4 附录

In [1]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.metrics import cohen_kappa_score, confusion_matrix
from operator import truediv
import time
%matplotlib inline
```

In [2]:
```python
sns.set_style("darkgrid", {"grid.color": ".6", "grid.linestyle": ":"})
sns.set_theme(font='Times New Roman', font_scale=1.2)
plt.rc("figure", autolayout=True)
# Chinese support
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

In [3]:
```python
data = loadmat('./Indian_pines_corrected.mat')['indian_pines_corrected']
labels = loadmat('./Indian_pines_gt.mat')['indian_pines_gt']
print(f"Dataset: {data.shape}\nGround Truth: {labels.shape}") # 打印形状
print(np.unique(labels))
pd.DataFrame(pd.Series(labels.reshape(-1,)).value_counts()).T
```

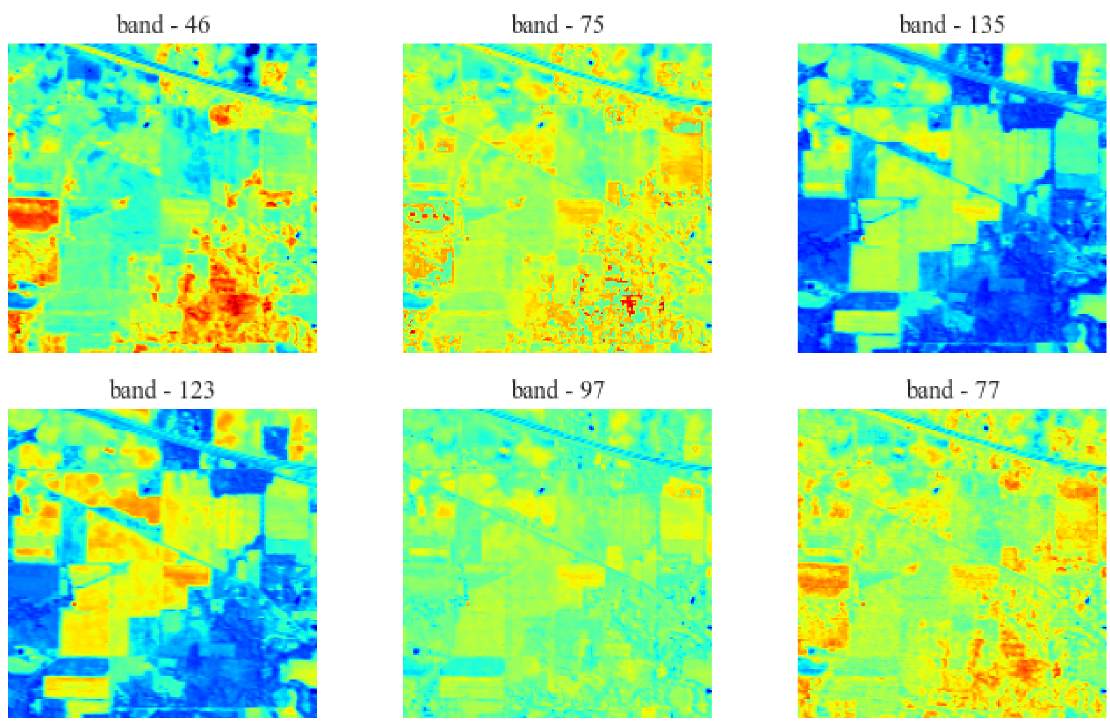
```
Dataset: (145, 145, 200)
Ground Truth: (145, 145)
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
```

Out[3]:

	0	11	2	14	10	3	6	12	5	8	15	4	13	16	\
0	10776	2455	1428	1265	972	830	730	593	483	478	386	237	205	93	

	1	7	9
0	46	28	20

```python
In [4]: sns.axes_style('whitegrid')
        fig = plt.figure(figsize=(12, 6))

        # 查看不同光谱下的图像示例
        for i in range(1, 1+6):
            fig.add_subplot(2, 3, i)
            band = np.random.randint(data.shape[2])
            plt.imshow(data[:, :, band], cmap='jet')
            plt.axis('off')
            plt.title(f'band - {band}')

        plt.savefig('./document/figure/diffband6.pdf')
        plt.show()
```




```python
In [5]: def extract_pixels(X, y, save_name='indian_pines_all'):
            q = X.reshape(-1, X.shape[2])
            df = pd.DataFrame(q)
            df = pd.concat([df, pd.DataFrame(y.ravel())], axis=1)
```

```
    df.columns= [f'band{i}' for i in range(1, 1+X.shape[2])]+['class']
    #df.to_csv(f'data/{save_name}.csv')
    return df

df = extract_pixels(data, labels, save_name='indian_pines_all')
df.head()
```

