sum((x <= cut) & ~truth) for cut in sorted(x)])
fn = np.array([np.sum((x <= cut) & truth) for cut in sorted(x)])
prec = tp/(tp+fp)
reca = tp/(tp+fn)
AUC=-np.nanmean(prec[0:-1]*np.diff(reca)).round(3)
ax.plot(reca,prec,'-',c='blue',label=f'$H \prime k_e$, {AUC=})'

noskill=np.sum(truth)/len(truth)
ax.hlines(noskill,0,1,color='k',linestyle='--',label="no skill classifier")

ax.set_xlim(0,1)
ax.set_ylim(noskill-0.05,1.05)

ax.set_ylabel('Recall', fontsize=fs)
ax.set_xlabel('Precision', fontsize=fs)
ax.legend(fontsize=fs, loc='lower left')
ax.tick_params(axis='both', which='major', labelsize=fs)
ax.set_title('PRCs for chaotic regime ($s>1$)', fontsize=fs)
plt.savefig(f'figures/PRC_S.pdf', bbox_inches='tight')
plt.savefig(f'figures/PRC_S.png', bbox_inches='tight')
plt.show()

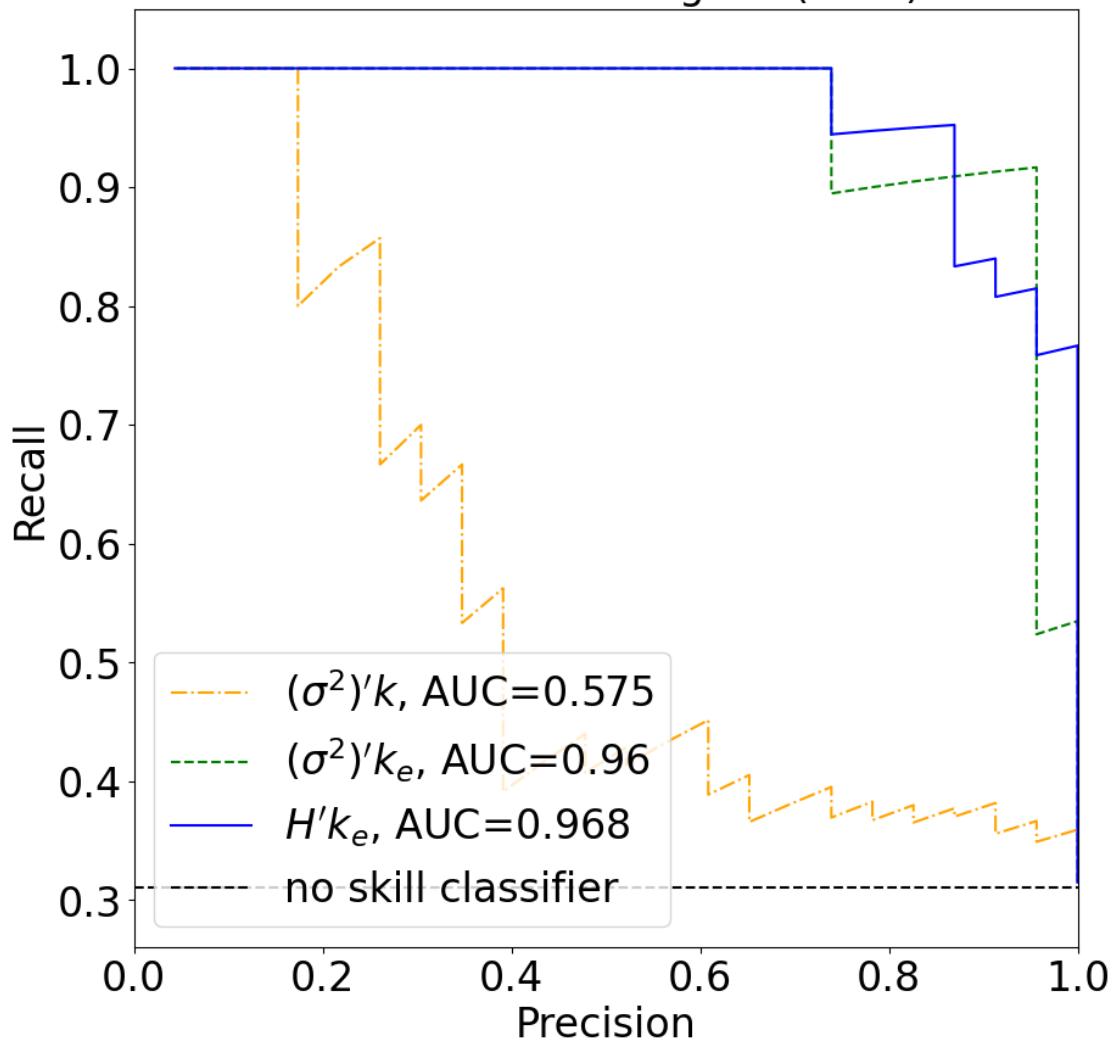
```

```

C:\Users\jcroz\AppData\Local\Temp\ipykernel_7764\3408119185.py:11:
RuntimeWarning: invalid value encountered in divide
    prec = tp/(tp+fp)
C:\Users\jcroz\AppData\Local\Temp\ipykernel_7764\3408119185.py:21:
RuntimeWarning: invalid value encountered in divide
    prec = tp/(tp+fp)
C:\Users\jcroz\AppData\Local\Temp\ipykernel_7764\3408119185.py:31:
RuntimeWarning: invalid value encountered in divide
    prec = tp/(tp+fp)

```

PRCs for chaotic regime ($s > 1$)



[]: