AI 4 Science Discovery Network+

Group: 14
Challenge: Defect Detection in Graphene Sheets
AI4SD ML Summer School Report
20-24th June 2022

Project Team: Peng Bao (University of Liverpool), Jack Macklin (University of Bath), Masood Gheasi (University of Southampton)

Report Date: 30/06/2022

Group: 14
Challenge: Defect Detection in Graphene Sheets
AI4SD-SummerSchool-Series:Report-7
Report Date: 30/06/2022
DOI: 10.5258/SOTON/AI3SD0250
Published by University of Southampton

Network: Artificial Intelligence and Augmented Intelligence for Automated Investigations for Scientific Discovery
This Network+ is EPSRC Funded under Grant No: EP/S000356/1

Principal Investigator: *Professor Jeremy Frey*
Co-Investigator: *Professor Mahesan Niranjan*
Network+ Coordinator: *Dr Samantha Kanza*

Contents

1 Project Details

Group Number	14
Challenge Name	Defect Detection in Graphene Sheets
Project Dates	20th June-30th June 2022

2 Project Team

Name and Title	Dr Peng Bao
Employer name / University Department Name	Department of Chemistry, University of Liverpool
Work Email	peng.bao@liverpool.ac.uk
Website Link	https://www.liverpool.ac.uk/chemistry/staff/peng-bao/

Name and Title	Jack Macklin
Employer name / University Department Name	Department of Chemistry, University of Bath
Work Email	jpm93@bath.ac.uk
Website Link	https://researchportal.bath.ac.uk/en/persons/jack-macklin

Name and Title	Dr Masood Gheasi
Employer name / University Department Name	Economic, Social and Political Sciences, University of Southampton
Work Email	masood.gheasi@soton.ac.uk
Website Link)	https://www.southampton.ac.uk/socsci/about/staff/mg1m21.page

2.1 Challenge Description

Try to use machine learning method to accurately classify different types of TEM images of atom lattice of monolayer Graphenes, see Figure 1.

3 Lay Summary

This is a typical classification problem. Many classification models could be used to solve this kind of problem. Here we choose the Decision tree classifier model [1]. A decision tree model is a useful tool for identifying non-linear problems. The advantages of this model are that the results are easily interpretable and do not need high statistical knowledge and it has an

accuracy test. The accuracy test is disputed among researchers and needs extra tools to confirm the result. There are two main types of decision tree models; categorical variable decision tree and continuous variable decision tree. In our application, we applied the categorical variable decision tree model as our variable of interest had a distribution of 0 and 1. A comparison of the different classifiers is shown in Figure 2 [2].

To classify the perfect TEM images of graphene and defective TEM images of graphene, as shown in Figure 1, we need to feed the classifier model with enough amount of training data. Then we can also use a small portion of test data to verify our model.

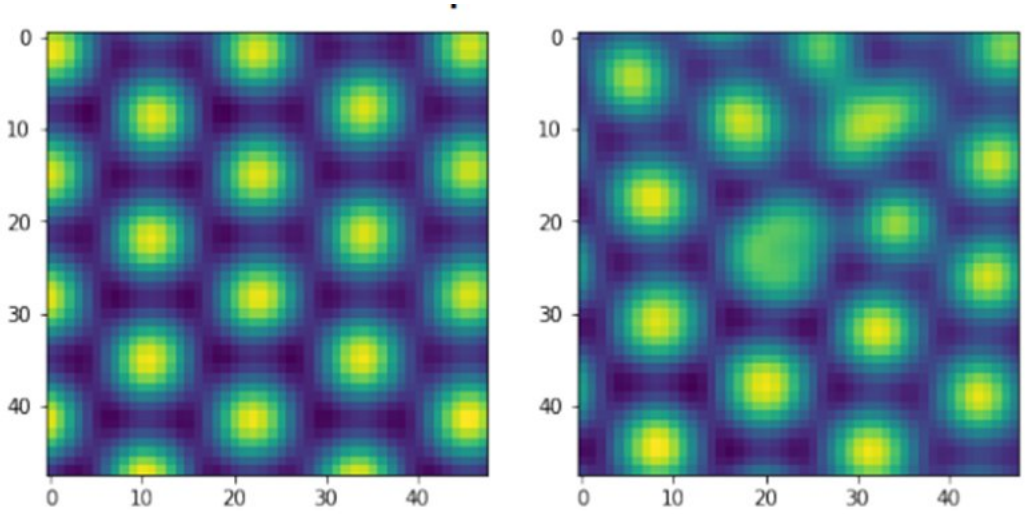

Figure 1: The perfect and defective TEM images of graphene in the training data set.

