Supplemental Document

applied optics

Machine learning algorithms to control concentrations of carbon nanocomplexes in a biological medium via optical absorption spectroscopy: how to choose and what to expect?: supplement

Olga Sarmanova,[1,*] Kirill Laptinskiy,[2] Sergey Burikov,[1,2] Maria Khmeleva,[1,2] Anna Fedyanina,[1] Alexandra Tomskaya,[3,4] Aleksandr Efitorov,[2] Sergey Dolenko,[2] and Tatiana Dolenko[1,2]

[1] *Faculty of Physics, Lomonosov Moscow State University, Moscow 119991, Russia*
[2] *Skobeltsyn Institute of Nuclear Physics, Lomonosov Moscow State University, Moscow 119991, Russia*
[3] *A.M. Prokhorov General Physics Institute, Russian Academy of Science, Moscow 119991, Russia*
[4] *North-Eastern Federal University, Yakutsk 677007, Russia*
[*] *Corresponding author: oe.sarmanova@physics.msu.ru*

This supplement published with The Optical Society on 13 Sepember 2021 by The Authors under the terms of the Creative Commons Attribution 4.0 License in the format provided by the authors and unedited. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

Supplement DOI: https://doi.org/10.6084/m9.figshare.15384747

Parent Article DOI: https://doi.org/10.1364/AO.434984

Machine learning algorithms to control concentrations of carbon nanocomplexes in biological medium via optical absorption spectroscopy: how to choose and what to expect?

Olga E. Sarmanova*[a], Kirill A. Laptinskiy[b], Sergey A. Burikov[a,b], Maria Yu. Khmeleva[a,b], Anna A. Fedyanina[a], Alexandra E. Tomskaya[c,d], Alexander O. Efitorov[b], Sergey A. Dolenko[b], Tatiana A. Dolenko[a,b]

[a]Department of Physics, Lomonosov Moscow State University, Moscow, Russia 119991;
[b]Skobeltsyn Institute of Nuclear Physics, Lomonosov Moscow State University, Moscow, Russia 119991;
[c]A.M. Prokhorov General Physics Institute, RAS, Moscow, Russia, 119991;
[d]North-Eastern Federal University, Yakutsk, Russia, 677007;

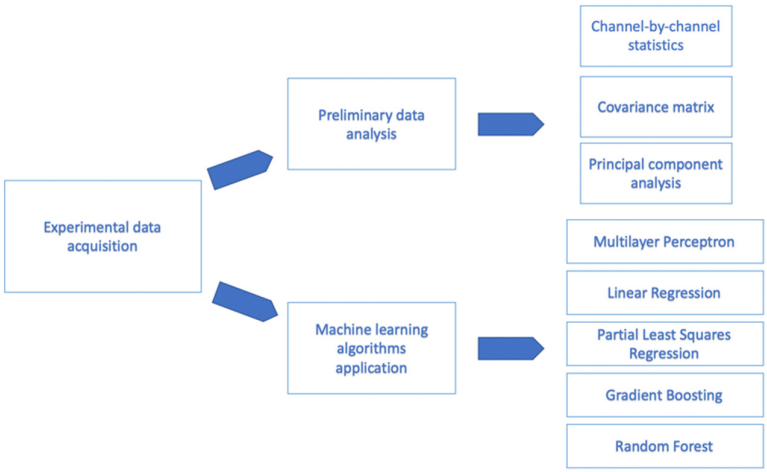

Fig. SI0. An explanatory flowchart of the study.

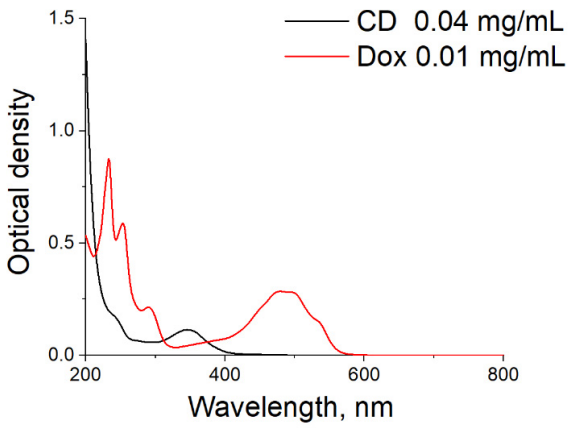

Fig. SI1. Optical absorption spectra of CD aqueous suspension and Dox aqueous solution.