Out[5]:

	band1	band2	band3	band4	band5	band6	band7	band8	band9	band10	...
0	3172	4142	4506	4279	4782	5048	5213	5106	5053	4750	...
1	2580	4266	4502	4426	4853	5249	5352	5353	5347	5065	...
2	3687	4266	4421	4498	5019	5293	5438	5427	5383	5132	...
3	2749	4258	4603	4493	4958	5234	5417	5355	5349	5096	...
4	2746	4018	4675	4417	4886	5117	5215	5096	5098	4834	...

	band192	band193	band194	band195	band196	band197	band198	band199
0	1094	1090	1112	1090	1062	1069	1057	1020
1	1108	1104	1117	1091	1079	1085	1064	1029
2	1111	1114	1114	1100	1065	1092	1061	1030
3	1122	1108	1109	1109	1071	1088	1060	1030
4	1110	1107	1112	1094	1072	1087	1052	1034

	band200	class
0	1020	3
1	1020	3
2	1016	3
3	1006	3
4	1019	3

[5 rows x 201 columns]

In [6]:
```
# x = df[df['class'] != 0] # 标签 0 没有关键的类别，去除
x = df
X = x.iloc[:, :-1].values
y = x.loc[:, 'class'].values
X.shape
```

Out[6]: (21025, 200)

In [7]:
```
from sklearn import preprocessing
```

```
def norm(X):
    min_max_scaler = preprocessing.MinMaxScaler()
    X = min_max_scaler.fit_transform(X)
    return X

def standartize(X): # 数据标准化
    scaler = preprocessing.StandardScaler().fit(X)
    X = scaler.transform(X)
    return X

X = norm(X)
```

4.1 KNN classification

```
In [8]: acc = np.zeros((15,20))

        start = time.time()
        for hell in range(2,17):
          start1 = time.time()
          model = PCA(n_components=hell)
          pca = model.fit_transform(X)
          X_train, X_test, y_train, y_test = train_test_split(pca,
                                                              y,
                                                              test_size=0.95,
                                                              random_state=42,
                                                              stratify=y)
          print("For band =", hell)
          for i in range(1,21):
            knn = KNeighborsClassifier(n_neighbors=i)
            knn.fit(X_train, y_train)
            y_pred=knn.predict(X_test)
            acc[hell-2][i-1]=metrics.accuracy_score(y_test, y_pred)
          end=time.time()
          print("Time taken for band", hell," is =",end-start1,"sec")
          max=np.argmax(acc[hell-2])
          print("Max accuracy =", acc[hell-2][max])
```

```python
        end=time.time()
        print("Total Time Taken =",end-start,"sec")
```

