MATH6350_HW4_part2

November 27, 2019

Statistical Learning and Data Mining
Dr. Azencott
Sable Levy

In [85]:
```
import time
start_time = time.time()

import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
from sklearn.model_selection import train_test_split


seed=713

data = pd.read_csv('dataset-har-PUC-Rio-ugulino.csv',
                   sep=';', low_memory=False)

#searching for missing data
cols = [col for col in data.columns if data[col].isnull().any()]
print('columns with missing data:', cols) #no missing data
print('\nThere are no columns with missing data.')

data['user'].nunique() #There are 4 unique users/test subjects


classes = data['class'].value_counts().sort_values(ascending=False)
print('\nAll classes:\n',classes)
#showing classes that are not in the top three largest
classes = classes[3:]
print('\nClasses to exclude:\n',classes)
#dropping cases where class is not in top 2,
#i.e. class = standingup or sittingdown
data = data[data['class'] != 'standingup']
data = data[data['class'] != 'sittingdown']
classes = data['class'].value_counts().sort_values(ascending=False);
data.reset_index(drop=True, inplace=True)
```

```
#fixing row 122076, which is erroniously coded as datetime
data.loc[122076, 'z4'] = -144
data['z4']=data['z4'].astype(float)

data_num = data.select_dtypes(exclude = 'object')
data_num = data_num.drop(['age','weight'], axis = 1, inplace = False)

#Step3 : Center and Rescale the whole RDS so that each feature will then
#have global mean = 0 and global stand. dev. =1
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data_scaled = pd.DataFrame(sc.fit_transform(data_num),
                           columns=data_num.columns) #we lose column names

y_categoric = data['class']

#Getting feature statistics per class
data_num_with_y=data_num.join(y_categoric)
CL1_unscaled_y=data_num_with_y[data_num_with_y['class']=='sitting']
CL2_unscaled_y=data_num_with_y[data_num_with_y['class']=='standing']
CL3_unscaled_y=data_num_with_y[data_num_with_y['class']=='walking']

CL1_unscaled=CL1_unscaled_y.drop(['class'], axis = 1)
CL2_unscaled=CL2_unscaled_y.drop(['class'], axis = 1)
CL3_unscaled=CL3_unscaled_y.drop(['class'], axis = 1)

def feature_stats(df1,df2,df3):
    for col in df1.columns:
        print('\nMean of', col, 'in CL1 = {:.2f}'.format(df1[col].mean()))
        print('Std of', col, 'in CL1 ={:.2f}'.format(df1[col].std()))
        print('Mean of', col, 'in CL2 = {:.2f}'.format(df2[col].mean()))
        print('Std of', col, 'in CL2 ={:.2f}'.format(df2[col].std()))
        print('Mean of', col, 'in CL3 = {:.2f}'.format(df3[col].mean()))
        print('Std of', col, 'in CL3 ={:.2f}'.format(df3[col].std()))

print('\n\nMean and std of all features within CL1, CL2, and CL3:')
feature_stats(CL1_unscaled,CL2_unscaled,CL3_unscaled)



#Splitting into X data and target y
#dropping target column, and pol(x) column
X = data_scaled
y = data['class']

Xy=X.join(y)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=seed)

#checking proportion of each class in test set
```

```python
    m1 = 0; m2=0; m3=0
    for i in range (len(y_test)):
        if y_test.iloc[i] == 'sitting': #1
            m1+= 1
        elif y_test.iloc[i] == 'standing': #2
            m2+=1
        elif y_test.iloc[i] == 'walking': #3
            m3+=1

    #size of test and training set.
    print('\nSize of training set:',len(X_train))
    print('Size of test set:',len(X_test))
    #verifying that the class ratios in the test set are roughly equal
    print('\nClass ratios in test set:')
    print('\nCl(sitting): {:.2f}, Cl(standing): {:.2f}, Cl(walking): {:.2f}'.format(m1/len(y_test)

    #Creating appropriate classes
    CL1 = Xy[Xy['class'] == 'sitting']
    CL2 = Xy[Xy['class'] == 'standing']
    CL3 = Xy[Xy['class'] == 'walking']

    #Merging classes
    CL12=pd.concat([CL1,CL2]).sort_index()
    CL13=pd.concat([CL1,CL3]).sort_index()
    CL23=pd.concat([CL2,CL3]).sort_index()

    end_time=time.time()
    total_time = round(end_time - start_time,3)
    print('\nTime:', total_time, 'seconds.')
```

```
columns with missing data: []

There are no columns with missing data.

All classes:
 sitting        50631
standing       47370
walking        43390
standingup     12415
sittingdown    11827
Name: class, dtype: int64

Classes to exclude:
 standingup     12415
sittingdown    11827
Name: class, dtype: int64


Mean and std of all features within CL1, CL2, and CL3:

Mean of x1 in CL1 = -7.14
Std of x1 in CL1 =12.75
Mean of x1 in CL2 = -6.49
Std of x1 in CL2 =4.78
```

```
Mean of x1 in CL3 = -7.91
Std of x1 in CL3 =14.20

Mean of y1 in CL1 = 65.99
Std of y1 in CL1 =25.30
Mean of y1 in CL2 = 97.75
Std of y1 in CL2 =5.06
Mean of y1 in CL3 = 100.89
Std of y1 in CL3 =20.99

Mean of z1 in CL1 = -49.53
Std of z1 in CL1 =25.04
Mean of z1 in CL2 = -106.78
Std of z1 in CL2 =21.04
Mean of z1 in CL3 = -115.45
Std of z1 in CL3 =19.97

Mean of x2 in CL1 = -58.92
Std of x2 in CL1 =90.53
Mean of x2 in CL2 = -18.57
Std of x2 in CL2 =108.41
Mean of x2 in CL3 = -191.40
Std of x2 in CL3 =231.27

Mean of y2 in CL1 = -55.17
Std of y2 in CL1 =116.25
Mean of y2 in CL2 = 53.83
Std of y2 in CL2 =128.36
Mean of y2 in CL3 = -153.77
Std of y2 in CL3 =285.42

Mean of z2 in CL1 = -87.20
Std of z2 in CL1 =134.36
Mean of z2 in CL2 = -144.93
Std of z2 in CL2 =106.41
Mean of z2 in CL3 = -311.59
Std of z2 in CL3 =239.26

Mean of x3 in CL1 = 23.40
Std of x3 in CL1 =42.22
Mean of x3 in CL2 = 23.00
Std of x3 in CL2 =20.45
Mean of x3 in CL3 = 13.81
Std of x3 in CL3 =62.84

Mean of y3 in CL1 = 88.48
Std of y3 in CL1 =23.42
Mean of y3 in CL2 = 108.12
Std of y3 in CL2 =27.01
Mean of y3 in CL3 = 132.44
Std of y3 in CL3 =51.78

Mean of z3 in CL1 = -95.39
Std of z3 in CL1 =21.21
```

```
Mean of z3 in CL2 = -87.98
Std of z3 in CL2 =23.96
Mean of z3 in CL3 = -96.48
Std of z3 in CL3 =48.86

Mean of x4 in CL1 = -131.73
Std of x4 in CL1 =32.13
Mean of x4 in CL2 = -178.18
Std of x4 in CL2 =17.63
Mean of x4 in CL3 = -185.84
Std of x4 in CL3 =24.56

Mean of y4 in CL1 = -109.84
Std of y4 in CL1 =18.01
Mean of y4 in CL2 = -85.32
Std of y4 in CL2 =10.84
Mean of y4 in CL3 = -79.63
Std of y4 in CL3 =17.64

Mean of z4 in CL1 = -161.98
Std of z4 in CL1 =12.01
Mean of z4 in CL2 = -157.32
Std of z4 in CL2 =7.11
Mean of z4 in CL3 = -166.26
Std of z4 in CL3 =11.82

Size of training set: 113112
Size of test set: 28279

Class ratios in test set:

Cl(sitting): 0.36, Cl(standing): 0.33, Cl(walking): 0.31

Time: 1.025 seconds.
```

```
In [81]: start_time = time.time()

         #Defining SVM class
         class SVM:
             def __init__(self, kernel, parameters, X, y, rand_state):
                 self.kernel = kernel
                 self.parameters = parameters
                 self.rand_state=rand_state
                 self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X,
                             y, test_size=0.2, random_state=self.rand_state)
                 self.best_cost = None
                 self.best_gamma=None
                 self.predictions_test=None
                 self.predictions_train=None

             #To display descriptive analytics
             def analytics(self):
                 #selecting the proper kernel
```

```python
if self.kernel == 'linear':
    clf = svm.SVC(kernel=self.kernel,
                  C=self.parameters['SVM__C'][0],
                  random_state=seed)
elif self.kernel == 'rbf':
    clf = svm.SVC(kernel=self.kernel, C=self.parameters['SVM__C'][0],
                  decision_function_shape='ovr',
                  gamma=self.parameters['SVM__gamma'][0], random_state=seed)
elif self.kernel == 'poly':
    clf = svm.SVC(kernel=self.kernel, C=self.parameters['SVM__C'][0],
                  decision_function_shape='ovr',
                  degree=self.parameters['degree'][0],
                  coef0=self.parameters['SVM__coef0'][0], gamma='auto',
                  random_state=seed)
#fitting pre-scaled data
clf.fit(self.X_train, self.y_train)
#predictions on test and training set
