Supplementary information 1

The source code of deep learning algorithm

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
os.environ['CUDA_VISIBLE_DEVICES'] = "0"
import warnings
warnings.filterwarnings('ignore')

from sklearn.metrics import r2_score, mean_squared_error

def split_train_test(data, frac=0.2, seed=None):
    test_data = data.sample(frac=frac, random_state=seed)
    train_data = data.drop( index=test_data.index)
    return train_data, test_data

def split_x_y(train_data, test_data, target):
    y_train = np.reshape(np.array(train_data[target]), [-1,1])
    x_train = train_data.drop(target, axis=1)
    y_test = np.reshape(np.array(test_data[target]), [-1,1])
    x_test = test_data.drop(target, axis=1)
    return x_train, y_train, x_test, y_test


data = pd.read_csv('STAT3_L40.csv')
data.head(5)
new_data=pd.read_csv('pr.csv')


def init():
    init = tf.global_variables_initializer()
    sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True,
log_device_placement=True))
    sess.run(init)
train_data, test_data = split_train_test(data, .2)
x_train, y_train, x_test, y_test = split_x_y(train_data, test_data, target='pIC50')
new_data_train = new_data

X = tf.placeholder(shape=(None, len(x_train.columns)), dtype=tf.float32)
Y = tf.placeholder(shape=(None, 1), dtype=tf.float32)
```

```
fc_1 = tf.layers.dense(inputs=X,
                       activation=tf.nn.relu,
                       use_bias=True,
                       units=100)
fc_2 = tf.layers.dense(inputs=fc_1,
                       activation=tf.nn.relu,
                       use_bias=True,
                       units=50)
fc_2 = tf.layers.dropout(inputs=fc_2,
                         rate=0.4)
fc_3 = tf.layers.dense(inputs=fc_1,
                       activation=tf.nn.relu,
                       use_bias=True,
                       units=20)
fc_3 = tf.layers.dropout(inputs=fc_2,
                         rate=0.4)
fc_4 = tf.layers.dense(inputs=fc_2,
                       activation=tf.nn.relu,
                       use_bias=True,
                       units=10,
                       )
fc_4 = tf.layers.dropout(inputs=fc_3,
                         rate=0.6)
for i in range(40):
    fc_4 = tf.layers.dense(inputs=fc_4,
                       activation=tf.nn.relu,
                       use_bias=True,
                       units=10,
                       )
    fc_4 = tf.layers.dropout(inputs=fc_4,
                         rate=0.3)
out = tf.layers.dense(inputs=fc_4,
                       activation=None,
                       use_bias=True,
                       units=1)

loss = tf.losses.mean_squared_error(out, Y)
step = tf.train.AdamOptimizer(learning_rate=.0001).minimize(loss)


def run(search_range=50):
    tmp_r2_train = -np.inf
    tmp_r2_test = -np.inf
```

```
    prediction1 = sess.run(out, feed_dict={X: new_data_train})
    for i in range(1, search_range):

        _, tmp_cost = sess.run([step, loss], feed_dict={X: x_train, Y: y_train})

        if i % 10 == 0:
            prediction = sess.run(out, feed_dict={X: x_train})
            prediction = np.reshape(prediction, [-1, 1])
            test_prediction = sess.run(out, feed_dict={X: x_test})
            test_prediction = np.reshape(test_prediction, [-1, 1])

            if (r2_score(prediction, y_train) > tmp_r2_train or
r2_score(test_prediction,y_test) > tmp_r2_test) and abs( r2_score(prediction, y_train)
- r2_score(test_prediction, y_test)) < .2:
                tmp_r2_train = r2_score(prediction, y_train)
                tmp_r2_test = r2_score(test_prediction, y_test)
            if tmp_r2_train > .8 and tmp_r2_test > .8:
                print('    saving...')
                saver = tf.train.Saver()
                path = saver.save(sess,
'./my_model{}_{}_{}_{}_.ckpt'.format(seed, num, str(tmp_r2_test)[:4],
str(tmp_r2_train)[:4]))
                prediction1 = sess.run(out, feed_dict={X: new_data_train})
                print(prediction1)
    return tmp_r2_train, tmp_r2_test, prediction1



r2_train_list=[]
r2_test_list = []
for num in range(50):
    print('epoch {}'.format(num))
    for seed in range(1):
        np.random.seed = seed
        train_data, test_data = split_train_test(data, .2, seed=seed)
        x_train, y_train, x_test, y_test = split_x_y(train_data, test_data,
target='pIC50')

        init = tf.global_variables_initializer()
        sess = tf.Session()
        sess.run(init)

        r2_train, r2_test ,prediction1= run(10001)
        r2_train_list.append(r2_train)
```

```
    r2_test_list.append(r2_test)

    print(r2_train_list)
    print(r2_test_list)

    sns.scatterplot(r2_train_list, r2_test_list)
    plt.scatter(r2_train_list, r2_test_list)
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.xlabel('r2 train')
    plt.ylabel('r2 test')
    colors='#008000'
    plt.savefig('new_pic_r2.png')
```

Supplementary information 2

The source code of six different machine learning algorithms

```
import numpy as np
import pandas as pd

raw_data = pd.read_csv('STAT3_L40.csv')

X = raw_data.iloc[:,1:]
y = raw_data.iloc[:,0]
from yellowbrick.features import Rank2D
visualizer = Rank2D(algorithm="pearson", show_feature_names=False)
visualizer.fit_transform(X)
visualizer.poof(outpath='pearson1.png')


from yellowbrick.features.pca import PCADecomposition
visualizer = PCADecomposition(scale=True, proj_dim=2)
visualizer.fit_transform(X, y)
visualizer.poof(outpath='pca_2d.png')


from yellowbrick.features.pca import PCADecomposition
visualizer = PCADecomposition(scale=True, proj_dim=3)
visualizer.fit_transform(X, y)
visualizer.poof(outpath='pca_3d.png')


from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()
X_minmax = min_max_scaler.fit_transform(X)


from sklearn.feature_selection import VarianceThreshold
var = VarianceThreshold(threshold=0.05)
X_var = var.fit_transform(X_minmax)
X_var.shape