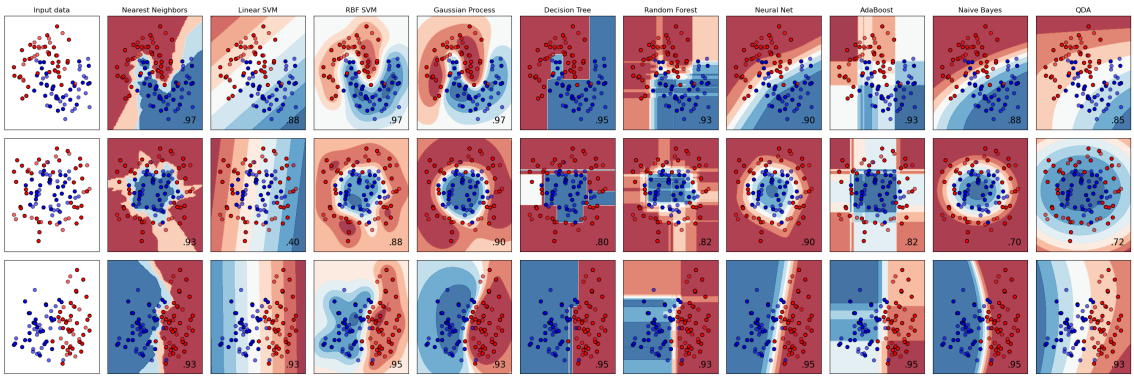

Figure 2: Classifier comparison.

4 Methodology

- Read images using np.load;
- Convert images into arrays; construct labels for these images;
- Put perfect/defect images together randomly;
- Convert images and their labels into Dataframe;
- Use train-test-split to generate training images and test images;

- Train DecisionTreeClassifier model using training images;
- Test trained model using test images;
- Evaluate the accuracy of the model using different scores;
- Test our model with bigger, cut up, TEM images, pull out the defective overall images.

5 Results

- We have successfully used the Sklean.tree.Decisiontreeclassifier Model to classify the two types of TEM images of graphenes. Some example images from the test data set are shown in Figure 3.
- We have evaluated the success of this method using the accuracy score, roc auc score, and F1 score. The comparison of these scores are shown in Figure 4.
- We have divided the original TEM images of graphene (256 x 256pixels) into 16 smaller images (48x48 pixels), and used our trained model to find the images with defects, then we convert these smaller images to images with the original size.
- We have improved our result using the following protocol: train the model with random allocated train/test data sets, and save the best-trained model, then test this model using random data sets of test images. We found the three scores could be 100per for any data sets in the perfect/defect images.

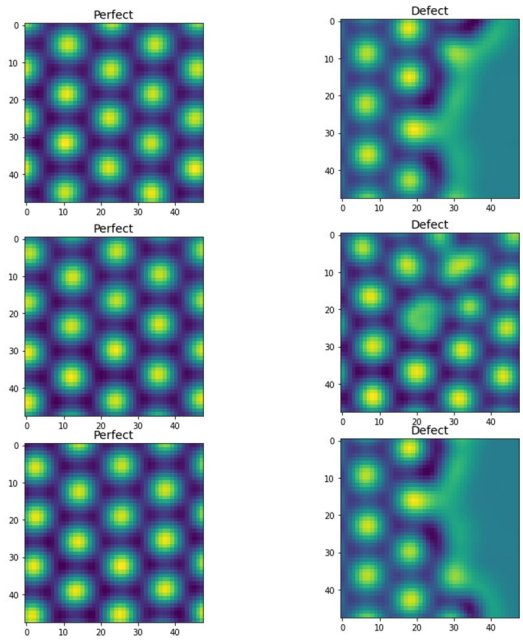

Figure 3: The classified perfect/defect images of graphene.