predictions_test = clf.predict(self.X_test)
predictions_train = clf.predict(self.X_train)

#finding number of support vectors (sum of number from each class)
S = (clf.n_support_[0])+(clf.n_support_[1])
print("Number of support vectors in training set:", S)
SV_ratio = S/len(self.X_train)
print("Ratio of support vectors in training set = %3.3f" %(SV_ratio))
#percentages of correct predictions on test and training sets
print("Test set prediction accuracy = %3.3f"
      %(clf.score(self.X_test,self.y_test)))
print("Training set prediction accuracy = %3.3f"
      %(clf.score(self.X_train,self.y_train)))
#printing confusion matrices for both sets
cm_test = pd.DataFrame(confusion_matrix(predictions_test, self.y_test))
cm_train = pd.DataFrame(confusion_matrix(predictions_train, self.y_train))
print('Test Set Confusion Matrix: \n{}'.format(cm_test))
print('Training Set Confusion Matrix: \n{}'.format(cm_train))
#confustion matrices in % form:
cm_test_p = cm_test.copy()
cm_train_p = cm_train.copy()
for i in list(cm_test.index):
    cm_test_p.iloc[i] = \
    cm_test.iloc[i].apply(lambda x: x/sum(cm_test.iloc[i]))
    cm_train_p.iloc[i] =  \
    cm_train.iloc[i].apply(lambda x: x/sum(cm_train.iloc[i]))
print('Test Set Confusion Matrix by %: \n{}'.format(cm_test_p.round(3)))
print('Training Set Confusion Matrix by %: \n{}'.format(cm_train_p.round(3)))
#confusion matrices in 95% CI form
print('Test Set Confusion Matrix by 95% CI: \n{}'\
      .format(cm_test_p.applymap(self.get_CI)))
print('Training Set Confusion Matrix by 95% CI: \n{}'\
      .format(cm_train_p.applymap(self.get_CI)))

#computing performance of testing and training sets, with 95% CI
p_test = clf.score(self.X_test,self.y_test)
p_train = clf.score(self.X_train,self.y_train)
```

```python
        print("95 percent CI for test set performance: %3.3f, %3.3f" \
              %(self.get_CI(p_test)))
        print("95 percent CI for training set performance: %3.3f, %3.3f" \
              %(self.get_CI(p_train)))

        self.clf = clf
        self.predictions_test=predictions_test
        self.predictions_train=predictions_train

    #Parameter optimization and displaying analytical results
    def param_optimize(self):
        if self.kernel == 'linear':
            parameters={k: self.parameters[k] for k in ['SVM__C']}
            clf = svm.SVC(kernel=self.kernel,
                          C=parameters['SVM__C'][0], random_state=seed)
        elif self.kernel == 'rbf':
            parameters={k: self.parameters[k] for k in ['SVM__C','SVM__gamma']}
            clf = svm.SVC(kernel=self.kernel, C=parameters['SVM__C'][0],
                          gamma=parameters['SVM__gamma'][0], random_state=seed)
        elif self.kernel == 'poly':
            parameters={k: self.parameters[k] for k in ['SVM__C','SVM__coef0']}
            clf = svm.SVC(kernel=self.kernel, C=parameters['SVM__C'][0],
                          degree=self.parameters['degree'][0],
                          coef0=parameters['SVM__coef0'][0],
                          gamma='auto',
                          random_state=seed)
        else:
            return("Error, invalid kernel name.")

        steps = [('scaler', StandardScaler()), ('SVM', clf)]
        pipeline = Pipeline(steps)
        #Iterating through all parameter combinations to find best performance parameters
        grid = GridSearchCV(pipeline, param_grid=parameters, cv=10)
        #uses 10-fold cross validation
        grid.fit(self.X_train, self.y_train)
        print("\nBest parameters from list of options:", grid.best_params_)
        #best parameters
        best_cost = grid.best_params_.get('SVM__C',None)
        best_gamma = grid.best_params_.get('SVM__gamma',None)
        best_a = grid.best_params_.get('SVM__coef0',None)
        #using best parameters to update svm classifier
        if self.kernel == 'linear':
            clf_best_params = svm.SVC(kernel=self.kernel, C=best_cost,
                                      random_state=seed)
        elif self.kernel == 'rbf':
            clf_best_params = svm.SVC(kernel=self.kernel, C=best_cost,
                          gamma=best_gamma, random_state=seed)
        elif self.kernel == 'poly':
            clf_best_params = svm.SVC(kernel=self.kernel,
                C=best_cost, degree=self.parameters['degree'][0], coef0=best_a,
                                      gamma='auto', random_state=seed)
        else:
            return("Error, invalid kernel name.")
```

```
        clf_best_params.fit(self.X_train, self.y_train)
        predictions_test = clf_best_params.predict(self.X_test)
        predictions_train = clf_best_params.predict(self.X_train)
       #finding number of support vectors (sum of number from each class)
        S = (clf_best_params.n_support_[0])+(clf_best_params.n_support_[1])
        print("Number of support vectors in training set:", S)
        SV_ratio = S/len(self.X_train)
        print("Ratio of support vectors in training set = %3.3f" %(SV_ratio))
        #percentages of correct predictions
        print("Test set accuracy = %3.3f"\
              %(clf_best_params.score(self.X_test,self.y_test)))
        print("Training set accuracy = %3.3f" \
              %(clf_best_params.score(self.X_train,self.y_train)))
        #printing confustion matrices for both sets
        cm_test = pd.DataFrame(confusion_matrix(predictions_test, self.y_test))
        cm_train = pd.DataFrame(confusion_matrix(predictions_train, self.y_train))
        print('Test Set Confusion Matrix: \n{}'.format(cm_test))
        print('Training Set Confusion Matrix: \n{}'.format(cm_train))
        #confustion matrices in % form:
        cm_test_p = cm_test.copy()
        cm_train_p = cm_train.copy()
        for i in list(cm_test.index):
            cm_test_p.iloc[i] =  cm_test.iloc[i].apply(lambda x: x/sum(cm_test.iloc[i]))
            cm_train_p.iloc[i] =  cm_train.iloc[i].apply(lambda x: x/sum(cm_train.iloc[i]))
        print('Test Set Confusion Matrix by %: \n{}'.format(cm_test_p.round(3)))
        print('Training Set Confusion Matrix by %: \n{}'.format(cm_train_p.round(3)))

        #confusion matrices in 95% CI form
        print('Test Set Confusion Matrix by 95% CI: \n{}'\
              .format(cm_test_p.applymap(self.get_CI)))
        print('Training Set Confusion Matrix by 95% CI: \n{}'\
              .format(cm_train_p.applymap(self.get_CI)))

        #computing performance of testing and training sets, with 95% CI
        p_test = clf_best_params.score(self.X_test,self.y_test)
        p_train = clf_best_params.score(self.X_train,self.y_train)

        print("95 percent CI for test set performance: %3.3f, %3.3f"\
              %(self.get_CI(p_test)))
        print("95 percent CI for training set performance: %3.3f, %3.3f"\
              %(self.get_CI(p_train)))

        self.best_cost=best_cost
        self.best_gamma=best_gamma
        #self.clf = clf_best_params

    def get_CI(self,p):
        self.p=p
        std = np.sqrt(p*(1-p)/len(self.X_test)/2)
        CI=round(p-(1.96*std),3),round(p+(1.96*std),3)
        return(CI)

    def get_best_cost(self):
```

```
            return(self.best_cost)

        def get_best_gamma(self):
            return(self.best_gamma)

        def plot(self, df, n):
            Z = df.iloc[:n]['pol(x)']
            Y = self.clf.decision_function(X)
            plt.figure(figsize=(9, 6))
            plt.scatter(Z, Y, marker='.',color='crimson',alpha=.5)
            plt.grid(True)
            plt.xlabel("Poly(X)")
            plt.ylabel("SVM(X)")
            plt.title('Poly(X) vs. SVM(X)')
            plt.show()


        def get_train_predictions(self):
            return(self.predictions_train)

        def get_test_predictions(self):
            return(self.predictions_test)

        def get_y_train(self):
            return (self.y_train)

        def get_y_test(self):
            return (self.y_test)


end_time=time.time()
total_time = round(end_time - start_time,3)
print('\nTime:', total_time, 'seconds.')
```

```
Time: 0.001 seconds.
```

