CommunitySim

In this vignette, we simulate the dynamics of a community of bacteria after inoculation, and use various popular machine learning algorithms to predict the steady state abundance of the bacteria using their initial abundance. In the process, we demonstrate how to use CommunitySim package to simulate the parameters for community simulation, how to simulate a community, and how to read off the steady state abundance of members of the community.

```
library(CommunitySim)
library(reticulate)
use_condaenv("CommunitySimDemo")
# Need a conda env with numpy, seaborn, and scikit-learn for Machine Learning Case Study
```

Basic Functions

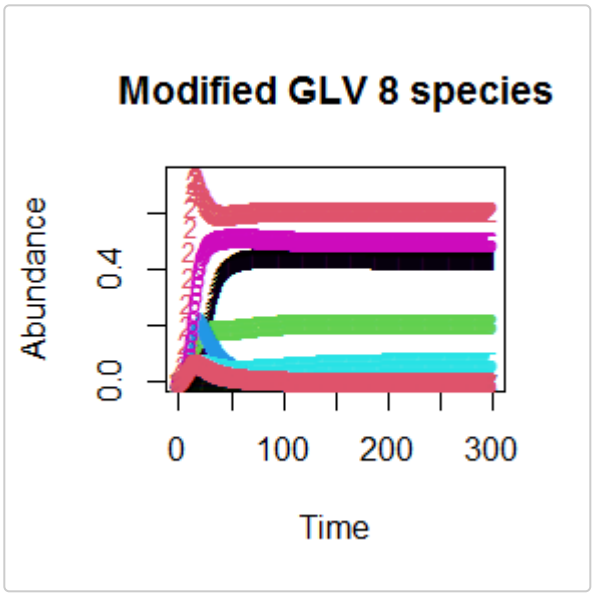
In this demonstration, we simulate a community of 8 possible species. 4 parameters are needed to describe and hence simulate the community: `alpha` is 1x8 vector of intrinsic growth rate of each species; `c0` is a 8x8 matrix of pairwise interaction intensity in the community, where `c0[i,j]` is the impact of species `j` on `i` on a per unit basis; `ck` is a list of 8 matrices, each with 8x8 entries, where `ck[[k]][i,j]` is the higher order impact from species `j` onto the pairwise interaction of `i` and `k`, felt by `k`; `init` is a 1x8 vector of initial abundance at the inoculation.

```
N <- 8
params <- simulateParam(N)
print(diag(params$c0))
#> [1] -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5
```

Notice the diagonal of `c0` are all set to `-0.5` to reflect the intra-species competition.

Then we simulate the dynamics of the abundance of species. A DataFrame of 8 columns (for each species) of abundance at each integration time points (default to 300 time points) is returned. A plot of all columns are also displayed.

```
abundance <- growthFunction(params$N, params$alpha, params$c0, params$ck, params$init)
```



Machine Learning Case Study

Data Preparation

Suppose we know an environment has 8 possible bacterial species. When we take sample from the environment and inoculate in a lab, we can cherry-pick particular species and add to a culture medium. For different batch of inoculations, typically the initial abundance of adding a particular species are fixed, which is determined by the concentration of bacteria stock and pipetting volume. What we can play with is whether to add each species. Following this reasoning, there are 2^8 possibilities of sampling 8 species. If we assume a fixed inoculation abundance for each species, we can generate 2^8 initial conditions.

```
mask <- binaryInitState(N)
head(mask)
#>     Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8
#> 48     1    1    1    1    0    1    0    0
#> 246    1    0    1    0    1    1    1    1
#> 34     1    0    0    0    0    1    0    0
#> 131    0    1    0    0    0    0    0    1
#> 197    0    0    1    0    0    0    1    1
#> 170    1    0    0    1    0    1    0    1
```

```
init <- t(t(mask) * params$init)
head(data.frame(init))
#>            Var1        Var2        Var3        Var4        Var5       Var6
#> 48  0.001146661 0.006504564 0.001295101 0.004169703 0.000000000 0.00307967
#> 246 0.001146661 0.000000000 0.001295101 0.000000000 0.005851208 0.00307967
#> 34  0.001146661 0.000000000 0.000000000 0.000000000 0.000000000 0.00307967
#> 131 0.000000000 0.006504564 0.000000000 0.000000000 0.000000000 0.00000000
#> 197 0.000000000 0.000000000 0.001295101 0.000000000 0.000000000 0.00000000
#> 170 0.001146661 0.000000000 0.000000000 0.004169703 0.000000000 0.00307967
#>             Var7        Var8
#> 48  0.0000000000 0.000000000
```

```
#> 246 0.0003479984 0.005823576
#> 34  0.0000000000 0.000000000
#> 131 0.0000000000 0.005823576
#> 197 0.0003479984 0.005823576
#> 170 0.0000000000 0.005823576
```

where each row correspond to an inoculation abundance when some species are present while the others are not.