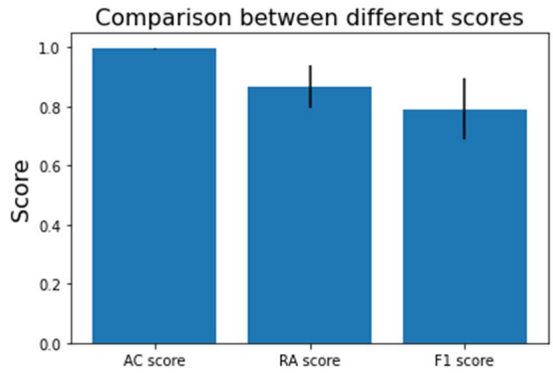

Figure 4: Comparison between different scores.

6 Conclusions & Future Work

We have successfully solved the problem using the Decisiontreeclassifier model, i.e. we can find defects in 256 x 256 pixels images. The accuracy score, f1 score, and roc auc score are compared for the test data set. Future work will include further increasing the accuracy by increasing the number of images with defects, optimizing the test size, and saving the best training model for later use, instead of running the model every time using random training data set. It will also include increasing speed by compressing the 2D training image (48 x 48 pixels) to 1D line (48 pixels) and doing fft on these 1D arrays before putting them into the model. We also could try other models such as Random Forrest, and Nearest neighbors to see if the results could be better.

7 Outputs, Data & Software Links

See attached Python code as below.

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


#!/usr/bin/env python
# coding: utf-8


# In[2]:


import numpy as np
import matplotlib.pyplot as plt
import random
import pandas as pd


# ## The data<br>
# <br>
# The data is from electron microscopy - images of grpahene sheets. The typical image is $(256, 256)$ but we would like to take small patches of this and identify - is there a defect in the patch.<br>
# <br>
# Lets go through a few of those things. Below we will look at <br>
# <br>
# * A full image<br>
# * A patch of an image with no defects<br>
# * A patch with a defect

# ### Full image

# In[3]:


graphene = np.load('full-stack.npy') ## This has 180 different microgrpahs each (256, 256)
sample = np.squeeze(graphene[23])

print(type(graphene))


# In[4]:


plt.imshow(sample)
plt.colorbar();


# #### Description<br>
# The blue (high electron density) correspnds to atoms and the green correspnds to background.<br>
# <br>
```

```
# You can see that in the middle of the sheet most of it is a pretty regular
array of hexagons, this is what we exepct for the perfect lattice. You will also
hopefully see some spots where the hexagon is broken. This many of these are the
result of missing atom defects. <br>
# <br>
# Notice the edges look a bit weird too. This is standard, and we will generally
ignore the edges when analysing.<br>
# <br>
#

# Cutting up the original full images to make patches that we can test on
#
# We also must remove some data from around the edges. The patches are 48 * 48.
48 * 4 = 192. Therefore need to remove (256-192)/2 = 32 data points from top,
right, left and bottom of the data.

# In[5]:


# Dropping the unwanted data
truncated_data=[]

for i in range(np.shape(graphene)[0]):
    #edge_removed_sample = np.squeeze(graphene[i])
    edge_removed_sample = graphene[i]
    edge_removed_sample=edge_removed_sample[32:-32,32:-32]
    truncated_data.append(edge_removed_sample)

# Cutting up the image
cut_images=[]
for i in range(np.shape(truncated_data)[0]):
     for n in range(4):
        for m in range(4):
            cut_images.append(truncated_data[i][0+n*48:48+n*48,0+m*48:48+m*48])

plt.imshow(cut_images[0])


# In[6]:


plt.imshow(truncated_data[0])


# ### Perfect patch<br>
# <br>
# There is a training dataset of perfect patches

# In[7]:


pp = np.load('./perfect_patches.npy')
perfect_sample = pp[17]
print(pp.shape)
```

```
# In[11]:

# In[8]:


plt.imshow(perfect_sample);


# ### Defect patch<br>
# <br>
# There is also a set of defective patches. Note - these are for testing only, not for training. I guess we could imagine training a classifier on these and the perfect patches - but that would be no fun :)

# In[14]:

# In[9]:


dp = np.load('./defect_patches.npy')
defect_sample = dp[0]
print(dp.shape)


