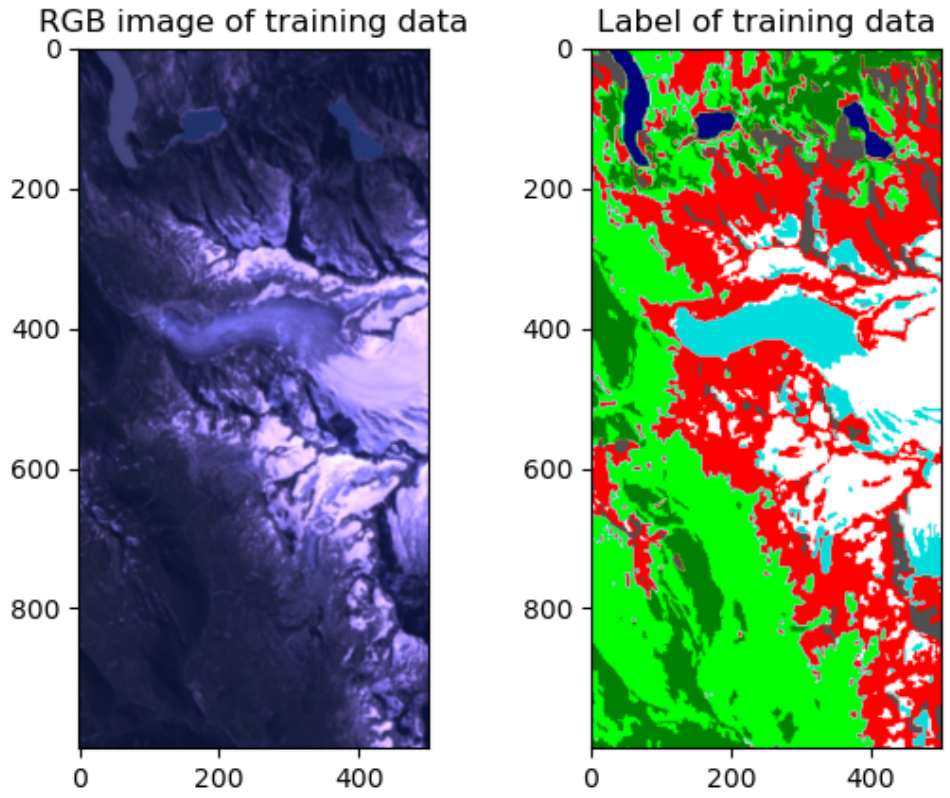
Introduction

In this exercise, we will use the maximum likelihood method to perform a Gaussian Naive Bayes classification. Two clips of Landsat satellite images with 7 different spectral channels are given. The first clip is used as training data and the second clip is used as test data. The goal is to classify the pixels of the test data into one of the 7 classes. The classes are: water, forest, vegetation, ice, snow, rock, shadow.

Training data and test data

Firstly we want to visualize the training data and the ground truth label of the training data.



We can see that is hard to distinguish the classes by only looking at the spectral channels. However, we can see that the water and the shadow classes are quite different from the other classes. The water class has a very low reflectance in all spectral channels. The shadow class has a very low reflectance in the visible spectral channels. The other classes have similar reflectance in the visible spectral channels. However, they have different reflectance in the infrared spectral channels. Therefore, we can expect that the infrared spectral channels are more important for the classification.

Maximum likelihood method

The maximum likelihood method is a method to estimate the parameters of a probability distribution. The probability distribution is assumed to be a Gaussian distribution. The probability density function of a Gaussian distribution is

$$p(\boldsymbol{x}|\omega_i) = \frac{1}{(2\pi)^{\frac{N}{2}}\sqrt{|\boldsymbol{K_{xxi}}|}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m_i})^T \boldsymbol{K_{xxi}}^{-1}(\boldsymbol{x}-\boldsymbol{m_i})\right\}$$