Next, for each row - an initial abundance, we can run the community simulation of such inoculation.

```
abun_list <- apply(init, 1, function(x) growthFunction(N, params$alpha, params$c0, params$ck, x))
```

We then use a function provided by `CommunitySim` to find the steady state abundance of each species for each initial state simulation.

```
matrixToSS <- function(densityMatrix){
  s <- apply(densityMatrix[, 2:(N+1)], 2, findSteadyState)
  return(s)
}

SS <- sapply(abun_list, matrixToSS)
SS <- t(SS)
#If SS abundance is too small, set to 0
SS[which(SS < 0.001)] = 0
```

Model Training and Prediction

We carry on the rest of the study in Python because we love Python (no). For 8 species community, we generated 2^8 = 256 data points. We use 80% of the data to train model and use the rest 20% to predict and assess accuracy. We compare the prediction accuracy of the linear regressor, SVM, and Random Forest.

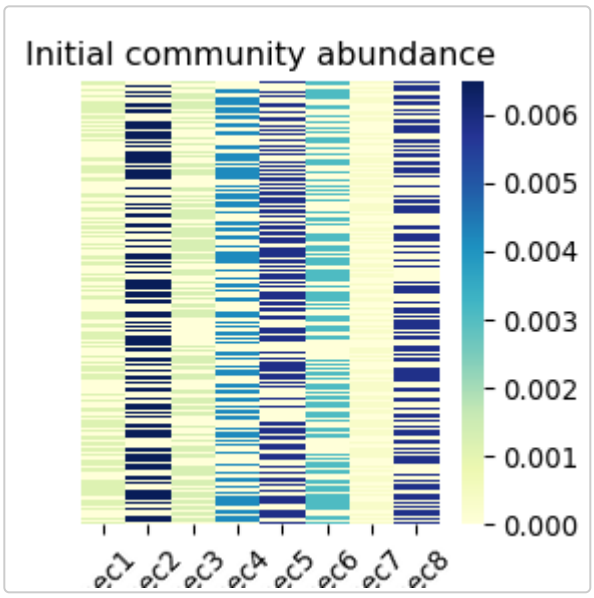
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

N = int(r.N)
init = r.init
SS = r.SS
```

SS may contain `nan` when a steady state is not reached. We need to remove those samples.

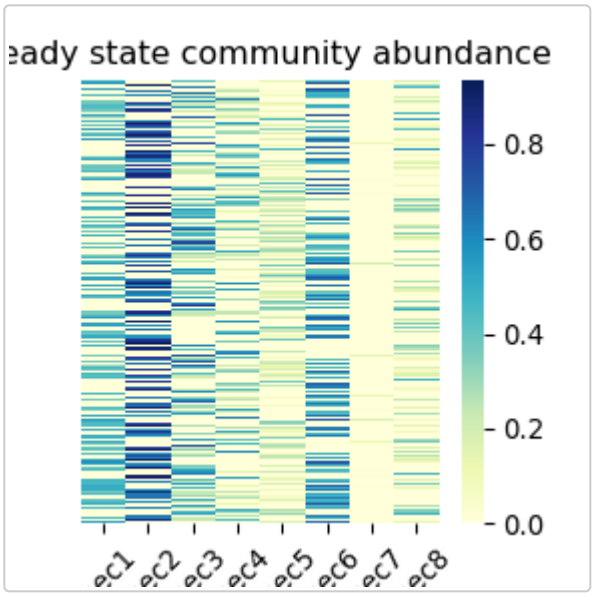
```
nanSamples = np.isnan(SS).any(axis = 1)
init_clean = init[~nanSamples, ]
SS_clean = SS[~nanSamples, ]
```

We can visualize the input and the output using a heatmap.

