```
# Dirk Bernhardt-Walther, January 2022: dirk.

import os
import pandas as pd
from google.colab import drive
drive.mount("/content/drive", force_remount=True)
os.chdir('/content/drive/MyDrive/Colab Notebooks/PSY3100')
allData = pd.read_csv('fMRI_Scenes/S01_PPA.csv',sep=r',',skipinitialspace = True,index_col='type');
allData
```

```
Mounted at /content/drive
```

type	category	run	vox1	vox2	vox3	vox4	vox5	vox6	vox7	vox8	vox!
lineDrawings	forests	1	-4.463114	3.385811	2.000706	-6.156020	4.001413	6.156020	-1.077303	-0.307801	-1.23120
lineDrawings	forests	1	-5.432294	-0.814844	4.345835	1.629688	5.160679	2.716147	2.987762	-5.975523	-3.25937
lineDrawings	forests	1	-0.543229	2.444532	8.691670	-3.259376	3.530991	1.086459	9.234900	1.901303	3.53099
lineDrawings	forests	1	-2.172918	7.061982	-4.074221	-0.271615	6.518753	10.321359	8.691670	5.160679	-0.81484
lineDrawings	forests	1	-1.358073	1.358073	13.580735	1.358073	7.061982	5.160679	5.432294	0.814844	3.25937
...	...	...	...	...	...	...	...	...	...	...	..
original	mountains	14	-3.396710	1.940977	-6.955168	0.646992	2.426222	1.779229	2.911466	2.749718	-4.52894
original	mountains	14	7.558376	0.419910	11.757474	2.519459	1.889594	0.839820	6.088692	-4.409053	3.77918
original	mountains	14	0.000000	1.358073	11.679432	2.444532	5.703909	8.148441	4.889065	-1.629688	-0.54322
original	mountains	14	0.543229	1.901303	-6.247138	-2.172918	4.617450	2.716147	5.432294	5.432294	-0.54322
original	mountains	14	-1.477095	-0.134281	0.939970	-0.805688	2.014221	3.088472	2.819909	5.371256	-5.90838

672 rows × 345 columns

Split data into training and test

```
photoData = allData.loc['lineDrawings']
leftOutRun = photoData['run'][0]

naturalCategories = ['beaches','forests','mountains']
manmadeCategories = ['city','highways','offices']

testData = photoData.loc[photoData['run'] == leftOutRun]
testLabels = testData['category']
binaryTestLabels = testLabels.replace(to_replace = naturalCategories,value = 'natural')\
                             .replace(to_replace = manmadeCategories,value = 'manmade').to_numpy()
testLabels = testLabels.to_numpy()
testSamples = testData.iloc[:,2:].to_numpy()

trainData = photoData.loc[photoData['run'] != leftOutRun]
trainLabels = trainData['category']
binaryTrainLabels = trainLabels.replace(to_replace = naturalCategories,value = 'natural')\
                               .replace(to_replace = manmadeCategories,value = 'manmade').to_numpy()
trainLabels = trainLabels.to_numpy()
trainSamples = trainData.iloc[:,2:].to_numpy()
```

Train the SVM classifier

```
from sklearn import svm
clf = svm.SVC(kernel='linear')
clf.fit(trainSamples, binaryTrainLabels)
binaryPredictLabels = clf.predict(testSamples)
```

Check the prediction

```
print('Predicted binary labels:')
print(binaryPredictLabels)
```

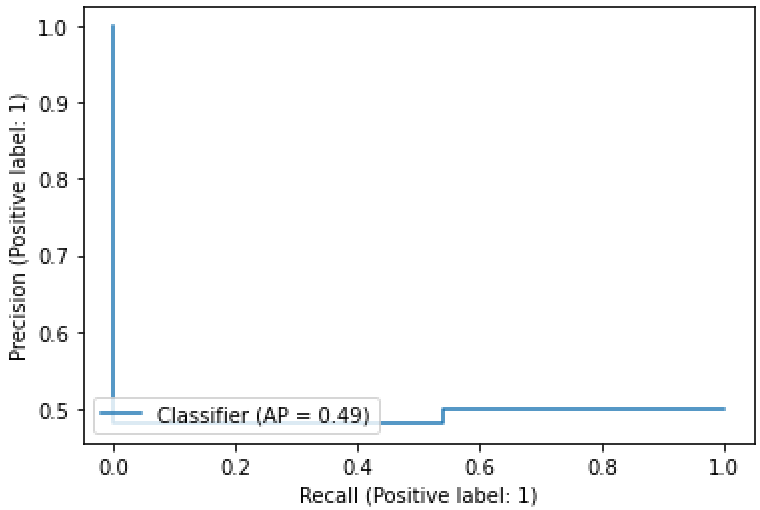
```
print('True binary labels:')
print(binaryTestLabels)
```