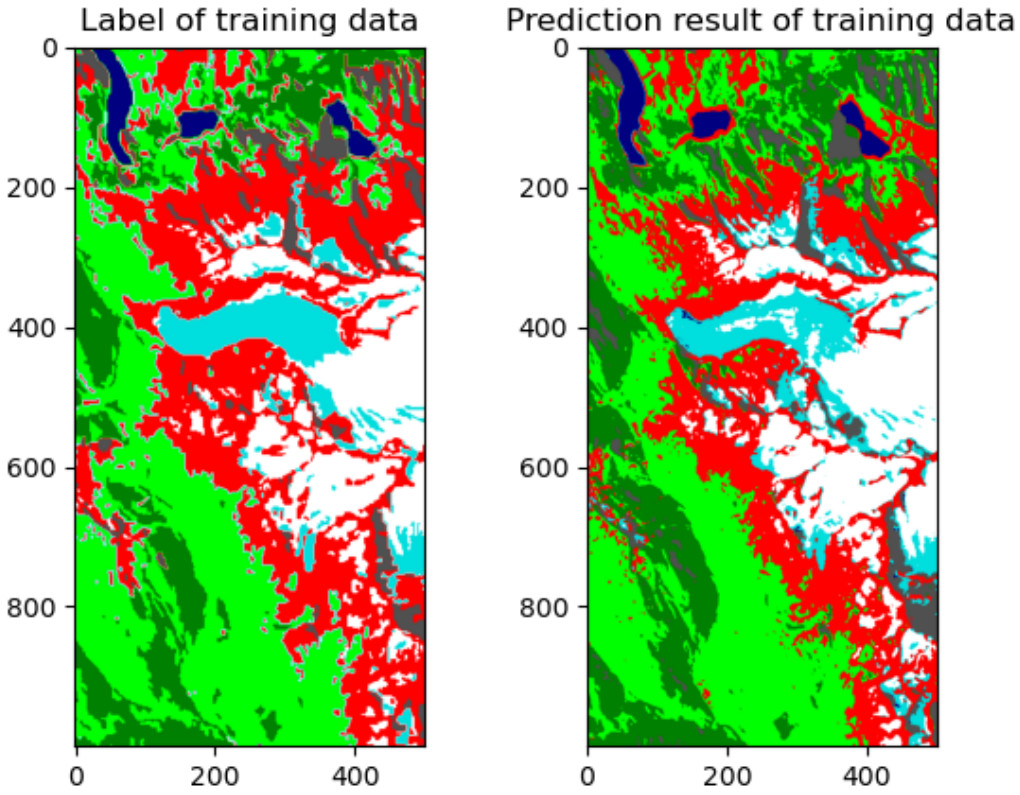
where $\boldsymbol{x}$ is a vector of the spectral channels, ω_i is the class, N is the number of spectral channels, $\boldsymbol{m_i}$ is the mean vector of the class ω_i and $\boldsymbol{K_{xxi}}$ is the covariance matrix of the class ω_i.

Training the classifier

Now we have to train our classifier with the training data. We use the maximum likelihood method to estimate the parameters of the Gaussian Naive Bayes classifier.

```
clf=GaussianNB()
clf.fit(X_train,y_train)
y_pred=clf.predict(X_train)
```