0.0.1 Question 2: SVM classification by radial kernel: CL1 vs CL2

```
In [20]: start_time = time.time()
         y=CL12['class'].copy()
         class_to_numerical = {"sitting": 1, "standing": 2}
         y.replace(class_to_numerical, inplace=True)
         y=y.astype('float')
         X=CL12.drop(['class'],axis=1)

         parameters={'SVM__C':[.1,1,5,25], 'SVM__gamma':[.01,.1,.5,1]}
         #'rbf' = radial kernel
         SVM_radial_CL12 = SVM('rbf', parameters, X, y, seed)
         SVM_radial_CL12.param_optimize()

         end_time=time.time()
         total_time = round(end_time - start_time,3)
         print('\nTime:', total_time, 'seconds.')
```

```
Best parameters from list of options: {'SVM__C': 25, 'SVM__gamma': 0.01}
Number of support vectors in training set: 91
Ratio of support vectors in training set = 0.001
Test set accuracy = 1.000
Training set accuracy = 1.000
Test Set Confusion Matrix:
       0     1
0  10115     1
1      0  9485
Training Set Confusion Matrix:
       0      1
0  40516      0
1      0  37884
Test Set Confusion Matrix by %:
     0    1
0  1.0  0.0
1  0.0  1.0
Training Set Confusion Matrix by %:
     0    1
0  1.0  0.0
1  0.0  1.0
Test Set Confusion Matrix by 95% CI:
            0           1
0  (1.0, 1.0)  (0.0, 0.0)
1  (0.0, 0.0)  (1.0, 1.0)
Training Set Confusion Matrix by 95% CI:
            0           1
0  (1.0, 1.0)  (0.0, 0.0)
1  (0.0, 0.0)  (1.0, 1.0)
95 percent CI for test set performance: 1.000, 1.000
95 percent CI for training set performance: 1.000, 1.000

Time: 779.404 seconds.
```

0.0.2 Step 2: Re-evaluation of tuning, CL1 vs CL3:

In [21]:
```
start_time = time.time()

y=CL13['class'].copy()
class_to_numerical = {"sitting":1, 'walking':3}
y.replace(class_to_numerical, inplace=True)
y=y.astype('float')
X=CL13.drop(['class'],axis=1)

parameters={'SVM__C':[.1,1,5,25], 'SVM__gamma':[.01,.1,.5,1]}

SVM_radial_CL13 = SVM('rbf', parameters, X, y, seed)
SVM_radial_CL13.param_optimize()

end_time=time.time()
total_time = round(end_time - start_time,3)
print('\nTime:', total_time, 'seconds.')
```

```
Best parameters from list of options: {'SVM__C': 5, 'SVM__gamma': 0.1}
Number of support vectors in training set: 550
Ratio of support vectors in training set = 0.007
Test set accuracy = 1.000
Training set accuracy = 1.000
Test Set Confusion Matrix:
       0     1
0  10152     1
1      2  8650
Training Set Confusion Matrix:
       0      1
0  40477      0
1      0  34739
Test Set Confusion Matrix by %:
     0    1
0  1.0  0.0
1  0.0  1.0
Training Set Confusion Matrix by %:
     0    1
0  1.0  0.0
1  0.0  1.0
Test Set Confusion Matrix by 95% CI:
            0           1
0  (1.0, 1.0)  (-0.0, 0.0)
1  (0.0, 0.0)   (1.0, 1.0)
Training Set Confusion Matrix by 95% CI:
            0          1
0  (1.0, 1.0)  (0.0, 0.0)
1  (0.0, 0.0)  (1.0, 1.0)
95 percent CI for test set performance: 1.000, 1.000
95 percent CI for training set performance: 1.000, 1.000

Time: 2637.981 seconds.
```

0.0.3 Question 3 : for the largest 3 classes CL1 CL2 CL3 , compute 3 SVMs:

0.0.4 Getting best parameters

In [24]:
```python
start_time = time.time()

#getting best cost from SVM_radial_CL12 and SVM_radial_CL13
best_cost_radial_1=SVM_radial_CL12.get_best_cost()
best_cost_radial_2=SVM_radial_CL13.get_best_cost()

#getting best gamma from SVM_radial_CL12 and SVM_radial_CL13
best_gamma_radial_1=SVM_radial_CL12.get_best_gamma()
best_gamma_radial_2=SVM_radial_CL13.get_best_gamma()

print('Best cost from SVM_radial_CL12:', best_cost_radial_1)
print('Best cost from SVM_radial_CL13:', best_cost_radial_2)
print('Best gamma from SVM_radial_CL12:', best_gamma_radial_1)
print('Best gamma from SVM_radial_CL13:', best_gamma_radial_2)
```

```
#Checking to see if the best parameters are the same
if best_cost_radial_1==best_cost_radial_2:
    best_cost_radial=best_cost_radial_1
    print('Best cost parameters are the same.')
else:
    best_cost_radial=(best_cost_radial_1+best_cost_radial_2)/2
    print('Best cost parameters are not the same. Taking average.')

if best_gamma_radial_1==best_gamma_radial_2:
    best_gamma_radial=best_gamma_radial_1
    print('Best gamma parameters are the same.')
else:
    best_gamma_radial=(best_gamma_radial_1+best_gamma_radial_2)/2
    print('Best gamma parameters are not the same. Taking average.')


end_time=time.time()
total_time = round(end_time - start_time,3)
print('\nTime:', total_time, 'seconds.')
```

```
Best cost from SVM_radial_CL12: 25
Best cost from SVM_radial_CL13: 5
Best gamma from SVM_radial_CL12: 0.01
Best gamma from SVM_radial_CL13: 0.1
Best cost parameters are not the same. Taking average.
Best gamma parameters are not the same. Taking average.

Time: 0.001 seconds.
```