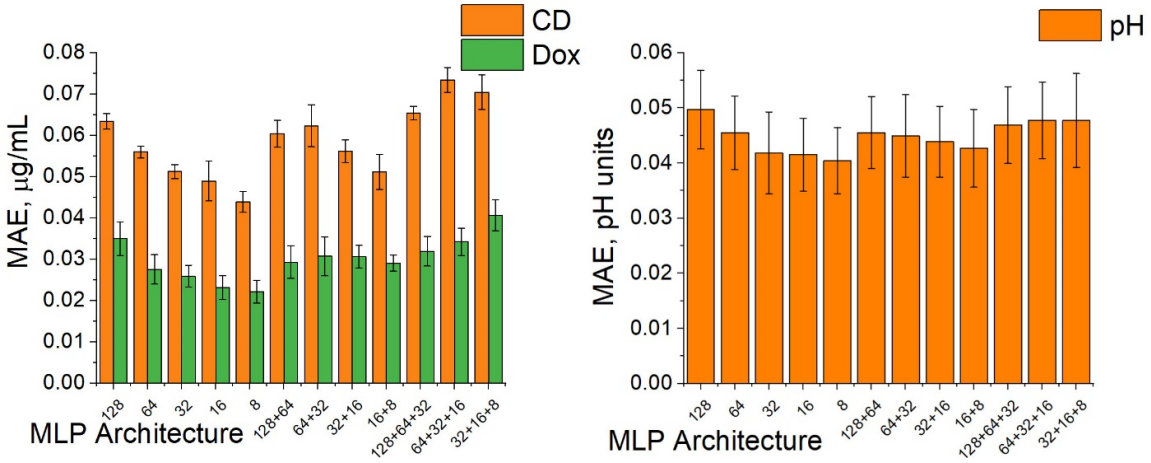

Fig. SI2. Average absolute error of the determination of A) the concentrations of the suspension components and B) the pH value on the test set.

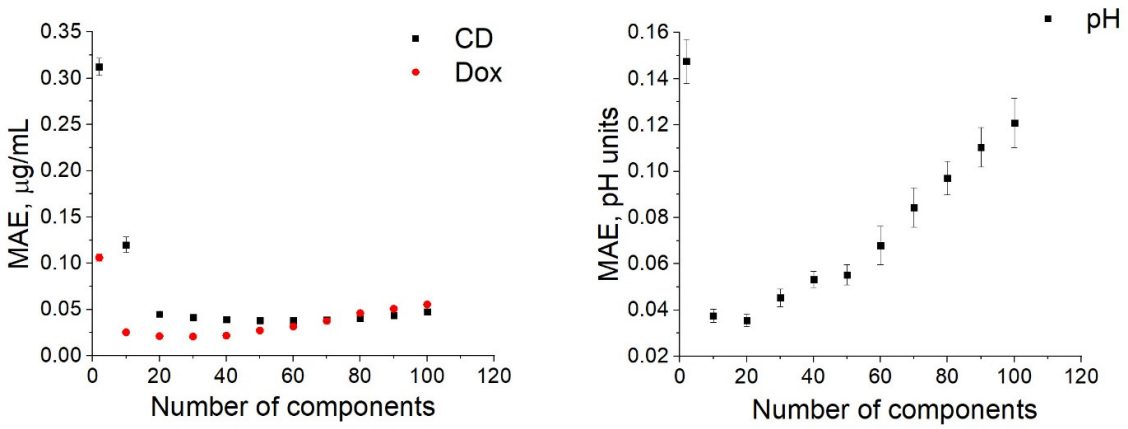

Fig. SI3. The quality of the solution of the inverse problem by the PLS method depending on the number of used principal components of the problem.