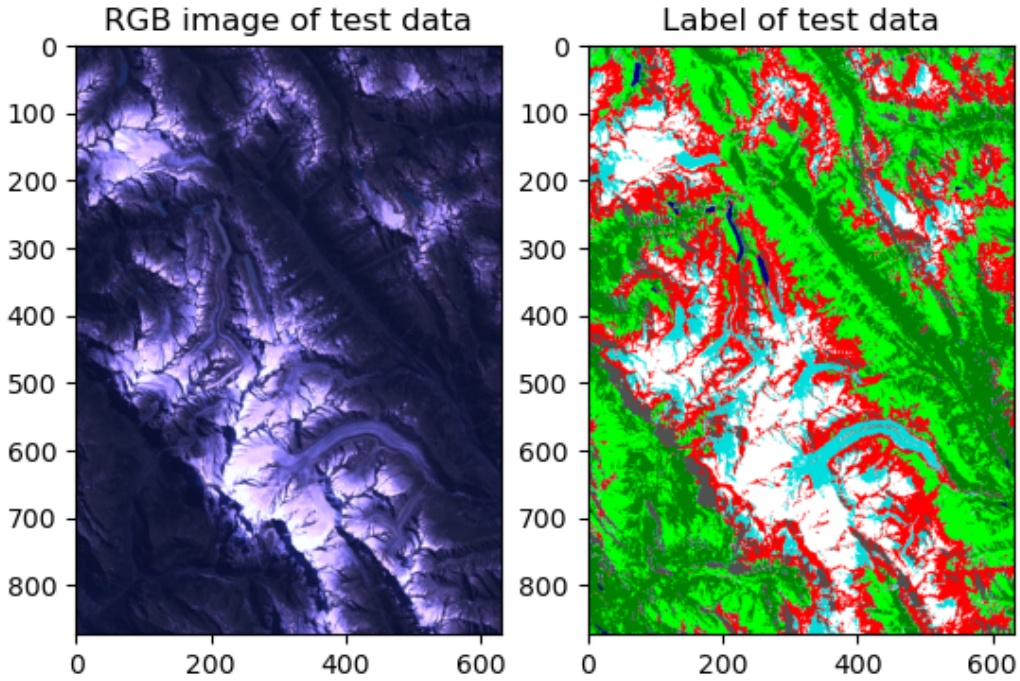
Now we want to visualize the classification result of the training data.



We can see that the water (dark blue) and rock (red) classes are classified very well. However, the other classes for example vegetation and forest, ice and snow are not classified very well. The reason is that the other classes have similar reflectance in the spectral channels. Therefore, it is hard to distinguish them.

Prediction on the test data

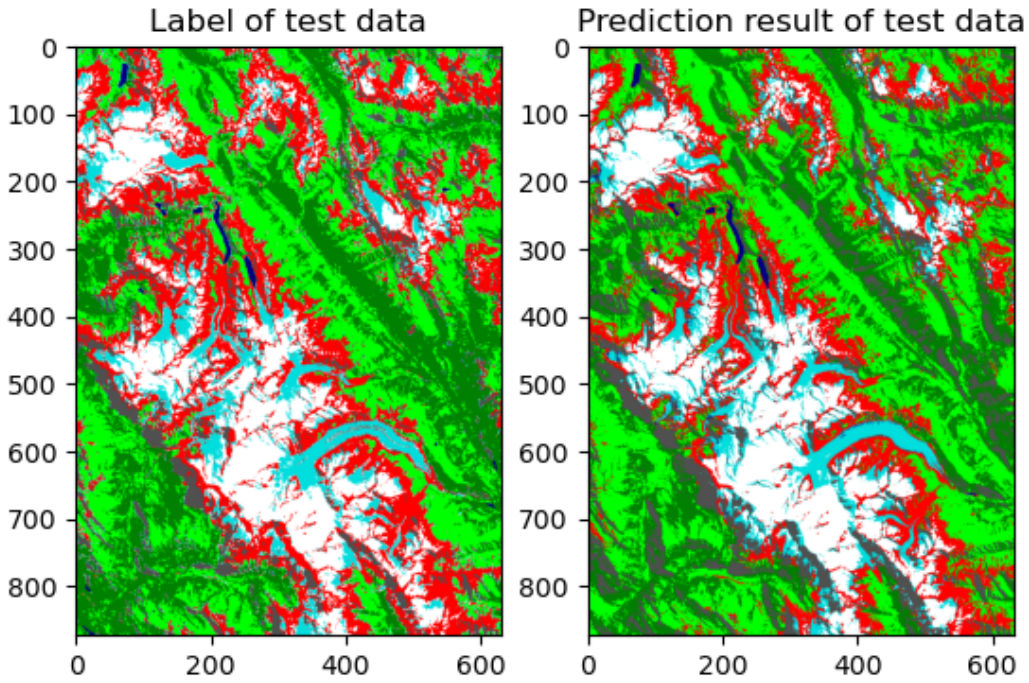
Now we want to visualize the test data and the ground truth label of the test data.