0.0.5 SVM1 to classify CL1 vs (not CL1) - using radial kernel

```
In [25]: start_time = time.time()

         class_to_numerical = {"sitting": 1, "standing": 23, 'walking':23}
         y=Xy['class'].copy()
         y.replace(class_to_numerical, inplace=True)
         y=y.astype('float')
         X=Xy.drop(['class'],axis=1)

         parameters={'SVM__C':[best_cost_radial], 'SVM__gamma':[best_gamma_radial]}

         SVM1_radial = SVM('rbf', parameters, X, y, seed)
         SVM1_radial.analytics()

         end_time=time.time()
         total_time = round(end_time - start_time,3)
         print('\nTime:', total_time, 'seconds.')
```

```
Number of support vectors in training set: 272
Ratio of support vectors in training set = 0.002
Test set prediction accuracy = 1.000
Training set prediction accuracy = 1.000
Test Set Confusion Matrix:
       0      1
```

```
0  10123      0
1      2  18154
Training Set Confusion Matrix:
       0      1
0  40506      0
1      0  72606
Test Set Confusion Matrix by %:
     0    1
0  1.0  0.0
1  0.0  1.0
Training Set Confusion Matrix by %:
     0    1
0  1.0  0.0
1  0.0  1.0
Test Set Confusion Matrix by 95% CI:
            0           1
0  (1.0, 1.0)  (0.0, 0.0)
1  (0.0, 0.0)  (1.0, 1.0)
Training Set Confusion Matrix by 95% CI:
            0           1
0  (1.0, 1.0)  (0.0, 0.0)
1  (0.0, 0.0)  (1.0, 1.0)
95 percent CI for test set performance: 1.000, 1.000
95 percent CI for training set performance: 1.000, 1.000

Time: 3.587 seconds.
```

0.0.6 SVM2 to classify CL2 vs (not CL2) - using radial kernel

In [26]:
```python
start_time = time.time()

class_to_numerical = {"sitting": 13, "standing": 2, 'walking':13}
y=Xy['class'].copy()
y.replace(class_to_numerical, inplace=True)
y=y.astype('float')
X=Xy.drop(['class'],axis=1)

parameters={'SVM__C':[best_cost_radial], 'SVM__gamma':[best_gamma_radial]}

SVM2_radial = SVM('rbf', parameters, X, y, seed)
SVM2_radial.analytics()


end_time=time.time()
total_time = round(end_time - start_time,3)
print('\nTime:', total_time, 'seconds.')
```

```
Number of support vectors in training set: 3355
Ratio of support vectors in training set = 0.030
Test set prediction accuracy = 0.993
Training set prediction accuracy = 0.994
Test Set Confusion Matrix:
      0      1
0  9355    162
```

```
1     24  18738
Training Set Confusion Matrix:
       0      1
0  37883    591
1    108  74530
Test Set Confusion Matrix by %:
       0      1
0  0.983  0.017
1  0.001  0.999
Training Set Confusion Matrix by %:
       0      1
0  0.985  0.015
1  0.001  0.999
Test Set Confusion Matrix by 95% CI:
                0               1
0  (0.982, 0.984)  (0.016, 0.018)
1  (0.001, 0.002)  (0.998, 0.999)
Training Set Confusion Matrix by 95% CI:
                0               1
0  (0.984, 0.986)  (0.014, 0.016)
1  (0.001, 0.002)  (0.998, 0.999)
95 percent CI for test set performance: 0.993, 0.994
95 percent CI for training set performance: 0.993, 0.994

Time: 41.062 seconds.
```

0.0.7 SVM3 to classify CL3 vs (not CL3) - using radial kernel

In [27]:
```
start_time = time.time()

class_to_numerical = {"sitting": 12, "standing": 12, 'walking':3}
y=Xy['class'].copy()
y.replace(class_to_numerical, inplace=True)
y=y.astype('float')
X=Xy.drop(['class'],axis=1)

parameters={'SVM__C':[best_cost_radial], 'SVM__gamma':[best_gamma_radial]}

SVM3_radial = SVM('rbf', parameters, X, y, seed)
SVM3_radial.analytics()


end_time=time.time()
total_time = round(end_time - start_time,3)
print('\nTime:', total_time, 'seconds.')
```

```
Number of support vectors in training set: 3758
Ratio of support vectors in training set = 0.033
Test set prediction accuracy = 0.993
Training set prediction accuracy = 0.993
Test Set Confusion Matrix:
      0      1
0  8598     26
1   177  19478
```

```
Training Set Confusion Matrix:
       0      1
0  33949    107
1    666  78390
Test Set Confusion Matrix by %:
       0      1
0  0.997  0.003
1  0.009  0.991
Training Set Confusion Matrix by %:
       0      1
0  0.997  0.003
1  0.008  0.992
Test Set Confusion Matrix by 95% CI:
                0               1
0  (0.997, 0.997)  (0.003, 0.003)
1   (0.008, 0.01)   (0.99, 0.992)
Training Set Confusion Matrix by 95% CI:
                0               1
0  (0.996, 0.997)  (0.003, 0.004)
1  (0.008, 0.009)  (0.991, 0.992)
95 percent CI for test set performance: 0.992, 0.994
95 percent CI for training set performance: 0.992, 0.994

Time: 50.184 seconds.
```