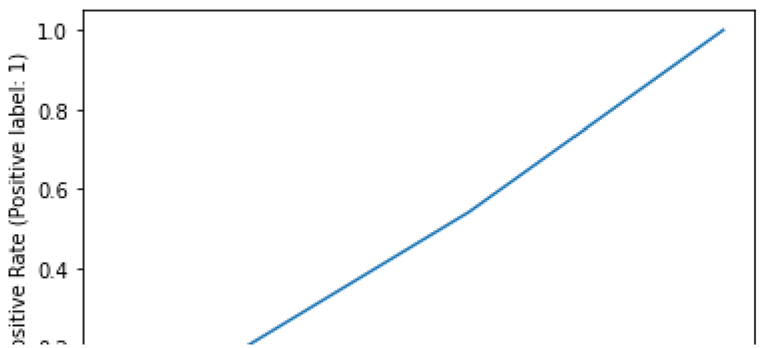
```
Predicted binary labels:
['manmade' 'manmade' 'manmade' 'manmade' 'manmade' 'manmade' 'natural'
 'natural' 'manmade' 'manmade' 'manmade' 'manmade' 'manmade' 'manmade'
 'manmade' 'manmade' 'manmade' 'natural' 'natural' 'manmade' 'manmade'
 'natural' 'natural' 'natural' 'natural' 'manmade' 'natural' 'natural'
 'manmade' 'manmade' 'manmade' 'manmade' 'manmade' 'manmade' 'natural'
 'natural' 'natural' 'natural' 'manmade' 'natural' 'natural' 'natural'
 'natural' 'manmade' 'manmade' 'natural' 'natural' 'natural']
True binary labels:
['natural' 'natural' 'natural' 'natural' 'natural' 'natural' 'natural'
 'natural' 'manmade' 'manmade' 'manmade' 'manmade' 'manmade' 'manmade'
 'manmade' 'manmade' 'natural' 'natural' 'natural' 'natural' 'natural'
 'natural' 'natural' 'natural' 'natural' 'natural' 'natural' 'natural'
 'natural' 'natural' 'natural' 'natural' 'manmade' 'manmade' 'manmade'
 'manmade' 'manmade' 'manmade' 'manmade' 'manmade' 'manmade' 'manmade'
 'manmade' 'manmade' 'manmade' 'manmade' 'manmade' 'manmade']
```

Classification Report

```
import sklearn.metrics as sm
print(sm.classification_report(binaryTestLabels,binaryPredictLabels))

# make binary labels for precision-recall and ROC curves
y_true = []; y_pred = []
for i in range(testLabels.size):
  y_true.append(binaryTestLabels[i] == 'manmade')
  y_pred.append(binaryPredictLabels[i] == 'manmade')

sm.PrecisionRecallDisplay.from_predictions(y_true,y_pred)
sm.RocCurveDisplay.from_predictions(y_true,y_pred)
```

```
              precision    recall  f1-score   support

     manmade       0.48      0.54      0.51        24
     natural       0.48      0.42      0.44        24

    accuracy                           0.48        48
   macro avg       0.48      0.48      0.48        48
weighted avg       0.48      0.48      0.48        48

<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f1cf8639dd0>
```





Now let's try multi-class, one-versus-one

```
clf.fit(trainSamples,trainLabels)  # this is doing one-versus-one
multiPredictLabels = clf.predict(testSamples)
print('Predicted labels:')
```

```
print(multiPredictLabels)
print('True labels:')
print(testLabels)
```

```
Predicted labels:
['offices' 'offices' 'beaches' 'offices' 'highways' 'highways' 'beaches'
 'highways' 'offices' 'offices' 'offices' 'offices' 'offices' 'offices'
 'offices' 'offices' 'mountains' 'beaches' 'beaches' 'highways' 'city'
 'forests' 'beaches' 'mountains' 'mountains' 'highways' 'forests'
 'forests' 'beaches' 'highways' 'offices' 'highways' 'forests' 'city'
 'mountains' 'city' 'forests' 'forests' 'highways' 'beaches' 'city' 'city'
 'city' 'city' 'highways' 'beaches' 'forests' 'beaches']
True labels:
['forests' 'forests' 'forests' 'forests' 'forests' 'forests' 'forests'
 'forests' 'offices' 'offices' 'offices' 'offices' 'offices' 'offices'
 'offices' 'offices' 'mountains' 'mountains' 'mountains' 'mountains'
 'mountains' 'mountains' 'mountains' 'mountains' 'beaches' 'beaches'
 'beaches' 'beaches' 'beaches' 'beaches' 'beaches' 'beaches' 'highways'
 'highways' 'highways' 'highways' 'highways' 'highways' 'highways'
 'highways' 'city' 'city' 'city' 'city' 'city' 'city' 'city' 'city']
```