We can predict the classes of the test data with our trained classifier.

```
y_pred=clf.predict(X_test)
```

Now we want to visualize the classification result of the test data and compare to the ground truth label of the test data.



The same as the training data, the classifation of the water and rocks are better than other classes. In the test data, the snow and ice are slightly better classified than in the train image. However, the vegetation and forest are still not classified very well. The reason is that the vegetation and forest have similar reflectance in the spectral channels.

Confusion matrix

We can also observe the confusion matrix of the classification result. The confusion matrix is a matrix which shows the number of samples of the true class and the predicted class. The diagonal elements of the confusion matrix are the number of samples which are correctly classified. The off-diagonal elements of the confusion matrix are the number of samples which are incorrectly classified.

There are several accuracies which can be calculated from the confusion matrix. The user's accuracy is the number of correctly classified samples divided by the number of samples which are predicted to be in the class. The producer's accuracy is the number of correctly classified samples divided by the number of samples which are actually in the class. The overall accuracy is the number of correctly classified samples divided by the total number of samples.

Class		Reference data								Accuracy
		Water	Forest	Vegetation	Ice	Snow	Rock	Shadow	Total	User's
Predict	Water	**1481**	39	12	619	189	94	91	2525	0.586
	Forest	34	**81218**	11094	2083	1074	7305	1832	104640	0.776
	Vegetation	0	17381	**112113**	5362	2123	13203	288	150470	0.745
	Ice	3	20	23	**29524**	7611	6353	369	43903	0.672
	Snow	0	0	0	8515	**70736**	2507	0	81758	0.865
	Rock	6	94	5717	4447	5252	**86618**	12710	102495	0.845
	Shadow	401	14564	457	2253	1575	12710	**33353**	65313	0.511
	Total	1925	113316	129416	52803	88560	128790	36294	**551104**	
Accuracy	Producer's	0.769	0.717	0.866	0.559	0.799	0.672	0.919		**0.769**

From the user's accuracy, we can see that the snow and rock classes are classified very well. The forest, vegetation, ice classes are classified quite well. The accuracies of water and shadow are just over 50%. We achieve an overall accuracy of 76.9%.

Kappa coefficient

The kappa coefficient is a measure of the agreement between the predicted class and the true class. The kappa coefficient is defined as

$$\kappa = \frac{p_o - p_e}{1 - p_e} = \frac{n \sum_{i=1}^{m} n_{ii} - \sum_{i=1}^{m} n_{i+} n_{+i}}{n^2 - \sum_{i=1}^{m} n_{i+} n_{+i}}$$

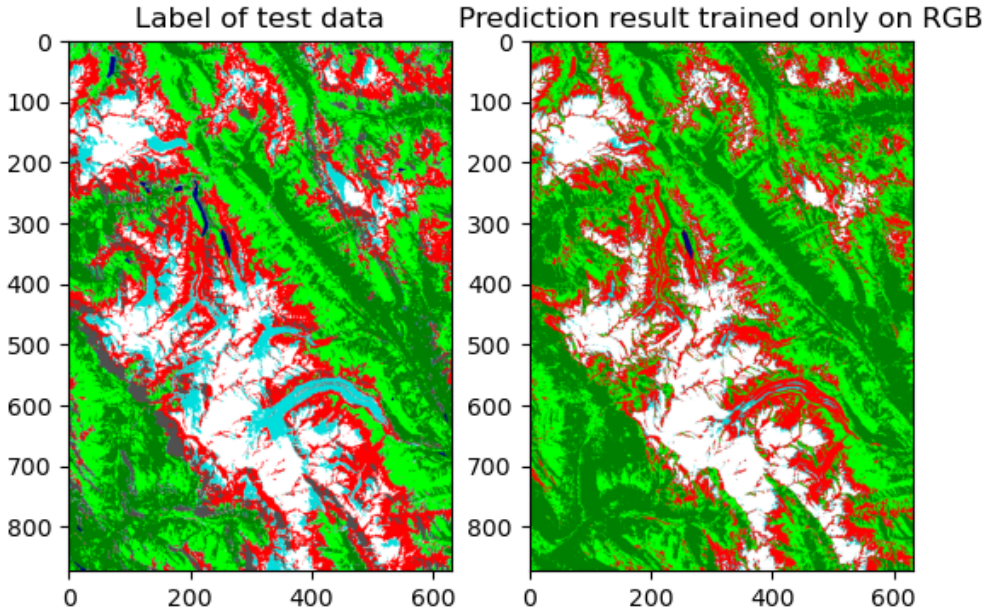
where p_o is the observed agreement, p_e is the expected agreement, n is the total number of samples, n_{ii} is the number of samples which are correctly classified, n_{i+} is the number of samples which are predicted to be in the class i, n_{+i} is the number of samples which are actually in the class i and m is the number of classes.

```
Kappa coefficient:  0.6967297666414108
```

