**Appendix A: Code For Spectral Signatures**

Source Code:<https://github.com/AdrianL769/CSCI-491-Senior-Seminar.git>

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** scipy.io **import** loadmat

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.preprocessing **import** StandardScaler, MinMaxScaler

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.metrics **import** confusion\_matrix, classification\_report

**from** sklearn.svm **import** SVC

**from** sklearn.decomposition **import** PCA

**from** sklearn.neural\_network **import** MLPClassifier

**from** sklearn **import** metrics

*# Load the hyperspectral data and ground truth from .mat files*

hyperspectral\_data = loadmat('/home/jovyan/PaviaU.mat')['paviaU']

ground\_truth\_data = loadmat('/home/jovyan/PaviaU\_gt.mat')['paviaU\_gt']

**def** plot\_spectralSignature(hyperspectral\_image, pixels):

fig, ax = plt.subplots(figsize=(12, 4))

**for** (x, y) **in** pixels:

reflectance = hyperspectral\_image[x, y, :]

ax.plot(reflectance, label=f'Pixel ({x}, {y})')

ax.set\_title('Spectral Signatures of Selected Pixels')

ax.set\_xlabel('Band')

ax.set\_ylabel('Reflectance')

ax.legend()

plt.show()

pixels\_to\_plot = [(10, 10), (20, 20), (30, 30)]

plot\_spectralSignature(hyperspectral\_data, pixels\_to\_plot)

*#Reshaping Ground Truth*

**def** reshape\_ground\_truth(hyperspectral\_data):

*# Get the ground truth image into a 1D array*

ground\_truth\_1d = hyperspectral\_data.flatten()

**return** ground\_truth\_1d

ground\_truth\_image = np.array([[1, 2, 3, 4]])

print(reshape\_ground\_truth(hyperspectral\_data))

plt.imshow(ground\_truth\_image)

*#Reshape Hyperspectral Image*

**def** reshape\_hyperspectral\_data(hyperspectral\_data):

shape\_image = hyperspectral\_data.shape

reshaped\_data = hyperspectral\_data.reshape((shape\_image[0] \* shape\_image[1], shape\_image[2]))

**return** reshaped\_data

print (reshape\_hyperspectral\_data(hyperspectral\_data))

*#Normalization Function*

**def** normalization(bands\_pixels\_HS):

r, c = bands\_pixels\_HS.shape

normalized\_pixels = np.zeros((r, c))

i = 0

*#For loop*

**for** x **in** bands\_pixels\_HS:

*#Minimum value*

min\_val = min(x)

*#Maximum Value*

max\_val = max(x)

*#NORMALIZATION Formula*

x\_norm = (x - min\_val) / (max\_val - min\_val)

normalized\_pixels[i,:] = x\_norm

i = i+1

**return** normalized\_pixels

*#Making a Standardization Function*

**def** standardization(bands\_pixels\_HS):

r, c = bands\_pixels\_HS.shape

standardized\_pixels = np.zeros((r, c))

i = 0

*#For loop*

**for** x **in** bands\_pixels\_HS:

*#Mean value*

mean\_val = np.mean(bands\_pixels\_HS)

*#Standard Deviation*

std\_val = np.std(bands\_pixels\_HS)

*#STANDARDIZATION Formula*

x\_stand = (x - mean\_val) / std\_val

standardized\_pixels[i,:] = x\_stand

i = i+1

**return** standardized\_pixels

**def** pixels\_per\_classes(data\_labels):

values, counts = np.unique(data\_labels, return\_counts= True)

print("Labels: ", values)

print("Quantity: ",counts)

**def** background\_off(bands\_pixels\_HS, labels):

*#this function select the pixels from the ground thruth but the background (labels=0)*

zero\_list = []

index = 0

new\_labels = labels

**for** label **in** labels:

**if** (label == 0):

zero\_list.append(index)

index = index + 1

bands\_pixels\_HS = np.delete(bands\_pixels\_HS, zero\_list, axis = 0)

new\_labels = np.delete(new\_labels, zero\_list)

**return** bands\_pixels\_HS, new\_labels

[ 647 499 464 ... 2416 2447 2485]

[[ 647 499 464 ... 3221 3238 3250]

[ 604 546 527 ... 2442 2464 2528]

[ 621 746 556 ... 2308 2345 2361]

...

[ 593 387 428 ... 2382 2407 2423]

[ 593 751 655 ... 2312 2308 2289]

[ 889 720 449 ... 2416 2447 2485]]

*#Formatting dataset into band by pixel format (Each of these functions is returning a value)*

ground\_truth\_labels = reshape\_ground\_truth(ground\_truth\_data)

reshape\_HS = reshape\_hyperspectral\_data(hyperspectral\_data)

*#Processing Image by Normalizing Pixel*

preprocessed\_data = normalization(reshape\_HS)

type(reshape\_HS)

numpy.ndarray

*#Splitting datasets into training and testing*

X = reshape\_HS

y = ground\_truth\_labels

*#PCA Reduction*

do\_pca = 0

**if** do\_pca == 1:

pca = PCA(n\_components = 5) *#these are the number of features*

pca.fit(X)

X\_data\_pca = pca.transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_data\_pca, Y, test\_size=0.2, random\_state=42)

**else**:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.2, random\_state=42)

*#Verifying datasets into training and testing*

print(X\_train.shape)

print(y\_train.shape)

(165920, 103)

(165920,)

*#Creating Machine Learning model data for Land Cover Classification*

lr\_model = LogisticRegression()

lr\_model.fit(X\_train, y\_train)

LogisticRegression()

*#Label Prediction*

predicted\_label = lr\_model.predict(X\_test)

print(predicted\_label)

[0 0 0 ... 0 0 0]

*#Displaying Confusion Matrix (Logistic Regression)*

cm = confusion\_matrix(y\_test, predicted\_label, labels = lr\_model.classes\_)

disp = metrics.ConfusionMatrixDisplay(confusion\_matrix = cm, display\_labels = lr\_model.classes\_)

disp.plot()

plt.show()

*#Printing Classification reports (Logistic Regression)*

num\_classes = 10

target\_names = ["Class {}".format(i) **for** i **in** range(num\_classes)]

print(classification\_report(y\_test, predicted\_label, target\_names = target\_names))

precision recall f1-score support

Class 0 0.81 0.97 0.88 33031

Class 1 0.03 0.00 0.00 1309

Class 2 0.45 0.17 0.25 3692

Class 3 0.00 0.00 0.00 396

Class 4 0.35 0.04 0.07 609

Class 5 0.75 0.79 0.77 272

Class 6 0.03 0.00 0.00 989

Class 7 0.00 0.00 0.00 260

Class 8 0.00 0.00 0.00 741

Class 9 0.00 0.00 0.00 181

accuracy 0.79 41480

macro avg 0.24 0.20 0.20 41480

weighted avg 0.70 0.79 0.73 41480

*#Predicting the whole HSI (Logistic Regression)*

data\_normalized\_ = normalization(reshape\_HS)

output\_map = lr\_model.predict(data\_normalized\_)

*#Shaping the LR Model*

output\_map.shape

(207400,)

row, columns, bands = hyperspectral\_data.shape

output\_map = output\_map.reshape((row,columns))

gt\_result = output\_map \* (ground\_truth\_data != 0)

plt.imshow(gt\_result, cmap='jet')

<matplotlib.image.AxesImage at 0x7f4f3489e810>

*#Using Support Vector Machine*

*#svm = SVC(C=1.0, kernel='rbf', random\_state=42)*

*#svm.fit(X\_train, y\_train)*

*#Training Model*

svm = SVC()

svm.fit(X\_train, y\_train)

SVC()

*#Predicting testing set of SVM*

predicted\_label = svm.predict(X\_test)

conf\_matrix = confusion\_matrix(y\_test, predicted\_label)

class\_report = classification\_report(y\_test, predicted\_label)

*#Displaying Confusion Matrix (SVM)*

cm = confusion\_matrix(y\_test, predicted\_label, labels = svm.classes\_)

disp = metrics.ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=svm.classes\_)

disp.plot()

plt.show()

*#Printing Classification reports (SVM)*

num\_classes = 10

target\_names = ["Class {}".format(i) **for** i **in** range(num\_classes)]

print(classification\_report(y\_test, predicted\_label, target\_names=target\_names))

precision recall f1-score support

Class 0 0.81 1.00 0.89 33031

Class 1 0.00 0.00 0.00 1309

Class 2 0.85 0.08 0.15 3692

Class 3 0.00 0.00 0.00 396

Class 4 0.00 0.00 0.00 609

Class 5 0.76 0.94 0.84 272

Class 6 0.00 0.00 0.00 989

Class 7 0.00 0.00 0.00 260

Class 8 0.00 0.00 0.00 741

Class 9 0.00 0.00 0.00 181

accuracy 0.81 41480

macro avg 0.24 0.20 0.19 41480

weighted avg 0.72 0.81 0.73 41480

*#predicting the whole HSI*

data\_normalized\_ = normalization(reshape\_HS)

output\_map = svm.predict(data\_normalized\_)

*#Shaping the SVM Model*

output\_map.shape

(207400,)

row, columns, bands = hyperspectral\_data.shape

output\_map = output\_map.reshape((row,columns))

gt\_result = output\_map \* (ground\_truth\_data != 0)

plt.imshow(gt\_result, cmap='jet')

<matplotlib.image.AxesImage at 0x7f1dc33ffcd0>

*#mlp\_model = MLPClassifier(layer\_sizes=0.1, random\_state=42)*

*#mlp\_model.fit(X\_train, y\_train)*

mlp = MLPClassifier()

mlp.fit(X\_train, y\_train)

predicted\_label\_mlp = mlp.predict(X\_test)

*#Logistic Regrssion*

*# Parameters*

C\_values = [1.0, 0.01, 0.05]

solvers = ['lbfgs', 'liblinear', 'newton-cg']

*# Train and evaluate models*

results = []

**for** solver **in** solvers:

**for** C **in** C\_values:

print("Training with C={C} and solver={solver}")

*#Train SVM Model*

**for** kernel **in** kernels:

**for** C **in** C\_values:

**if** kernel == 'poly':

**for** degree **in** degrees:

print(f"Training with C={C}, kernel={kernel}, degree={degree}")

svm\_model = SVC(C=C, kernel=kernel, degree=degree)

svm\_model.fit(X\_train, y\_train)

y\_pred = svm\_model.predict(X\_test)

print("C={result['C']} Kernel={result['kernel']} Degree={result['degree']}")

*# MLP Parameters*

activations = ['logistic', 'relu']

solvers = ['sgd', 'adam']

layer\_sizes = [(50, 30), (100,), (100, 50)]

mlp\_results = []

*#loading dataset*

path = ('/home/jovyan/PaviaU.mat')

path\_gt = ('/home/jovyan/PaviaU.mat')

path = "/Indian\_pines\_corrected.mat"

path\_gt = "/Indian\_pines\_gt.mat"

image\_name = "indian\_pines\_corrected"

gt\_image = "indian\_pines\_gt"

band=[20]

data\_HSI = read\_HSI(path, image\_name, band)

data\_gt= read\_groundTruth(path\_gt,gt\_image)

plt.imshow(np.transpose(pavia\_gt['paviaU\_gt']))

plt.imshow(np.transpose(hsi\_pavia['paviaU'][:,:,5]))

*#Printing Classification reports*

num\_classes = 9

target\_names = ["Class {}".format(i) **for** i **in** range(num\_classes)]

print(classification\_report(y\_test, predicted\_label\_svm, target\_names=target\_names))

*#Predicting the whole Hyperspectral Image*

data\_normalized\_ = normalization(data\_pixel\_by\_bands)

output\_map = svm.predict(data\_normalized\_)

row, columns, bands = data\_HSI.shape

output\_map = output\_map.reshape((row,columns))

gt\_result = output\_map \* (data\_gt != 0)

plt.imshow(gt\_result, cmap='jet')