Classification Report

```
print(sm.classification_report(testLabels,multiPredictLabels))
sm.ConfusionMatrixDisplay.from_predictions(testLabels,multiPredictLabels)
```

```
              precision    recall  f1-score   support

     beaches       0.11      0.12      0.12         8
        city       0.57      0.50      0.53         8
     forests       0.00      0.00      0.00         8
    highways       0.11      0.12      0.12         8
   mountains       0.50      0.25      0.33         8
     offices       0.67      1.00      0.80         8

    accuracy                           0.33        48
   macro avg       0.33      0.33      0.32        48
weighted avg       0.33      0.33      0.32        48
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f1cf85138d0>
```

Trying one-versus-rest

```
lin_clf = svm.LinearSVC()            # this is doing one-versus-rest
lin_clf.fit(trainSamples,trainLabels)
linPredictLabels = lin_clf.predict(testSamples)

print(sm.classification_report(testLabels,linPredictLabels))
sm.ConfusionMatrixDisplay.from_predictions(testLabels,linPredictLabels)
```

```
              precision    recall  f1-score   support

     beaches       0.14      0.12      0.13         8
        city       0.67      0.75      0.71         8
     forests       0.12      0.12      0.12         8
    highways       0.11      0.12      0.12         8
   mountains       0.33      0.12      0.18         8
     offices       0.33      0.50      0.40         8

    accuracy                           0.29        48
   macro avg       0.29      0.29      0.28        48
weighted avg       0.29      0.29      0.28        48

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f1cf7fbb990>
```

Using Feature scaling

```
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
scaled_clf = make_pipeline(StandardScaler(),svm.SVC(kernel='linear'))
scaled_clf.fit(trainSamples,trainLabels)
scaledPredictLabels = scaled_clf.predict(testSamples)
print('Predicted labels:')
print(scaledPredictLabels)
print('True labels:')
print(testLabels)
```

```
Predicted labels:
['offices' 'offices' 'highways' 'offices' 'highways' 'forests' 'beaches'
 'offices' 'mountains' 'offices' 'offices' 'offices' 'offices' 'offices'
 'offices' 'offices' 'highways' 'beaches' 'beaches' 'highways' 'city'
 'forests' 'beaches' 'mountains' 'mountains' 'highways' 'forests'
 'forests' 'beaches' 'highways' 'offices' 'highways' 'forests' 'highways'
 'beaches' 'beaches' 'city' 'forests' 'highways' 'beaches' 'city' 'city'
 'city' 'city' 'highways' 'mountains' 'forests' 'beaches']
True labels:
['forests' 'forests' 'forests' 'forests' 'forests' 'forests' 'forests'
 'forests' 'offices' 'offices' 'offices' 'offices' 'offices' 'offices'
 'offices' 'offices' 'mountains' 'mountains' 'mountains' 'mountains'
```

```
 'mountains' 'mountains' 'mountains' 'mountains' 'beaches' 'beaches'
 'beaches' 'beaches' 'beaches' 'beaches' 'beaches' 'beaches' 'highways'
 'highways' 'highways' 'highways' 'highways' 'highways' 'highways'
 'highways' 'city' 'city' 'city' 'city' 'city' 'city' 'city' 'city']
```

Classification Report

```
print(sm.classification_report(testLabels,scaledPredictLabels))
sm.ConfusionMatrixDisplay.from_predictions(testLabels,scaledPredictLabels)
```

Explore PCA

```
        beaches       0.11      0.12      0.12         8
```

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(trainSamples)
```

```
PCA()
      accuracy                           0.33        48
```

Determine the cumulative amount of variance explained