0.0.8 Question 4 : for the largest 3 classes CL1 CL2 CL3 , combine the three SVMs to classify all cases

0.0.9 Training Set:

In [78]:
```
#Combining three SVMs to classify all cases for TRAIN set

start_time = time.time()

#creating empty dataframe, to store reliability, predictions,
#and results of weighted voting
classify_all_radial_train=pd.DataFrame(None)

#Getting class predictions from all three SVMs
pred_train_SVM1_radial = SVM1_radial.get_train_predictions()
pred_train_SVM2_radial = SVM2_radial.get_train_predictions()
pred_train_SVM3_radial = SVM3_radial.get_train_predictions()

#storing class predictions for all three SVMs
classify_all_radial_train['SVM1']=pred_train_SVM1_radial
classify_all_radial_train['SVM2']=pred_train_SVM2_radial
classify_all_radial_train['SVM3']=pred_train_SVM3_radial

#Computing prediction reliability for SVM1: CL1 vs (not CL1)
def reliability_SVM1_train(class_prediction):
    if class_prediction==1:
        rel_sit=1
        rel_stand=0
        rel_walk=0
    if class_prediction==23:
```

```
        rel_sit=0
        rel_stand=1/2
        rel_walk=1/2
    return(rel_sit,rel_stand,rel_walk)

reliability_radial_train1=\
classify_all_radial_train['SVM1'].apply(reliability_SVM1_train)

classify_all_radial_train['SVM1_reli_sit']=\
reliability_radial_train1.apply(lambda x: x[0])
classify_all_radial_train['SVM1_reli_stand']=\
reliability_radial_train1.apply(lambda x: x[1])
classify_all_radial_train['SVM1_reli_walk']=\
reliability_radial_train1.apply(lambda x: x[2])

#Computing prediction reliability for SVM2: CL2 vs (not CL2)
def reliability_SVM2_train(class_prediction):
    if class_prediction==2:
        rel_sit=1
        rel_stand=.985
        rel_walk=0
    if class_prediction==13:
        rel_sit=.999/2
        rel_stand=0
        rel_walk=.999/2
    return(rel_sit,rel_stand,rel_walk)

reliability_radial_train2=\
classify_all_radial_train['SVM2'].apply(reliability_SVM2_train)

classify_all_radial_train['SVM2_reli_sit']=\
reliability_radial_train2.apply(lambda x: x[0])
classify_all_radial_train['SVM2_reli_stand']=\
reliability_radial_train2.apply(lambda x: x[1])
classify_all_radial_train['SVM2_reli_walk']=\
reliability_radial_train2.apply(lambda x: x[2])

#Computing prediction reliability for SVM3: CL3 vs (not CL3)
def reliability_SVM3_train(class_prediction):
    if class_prediction==3:
        rel_sit=0
        rel_stand=0
        rel_walk=.997
    if class_prediction==12:
        rel_sit=.992/2
        rel_stand=.992/2
        rel_walk=0
    return(rel_sit,rel_stand,rel_walk)

reliability_radial_train3=\
classify_all_radial_train['SVM3'].apply(reliability_SVM3_train)

classify_all_radial_train['SVM3_reli_sit']=\
reliability_radial_train3.apply(lambda x: x[0])
```

```
classify_all_radial_train['SVM3_reli_stand']=\
reliability_radial_train3.apply(lambda x: x[1])
classify_all_radial_train['SVM3_reli_walk']=\
reliability_radial_train3.apply(lambda x: x[2])

#Summing the scores to be able to choose the highest one
classify_all_radial_train['score_sit']=\
classify_all_radial_train['SVM1_reli_sit']\
+classify_all_radial_train['SVM2_reli_sit']\
+classify_all_radial_train['SVM3_reli_sit']

classify_all_radial_train['score_stand']=\
classify_all_radial_train['SVM1_reli_stand']\
+classify_all_radial_train['SVM2_reli_stand']\
+classify_all_radial_train['SVM3_reli_stand']

classify_all_radial_train['score_walk']=\
classify_all_radial_train['SVM1_reli_walk']\
+classify_all_radial_train['SVM2_reli_walk']\
+classify_all_radial_train['SVM3_reli_walk']

#Decision function to classify all cases into the highest-scoring class
def decision(row):
    if (row[0]>row[1])&(row[0]>row[2]):
        decision=1
        score=row[0]/sum(row)
    elif (row[1]>row[0])&(row[1]>row[2]):
        decision=2
        score=row[1]/sum(row)
    elif (row[2]>row[0])&(row[2]>row[1]):
        decision=3
        score=row[2]/sum(row)
    return (decision, score)

#subset of the three scores
radial_train_prediction_scores=\
classify_all_radial_train[['score_sit','score_stand','score_walk']]

#applying decision function to the three scores
decision_score_train=radial_train_prediction_scores.apply(decision, axis=1)

classify_all_radial_train['class_decision']=decision_score_train.apply(lambda x:x[0])
#defining reliability as (highest score)/(sum of scores) for each case
classify_all_radial_train['reliability']=decision_score_train.apply(lambda x:x[1])
classify_all_radial_train['true_class']=np.array(y_train)


class_to_numerical = {"sitting": 1, "standing": 2, 'walking':3}
y_true=y_train.copy()
y_true.replace(class_to_numerical, inplace=True)
y_true=pd.Series(y_true.astype('int').tolist(), name='True')

y_pred=pd.Series(classify_all_radial_train['class_decision'],name = 'Predicted')
```

```
classify_all_radial_train['true_class']=np.array(y_true)

#print(classify_all_radial_train.head())

df_confusion = pd.crosstab(y_true, y_pred)
df_conf_norm = round(df_confusion / df_confusion.sum(axis=1),3)
print('Confusion Matrix for Radial Kernel Training Set:\n')
print(df_conf_norm)


end_time=time.time()
total_time = round(end_time - start_time,3)
print('\nTime:', total_time, 'seconds.')
```

```
Confusion Matrix for Radial Kernel Training Set:

Predicted    1      2      3
True
1          1.0  0.000  0.000
2          0.0  0.997  0.003
3          0.0  0.014  0.984

Time: 8.704 seconds.
```

0.0.10 Test Set:

In [90]:

```
#Combining three SVMs to classify all cases for TEST set
start_time = time.time()

#creating empty dataframe, to store reliability and predictions
classify_all_radial_test=pd.DataFrame(None)

#Getting class predictions from all three SVMs
pred_test_SVM1_radial = SVM1_radial.get_test_predictions()
pred_test_SVM2_radial = SVM2_radial.get_test_predictions()
pred_test_SVM3_radial = SVM3_radial.get_test_predictions()

#storing class predictions for all three SVMs
classify_all_radial_test['SVM1']=pred_test_SVM1_radial
classify_all_radial_test['SVM2']=pred_test_SVM2_radial
classify_all_radial_test['SVM3']=pred_test_SVM3_radial

#Computing prediction reliability for SVM1: CL1 vs (not CL1)
def reliability_SVM1_test(class_prediction):
    if class_prediction==1:
        rel_sit=1
        rel_stand=0
        rel_walk=0
    if class_prediction==23:
        rel_sit=0
        rel_stand=1/2
        rel_walk=1/2
    return(rel_sit,rel_stand,rel_walk)
```

```
reliability_radial_test1=\
classify_all_radial_test['SVM1'].apply(reliability_SVM1_test)

classify_all_radial_test['SVM1_reli_sit']=\
reliability_radial_test1.apply(lambda x: x[0])
classify_all_radial_test['SVM1_reli_stand']=\
reliability_radial_test1.apply(lambda x: x[1])
classify_all_radial_test['SVM1_reli_walk']=\
reliability_radial_test1.apply(lambda x: x[2])

#Computing prediction reliability for SVM2: CL2 vs (not CL2)
def reliability_SVM2_test(class_prediction):
    if class_prediction==2:
        rel_sit=1
        rel_stand=.983
        rel_walk=0
    if class_prediction==13:
        rel_sit=.999/2
        rel_stand=0
        rel_walk=.999/2
    return(rel_sit,rel_stand,rel_walk)

reliability_radial_test2=\
classify_all_radial_test['SVM2'].apply(reliability_SVM2_test)

classify_all_radial_test['SVM2_reli_sit']=\
reliability_radial_test2.apply(lambda x: x[0])
classify_all_radial_test['SVM2_reli_stand']=\
reliability_radial_test2.apply(lambda x: x[1])
classify_all_radial_test['SVM2_reli_walk']=\
reliability_radial_test2.apply(lambda x: x[2])

#Computing prediction reliability for SVM3: CL3 vs (not CL3)
def reliability_SVM3_test(class_prediction):
    if class_prediction==3:
        rel_sit=0
        rel_stand=0
        rel_walk=.997
    if class_prediction==12:
        rel_sit=.991/2
        rel_stand=.991/2
        rel_walk=0
    return(rel_sit,rel_stand,rel_walk)

reliability_radial_test3=\
classify_all_radial_test['SVM3'].apply(reliability_SVM3_test)

classify_all_radial_test['SVM3_reli_sit']=\
reliability_radial_test3.apply(lambda x: x[0])
classify_all_radial_test['SVM3_reli_stand']=\
reliability_radial_test3.apply(lambda x: x[1])
classify_all_radial_test['SVM3_reli_walk']=\
reliability_radial_test3.apply(lambda x: x[2])
```

```
#Summing the scores to be able to choose the highest one
classify_all_radial_test['score_sit']=\
classify_all_radial_test['SVM1_reli_sit']\
+classify_all_radial_test['SVM2_reli_sit']\
+classify_all_radial_test['SVM3_reli_sit']

classify_all_radial_test['score_stand']=\
classify_all_radial_test['SVM1_reli_stand']\
+classify_all_radial_test['SVM2_reli_stand']\
+classify_all_radial_test['SVM3_reli_stand']

classify_all_radial_test['score_walk']=\
classify_all_radial_test['SVM1_reli_walk']\
+classify_all_radial_test['SVM2_reli_walk']\
+classify_all_radial_test['SVM3_reli_walk']

#Decision function to classify all cases into the highest-scoring class
def decision(row):
    if (row[0]>row[1])&(row[0]>row[2]):
        decision=1
        score=row[0]/sum(row)
    elif (row[1]>row[0])&(row[1]>row[2]):
        decision=2
        score=row[1]/sum(row)
    elif (row[2]>row[0])&(row[2]>row[1]):
        decision=3
        score=row[2]/sum(row)
    return (decision, score)

#subset of the three scores
radial_test_prediction_scores=\
classify_all_radial_test[['score_sit','score_stand','score_walk']]

#applying decision function to the three scores
decision_score_test=radial_test_prediction_scores.apply(decision, axis=1)

classify_all_radial_test['class_decision']=decision_score_test.apply(lambda x:x[0])
#defining reliability as (highest score)/(sum of scores) for each case
classify_all_radial_test['reliability']=decision_score_test.apply(lambda x:x[1])

class_to_numerical = {"sitting": 1, "standing": 2, 'walking':3}
y_true=y_test.copy()
y_true.replace(class_to_numerical, inplace=True)
y_true=pd.Series(y_true.astype('int').tolist(), name='True')

y_pred=pd.Series(classify_all_radial_test['class_decision'],name = 'Predicted')

classify_all_radial_test['true_class']=np.array(y_true)

#print(classify_all_radial_test.tail(10))

df_confusion = pd.crosstab(y_true, y_pred)
df_conf_norm = round(df_confusion / df_confusion.sum(axis=1),3)
print('Confusion Matrix for Radial Kernel Test Set:\n')
```

```
        print(df_conf_norm)


        end_time=time.time()
        total_time = round(end_time - start_time,3)
        print('\nTime:', total_time, 'seconds.')
```

```
Confusion Matrix for Radial Kernel Test Set:

Predicted    1      2      3
True
1          1.0  0.000  0.000
2          0.0  0.997  0.003
3          0.0  0.016  0.983

Time: 2.22 seconds.
```

