accelerometer-svm-Copy1

November 21, 2017

0.0.1 1. Processing data

```
In [2]: '''
        #Setting up SVM; processing input data in right format as input to svm.
        '''

        from sklearn import svm
        from sklearn import svm, datasets
        from sklearn.model_selection import GridSearchCV

        import numpy as np

        #--------------------------------

        def processData(indata):
            sdata = np.hsplit(indata, np.array([1,4,5,8,9,12,13,16,17,20]))

            timestamps = sdata[0]
        #     print(timestamps)

            odata1 = sdata[1]
            otdata = sdata[2]
            odata2 = (odata1*odata1).sum(axis=1)**0.5
            for i in range(1,5):
        #         print(i)
                odata1 = np.vstack((odata1,sdata[2*i+1]))
                otdata = np.vstack((otdata,sdata[2*i+2]))
                dd = (sdata[2*i+1]*sdata[2*i+1]).sum(axis=1)**0.5
                odata2 = np.concatenate((odata2,dd))

            otdata = np.ravel(otdata)
            odata2 = odata2.reshape(-1,1)

            return odata1, odata2, otdata, timestamps

        #-------------------------------
```

```
indata1 = np.genfromtxt('sitting2.csv',delimiter=',')
indata2 = np.genfromtxt('standing2.csv',delimiter=',')
#print(indata)
#print(indata.shape)

n1 = indata1.shape[0]
n2 = indata2.shape[0]

trtar1 = np.zeros(n1) + 1
trtar2 = np.zeros(n2) + 2

#training data
dtrdata1 = np.vstack((indata1,indata2))

#test data
dtrtar1 = np.concatenate((trtar1,trtar2))

n3 = (n1*9)//10
n4 = (n2*9)//10
print(n1,n2,n3,n4)
trdata3 = indata1[:n3]
ttdata3 = indata1[n3:]

trtar3 = trtar1[:n3]
tttar3 = trtar1[n3:]

trdata4 = indata2[:n4]
ttdata4 = indata2[n4:]

trtar4 = trtar2[:n4]
tttar4 = trtar2[n4:]

#training data
dtrdata2 = np.vstack((trdata3,trdata4))
#print(dtrdata2.shape[0])

#test data
dtrtar2 = np.concatenate((trtar3,trtar4))
#print(dtrtar2.shape[0])

#training data
dttdata2 = np.vstack((ttdata3,ttdata4))
#print(dttdata2.shape[0])

#test data
dtttar2 = np.concatenate((tttar3,tttar4))
#print(dtttar2.shape[0])
```

```
6311 6387 5679 5748
```

0.0.2 2. GridSearchCV to search for best parameters 'C' and 'gamma'

```
In [3]: #using GridSearchCV to check for the best parameter values.

        param_grid = [
          {'C': [1, 10, 100, 1000], 'gamma': [0.3, 0.333, 0.4, 0.5, 0.6, 0.7], 'ke
         ]

        svc = svm.SVC()
        clf = GridSearchCV(svc, param_grid)
        print(clf)
        clf.fit(dtrdata2, dtrtar2)
        print(sorted(clf.cv_results_.keys()))

        scoreval = clf.score(dttdata2,dtttar2)
        print("score:", scoreval)
        print('accuracy:', scoreval*100, '%')
```

```
GridSearchCV(cv=None, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'C': [1, 10, 100, 1000], 'kernel': ['rbf'], 'gamma': [0.3, 0.3
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
['mean_fit_time', 'mean_score_time', 'mean_test_score', 'mean_train_score', 'param_
score: 0.940991345397
accuracy: 94.0991345397 %
```

```
In [4]: clf.get_params()
```

```
Out[4]: {'cv': None,
         'error_score': 'raise',
         'estimator': SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False),
         'estimator__C': 1.0,
         'estimator__cache_size': 200,
         'estimator__class_weight': None,
         'estimator__coef0': 0.0,
         'estimator__decision_function_shape': 'ovr',
         'estimator__degree': 3,
         'estimator__gamma': 'auto',
         'estimator__kernel': 'rbf',
         'estimator__max_iter': -1,
         'estimator__probability': False,
         'estimator__random_state': None,
         'estimator__shrinking': True,
         'estimator__tol': 0.001,
         'estimator__verbose': False,
         'fit_params': None,
         'iid': True,
         'n_jobs': 1,
         'param_grid': [{'C': [1, 10, 100, 1000],
           'gamma': [0.3, 0.333, 0.4, 0.5, 0.6, 0.7],
           'kernel': ['rbf']}],
         'pre_dispatch': '2*n_jobs',
         'refit': True,
         'return_train_score': 'warn',
         'scoring': None,
         'verbose': 0}
```

GridSearchCV with different combinations of the parameters 'C' and 'gamma' results in the best fit SVM model for values 'C' = 1 and 'gamma' = auto. These are the default parameters.

0.0.3 3. Cross validation. Feature set: x, y, z values. 'C'=1, 'gamma'=auto

```
In [5]: #cross validation over 10 diffent combinations of the data set

        from sklearn.model_selection import cross_val_score

        clf = svm.SVC()
        print(clf)

        scores = cross_val_score(clf, dtrdata1, dtrtar1, cv=10)
        print(scores)
        print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
[ 0.68056648  0.97086614  0.98346457  0.99606299  0.9511811   0.94094488
  0.94724409  0.95429472  0.9605989   0.94089835]
Accuracy: 0.93 (+/- 0.17)
```

Cross validation show that the model fitted with paramaeters 'C'=1 and 'gamma'=auto (default parameters) results in a consistently good accuracy model.

0.0.4 4. SVM model fit. Feature set x, y, z values.

```
In [7]: #SVM classifier.  Feature set: X, y,  z values
        from sklearn import svm
        from sklearn import svm, datasets
        from sklearn.model_selection import GridSearchCV

        import numpy as np


        classifier1 = svm.SVC()
        print(classifier1)
        classifier1.fit(dtrdata1,dtrtar1)

        pdata = np.genfromtxt('combined2.csv', delimiter=',')
        pdata2 = np.hsplit(pdata, np.array([2]))

        #print(pdata2)

        yy = classifier1.predict(pdata2[1])
        accur = (yy==2.).sum()
        print(accur, yy.shape[0])
        accurp = (accur/yy.shape[0])*100
        err = 100-accurp
        print("accuracy%, error%: ", accurp, err)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
29789 29813
accuracy%, error%:  99.9194982055 0.0805017945192
```

The SVM model fit with x,y,z values as feature set results prediction error rate 0.08(%)

```
In [ ]:
```