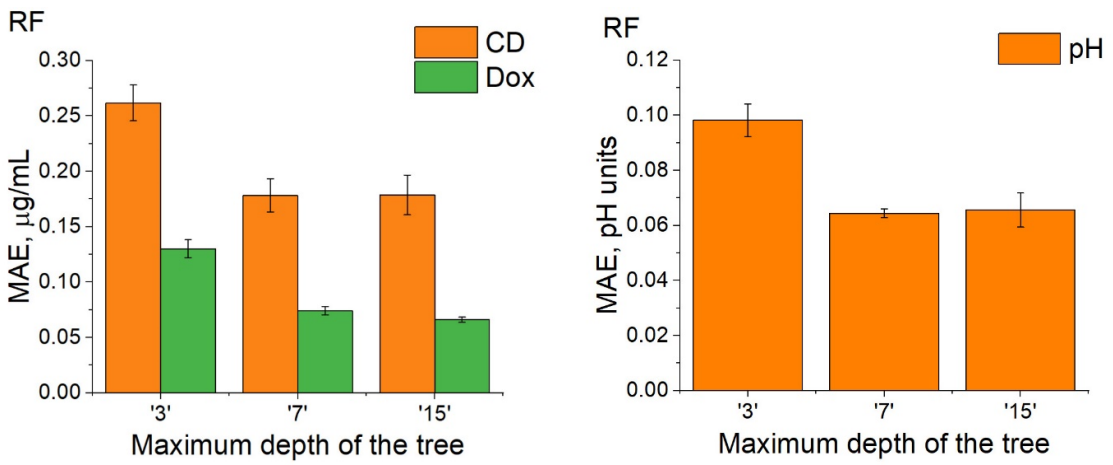

Fig. SI4. The quality of the solution of the inverse problem by the RF method depending on the maximum depth of trees.

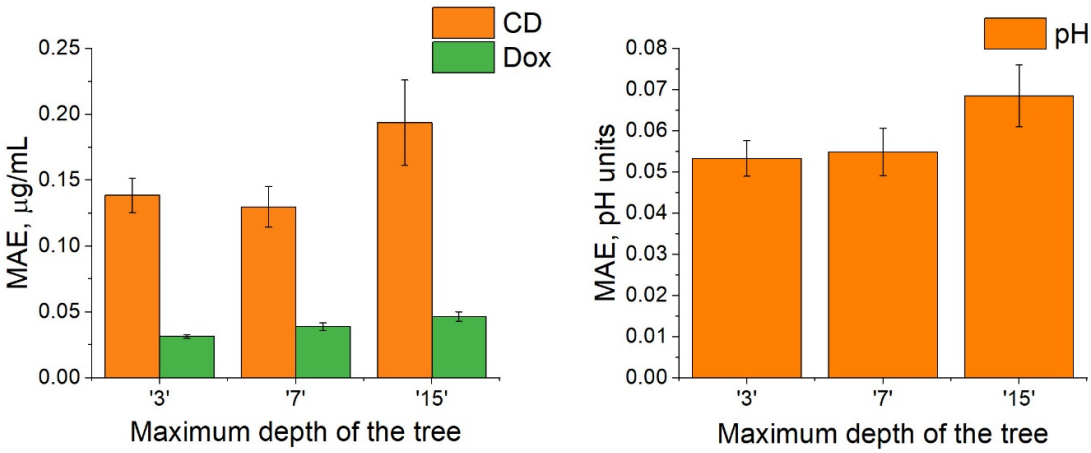

Fig. SI5. The quality of the inverse problem solution with GB method depending on the maximum depth of the trees.

The authors used the following pseudocode to create and fit ANN:

```
def best_epoch_finder(path2weights):
    best_epoch = max([int(x.split('-')[2]) for x in os.listdir(path2weights) if x[len(x)-4:len(x)] == 'hdf5'])
    out_filename = [np.sort(os.listdir(path2weights))[i] for i in range(0, len(os.listdir(path2weights))) if
int(re.split('-', np.sort(os.listdir(path2weights))[i])[2]) == best_epoch][0]

    #del all garbage
    dustbin_list = os.listdir(path2weights)
    dustbin_list.remove(out_filename)

    [os.remove(path2weights + i) for i in dustbin_list]
    #del done!

    return(best_epoch, out_filename)

# Define X, Y
X = data[:, :-3]
Y = data[:, -3:]

# Split data into train, validation and test sets
X_trn, X_30, Y_trn, Y_30 = train_test_split(X, Y, test_size=0.3, random_state=splt[0])
X_vld, X_tst, Y_vld, Y_tst = train_test_split(X_30, Y_30, test_size = 0.3333, random_state=splt[1])

#Data Scaling

scalerX = preprocessing.StandardScaler()
scalerY = preprocessing.MinMaxScaler()

x_trn = scalerX.fit_transform(X_trn)
x_vld = scalerX.transform(X_vld)
x_tst = scalerX.transform(X_tst)

y_trn = scalerY.fit_transform(Y_trn)
y_vld = scalerY.transform(Y_vld)
```

```
y_tst = scalerY.transform(Y_tst)

#Model Creation (three layer perceptron 128+64+32)
model = Sequential()
model.add(Dense(128, input_dim=x_trn.shape[1], kernel_initializer='uniform', activation='sigmoid'))
model.add(Dense(64, kernel_initializer='uniform', activation='sigmoid'))
model.add(Dense(32, kernel_initializer='uniform', activation='sigmoid'))
model.add(Dense(y_trn.shape[1], kernel_initializer='uniform', activation='linear', use_bias=True))

path2model = loc_path+str(i)+ '/'
model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mae'])

filename = 'weights-improvement-{epoch:02d}-{val_loss:.5f}.hdf5'
checkpoint = ModelCheckpoint(loc_path+str(i)+ '/'+filename, monitor='val_loss', verbose=1,
save_best_only=True, mode='min')
callbacks_list = [checkpoint]

#Fitting
tst_stop = 500
for epoch_step in range(0, 100000, tst_stop):
    if (epoch_step!=0):
        last_best_epoch, best_weights_file = best_epoch_finder(path2model)
        print(last_best_epoch, epoch_step)
        if ((tst_stop+int(last_best_epoch))>epoch_step):
            model.load_weights(path2model+best_weights_file)
            model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mae'])
            model.fit(x_trn, y_trn, epochs=epoch_step+tst_stop, batch_size=30,  verbose=0,
callbacks=callbacks_list, shuffle=True,validation_data = (x_vld, y_vld), initial_epoch=epoch_step)
        else:
            continue
    if (epoch_step==0):
        model.fit(x_trn, y_trn, epochs=epoch_step+tst_stop, batch_size=30,  verbose=0,
callbacks=callbacks_list, shuffle=True,validation_data = (x_vld, y_vld))

best_epoch_finder(path2model)

model_json = model.to_json()
with open(path2model + 'HM_model'+'.json', 'w') as json_file: json_file.write(model_json)

y_predicted_trn = scalerY.inverse_transform(model.predict(x_trn))
y_predicted_trn[y_predicted_trn<0]=0
y_predicted_vld = scalerY.inverse_transform(model.predict(x_vld))
y_predicted_vld[y_predicted_vld<0]=0
y_predicted_tst = scalerY.inverse_transform(model.predict(x_tst))
y_predicted_tst[y_predicted_tst<0]=0
```

The authors used the following pseudocode to create and fit the models (LinearRegression, PLSRegression, RandomForestRegressor, GradientBoostingRegressor):

```
# Define X, Y
X = data[:,:-3]
Y = data[:, -3:]

# Split data into train and test sets
X_trn, X_tst, Y_trn, Y_tst = train_test_split(X, Y, test_size=0.2, random_state=i)
```

```
#Data Scaling
scalerX = preprocessing.StandardScaler()
scalerY = preprocessing.MinMaxScaler()

x_trn = scalerX.fit_transform(X_trn)
x_tst = scalerX.transform(X_tst)
y_trn = scalerY.fit_transform(Y_trn)
y_tst = scalerY.transform(Y_tst)

Model = one of {LinearRegression, PLSRegression, RandomForestRegressor, GradientBoostingRegressor}

for parameter in parameters:
    if Model == GradientBoostingRegressor then:
        for each output:
            reg = Model(parameter)
            reg.fit(x_trn, y_trn)

            y_predicted_trn = scalerY.inverse_transform(reg.predict(x_trn))
            y_predicted_trn[y_predicted_trn<0]=0
            y_predicted_tst = scalerY.inverse_transform(reg.predict(x_tst))
            y_predicted_tst[y_predicted_tst<0]=0
    else:
        reg = Model(parameter)
        reg.fit(x_trn, y_trn)

        y_predicted_trn = scalerY.inverse_transform(reg.predict(x_trn))
        y_predicted_trn[y_predicted_trn<0]=0
        y_predicted_tst = scalerY.inverse_transform(reg.predict(x_tst))
        y_predicted_tst[y_predicted_tst<0]=0
```