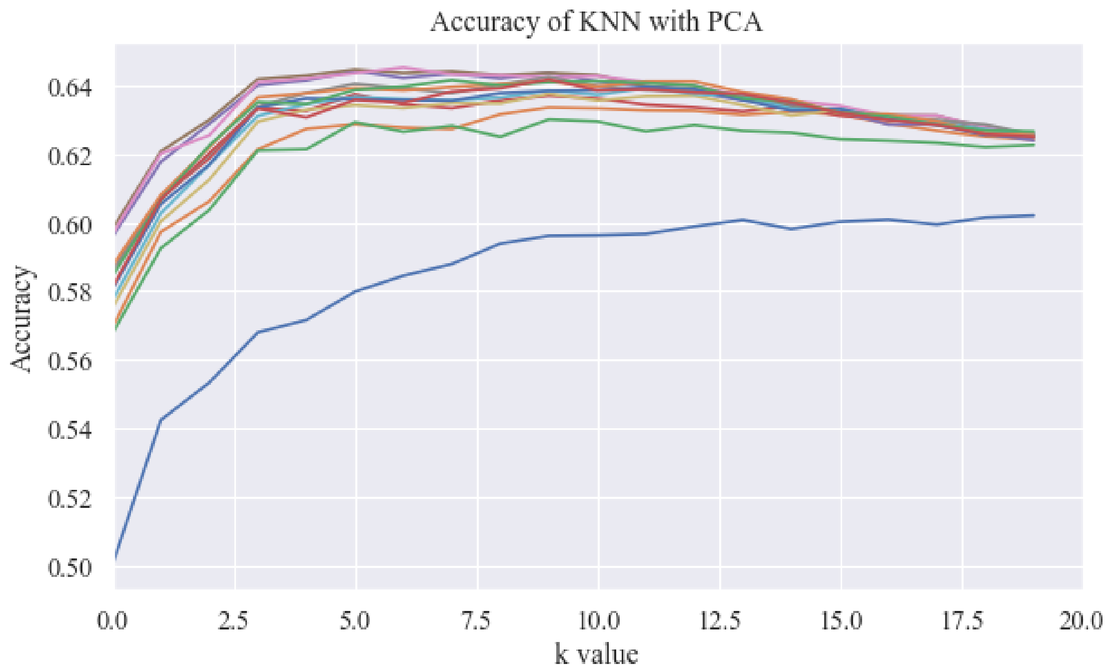
```
For band = 2
Time taken for band 2  is = 1.5301227569580078 sec
Max accuracy = 0.6020827075197757
For band = 3
Time taken for band 3  is = 1.7832973003387451 sec
Max accuracy = 0.6335736457394613
For band = 4
Time taken for band 4  is = 1.9694814682006836 sec
Max accuracy = 0.6300690898167618
For band = 5
Time taken for band 5  is = 2.1276233196258545 sec
Max accuracy = 0.6373785921698207
For band = 6
Time taken for band 6  is = 2.4373369216918945 sec
Max accuracy = 0.6441874436767798
For band = 7
Time taken for band 7  is = 2.6667490005493164 sec
Max accuracy = 0.6446380294382698
For band = 8
Time taken for band 8  is = 2.8386828899383545 sec
Max accuracy = 0.6452388104535897
For band = 9
Time taken for band 9  is = 3.0289556980133057 sec
Max accuracy = 0.64248523080004
For band = 10
Time taken for band 10  is = 3.4666008949279785 sec
Max accuracy = 0.6375287874236507
For band = 11
Time taken for band 11  is = 3.4986419677734375 sec
Max accuracy = 0.6391809352157806
For band = 12
Time taken for band 12  is = 3.7522194385528564 sec
Max accuracy = 0.6394813257234405
For band = 13
```

```
Time taken for band 13  is = 3.7696914672851562 sec
Max accuracy = 0.6411334735155703
For band = 14
Time taken for band 14  is = 3.9831297397613525 sec
Max accuracy = 0.6415339941924502
For band = 15
Time taken for band 15  is = 4.22843599319458 sec
Max accuracy = 0.6417342545308902
For band = 16
Time taken for band 16  is = 10.74459195137024 sec
Max accuracy = 0.6419845799539401
Total Time Taken = 51.828561305999756 sec
```

In [9]:
```python
acc.shape
```

Out[9]:
```
(15, 20)
```

In [10]:
```python
plt.figure(figsize=(8, 5), dpi=80)
for i in range(0,14):
  plt.plot(acc[i])
plt.xlim(0,20)
plt.title('Accuracy of KNN with PCA')
plt.xlabel('k value')
plt.ylabel('Accuracy')
plt.savefig('./document/figure/KNN_with_PCA.pdf')
plt.show()
```