```
ax = sns.heatmap(init_clean, cmap="YlGnBu")
ax.set_xticklabels(["Spec"+str(i) for i in range(1,N+1)])
plt.setp(ax.get_xticklabels(), rotation=45)
#> [None, None, None, None, None, None, None, None, None, None, None, None, None, None, None]
ax.set_yticks([])
#> []
ax.set_title('Initial community abundance')
plt.show()
```




```
plt.clf()

ax = sns.heatmap(SS_clean, cmap="YlGnBu")
ax.set_xticklabels(["Spec"+str(i) for i in range(1,N+1)])
plt.setp(ax.get_xticklabels(), rotation=45)
#> [None, None, None, None, None, None, None, None, None, None, None, None, None, None, None]
ax.set_yticks([])
#> []
ax.set_title('Steady state community abundance')
plt.show()
```




```
plt.clf()
```

For model training and testing, we use 5-fold cross validation.

```
K = 5
kf = KFold(n_splits = K, shuffle = True)
train_idx = {}
test_idx = {}
i = 0
for train_i, test_i in kf.split(init_clean):
  train_idx[i] = train_i
  test_idx[i] = test_i
  i += 1
```

Generic model training and accuracy assessment function.

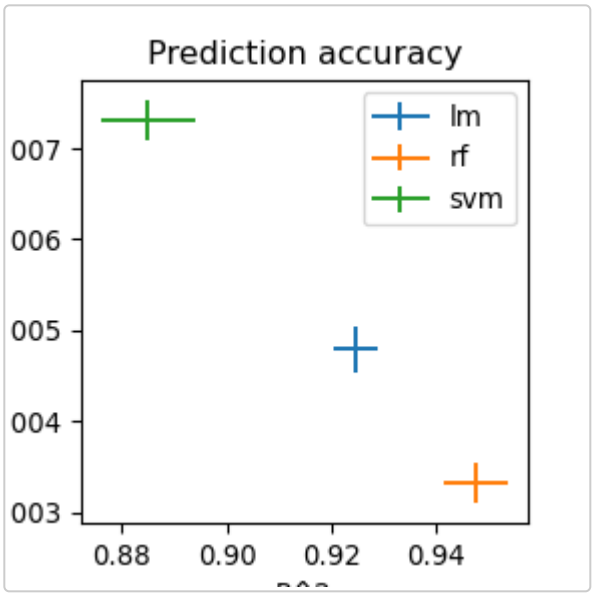
```
def trainAssess(X_train, y_train, X_test, y_test, model):

  prediction = np.zeros(y_test.shape)

  for i in range(N):
    reg = model.fit(X_train, y_train[:, i])
    pred = reg.predict(X_test)
    prediction[:,i] = pred

  rmse = mean_squared_error(y_test.flatten(), prediction.flatten())
  r2 = r2_score(y_test.flatten(), prediction.flatten())

  return rmse, r2
```

Initialize models and run trainAssess routine on K fold.

```
lm = LinearRegression()
svm = SVR()
rf = RandomForestRegressor(max_depth=2)

output = []

for k in range(K):
  X_train = init_clean[train_idx[k], :]
  X_test = init_clean[test_idx[k], :]
  y_train = SS_clean[train_idx[k], :]
  y_test = SS_clean[test_idx[k], :]

  rmse, r2 = trainAssess(X_train, y_train, X_test, y_test, lm)
  output.append([rmse, r2, 'lm'])

  rmse, r2 = trainAssess(X_train, y_train, X_test, y_test, svm)
  output.append([rmse, r2, 'svm'])

  rmse, r2 = trainAssess(X_train, y_train, X_test, y_test, rf)
  output.append([rmse, r2, 'rf'])

output = pd.DataFrame(output, columns = ['rmse', 'r2', 'model'])
```

Finally let's plot the output.

```
x = output.groupby(['model']).mean().loc[:,'r2']
y = output.groupby(['model']).mean().loc[:,'rmse']
xstd = output.groupby(['model']).std().loc[:,'r2']
ystd = output.groupby(['model']).std().loc[:,'rmse']

color = range(3)
labels = output.groupby(['model']).mean().index

ax = plt.errorbar(x[0], y[0], xerr = xstd[0], yerr = ystd[0], label = labels[0])
plt.errorbar(x[1], y[1], xerr = xstd[1], yerr = ystd[1], label = labels[1])
#> <ErrorbarContainer object of 3 artists>
plt.errorbar(x[2], y[2], xerr = xstd[2], yerr = ystd[2], label = labels[2])
#> <ErrorbarContainer object of 3 artists>
plt.legend()
plt.title('Prediction accuracy')
plt.xlabel('R^2')
plt.ylabel('RMSE')
plt.show()
```




```
plt.clf()
```

Therefore we may conclude that for a web-lab experiment designed as we stated above, i.e. permutate species presence and observe steady state abundance, random forest can be used to predict the steady state of the community.