```
import numpy as np

explainedVar = np.cumsum(pca.explained_variance_ratio_ * 100)
print(explainedVar)

expl80idx = np.argwhere(explainedVar >= 80)[0][0]
expl80var = explainedVar[expl80idx]
expl90idx = np.argwhere(explainedVar >= 90)[0][0]
expl90var = explainedVar[expl90idx]
```

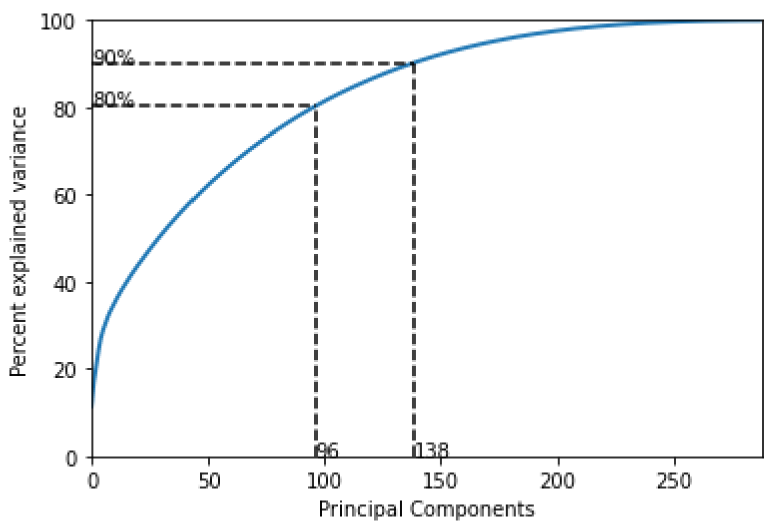
```
[ 11.49209395  16.92399045  20.63513646  24.11125507  26.96805791
  28.90242354  30.4263271   31.8746449   33.11155483  34.19474503
  35.25082679  36.2474679   37.19901505  38.12666263  38.98815656
  39.83606987  40.6750485   41.49579877  42.2785026   43.0489144
  43.79873112  44.53582629  45.25971007  45.97963202  46.6942942
  47.40200349  48.09656647  48.78390422  49.45894608  50.12293259
  50.77674125  51.42614329  52.06589827  52.69133563  53.30682821
  53.91741965  54.52430121  55.12714594  55.71929126  56.30153028
  56.8768468   57.44220791  57.99929078  58.55103877  59.0982878
  59.63973793  60.17419475  60.70045509  61.22560504  61.7418741
  62.25334354  62.7589599   63.25380247  63.74345486  64.22881735
  64.70262702  65.17382087  65.63685421  66.09704821  66.54953716
  66.99967377  67.44812581  67.88857409  68.3237616   68.75562446
  69.18314828  69.60375141  70.02101053  70.43614258  70.84133108
  71.24574854  71.64203318  72.03486749  72.42451227  72.8113924
  73.19055002  73.56695667  73.9407792   74.30282014  74.66332303
```