The result of kappa coefficient is 0.6967. The kappa coefficient is between 0 and 1. The kappa coefficient is 1 if the predicted class is the same as the true class. The kappa coefficient is 0 if the predicted class is the same as the true class by chance. The kappa coefficient is negative if the predicted class is worse than the true class. Therefore, we can say that our classifier is much better than random guessing.

Train only on RGB channels

Now we want to train our classifier only on the RGB channels. We want to see if the infrared channels are important for the classification.



We can see that the classification result is worse than the classification result with all spectral channels. The reason is that the infrared channels are important for the classification. Other than the vegetation and forest classes, the other classes have large areas of misclassification in the classification result with only RGB channels. The reason is that the absorption ability of the infrared channels between the classes behave very differently.

Class		Reference data								Accuracy
		Water	Forest	Vegetation	Ice	Snow	Rock	Shadow	Total	User's
Predict	Water	**349**	345	112	0	0	1119	0	1925	0.181
	Forest	6	**100562**	12576	0	0	172	0	113316	0.887
	Vegetation	9	29693	**92876**	0	0	6838	0	129416	0.718
	Ice	5	2771	7041	**9826**	9467	23693	0	52803	0.186
	Snow	2	1338	3815	2868	**70922**	9615	0	88560	0.801
	Rock	1	6854	33914	5716	4546	**77753**	6	128790	0.604
	Shadow	0	30079	5670	0	0	524	**21**	36294	$5.786 \cdot 10^{-4}$
	Total	372	171642	156004	18410	84935	119714	27	**551104**	
Accuracy	Producer's	0.938	0.586	0.595	0.534	0.835	0.650	0.778		**0.639**

To check the accuracy of the classification result, we can observe the confusion matrix. The user's accuracy of the water, ice and shadow classes are very low. The overall accuracy is 63.9%. The calculated accuracies correspond to our previous visual assumption that

the result of training only with RGB channels is much worse than the classification result with all spectral channels.