0.0.11 Question 5: Repeat the whole preceding procedure using the polynomial kernel

0.0.12 First, we tune cost parameter for classification CL1 vs. CL2

```
In [36]: start_time = time.time()
         y=CL12['class'].copy()
         class_to_numerical = {"sitting": 1, "standing": 2, 'walking':3}
         y.replace(class_to_numerical, inplace=True)
         y=y.astype('float')
         X=CL12.drop(['class'],axis=1)

         parameters={'SVM__C':[.1,1,5,25], 'SVM__coef0':[1], 'degree': [2]}

         SVM_poly_CL12 = SVM('poly', parameters, X, y, seed)
         SVM_poly_CL12.param_optimize()


         end_time=time.time()
         total_time = round(end_time - start_time,3)
         print('\nTime:', total_time, 'seconds.')
```

```
Best parameters from list of options: {'SVM__C': 1, 'SVM__coef0': 1}
Number of support vectors in training set: 118
Ratio of support vectors in training set = 0.002
Test set accuracy = 1.000
Training set accuracy = 1.000
Test Set Confusion Matrix:
       0     1
0  10115     0
1      0  9486
Training Set Confusion Matrix:
       0      1
0  40516      1
1      0  37883
Test Set Confusion Matrix by %:
     0    1
```

```
0  1.0  0.0
1  0.0  1.0
Training Set Confusion Matrix by %:
     0    1
0  1.0  0.0
1  0.0  1.0
Test Set Confusion Matrix by 95% CI:
            0           1
0  (1.0, 1.0)  (0.0, 0.0)
1  (0.0, 0.0)  (1.0, 1.0)
Training Set Confusion Matrix by 95% CI:
            0            1
0  (1.0, 1.0)  (-0.0, 0.0)
1  (0.0, 0.0)   (1.0, 1.0)
95 percent CI for test set performance: 1.000, 1.000
95 percent CI for training set performance: 1.000, 1.000

Time: 21.81 seconds.
```

0.0.13 Step 2: Re-evaluation of tuning cost parameter for classification CL1 vs. CL3

In [37]:
```
start_time = time.time()

y=CL13['class'].copy()
class_to_numerical = {"sitting": 1, "standing": 2, 'walking':3}
y.replace(class_to_numerical, inplace=True)
y=y.astype('float')
X=CL13.drop(['class'],axis=1)

parameters={'SVM__C':[.1,1,5,25], 'SVM__coef0':[1], 'degree': [2]}

SVM_poly_CL13 = SVM('poly', parameters, X, y, seed)
SVM_poly_CL13.param_optimize()


end_time=time.time()
total_time = round(end_time - start_time,3)
print('\nTime:', total_time, 'seconds.')
```

```
Best parameters from list of options: {'SVM__C': 1, 'SVM__coef0': 1}
Number of support vectors in training set: 150
Ratio of support vectors in training set = 0.002
Test set accuracy = 1.000
Training set accuracy = 1.000
Test Set Confusion Matrix:
       0     1
0  10154     1
1      0  8650
Training Set Confusion Matrix:
       0      1
0  40477      0
1      0  34739
Test Set Confusion Matrix by %:
```

```
     0    1
0  1.0  0.0
1  0.0  1.0
Training Set Confusion Matrix by %:
     0    1
0  1.0  0.0
1  0.0  1.0
Test Set Confusion Matrix by 95% CI:
            0            1
0  (1.0, 1.0)  (-0.0, 0.0)
1  (0.0, 0.0)   (1.0, 1.0)
Training Set Confusion Matrix by 95% CI:
            0           1
0  (1.0, 1.0)  (0.0, 0.0)
1  (0.0, 0.0)  (1.0, 1.0)
95 percent CI for test set performance: 1.000, 1.000
95 percent CI for training set performance: 1.000, 1.000

Time: 29.634 seconds.
```

0.0.14 Getting best cost parameter for polynomial kernel:

```
In [38]: start_time = time.time()

         #getting best cost from SVM_poly_CL12 and SVM_poly_CL13
         best_cost_poly_1=SVM_poly_CL12.get_best_cost()
         best_cost_poly_2=SVM_poly_CL13.get_best_cost()

         print('Best cost from SVM_radial_CL12:', best_cost_poly_1)
         print('Best cost from SVM_radial_CL13:', best_cost_poly_2)

         #Checking to see if the best parameters are the same
         if best_cost_poly_1==best_cost_poly_2:
             best_cost_poly=best_cost_poly_1
             print('Best cost parameters are the same.')
         else:
             best_cost_poly=(best_cost_poly_1+best_cost_poly_2)/2
             print('Best cost parameters are not the same.')


         end_time=time.time()
         total_time = round(end_time - start_time,3)
         print('\nTime:', total_time, 'seconds.')
```

```
Best cost from SVM_radial_CL12: 1
Best cost from SVM_radial_CL13: 1
Best cost parameters are the same.

Time: 0.0 seconds.
```

0.0.15 SVM1 to classify CL1 vs (not CL1) - using polynomial kernel

In [40]:
```
start_time = time.time()

class_to_numerical = {"sitting": 1, "standing": 23, 'walking':23}
y=Xy['class'].copy()
y.replace(class_to_numerical, inplace=True)
y=y.astype('float')
X=Xy.drop(['class'],axis=1)

parameters={'SVM__C':[best_cost_poly], 'SVM__coef0':[1], 'degree': [2]}

SVM1_poly = SVM('poly', parameters, X, y, seed)
SVM1_poly.analytics()

end_time=time.time()
total_time = round(end_time - start_time,3)
print('\nTime:', total_time, 'seconds.')
```

```
Number of support vectors in training set: 178
Ratio of support vectors in training set = 0.002
Test set prediction accuracy = 1.000
Training set prediction accuracy = 1.000
Test Set Confusion Matrix:
       0      1
0  10125      1
1      0  18153
Training Set Confusion Matrix:
       0      1
0  40506      0
1      0  72606
Test Set Confusion Matrix by %:
     0    1
0  1.0  0.0
1  0.0  1.0
Training Set Confusion Matrix by %:
     0    1
0  1.0  0.0
1  0.0  1.0
Test Set Confusion Matrix by 95% CI:
            0           1
0  (1.0, 1.0)  (0.0, 0.0)
1  (0.0, 0.0)  (1.0, 1.0)
Training Set Confusion Matrix by 95% CI:
            0           1
0  (1.0, 1.0)  (0.0, 0.0)
1  (0.0, 0.0)  (1.0, 1.0)
95 percent CI for test set performance: 1.000, 1.000
95 percent CI for training set performance: 1.000, 1.000

Time: 2.136 seconds.
```