```
 75.02040164  75.37632852  75.72361149  76.06991536  76.41399907
 76.75685574  77.09932356  77.4286384   77.75659071  78.0817858
 78.40219495  78.72061854  79.02972996  79.33776728  79.64058445
 79.94189708  80.24062191  80.53569014  80.83035795  81.12229969
 81.41319119  81.69880597  81.98072447  82.25815426  82.52960266
 82.79900615  83.06364944  83.32601276  83.58804686  83.8465868
 84.10177201  84.35198917  84.59992805  84.84551571  85.08729151
 85.3278448   85.56486181  85.8003929   86.03265659  86.26107345
 86.48848092  86.71503891  86.93739713  87.15829286  87.37641153
 87.59100704  87.80447431  88.01639815  88.22412562  88.42850499
 88.62942657  88.8277084   89.02409073  89.21642325  89.40849048
 89.59689614  89.78423025  89.9691553   90.15338035  90.332625
 90.50933106  90.68528385  90.85819337  91.03067299  91.19834403
 91.36461917  91.5298651   91.69187477  91.85360805  92.0090925
 92.16413912  92.31475687  92.46510063  92.61380836  92.76117074
 92.90507627  93.04677436  93.1879659   93.3260791   93.46149808
 93.59520747  93.72852951  93.86020202  93.9875366   94.11404621
 94.24008657  94.3636304   94.4862647   94.6080818   94.72593762
 94.84251446  94.95675518  95.06792478  95.17716416  95.28542024
 95.39297401  95.50049552  95.6045959   95.70800542  95.81003067
 95.911494    96.0089594   96.1059645   96.20144186  96.29484866
 96.38627152  96.47542894  96.56203112  96.6483684   96.73369421
 96.81896034  96.90145545  96.98357113  97.06247438  97.14069252
 97.21818847  97.2929185   97.36701858  97.43915694  97.50987531
 97.5798327   97.64907244  97.71721993  97.78411826  97.85038552
 97.91604173  97.97918377  98.04126179  98.10039513  98.15747139
 98.21434451  98.26882209  98.32260006  98.37480256  98.42651907
 98.47811175  98.52894591  98.57768251  98.62514816  98.67202274
 98.71820259  98.76365873  98.80749301  98.84973644  98.89181928
 98.93155649  98.97046831  99.00851625  99.0458596   99.08243139
 99.11805376  99.15345891  99.18798506  99.22154696  99.25286198
 99.28338936  99.31318811  99.34242241  99.3709743   99.39847496
 99.4255309   99.4518263   99.47733258  99.50202216  99.52547145
 99.54805303  99.57000129  99.59129275  99.6119466   99.63204931
 99.65104938  99.66945628  99.68732642  99.70454417  99.72146427
 99.73797707  99.75377211  99.7692679   99.7838176   99.79827791
 99.81176412  99.82516724  99.83754119  99.84935299  99.86090638
 99.87202002  99.8823301   99.89223332  99.90189233  99.91101784
 99.91995835  99.92842079  99.93675547  99.94443638  99.95110936
 99.95736821  99.96331867  99.9687872   99.97408601  99.97880292
 99.98311718  99.9864675   99.9894966   99.99247537  99.99519159
 99.99766578 100.         100.        ]
```

Plot explained Variance

```
import matplotlib.pyplot as plt

plt.plot(range(len(explainedVar)), explainedVar, linewidth=2)

plt.plot([0,expl80idx],[expl80var,expl80var], color='black', linestyle='--')
plt.plot([expl80idx,expl80idx],[0,expl80var], color='black', linestyle='--')
plt.text(0,expl80var,'80%')
plt.text(expl80idx,0,str(expl80idx))

plt.plot([0,expl90idx],[expl90var,expl90var], color='black', linestyle='--')
plt.plot([expl90idx,expl90idx],[0,expl90var], color='black', linestyle='--')
plt.text(0,expl90var,'90%')
plt.text(expl90idx,0,str(expl90idx))

plt.xlabel('Principal Components')
plt.ylabel('Percent explained variance')
plt.xlim([0,len(explainedVar)])
plt.ylim([0,100])
```

```
(0.0, 100.0)
```



Now use PCA in the classification

```
pca80clf = make_pipeline(StandardScaler(),PCA(n_components=expl80idx+1), svm.SVC(kernel='linear'))
pca80clf.fit(trainSamples,trainLabels)
pca80predictLabels = pca80clf.predict(testSamples)
print(sm.classification_report(testLabels,pca80predictLabels))
sm.ConfusionMatrixDisplay.from_predictions(testLabels,pca80predictLabels)
```

```
              precision    recall  f1-score   support

     beaches       0.20      0.25      0.22         8
        city       0.83      0.62      0.71         8
     forests       0.18      0.25      0.21         8
    highways       0.33      0.25      0.29         8
   mountains       0.33      0.12      0.18         8
     offices       0.50      0.75      0.60         8

    accuracy                           0.38        48
   macro avg       0.40      0.38      0.37        48
weighted avg       0.40      0.38      0.37        48

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f1cf7ca4f90>
```

```
pca90clf = make_pipeline(StandardScaler(),PCA(n_components=expl90idx+1), svm.SVC(kernel='linear'))
pca90clf.fit(trainSamples,trainLabels)
pca90predictLabels = pca90clf.predict(testSamples)
print(sm.classification_report(testLabels,pca90predictLabels))
sm.ConfusionMatrixDisplay.from_predictions(testLabels,pca90predictLabels)
```

```
              precision    recall  f1-score   support

     beaches       0.25      0.25      0.25         8
        city       0.62      0.62      0.62         8
     forests       0.22      0.25      0.24         8
    highways       0.20      0.12      0.15         8
   mountains       0.25      0.12      0.17         8
     offices       0.43      0.75      0.55         8

    accuracy                           0.35        48
   macro avg       0.33      0.35      0.33        48
weighted avg       0.33      0.35      0.33        48