Code

```
# -*- coding: utf-8 -*-
"""
Spectral channels
1 0.45-0.52  m , blue-green
2 0.52-0.60  m , green
3 0.63-0.69  m , red
4 0.76-0.90  m , near infrared
5 1.55-1.75  m , mid infrared
6 10.4-12.5  m  (60    60 m) Thermal channel
7 2.08-2.35  m , mid infrared

"""
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB


label_to_color = {
    1: [0,  0, 128], # water
    2: [0,  128, 0], # forest
    3: [ 0,  255,  0], # vegetation
    4: [0, 221, 221], # ice
    5: [255, 255, 255], # snow
    6: [255, 0, 0], # rock
    7: [80, 80, 80] # shadow
}

# convert one channel label image [nxm] to a given colormap
# resutling in a rgb image [nxmx3]
def label2rgb(img_label, label_to_color):
    h, w = img_label.shape
    img_rgb = np.zeros((h, w, 3), dtype=np.uint8)
    for gray, rgb in label_to_color.items():
        img_rgb[img_label == gray, :] = rgb
    return img_rgb

# Band 3 red, band 2 green, band 1 blue
# load image as rgb image
band3=plt.imread('./data/band3_train.tif')
band2=plt.imread('./data/band2_train.tif')
band1=plt.imread('./data/band1_train.tif')
# normalize bands to be in range [0, 1]
band3 = band3 / np.max(band3)
band2 = band2 / np.max(band2)
band1 = band1 / np.max(band1)
```

```
46 img_rgb_train=np.zeros((band1.shape[0],band1.shape[1],3))
47 img_rgb_train[:,:,0]=band3
48 img_rgb_train[:,:,1]=band2
49 img_rgb_train[:,:,2]=band1
50 # plt.figure()
51 # plt.imshow(img_rgb)
52 # plt.title('RGB image of traning data')
53 #plt.savefig('rgb_train.png')
54
55 # load test image as rgb image
56 band3=plt.imread('./data/band3_test.tif')
57 band2=plt.imread('./data/band2_test.tif')
58 band1=plt.imread('./data/band1_test.tif')
59 # normalize bands to be in range [0, 1]
60 band3 = band3 / np.max(band3)
61 band2 = band2 / np.max(band2)
62 band1 = band1 / np.max(band1)
63 img_rgb_test=np.zeros((band1.shape[0],band1.shape[1],3))
64 img_rgb_test[:,:,0]=band3
65 img_rgb_test[:,:,1]=band2
66 img_rgb_test[:,:,2]=band1
67 # plt.figure()
68 # plt.imshow(img_rgb)
69 # plt.title('RGB image of test data')
70 # plt.savefig('rgb_test.png')
71
72
73 # load label image (values from 1 to 7)
74 img_label_train = plt.imread('./data/labels_train.tif')
75 img_label_train_color = label2rgb(img_label_train,label_to_color)
76 plt.figure()
77 plt.subplot(122)
78 plt.imshow(img_label_train_color)
79 plt.title('Label of training data')
80 plt.subplot(121)
81 plt.imshow(img_rgb_train)
82 plt.title('RGB image of training data')
83 # plt.savefig('plots/train.png')
84
85 # load traindata (7 bands)
86 traindata =
      np.zeros((img_label_train.shape[0],img_label_train.shape[1],7))
87 for ii in range(0,7):
88     I = plt.imread('./data/band' + str(ii+1) + '_train.tif')
89     traindata[:,:,ii] = I
90 img_label_test = plt.imread('./data/labels_test.tif')
91 img_label_test_color = label2rgb(img_label_test,label_to_color)
92 # load testdata (7 bands)
93 testdata =
      np.zeros((img_label_test.shape[0],img_label_test.shape[1],7))
94 for ii in range(0,7):
```

```
    I = plt.imread('./data/band' + str(ii+1) + '_test.tif')
    testdata[:,:,ii] = I

# flatten the train data and labels
X_train = traindata.reshape(traindata.shape[0]*traindata.shape[1],
                            traindata.shape[2])

y_train = img_label_train.reshape(img_label_train.shape[0]
                                  *img_label_train.shape[1])

clf=GaussianNB()
clf.fit(X_train,y_train)
y_pred=clf.predict(X_train)
print('Accuracy on training data: ', np.mean(y_pred==y_train))

img_label_train_pred=y_pred.reshape(img_label_train.shape[0],
                                    img_label_train.shape[1])
