Ni_ML

November 12, 2018

```
In [1]: import numpy as np
        from sklearn import svm
        from Ni_ML import *
        %load_ext autoreload
        %autoreload 2
```

0.1 list all files present in the currect directory

1 atom_property.c: C code to read input atomic configuration and create neighor list for atoms **2 atom_property.h:** header file for atom_property.c **3 createfeature.c:** C code to feature feature vector for atoms using their atomic coordinates **4 createfeature.h:** header file for createfeature.c **5 Makefile:** create executabe c_feature so generate feature vector from atomic coordinates **6 Ni_train.xyz:** Atomic coordinate of FCC Nickel indentation simulation in xyz format (use it for training) **7 Ni_test1.xyz:** Atomic coordinate of FCC Nickel indentation simulation in xyz format (use it for test) **8 Ni_test2.xyz:** Atomic coordinate of FCC Nickel indentation simulation in xyz format (use it for test) **9 feature:** containes converted atomic data into 18 dimension feature vector format for each atoms. It contains following files. a) train_XX.py: 18 dimension feature vector represntation for each atom, b) train_YY.npy : labels for each atom, c) train_pos.npy actual atomic coordinates for each atoms. It also contains files test_XX.npy,test_YY.npy and test_pos.npy (use it for testing)

```
In [2]: ###Type !ls to list all the files
        !ls
```

```
ML_Ni.pptx        Ni_ML.pyc         atom_property.o  output.xyz
ML_Ni_V_1.0.pptx  Ni_test1.xyz      c_feature        readinput.py
ML_Ni_V_2.0.pptx  Ni_test2.xyz      createfeature.c  readinput.pyc
Makefile          Ni_train.xyz      createfeature.h  test.py
Ni_ML.ipynb       atom_property.c   createfeature.o
Ni_ML.py          atom_property.h   feature
```

```
In [3]: # create executable to generate featue vector using createfeature.c
        # it create an executable called c_feature
        !rm -f feature/*
        !make clean
        !make
```

```
rm -f *.o c_feature
gcc  -c -Wall -std=c99 createfeature.c
gcc  -c -Wall -std=c99 atom_property.c
gcc -o c_feature createfeature.o atom_property.o -lm
```

0.2 To create feature vector from atomic data run this command

./c_feature input_file output_file where, input_file: name of the input file containing atomic coordinates (EX: Ni_train.xyz) output_file: name of the output file containing feature vector (EX: feature/train.txt)

```
In [4]: #create training data using atomic coordinates
        !./c_feature "Ni_train.xyz" "feature/train.txt"
        print("File created inside feature folder:" )
        !ls feature
```

```
Total number of atoms     114376
Box size   101.481003   130.103226   101.481003
File created inside feature folder:
train.txt
```

0.3 Create a linear classifer using the training data

Step 1 : Extract feature vector, lables and cooridnates from train.txt. Creates three files: a) train_XX.npy: 18 dimension feature vector represntation for each atom. b) train_YY.npy : labels for each atom. c) train_pos.npy actual atomic coordinates for each atoms. **Step 2** : Build a classifer using train_XX.npy and train_YY.npy

```
In [5]: inputfile='feature/train.txt'
        outfile='feature/train'
        #extract feature vector, lables and cooridnates from the input file.
        create_input_data(inputfile,outfile)
        #build classifier
        Ni_model = build_classifier('feature/train_XX.npy','feature/train_YY.npy')
        Ni_model.train()
```

```
('Natoms  and number of features per atom', 114376, 17)
('Atom_frac', [109786, 1367, 3223])
('Atom_select', array([0.1, 2.1, 1.1]))
('Number of training examples: ', 15663)
('training error: ', 3.5561514396986516)
('training accuracy: ', 96.44384856030135)
```

0.4 check the accuracy of the model on a test data set

Step 1: Convert atomic coordinates for test data Ni_test1.xyz into feature vector called test_1.txt
Step 2: Extract feature vector, lables and cooridnates from test_1.txt. It creates three files inside

feature folder : test_1_XX.npy, test_1_YY.npy, test_1_pos.npy **Step 3:** Precit lables of the test data using the build classifer.

```
In [6]: #Step 1: Convert atomic coordinates for test data Ni_test1.xyz into feature
        !./c_feature "Ni_test1.xyz" "feature/test_1.txt"
        #Step 2: extract feature vector, lables and cooridnates from the input file
        inputfile='feature/test_1.txt'
        outfile='feature/test_1'
        create_input_data(inputfile,outfile)
        test_X = np.load('feature/test_1_XX.npy')
        test_Y = np.load('feature/test_1_YY.npy')
        test_pos = np.load('feature/test_1_pos.npy')
        test_Y = test_Y.ravel()
        #Step 3: Precit lables of the test data using the build classier
        test_1_predict = Ni_model.predict(test_X)
        test_1_accuracy = Ni_model.accuracy(test_Y,test_1_predict)
        print("Test error: ",np.mean(test_1_accuracy)*100.0)
        print("Test accuracy: ",100.00-np.mean(test_1_accuracy)*100.0)
```

```
Total number of atoms     114376
Box size   101.521004   130.103226   101.521004
('Natoms  and number of features per atom', 114376, 17)
('Test error: ', 1.012450164370149)
('Test accuracy: ', 98.98754983562985)
```

0.4.1 Visulize the predicted lables on test data using ovito

writexyz creates a xyz file containing the true and predicted labels in a file called output.xyz

```
In [7]: #writexyz creates a xyz file containing the true
        #and predicted labels in a file called output.xyz
        writexyz(len(test_Y),test_pos,test_1_predict,test_Y)
        !ls
```

```
ML_Ni.pptx        Ni_ML.pyc         atom_property.o  output.xyz
ML_Ni_V_1.0.pptx  Ni_test1.xyz      c_feature        readinput.py
ML_Ni_V_2.0.pptx  Ni_test2.xyz      createfeature.c  readinput.pyc
Makefile          Ni_train.xyz      createfeature.h  test.py
Ni_ML.ipynb       atom_property.c   createfeature.o
Ni_ML.py          atom_property.h   feature
```

0.4.2 check the accuracy of the model on a another test data set

Step 1: Convert atomic coordinates for test data Ni_test1.xyz into feature vector called test_2.txt **Step 2:** Extract feature vector, lables and cooridnates from test_2.txt. It creates three files inside feature folder : test_2_XX.npy, test_2_YY.npy, test_2_pos.npy **Step 3:** Precit lables of the test data using the build classifer.

In [8]:
```
#Step 1: Convert atomic coordinates for test data Ni_test1.xyz into feature
!./c_feature "Ni_test2.xyz" "feature/test_2.txt"
#Step 2: extract feature vector, lables and cooridnates from the input file
inputfile='feature/test_2.txt'
outfile='feature/test_2'
create_input_data(inputfile,outfile)
test_X = np.load('feature/test_2_XX.npy')
test_Y = np.load('feature/test_2_YY.npy')
test_pos = np.load('feature/test_2_pos.npy')
test_Y = test_Y.ravel()
#Step 3: Precit lables of the test data using the build classier
test_2_predict = Ni_model.predict(test_X)
test_2_accuracy = Ni_model.accuracy(test_Y,test_1_predict)
print("test 2 error: ",np.mean(test_1_accuracy)*100.0)
print("test 2 accuracy: ",100.00-np.mean(test_1_accuracy)*100.0)
```

```
Total number of atoms     114376
Box size   101.481003   130.103226   101.481003
('Natoms  and number of features per atom', 114376, 17)
('test 2 error: ', 1.012450164370149)
('test 2 accuracy: ', 98.98754983562985)
```