0.0.16 SVM2 to classify CL2 vs (not CL2) - using polynomial kernel

```
In [41]: start_time = time.time()

         class_to_numerical = {"sitting": 13, "standing": 2, 'walking':13}
         y=Xy['class'].copy()
         y.replace(class_to_numerical, inplace=True)
         y=y.astype('float')
         X=Xy.drop(['class'],axis=1)

         parameters={'SVM__C':[best_cost_poly], 'SVM__coef0':[1], 'degree': [2]}

         SVM2_poly = SVM('poly', parameters, X, y, seed)
         SVM2_poly.analytics()

         end_time=time.time()
         total_time = round(end_time - start_time,3)
         print('\nTime:', total_time, 'seconds.')
```

```
Number of support vectors in training set: 7628
Ratio of support vectors in training set = 0.067
Test set prediction accuracy = 0.985
Training set prediction accuracy = 0.985
Test Set Confusion Matrix:
      0      1
0  9316    369
1    63  18531
Training Set Confusion Matrix:
       0      1
0  37706   1445
1    285  73676
Test Set Confusion Matrix by %:
       0      1
0  0.962  0.038
1  0.003  0.997
Training Set Confusion Matrix by %:
       0      1
0  0.963  0.037
1  0.004  0.996
Test Set Confusion Matrix by 95% CI:
                0               1
0   (0.96, 0.963)   (0.037, 0.04)
1  (0.003, 0.004)  (0.996, 0.997)
Training Set Confusion Matrix by 95% CI:
                0               1
0  (0.962, 0.965)  (0.035, 0.038)
1  (0.003, 0.004)  (0.996, 0.997)
95 percent CI for test set performance: 0.984, 0.986
95 percent CI for training set performance: 0.984, 0.986

Time: 57.253 seconds.
```

0.0.17 SVM3 to classify CL3 vs (not CL3) - using polynomial kernel

```
In [42]: start_time = time.time()

         class_to_numerical = {"sitting": 12, "standing": 12, 'walking':3}
         y=Xy['class'].copy()
         y.replace(class_to_numerical, inplace=True)
         y=y.astype('float')
         X=Xy.drop(['class'],axis=1)

         parameters={'SVM__C':[best_cost_poly], 'SVM__coef0':[1], 'degree': [2]}

         SVM3_poly = SVM('poly', parameters, X, y, seed)
         SVM3_poly.analytics()

         end_time=time.time()
         total_time = round(end_time - start_time,3)
         print('\nTime:', total_time, 'seconds.')
```

```
Number of support vectors in training set: 9411
Ratio of support vectors in training set = 0.083
Test set prediction accuracy = 0.978
Training set prediction accuracy = 0.978
Test Set Confusion Matrix:
      0      1
0  8233     72
1   542  19432
Training Set Confusion Matrix:
       0      1
0  32471    325
1   2144  78172
Test Set Confusion Matrix by %:
       0      1
0  0.991  0.009
1  0.027  0.973
Training Set Confusion Matrix by %:
       0      1
0  0.990  0.010
1  0.027  0.973
Test Set Confusion Matrix by 95% CI:
                0               1
0  (0.991, 0.992)  (0.008, 0.009)
1  (0.026, 0.028)  (0.972, 0.974)
Training Set Confusion Matrix by 95% CI:
                0               1
0  (0.989, 0.991)  (0.009, 0.011)
1  (0.025, 0.028)  (0.972, 0.975)
95 percent CI for test set performance: 0.977, 0.979
95 percent CI for training set performance: 0.977, 0.979

Time: 75.314 seconds.
```

0.0.18 For the largest 3 classes CL1 CL2 CL3 , combine the three Polynomial SVMs to classify all cases

0.0.19 Training set:

In [76]:
```
#Combining three SVMs to classify all cases for TRAIN set

start_time = time.time()

#creating empty dataframe, to store reliability, predictions,
#and results of weighted voting
classify_all_poly_train=pd.DataFrame(None)

#Getting class predictions from all three SVMs
pred_train_SVM1_poly = SVM1_poly.get_train_predictions()
pred_train_SVM2_poly = SVM2_poly.get_train_predictions()
pred_train_SVM3_poly = SVM3_poly.get_train_predictions()

#storing class predictions for all three SVMs
classify_all_poly_train['SVM1']=pred_train_SVM1_poly
classify_all_poly_train['SVM2']=pred_train_SVM2_poly
classify_all_poly_train['SVM3']=pred_train_SVM3_poly

#Computing prediction reliability for SVM1: CL1 vs (not CL1)
def reliability_SVM1_train(class_prediction):
    if class_prediction==1:
        rel_sit=1
        rel_stand=0
        rel_walk=0
    if class_prediction==23:
        rel_sit=0
        rel_stand=1/2
        rel_walk=1/2
    return(rel_sit,rel_stand,rel_walk)

reliability_poly_train1=\
classify_all_poly_train['SVM1'].apply(reliability_SVM1_train)

classify_all_poly_train['SVM1_reli_sit']=\
reliability_poly_train1.apply(lambda x: x[0])
classify_all_poly_train['SVM1_reli_stand']=\
reliability_poly_train1.apply(lambda x: x[1])
classify_all_poly_train['SVM1_reli_walk']=\
reliability_poly_train1.apply(lambda x: x[2])

#Computing prediction reliability for SVM2: CL2 vs (not CL2)
def reliability_SVM2_train(class_prediction):
    if class_prediction==2:
        rel_sit=1
        rel_stand=.963
        rel_walk=0
    if class_prediction==13:
        rel_sit=.996/2
        rel_stand=0
        rel_walk=.996/2
    return(rel_sit,rel_stand,rel_walk)
```

```
reliability_poly_train2=\
classify_all_poly_train['SVM2'].apply(reliability_SVM2_train)

classify_all_poly_train['SVM2_reli_sit']=\
reliability_poly_train2.apply(lambda x: x[0])
classify_all_poly_train['SVM2_reli_stand']=\
reliability_poly_train2.apply(lambda x: x[1])
classify_all_poly_train['SVM2_reli_walk']=\
reliability_poly_train2.apply(lambda x: x[2])

#Computing prediction reliability for SVM3: CL3 vs (not CL3)
def reliability_SVM3_train(class_prediction):
    if class_prediction==3:
        rel_sit=0
        rel_stand=0
        rel_walk=.99
    if class_prediction==12:
        rel_sit=.973/2
        rel_stand=.973/2
        rel_walk=0
    return(rel_sit,rel_stand,rel_walk)

reliability_poly_train3=\
classify_all_poly_train['SVM3'].apply(reliability_SVM3_train)

classify_all_poly_train['SVM3_reli_sit']=\
reliability_poly_train3.apply(lambda x: x[0])
classify_all_poly_train['SVM3_reli_stand']=\
reliability_poly_train3.apply(lambda x: x[1])
classify_all_poly_train['SVM3_reli_walk']=\
reliability_poly_train3.apply(lambda x: x[2])

#Summing the scores to be able to choose the highest one
classify_all_poly_train['score_sit']=\
classify_all_poly_train['SVM1_reli_sit']\
+classify_all_poly_train['SVM2_reli_sit']\
+classify_all_poly_train['SVM3_reli_sit']

classify_all_poly_train['score_stand']=\
classify_all_poly_train['SVM1_reli_stand']\
+classify_all_poly_train['SVM2_reli_stand']\
+classify_all_poly_train['SVM3_reli_stand']

classify_all_poly_train['score_walk']=\
classify_all_poly_train['SVM1_reli_walk']\
+classify_all_poly_train['SVM2_reli_walk']\
+classify_all_poly_train['SVM3_reli_walk']

#Decision function to classify all cases into the highest-scoring class
def decision(row):
    if (row[0]>row[1])&(row[0]>row[2]):
        decision=1
        score=row[0]/sum(row)
```

```
        elif (row[1]>row[0])&(row[1]>row[2]):
            decision=2
            score=row[1]/sum(row)
        elif (row[2]>row[0])&(row[2]>row[1]):
            decision=3
            score=row[2]/sum(row)
        return (decision, score)

    #subset of the three scores
    poly_train_prediction_scores=\
    classify_all_poly_train[['score_sit','score_stand','score_walk']]

    #applying decision function to the three scores
    decision_score_train=poly_train_prediction_scores.apply(decision, axis=1)

    classify_all_poly_train['class_decision']=decision_score_train.apply(lambda x:x[0])
    #defining reliability as (highest score)/(sum of scores) for each case
    classify_all_poly_train['reliability']=decision_score_train.apply(lambda x:x[1])

    class_to_numerical = {"sitting": 1, "standing": 2, 'walking':3}
    y_true=y_train.copy()
    y_true.replace(class_to_numerical, inplace=True)
    y_true=pd.Series(y_true.astype('int').tolist(), name='True')

    y_pred=pd.Series(classify_all_poly_train['class_decision'],name = 'Predicted')

    classify_all_poly_train['true_class']=np.array(y_true)
    #print(classify_all_poly_train.head())


    df_confusion = pd.crosstab(y_true, y_pred)
    df_conf_norm = round(df_confusion / df_confusion.sum(axis=1),3)
    print('Confusion Matrix for Polynomial Kernel Training Set:\n')
    print(df_conf_norm)



    end_time=time.time()
    total_time = round(end_time - start_time,3)
    print('\nTime:', total_time, 'seconds.')
```

```
Confusion Matrix for Polynomial Kernel Training Set:

Predicted    1      2      3
True
1          1.0  0.000  0.000
2          0.0  0.990  0.011
3          0.0  0.034  0.963

Time: 8.652 seconds.
```