img_label_train_pred_color=label2rgb(img_label_train_pred,label_to_color)
plt.figure()
plt.subplot(122)
plt.imshow(img_label_train_pred_color)
plt.title('Prediction result of training data')
plt.subplot(121)
plt.imshow(img_label_train_color)
plt.title('Label of training data')
# plt.savefig('plots/train_result.png')

# predict the test data

X_test=testdata.reshape(testdata.shape[0]*testdata.shape[1],testdata.shape[2])
y_test=img_label_test.reshape(img_label_test.shape[0]*img_label_test.shape[1])

y_pred_test=clf.predict(X_test)
img_label_test_pred=y_pred_test.reshape(testdata.shape[0],testdata.shape[1])
img_label_test_pred_color=label2rgb(img_label_test_pred,label_to_color)
print('Accuracy on test data: ', np.mean(y_pred_test==y_test))
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred_test)
print('Confusion matrix: ', cm)
print('sum1=', np.sum(cm, axis=1))
print('sum2=', np.sum(cm, axis=0))
print('sum3=', np.sum(cm))
# calculate the overall accuracy
OA = np.sum(np.diag(cm))/np.sum(cm)
# calculate the producer's accuracy and user's accuracy
# producer's accuracy
PA = np.zeros((7,1))
for ii in range(0,7):
    PA[ii] = cm[ii,ii]/np.sum(cm[ii,:])
print('Producer''s accuracy: ', PA)
# user's accuracy
```

```
UA = np.zeros((7,1))
for ii in range(0,7):
    UA[ii] = cm[ii,ii]/np.sum(cm[:,ii])
print('User''s accuracy: ', UA)
# kappa coefficient
p0 = OA
pe = np.sum(np.sum(cm, axis=1)*np.sum(cm, axis=0))/np.sum(cm)**2
kappa = (p0-pe)/(1-pe)
print('Kappa coefficient: ', kappa)
plt.figure()
plt.subplot(122)
plt.imshow(img_label_test_color)
plt.title('Label of test data')
plt.subplot(121)
plt.imshow(img_rgb_test)
plt.title('RGB image of test data')
# plt.savefig('plots/test.png')

plt.figure()
plt.subplot(122)
plt.imshow(img_label_test_pred_color)
plt.title('Prediction result of test data')
plt.subplot(121)
plt.imshow(img_label_test_color)
plt.title('Label of test data')
# plt.savefig('plots/test_result.png')

# predict only use rgb chanels
X_train_rgb =
   img_rgb_train.reshape(img_rgb_train.shape[0]*img_rgb_train.shape[1]
                                    ,img_rgb_train.shape[2])
X_test_rgb =
   img_rgb_test.reshape(img_rgb_test.shape[0]*img_rgb_test.shape[1],
                                  img_rgb_test.shape[2])
clf_rgb=GaussianNB()
clf_rgb.fit(X_train_rgb,y_train)
y_pred_rgb=clf_rgb.predict(X_test_rgb)
img_label_test_pred_rgb=y_pred_rgb.reshape(img_rgb_test.shape[0],
                                           img_rgb_test.shape[1])
img_label_test_pred_color_rgb=label2rgb(img_label_test_pred_rgb,
                                        label_to_color)
plt.figure()
plt.subplot(122)
plt.imshow(img_label_test_pred_color_rgb)
plt.title('Prediction result trained only on RGB')
plt.subplot(121)
plt.imshow(img_label_test_color)
plt.title('Label of test data')
plt.savefig('plots/test_result_rgb.png')
cm_rgb=confusion_matrix(y_test,y_pred_rgb)
print('Confusion matrix: ', cm_rgb)
```

```
195 print('sum1=', np.sum(cm_rgb, axis=1))
196 print('sum2=', np.sum(cm_rgb, axis=0))
197 print('sum3=', np.sum(cm_rgb))
198 # calculate the overall accuracy
199 OA_rgb = np.sum(np.diag(cm_rgb))/np.sum(cm_rgb)
200 print('Overall accuracy: ', OA_rgb)
201 # calculate the producer's accuracy and user's accuracy
202 # producer's accuracy
203 PA_rgb = np.zeros((7,1))
204 for ii in range(0,7):
205     PA_rgb[ii] = cm_rgb[ii,ii]/np.sum(cm_rgb[ii,:])
206 print('Producer''s accuracy: ', PA_rgb)
207 # user's accuracy
208 UA_rgb = np.zeros((7,1))
209 for ii in range(0,7):
210     UA_rgb[ii] = cm_rgb[ii,ii]/np.sum(cm_rgb[:,ii])
211 print('User''s accuracy: ', UA_rgb)
212 # plt.show()
```

ex04_classification.py