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f1cf7d5ecd0>
```

Try feature selection based on L1 regularization

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
```

```
lasso = LogisticRegression(penalty='l1', solver='liblinear')
featSelect_clf = make_pipeline(StandardScaler(),\
                               SelectFromModel(lasso),\
                               svm.SVC(kernel='linear'))
featSelect_clf.fit(trainSamples,trainLabels)
featSelectPredictLabels = featSelect_clf.predict(testSamples)
print(sm.classification_report(testLabels,featSelectPredictLabels))
sm.ConfusionMatrixDisplay.from_predictions(testLabels,featSelectPredictLabels)
```

```
              precision    recall  f1-score   support

     beaches       0.18      0.25      0.21         8
        city       1.00      0.50      0.67         8
     forests       0.11      0.12      0.12         8
    highways       0.25      0.25      0.25         8
   mountains       0.33      0.12      0.18         8
     offices       0.54      0.88      0.67         8

    accuracy                           0.35        48
   macro avg       0.40      0.35      0.35        48
weighted avg       0.40      0.35      0.35        48
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f1ced23ebd0>
```

Try feature selection based on F-statistic

```
from sklearn.feature_selection import SelectPercentile, f_classif

featSelectF_clf = make_pipeline(StandardScaler(),\
                                SelectPercentile(score_func=f_classif,percentile=20),\
                                svm.SVC(kernel='linear'))
featSelectF_clf.fit(trainSamples,trainLabels)
featSelectF_PredictLabels = featSelectF_clf.predict(testSamples)
print(sm.classification_report(testLabels,featSelectF_PredictLabels))
sm.ConfusionMatrixDisplay.from_predictions(testLabels,featSelectF_PredictLabels)
```

```
              precision    recall  f1-score   support

     beaches       0.33      0.50      0.40         8
        city       0.58      0.88      0.70         8
     forests       0.00      0.00      0.00         8
    highways       0.10      0.12      0.11         8
   mountains       1.00      0.62      0.77         8
     offices       0.33      0.25      0.29         8

    accuracy                           0.40        48
   macro avg       0.39      0.40      0.38        48
weighted avg       0.39      0.40      0.38        48
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f1cf5b6ed50>
```

Leave-one-run-out cross-validation

```
from sklearn.model_selection import GroupKFold

allLabels = photoData['category'].to_numpy()
allSamples = photoData.iloc[:,2:].to_numpy()
runIdx = photoData['run'].to_numpy()

numSplits = len(np.unique(runIdx))
gkf = GroupKFold(n_splits=numSplits)

clf = make_pipeline(StandardScaler(),svm.SVC(kernel='linear')) # one-versus-one

allAcc = 0
foldNum = 1
allPredLabels = [''] * len(allLabels)

for train,test in gkf.split(allSamples,allLabels,groups=runIdx):
  clf.fit(allSamples[train,:],allLabels[train])
  predLabels = clf.predict(allSamples[test,:])

  for i in range(len(test)):
    allPredLabels[test[i]] = predLabels[i]

  acc = sm.accuracy_score(allLabels[test],predLabels)
  print('Fold ' + str(foldNum) + ': Accuracy = ' + str(acc))
  allAcc += acc
  foldNum +=1

meanAcc = allAcc/numSplits
print('Mean accuracy across all folds: ' + str(meanAcc))

print(sm.classification_report(allLabels,allPredLabels))
sm.ConfusionMatrixDisplay.from_predictions(allLabels,allPredLabels,normalize='true')
```

```
Fold 1: Accuracy = 0.22916666666666666
Fold 2: Accuracy = 0.3333333333333333
Fold 3: Accuracy = 0.1875
Fold 4: Accuracy = 0.20833333333333334
Fold 5: Accuracy = 0.3958333333333333
Fold 6: Accuracy = 0.22916666666666666
Fold 7: Accuracy = 0.3333333333333333
Mean accuracy across all folds: 0.27380952380952384
              precision    recall  f1-score   support

     beaches       0.16      0.21      0.18        56
        city       0.27      0.27      0.27        56
     forests       0.23      0.20      0.21        56
    highways       0.12      0.11      0.11        56
   mountains       0.52      0.46      0.49        56
     offices       0.41      0.39      0.40        56

    accuracy                           0.27       336
   macro avg       0.28      0.27      0.28       336
weighted avg       0.28      0.27      0.28       336

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f1cf8d79150>
```