X_scaled = preprocessing.scale(X_var)
```

```
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1, max_iter=500)
lasso.fit(X_scaled, y)
model = SelectFromModel(lasso, prefit=True)
X_new = model.transform(X_scaled)
X_new.shape

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=3,
random_state=0)



visualizer = Rank2D(algorithm="pearson")
visualizer.fit_transform(X_new)
visualizer.poof(outpath='pearson2.png')



from sklearn import ensemble
rf = ensemble.RandomForestRegressor(n_estimators=20, oob_score=True, n_jobs=-
1, random_state=1)



rf.fit(X_train, y_train)



from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
predict = rf.predict(X_train)
true = y_train
print('MSE', mean_squared_error(true, predict))
print('R2', r2_score(true, predict))



predict = rf.predict(X_test)
true = y_test
print('MSE', mean_squared_error(true, predict))
print('R2', r2_score(true, predict))

yc = pd.read_csv('pr.csv')
Z = yc.iloc[:, 0:]
print("Z:", Z.shape)
Z_minMax = min_max_scaler.fit_transform(Z)
```

```
print("Z_minMax:", Z_minMax.shape)
Z_var = var.transform(Z_minMax)
print("Z_var:", Z_var.shape)
Z_scaled = preprocessing.scale(Z_var)
print("Z_scaled:", Z_scaled.shape)
Z_new = model.transform(Z_scaled)

predict = rf.predict(Z_new)
print("final predict", predict)
```

Supplementary information 3

GROMACS parameters

ions.mdp

```
integrator          = steep
emtol               = 1000.0
emstep          = 0.01
nsteps              = 50000
nstlist                 = 1
cutoff-scheme       = Verlet
ns_type                 = grid
coulombtype             = PME
rcoulomb                = 1.0
rvdw                    = 1.0
pbc                         = xyz
```

em.mdp

```
integrator          = steep
emtol               = 1000.0
emstep          = 0.01
nsteps              = 5000
nstlist                 = 1
cutoff-scheme       = Verlet
ns_type                 = grid
coulombtype             = PME
rcoulomb                = 1.0
rvdw                    = 1.0
pbc                         = xyz
```

nvt.mdp

```
title               = NVT equilibration
define              = -DPOSRES
integrator          = md
nsteps              = 10000000
dt                      = 0.002
nstxout             = 500
nstvout             = 5000
nstenergy           = 50
nstlog              = 500
continuation                = no
constraint_algorithm        = lincs
constraints                         = all-bonds
```

```
lincs_iter                = 1
lincs_order                        = 4
cutoff-scheme     = Verlet
ns_type                   = grid
nstlist                   = 10
rcoulomb                  = 1.0
rvdw                      = 1.0
coulombtype               = PME
pme_order                 = 4
fourierspacing      = 0.16
tcoupl                    = V-rescale
tc-grps                   = Protein Non-Protein
tau_t                     = 0.1          0.1
ref_t                     = 310          310
pcoupl                    = no
pbc                       = xyz
DispCorr= EnerPres
gen_vel                   = yes
gen_temp                  = 310
gen_seed= -1
```

npt.mdp

```
title                     = OPLS Lysozyme NPT equilibration
define                    = -DPOSRES
integrator                = md
nsteps                    = 10000000
dt                        = 0.002
nstxout                   = 500
nstvout                   = 500
nstenergy                 = 500
nstlog                    = 500
continuation                    = yes
constraint_algorithm       = lincs
constraints                        = all-bonds
lincs_iter                = 1
lincs_order                        = 4
cutoff-scheme     = Verlet
ns_type                   = grid
nstlist                   = 10
rcoulomb                  = 1.0
rvdw                      = 1.0
coulombtype               = PME
pme_order                 = 4
```

```
fourierspacing      = 0.16
tcoupl              = V-rescale
tc-grps             = Protein Non-Protein
tau_t               = 0.1       0.1
ref_t               = 300       300
pcoupl                      = Parrinello-Rahman
pcoupltype                  = isotropic
tau_p                       = 2.0
ref_p                       = 1.0
compressibility       = 4.5e-5
refcoord_scaling      = com
pbc                 = xyz
DispCorr= EnerPres
gen_vel             = no
```

md.mdp

```
title               = OPLS Lysozyme MD simulation
integrator          = md
nsteps              = 5000000
dt                        = 0.002
nstxout                   = 5000
nstvout                   = 5000
nstenergy                 = 5000
nstlog                    = 5000
nstxout-compressed    = 5000
compressed-x-grps     = System
continuation              = yes
constraint_algorithm      = lincs
constraints                       = all-bonds
lincs_iter                = 1
lincs_order                       = 4
cutoff-scheme     = Verlet
ns_type                   = grid
nstlist                   = 10
rcoulomb                  = 1.0
rvdw                      = 1.0
coulombtype               = PME
pme_order                 = 4
fourierspacing      = 0.16
tcoupl              = V-rescale
tc-grps             = Protein Non-Protein
tau_t               = 0.1       0.1
ref_t               = 310       310
```

```
pcoupl                          = Parrinello-Rahman
pcoupltype                      = isotropic
tau_p                           = 2.0
ref_p                           = 1.0
compressibility         = 4.5e-5
pbc                     = xyz
DispCorr= EnerPres
gen_vel                 = no
```