0.0.20 Test set:

```
In [75]: #Combining three SVMs to classify all cases for TEST set
         start_time = time.time()

         #creating empty dataframe, to store reliability and predictions
         classify_all_poly_test=pd.DataFrame(None)

         #Getting class predictions from all three SVMs
         pred_test_SVM1_poly = SVM1_poly.get_test_predictions()
         pred_test_SVM2_poly = SVM2_poly.get_test_predictions()
         pred_test_SVM3_poly = SVM3_poly.get_test_predictions()

         #storing class predictions for all three SVMs
         classify_all_poly_test['SVM1']=pred_test_SVM1_poly
         classify_all_poly_test['SVM2']=pred_test_SVM2_poly
         classify_all_poly_test['SVM3']=pred_test_SVM3_poly

         #Computing prediction reliability for SVM1: CL1 vs (not CL1)
         def reliability_SVM1_test(class_prediction):
             if class_prediction==1:
                 rel_sit=1
                 rel_stand=0
                 rel_walk=0
             if class_prediction==23:
                 rel_sit=0
                 rel_stand=1/2
                 rel_walk=1/2
             return(rel_sit,rel_stand,rel_walk)

         reliability_poly_test1=\
         classify_all_poly_test['SVM1'].apply(reliability_SVM1_test)

         classify_all_poly_test['SVM1_reli_sit']=\
         reliability_poly_test1.apply(lambda x: x[0])
         classify_all_poly_test['SVM1_reli_stand']=\
         reliability_poly_test1.apply(lambda x: x[1])
         classify_all_poly_test['SVM1_reli_walk']=\
         reliability_poly_test1.apply(lambda x: x[2])

         #Computing prediction reliability for SVM2: CL2 vs (not CL2)
         def reliability_SVM2_test(class_prediction):
             if class_prediction==2:
                 rel_sit=1
                 rel_stand=.962
                 rel_walk=0
             if class_prediction==13:
                 rel_sit=.997/2
                 rel_stand=0
                 rel_walk=.997/2
             return(rel_sit,rel_stand,rel_walk)

         reliability_poly_test2=\
         classify_all_poly_test['SVM2'].apply(reliability_SVM2_test)
```

```
classify_all_poly_test['SVM2_reli_sit']=\
reliability_poly_test2.apply(lambda x: x[0])
classify_all_poly_test['SVM2_reli_stand']=\
reliability_poly_test2.apply(lambda x: x[1])
classify_all_poly_test['SVM2_reli_walk']=\
reliability_poly_test2.apply(lambda x: x[2])

#Computing prediction reliability for SVM3: CL3 vs (not CL3)
def reliability_SVM3_test(class_prediction):
    if class_prediction==3:
        rel_sit=0
        rel_stand=0
        rel_walk=.991
    if class_prediction==12:
        rel_sit=.973/2
        rel_stand=.973/2
        rel_walk=0
    return(rel_sit,rel_stand,rel_walk)

reliability_poly_test3=\
classify_all_poly_test['SVM3'].apply(reliability_SVM3_test)

classify_all_poly_test['SVM3_reli_sit']=\
reliability_poly_test3.apply(lambda x: x[0])
classify_all_poly_test['SVM3_reli_stand']=\
reliability_poly_test3.apply(lambda x: x[1])
classify_all_poly_test['SVM3_reli_walk']=\
reliability_poly_test3.apply(lambda x: x[2])

#Summing the scores to be able to choose the highest one
classify_all_poly_test['score_sit']=\
classify_all_poly_test['SVM1_reli_sit']\
+classify_all_poly_test['SVM2_reli_sit']\
+classify_all_poly_test['SVM3_reli_sit']

classify_all_poly_test['score_stand']=\
classify_all_poly_test['SVM1_reli_stand']\
+classify_all_poly_test['SVM2_reli_stand']\
+classify_all_poly_test['SVM3_reli_stand']

classify_all_poly_test['score_walk']=\
classify_all_poly_test['SVM1_reli_walk']\
+classify_all_poly_test['SVM2_reli_walk']\
+classify_all_poly_test['SVM3_reli_walk']

#Decision function to classify all cases into the highest-scoring class
def decision(row):
    if (row[0]>row[1])&(row[0]>row[2]):
        decision=1
        score=row[0]/sum(row)
    elif (row[1]>row[0])&(row[1]>row[2]):
        decision=2
        score=row[1]/sum(row)
```

```python
        elif (row[2]>row[0])&(row[2]>row[1]):
            decision=3
            score=row[2]/sum(row)
        return (decision, score)

    #subset of the three scores
    poly_test_prediction_scores=\
    classify_all_poly_test[['score_sit','score_stand','score_walk']]

    #applying decision function to the three scores
    decision_score_test=poly_test_prediction_scores.apply(decision, axis=1)

    classify_all_poly_test['class_decision']=decision_score_test.apply(lambda x:x[0])
    #defining reliability as (highest score)/(sum of scores) for each case
    classify_all_poly_test['reliability']=decision_score_test.apply(lambda x:x[1])

    class_to_numerical = {"sitting": 1, "standing": 2, 'walking':3}
    y_true=y_test.copy()
    y_true.replace(class_to_numerical, inplace=True)
    y_true=pd.Series(y_true.astype('int').tolist(), name='True')

    y_pred=pd.Series(classify_all_poly_test['class_decision'],name = 'Predicted')

    classify_all_poly_test['true_class']=np.array(y_true)

    #print(classify_all_poly_test.head())

    df_confusion = pd.crosstab(y_true, y_pred)
    df_conf_norm = round(df_confusion / df_confusion.sum(axis=1),3)
    print('Confusion Matrix for Polynomial Kernel Test Set:\n')
    print(df_conf_norm)



    end_time=time.time()
    total_time = round(end_time - start_time,3)
    print('\nTime:', total_time, 'seconds.')
```

```
Confusion Matrix for Polynomial Kernel Test Set:

Predicted    1      2      3
True
1          1.0  0.000  0.000
2          0.0  0.991  0.009
3          0.0  0.036  0.962

Time: 2.288 seconds.
```

In [ ]: