11TH ADVANCED TRAINING COURSE ON LAND REMOTE SENSING

SAR & Optical data for Forestry: Python classification&regression exercise

Oleg Antropov, VTT Technical Research Centre of Finland

Exercise 1: Windstorm damage detection

Winstorm damage: definition

Input data represents a feature table X of stand-wise averages of “gamma-naught” values of several Sentinel-1 images acquired during winter season in Kainuu province of Finland, with target vector Y denoting damaged stands as 1 and non-damaged as 0.

Feature values are stored as HH_1, HV_1, HH_2, HV_2, … HH_*i*, HV_*i*, where *i* indexes Sentinel-1 datatakes.

Altogether 24 features available.

	0	1	2	3	4	5	6	7	8
0	0.042238	0.204763	0.0473664	0.212915	0.0498249	0.248792	0.0511011	0.228503	0.042
1	0.0400261	0.167429	0.0408453	0.162396	0.0453671	0.198624	0.0448717	0.206028	0.052
2	0.0384471	0.17716	0.0471163	0.172263	0.0397206	0.235421	0.0418255	0.238336	0.051
3	0.0386781	0.16937	0.0396473	0.167743	0.0457365	0.209459	0.0415521	0.199609	0.046
4	0.0346316	0.244109	0.052797	0.169443	0.0421452	0.251942	0.0412769	0.235223	0.057
5	0.0395057	0.172088	0.0424748	0.156928	0.0512316	0.232528	0.0499557	0.207746	0.050
6	0.0336381	0.164851	0.0284435	0.137749	0.0481644	0.205583	0.0518456	0.242115	0.054
7	0.0374011	0.15174	0.0362223	0.145424	0.0444247	0.209713	0.0413482	0.147064	0.049
8	0.0368379	0.172461	0.0305054	0.152425	0.0418559	0.201357	0.0360327	0.165862	0.038
9	0.0446427	0.20334	0.0381823	0.16902	0.0414162	0.262692	0.0438903	0.186676	0.055
10	0.0329553	0.175704	0.0384691	0.154791	0.0383618	0.210949	0.0344346	0.185068	0.042
11	0.0366533	0.197246	0.0382918	0.168713	0.0451951	0.20523	0.0372095	0.177725	0.046
12	0.0425926	0.178697	0.035966	0.163082	0.0469177	0.21568	0.0400774	0.195603	0.051
13	0.0434686	0.198629	0.0401603	0.148979	0.0495377	0.238204	0.0427372	0.197227	0.053
14	0.0497256	0.211234	0.0601379	0.160485	0.0633223	0.285979	0.0479872	0.225763	0.052

Winstorm damage: processing pipeline

1. Importing necessary packages

2. Reading in the feature datasets

3. Converting to dB

4. Forming target vector

```
"""
Created on Mon Jun 27 22:03:01 2022

@author: oleg.antropov@vtt.fi
"""
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
from sklearn.decomposition import PCA
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import accuracy_score



dtrain=np.loadtxt("C:\\data\\EOtraining\\python\\train.out");
dtest=np.loadtxt("C:\\data\\EOtraining\\python\\test.out");


X_train=10*np.log10(dtrain[:,:24])
X_test=10*np.log10(dtest[:,:24])
y_train=np.ones((200,1),dtype=int)
y_train[100:,:]=0
y_test=np.ones((200,1),dtype=int)
y_test[100:,:]=0
```

Winstorm damage: processing pipeline

5. Creating processing pipeline

6. Fitting the model.

7. Predicting on non-overlapping testing dataset

8. Plotting confusion matrix and calculating accuracies

*Evaluate performance of alternative classification methods,
*Examine role and dependence on PCA components.

```
clf = make_pipeline(StandardScaler(), PCA(n_components=2), SVC(gamma='auto'))


clf.fit(X_train, np.ravel(y_train))

pred = clf.predict(X_test)

disp = plot_confusion_matrix(clf, X_test, y_test, cmap=plt.cm.Blues, normalize=None)

acc=accuracy_score(pred,y_test)

kappa=cohen_kappa_score(pred,y_test)


print('\nPrediction accuracy for the normal test dataset with PCA:')
print('{:.2%}'.format(acc))

print('\nKappa equals:')
print('{:.2%}'.format(kappa))
```

Exercise 2: Forest variable prediction

Forest variable prediction: definition

Input data represents a feature table X of plot-level features calculated using bands of Sentinel-2 images, multitemporal Sentinel-1 composite, ALOS-2 PALSAR-2 mosaic data, and TanDEM-X interferometric height and coherence magnitude.

Study site was located in the Kymenlaakso province of Finland.

Target vector Y contains values of forest variables ['G', 'V', 'D', 'H', 'PINE','SPRUCE', 'BL'].

Feature organization is shown in the Table:

	Sentinel-2							Sentinel-1		ALOS-2 PALSAR-2		TanDEM-X	
	2	3	4	8	5	11	12	VH	VV	HH	HV	CHM	Coh
1	138.8	214.3	123.14	1862.71	423.23	832.57	359.02	-15.2	-8.79	-5.67	-12.34	6.45	0.5614
2	110.1	190.95	123.93	1977.24	390.52	1124.36	507.66	-12.57	-7.73	-5.79	-11.64	3.27	0.5536
3	159.66	267.8	136.61	2276.99	449.41	797.23	327.7	-14.71	-8.68	-4.24	-10.19	4.06	0.7272
4	159.87	249.02	176.36	1392.5	477.12	909.49	444.14	-14.83	-8.35	-5.3	-10.41	4.83	0.6525
5	245.05	359.61	291.87	1855.65	610.65	1286.38	663.24	-14.44	-9.19	-5.13	-10.86	9.55	0.5495
6	176.87	240.04	190.63	1562.21	426.97	910.35	427.46	-14.03	-8.06	-4.48	-11.17	12.4	0.4396

Forest variable prediction: definition

1. Import packages

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.metrics as metrics
from sklearn.preprocessing import StandardScaler
from matplotlib.colors import Normalize
from scipy.interpolate import interpn
from sklearn.linear_model import Lasso, LinearRegression
from sklearn.neighbors import KNeighborsRegressor
```

2. Extract modeled forest variables and features

```
#open files
xtrain=np.loadtxt("C:\\data\\EOtraining\\python\\reg_xtrain.out", delimiter=',');
xtest=np.loadtxt("C:\\data\\EOtraining\\python\\reg_xtest.out", delimiter=',');
ytrain=np.loadtxt("C:\\data\\EOtraining\\python\\reg_ytrain.out", delimiter=',');
ytest=np.loadtxt("C:\\data\\EOtraining\\python\\reg_ytest.out", delimiter=',');

#choose forest variable for modeling
bands=['G', 'V','D','H','PINE','SPRUCE','BL']
var=3
band_name=bands[var]

y_V=ytrain[:,var]
rs=0
re=13
X=xtrain[:,rs:re]
```

3. Form various data feature combinations – test separately
   a) Sentinel-2 bands,
   b) Sentinel-1 and Sentinel-2 bands,
   c) TanDEM-X bands,
   d) All radar bands, etc

Forest variable prediction: definition

4. Scale the data

5. Perform model training

6. Perform prediction on test dataset

7. Calculate accuracies and visualize scatterplots

7. Repeat analysis for another forest variable

```
#scaler
scaler = StandardScaler().fit(X)
X = scaler.transform(X)

neigh = KNeighborsRegressor(n_neighbors=5, weights='distance', algorithm='auto',
                            leaf_size=30, p=1, metric='minkowski')

neigh.fit(X, y_V)

ay_V=ytest[:,var]
aX=xtest[:,rs:re]

aX = scaler.transform(aX)

ay_p=neigh.predict(aX)


mse = metrics.mean_squared_error(ay_V, ay_p)
bias=np.mean(ay_V-ay_p)
rmse = np.sqrt(mse) # or mse**(0.5)
rmsep=rmse/np.mean(ay_V)*100
r2 = metrics.r2_score(ay_V,ay_p)
```