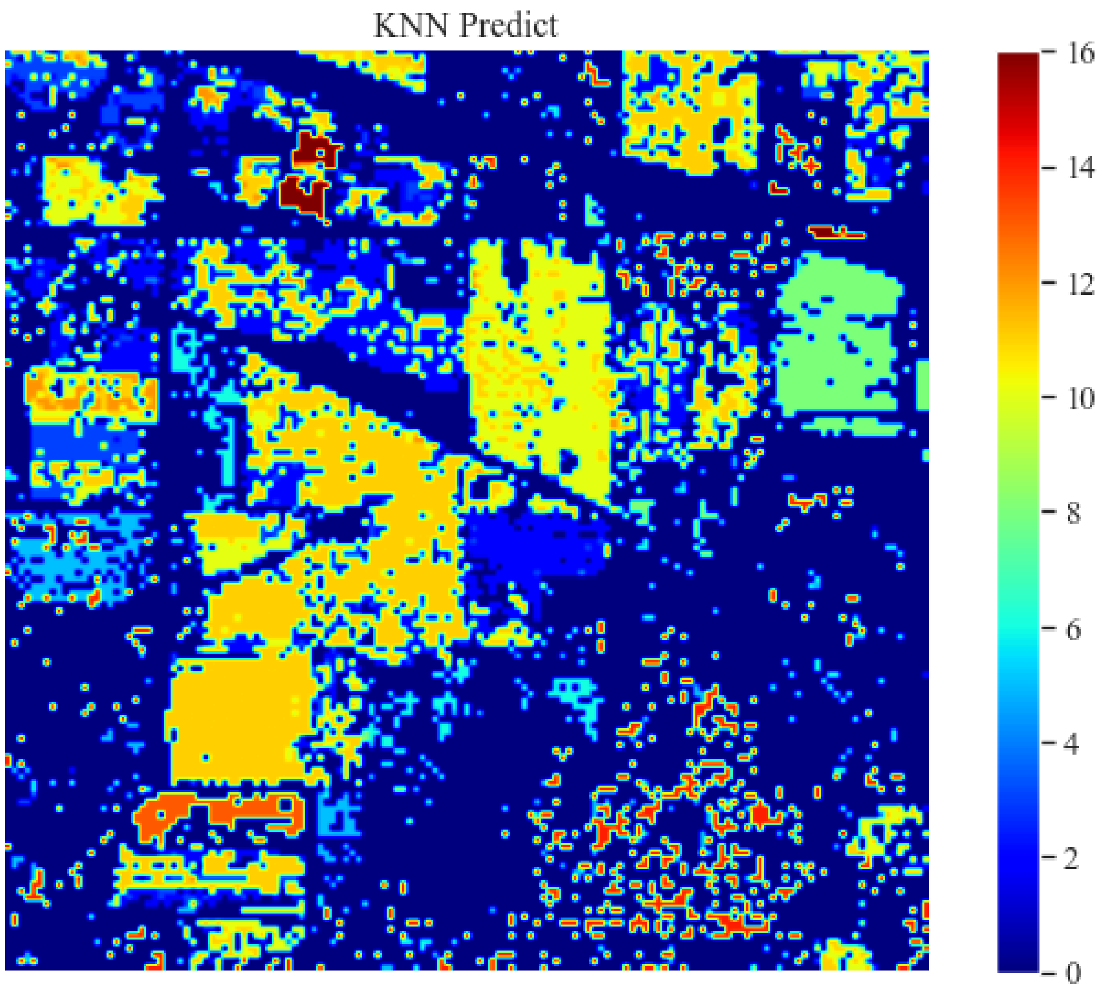

```
In [11]: temp = band = k = 0
         for i in range(0,14):
           for j in range(0,20):
             if temp < acc[i][j]:
               temp = acc[i][j]
               band = i + 2
               k = j + 1
         print("BAND =", band, "K-Value =",k, "MAX ACCURACY =", acc[band-2][k-1])
```

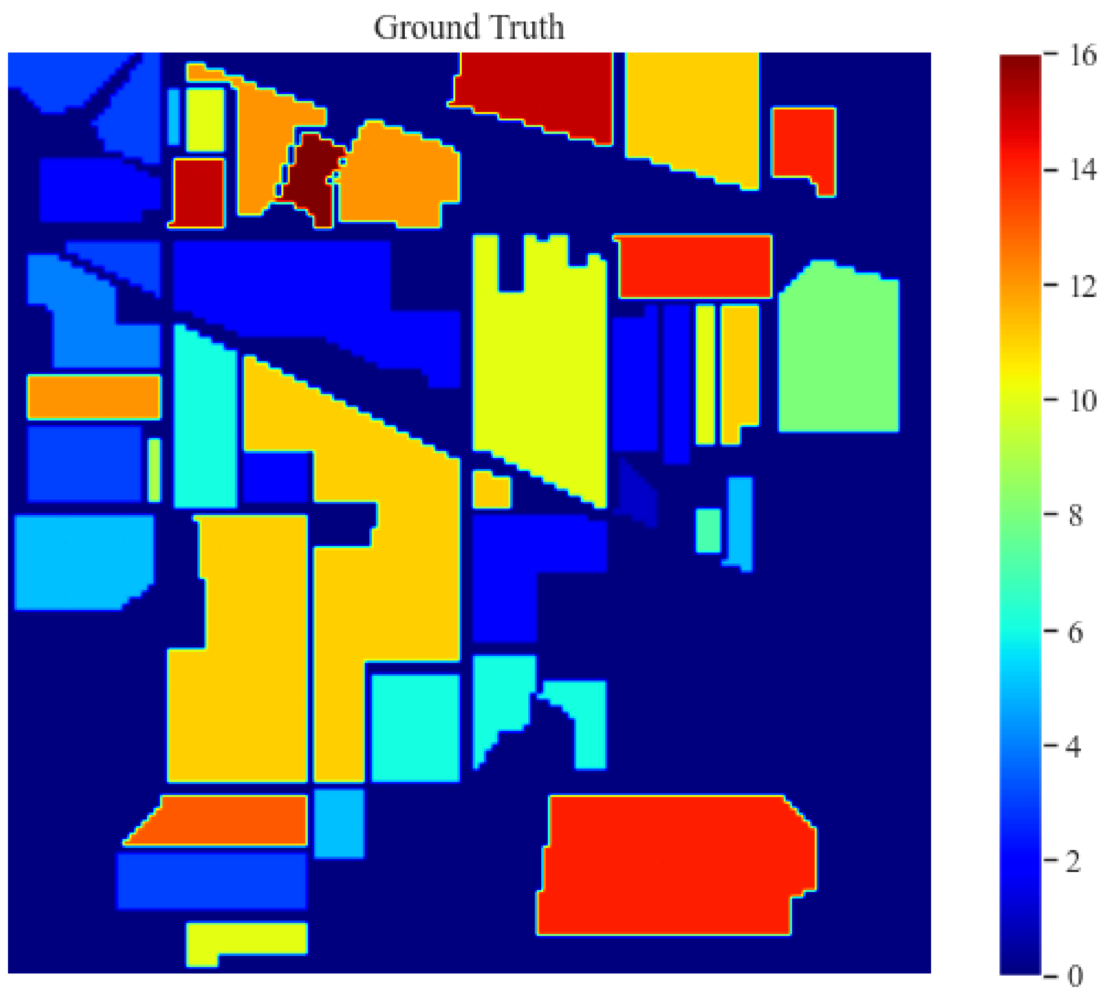
```
BAND = 8 K-Value = 7 MAX ACCURACY = 0.6452388104535897
```

```
In [12]: def AA_andEachClassAccuracy(confusion_matrix):
             counter = confusion_matrix.shape[0]
             list_diag = np.diag(confusion_matrix)
             list_raw_sum = np.sum(confusion_matrix, axis=1)
             each_acc = np.nan_to_num(truediv(list_diag, list_raw_sum))
             average_acc = np.mean(each_acc)
             return each_acc, average_acc
```

In [13]:
```python
model = PCA(n_components=band)
pca = model.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(pca,
                                                    y,
                                                    test_size =0.95,
                                                    random_state=42,
                                                    stratify=y)
knn = KNeighborsClassifier(n_neighbors = k)
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)

confusion = confusion_matrix(y_test, y_pred)
each_acc, aa = AA_andEachClassAccuracy(confusion)

print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Average Accuracy:", aa)
print("Kappa Coefficient:", metrics.cohen_kappa_score(y_test, y_pred))
print("Average Time:",(end-start)/320)

output = knn.predict(pca)
output = np.reshape(output,(145,145))
gt = np.reshape(y, (145,145))

plt.figure(figsize=(8, 6), dpi=80)
plt.imshow(output, cmap='jet')
plt.colorbar()
plt.axis('off')
plt.title('KNN Predict')
plt.savefig('./document/figure/KNN_Predict.pdf')

plt.figure(figsize=(8, 6), dpi=80)
plt.imshow(gt, cmap='jet')
plt.colorbar()
plt.axis('off')
plt.title('Ground Truth')
plt.savefig('./document/figure/GT.pdf')
plt.show()
```

```
Accuracy: 0.6452388104535897
Average Accuracy: 0.3522210175641218
Kappa Coefficient: 0.4661555315619679
Average Time: 0.16196425408124923
```






```
In [14]: from sklearn.model_selection import cross_val_score
         knn = KNeighborsClassifier(n_neighbors = k)
         scores = cross_val_score(knn, pca, y, cv=10, scoring='accuracy')
         print(scores)
         print(scores.mean())
```

```
[0.52829291 0.68283405 0.65382786 0.65430338 0.73941988 0.75261656
 0.72549952 0.65889629 0.70599429 0.59705043]
0.669873516742201
```

4.2 KNN report

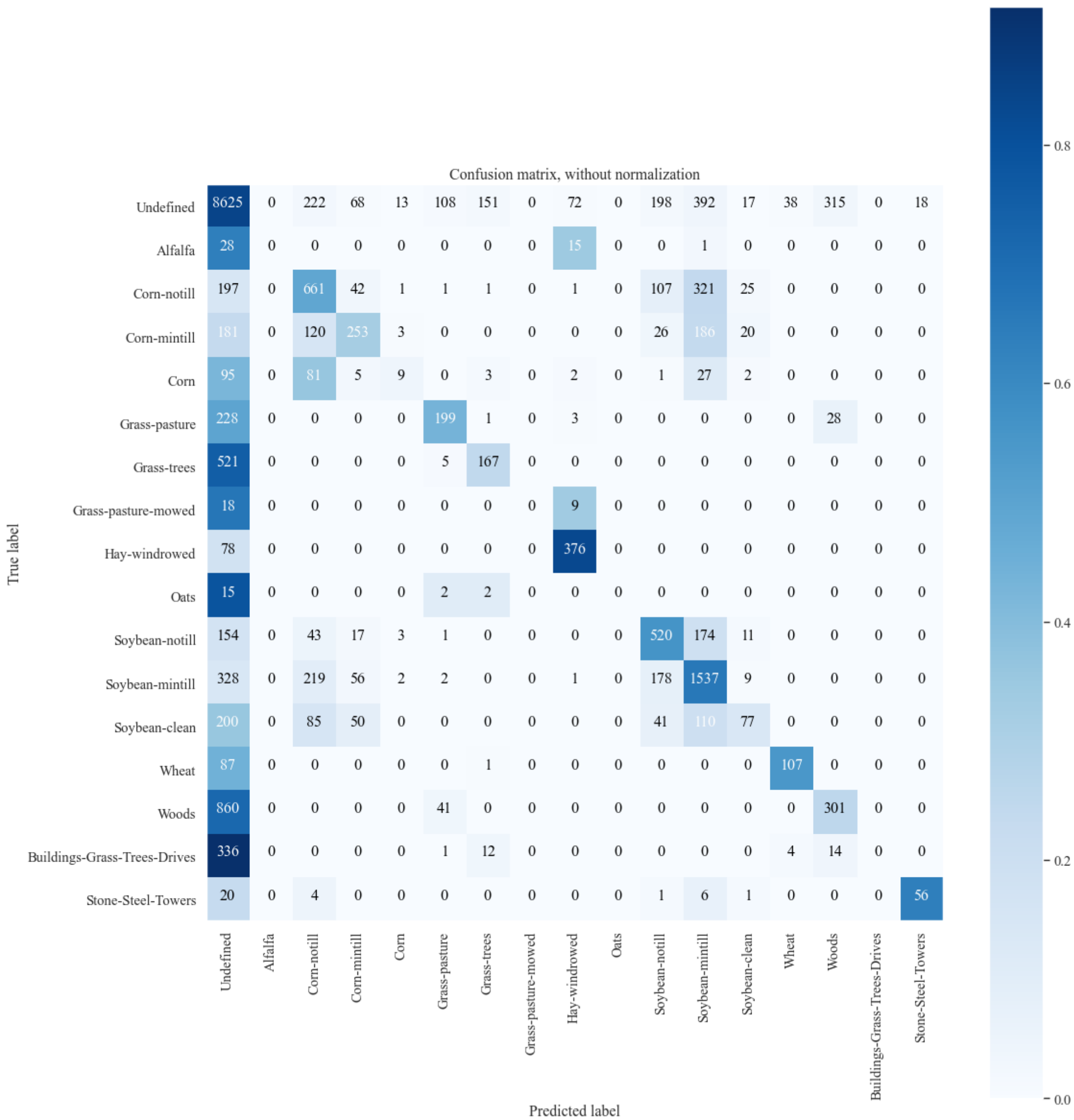
In [15]:
```
model = PCA(n_components=7)
pca = model.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(pca,
                                                    y,
                                                    test_size=0.95,
                                                    random_state=42,
                                                    stratify=y)
model = KNeighborsClassifier(n_neighbors=14)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

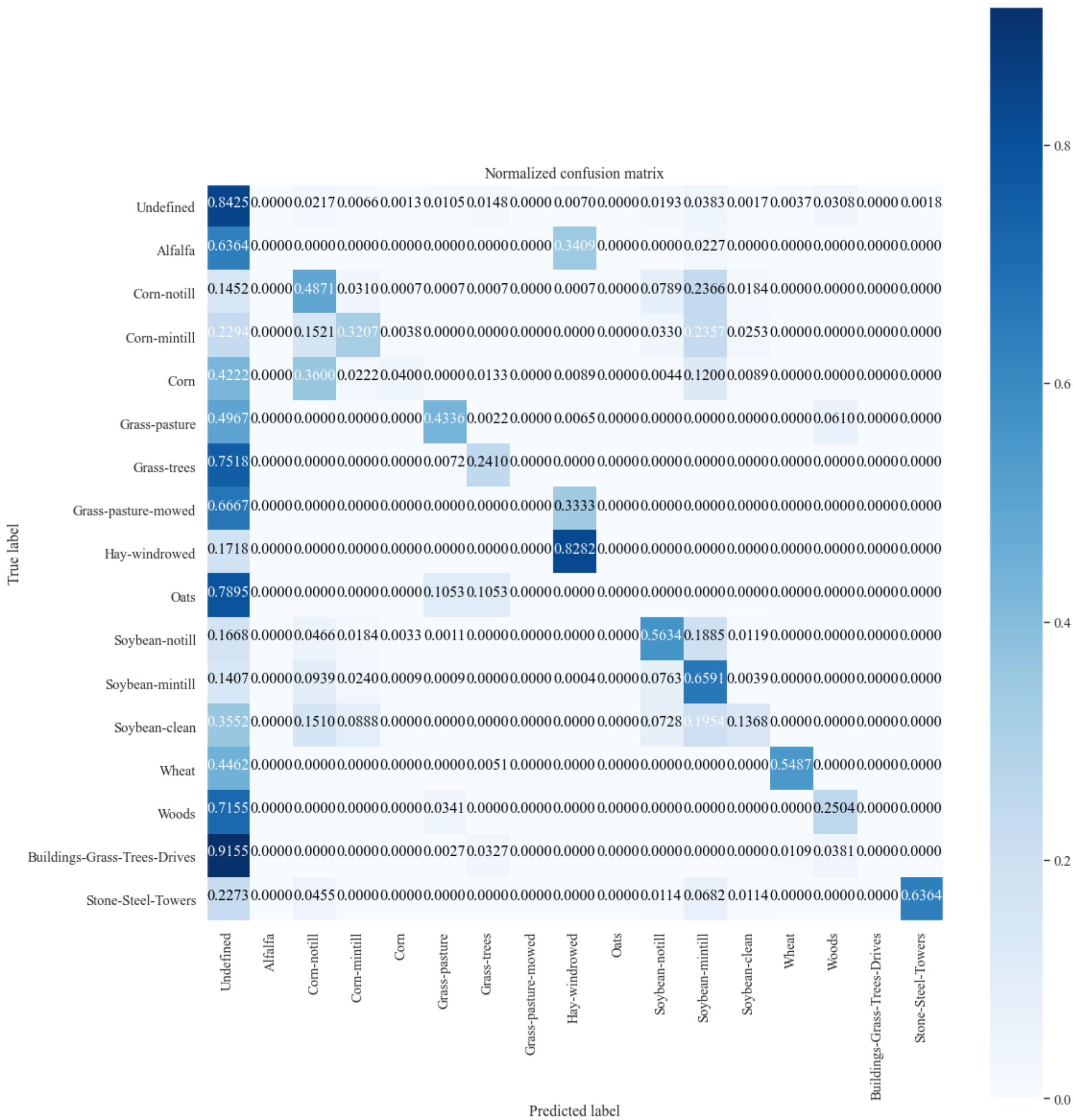
In [16]:
```
target_names = ['Undefined','Alfalfa', 'Corn-notill', 'Corn-mintill', 'Corn'
           ,'Grass-pasture', 'Grass-trees', 'Grass-pasture-mowed',
            'Hay-windrowed', 'Oats', 'Soybean-notill', 'Soybean-mintill',
           'Soybean-clean', 'Wheat', 'Woods', 'Buildings-Grass-Trees-Drives',
           'Stone-Steel-Towers']
```

In [17]:
```
from sklearn.metrics import classification_report
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.get_cmap("Blues")):
    Normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    if normalize:
        cm = Normalized
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    plt.imshow(Normalized, interpolation='nearest', cmap=cmap)
    plt.colorbar()
    plt.title(title)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
```

```python
        plt.yticks(tick_marks, classes)

        fmt = '.4f' if normalize else 'd'
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            thresh = cm[i].max() / 2.
            plt.text(j, i, format(cm[i, j], fmt),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')

    plt.figure(figsize=(15,15),dpi=80)
    plt.grid(False)
    plot_confusion_matrix(confusion, classes=target_names, normalize=False,
                          title='Confusion matrix, without normalization')
    plt.savefig('./document/figure/confusion_mat.pdf')

    plt.figure(figsize=(15,15),dpi=80)
    plt.grid(False)
    plot_confusion_matrix(confusion, classes=target_names, normalize=True,
                          title='Normalized confusion matrix')
    plt.savefig('./document/figure/confusion_mat_norm.pdf')
    plt.show()
```

```
Confusion matrix, without normalization
Normalized confusion matrix
```


Confusion matrix, without normalization
True label
Predicted label
Undefined 8625 0 222 68 13 108 151 0 72 0 198 392 17 38 315 0 18
Alfalfa 28 0 0 0 0 0 0 0 15 0 0 1 0 0 0 0 0
Corn-notill 197 0 661 42 1 1 1 0 1 0 107 321 25 0 0 0 0
Corn-mintill 181 0 120 253 3 0 0 0 0 0 26 186 20 0 0 0 0
Corn 95 0 81 5 9 0 3 0 2 0 1 27 2 0 0 0 0
Grass-pasture 228 0 0 0 0 199 1 0 3 0 0 0 0 0 28 0 0
Grass-trees 521 0 0 0 0 5 167 0 0 0 0 0 0 0 0 0 0
Grass-pasture-mowed 18 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0
Hay-windrowed 78 0 0 0 0 0 0 0 376 0 0 0 0 0 0 0 0
Oats 15 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0
Soybean-notill 154 0 43 17 3 1 0 0 0 0 520 174 11 0 0 0 0
Soybean-mintill 328 0 219 56 2 2 0 0 1 0 178 1537 9 0 0 0 0
Soybean-clean 200 0 85 50 0 0 0 0 0 0 41 110 77 0 0 0 0
Wheat 87 0 0 0 0 0 1 0 0 0 0 0 0 107 0 0 0
Woods 860 0 0 0 0 41 0 0 0 0 0 0 0 0 301 0 0
Buildings-Grass-Trees-Drives 336 0 0 0 0 1 12 0 0 0 0 0 0 4 14 0 0
Stone-Steel-Towers 20 0 4 0 0 0 0 0 0 0 1 6 1 0 0 0 56
Undefined
Alfalfa
Corn-notill
Corn-mintill
Corn
Grass-pasture
Grass-trees
Grass-pasture-mowed
Hay-windrowed
Oats
Soybean-notill
Soybean-mintill
Soybean-clean
Wheat
Woods
Buildings-Grass-Trees-Drives
Stone-Steel-Towers
0.8
0.6
0.4
0.2
0.0


Normalized confusion matrix
True label
Predicted label
Undefined 0.8425 0.0000 0.0217 0.0066 0.0013 0.0105 0.0148 0.0000 0.0070 0.0000 0.0193 0.0383 0.0017 0.0037 0.0308 0.0000 0.0018
Alfalfa 0.6364 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.3409 0.0000 0.0000 0.0227 0.0000 0.0000 0.0000 0.0000 0.0000
Corn-notill 0.1452 0.0000 0.4871 0.0310 0.0007 0.0007 0.0007 0.0000 0.0007 0.0000 0.0789 0.2366 0.0184 0.0000 0.0000 0.0000 0.0000
Corn-mintill 0.2294 0.0000 0.1521 0.3207 0.0038 0.0000 0.0000 0.0000 0.0000 0.0000 0.0330 0.2357 0.0253 0.0000 0.0000 0.0000 0.0000
Corn 0.4222 0.0000 0.3600 0.0222 0.0400 0.0000 0.0133 0.0000 0.0089 0.0000 0.0044 0.1200 0.0089 0.0000 0.0000 0.0000 0.0000
Grass-pasture 0.4967 0.0000 0.0000 0.0000 0.0000 0.4336 0.0022 0.0000 0.0065 0.0000 0.0000 0.0000 0.0000 0.0000 0.0610 0.0000 0.0000
Grass-trees 0.7518 0.0000 0.0000 0.0000 0.0000 0.0072 0.2410 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
Grass-pasture-mowed 0.6667 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.3333 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
Hay-windrowed 0.1718 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.8282 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
Oats 0.7895 0.0000 0.0000 0.0000 0.0000 0.1053 0.1053 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
Soybean-notill 0.1668 0.0000 0.0466 0.0184 0.0033 0.0011 0.0000 0.0000 0.0000 0.0000 0.5634 0.1885 0.0119 0.0000 0.0000 0.0000 0.0000
Soybean-mintill 0.1407 0.0000 0.0939 0.0240 0.0009 0.0009 0.0000 0.0000 0.0004 0.0000 0.0763 0.6591 0.0039 0.0000 0.0000 0.0000 0.0000
Soybean-clean 0.3552 0.0000 0.1510 0.0888 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0728 0.1954 0.1368 0.0000 0.0000 0.0000 0.0000
Wheat 0.4462 0.0000 0.0000 0.0000 0.0000 0.0000 0.0051 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.5487 0.0000 0.0000 0.0000
Woods 0.7155 0.0000 0.0000 0.0000 0.0000 0.0341 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.2504 0.0000 0.0000
Buildings-Grass-Trees-Drives 0.9155 0.0000 0.0000 0.0000 0.0000 0.0027 0.0327 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0109 0.0381 0.0000 0.0000
Stone-Steel-Towers 0.2273 0.0000 0.0455 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0114 0.0682 0.0114 0.0000 0.0000 0.0000 0.6364
Undefined
Alfalfa
Corn-notill
Corn-mintill
Corn
Grass-pasture
Grass-trees
Grass-pasture-mowed
Hay-windrowed
Oats
Soybean-notill
Soybean-mintill
Soybean-clean
Wheat
Woods
Buildings-Grass-Trees-Drives
Stone-Steel-Towers
0.8
0.6
0.4
0.2
0.0