# In[15]:

# In[10]:


plt.imshow(defect_sample);


# Get patches perfect and defect, and put them randomly in a list, with the ground truth saved

# In[16]:

# In[11]:


patches = np.concatenate([pp, dp])
gt = np.concatenate([np.zeros(len(pp)), np.ones(len(dp))])
xy = list(zip(patches, gt)) # put labels with data in (data, label) tuple
random.shuffle(xy) # shuffle the tuples


# In[17]:

# In[12]:


patches, gt = list(zip(*xy)) # unzip the tuples to get patches and ground truths
```

```
as vectors


# Have fun :-)

# From here on is Peng's code

# In[13]:


gt=np.array(gt)
patches=np.array(patches)
cut_images=np.array(cut_images)


# In[14]:


patches=patches.flatten()
cut_images=cut_images.flatten()


# In[15]:


patches=patches.reshape(2311,2304)
cut_images=cut_images.reshape(2880,2304)


# In[16]:


print(patches.shape)
print(cut_images.shape)


# In[17]:


patches=np.array(patches)


# In[19]:


import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# In[20]:
```

```
gt_f=pd.DataFrame(gt)
patches_f=pd.DataFrame(patches)
cut_images_f=pd.DataFrame(cut_images)


# In[21]:


X_train,X_test,y_train,y_test=train_test_split(patches_f,gt_f,test_size=0.2)


# In[23]:


model=DecisionTreeClassifier()
model.fit(X_train,y_train)
predictions=model.predict(X_test)

predictions_new=model.predict(cut_images_f)


# In[24]:


for i in y_test:
 if i==1:
    print(y_test[i],predictions[i])

accuracy_score(y_test,predictions)
print(accuracy_score)


# In[25]:


X_test_r=np.array(X_test)


# In[31]:


for i in range(len(y_test)):

    if predictions[i]==1:
       #print(predictions[i])
       x=X_test_r[i,:]
       print(x.shape)
       x=np.reshape(x,(48,48))
       print(x.shape)
       #plt.imshow(x);
       plt.imshow(x)
       plt.legend("True")
       plt.colorbar()
       plt.show()
```

```
       #plt.colorbar();


# In[ ]:


for i in range(len(y_test)):

    if predictions[i]==0:
       #print(predictions[i])
       x=X_test_r[i,:]
       print(x.shape)
       x=np.reshape(x,(48,48))
       print(x.shape)
       #plt.imshow(x);
       plt.imshow(x)
       plt.show()
       #plt.colorbar();


# In[34]:


for i in range(len(cut_images)):

    if predictions_new[i]==1:
       #print(predictions[i])
       x=cut_images[i,:]
       #print(x.shape)
       x=np.reshape(x,(48,48))
       #print(x.shape)
       #plt.imshow(x);
       #plt.imshow(x)
       #plt.legend("false")
       #plt.show()
       #plt.colorbar();


for i in range(len(graphene)):
    x=np.count_nonzero(predictions_new[i*16:i*16+16])
    if x > 0:
        print(i,"defective sample")
        #plt.imshow(graphene[i])
        plt.imshow(truncated_data[i])
        plt.colorbar()
        plt.show()
    #else:
        #pass




# In[28]:
```

```
RA_score=[]
AC_score=[]
F1_score=[]
for i in range(50):
    X_train,X_test,y_train,y_test=train_test_split(patches_f,gt_f,test_size=0.8)
    predictions=model.predict(X_test)
    predictions=np.array(predictions)

    RA_score.append(roc_auc_score(y_test,predictions))
    AC_score.append(accuracy_score(y_test,predictions))
    F1_score.append(f1_score(y_test,predictions))
    print(RA_score[i],AC_score[i],F1_score[i])




# In[ ]:


from pandas import Series, DataFrame

data = [np.mean(AC_score),np.mean(RA_score),np.mean(F1_score)]
Y_error=[np.std(AC_score),np.std(RA_score),np.std(F1_score)]
plt.bar(["AC score","RA score","F1 score"], data,yerr=Y_error)
plt.title('Comparison between different scores', fontsize=16)
plt.ylabel("Score",fontsize=16)
plt.show()
```

References

[1] https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

[2] https://www.youtube.com/watch?v=7eh4d6sabA0

List of Figures