CS244 Assignment 5 - Group 8

- Kevin Rothi (leader)
- J.Y. Ku
- John Lanier

Github Repository https://github.com/JBLanier/cs244fall2017/ (https://github.com/JBLanier/cs244fall2017/)

Overview

Our team used Python for this project. The venerable data science library SciKit Learn was the engine of our machine learning. We first parsed the data out of the xlsx file with the xlrd library, partitioned it, broke it up into windowed chunks, and then performed some basic feature engineering which resulted in linearly seperable data. A Support Vector Machine (SVM) was used as our classifier. We compare and contrast the performance of the extracted features against the raw data, and then test the trained learner on data we recorded from our sensor.

Parsing Data

Since the data was delivered in a xlsx file, we had to use a third party library (xlrd) to extract the data from the spreadsheet. The resulting data was was in a list of lists, and we needed to convert the data into a numpy array.

Windowing

This step was very similar to the kind of windowing we performed for assignment 3. The data was partitioned such that each chunk corresponded to a specific time window in the whole dataset. Note that we tried varying window sizes, from 6 to 12 seconds, and the impact of this was negligible on our final result due to the low noise in the data.

Feature Engineering

We computed the standard deviation of the signal magnitude vector (SMV) as a feature to perform machine learning on. We reasoned that the variation in motion as represented by the feature would be an excellent indicator of activity. It turns out that using this feature results in a linearly separable dataset, and the support vector machine had a perfect classification rate using this feature alone. The standard deviation of the motion data has a robust statistical correlation to the activity being performed, at least for this dataset.

Machine Learning

As per instruction, we used a Support Vector Machine (SVM) to build a statistical model between the feature vector and the target vector. The SVM builds a decision boundary which maximally separates the data, and this learner fits the data well (hopefully) without overfitting. It generalizes well since the data exhibits such statistical regularity when the appropraite features are generated.

Engineered Features vs Unconditioned Data

For comparison, we trained on the raw XYZ data. As expected, it performed worse in cross validation since the raw data isn't as separable as the engineered feature. The learner trained on the XYZ data was able to successfully classify the data ~93.8% of the time, compared to the 100% of the time the learner trained on the standard deviation of the SMV performed at.

Testing Real Data

As an exercise, we tested our learner on data we collected from our sensor. The data was generated from a walking person, but the label the SVM assigned to it was a "1" (sleeping). The training data provided for this assignment is at a different scale than the data provided by our sensor. While the provided data lists values of around "6" on some axis during sleeping, our sensor would never provide more than around "1.1" for the same activity. When classifying our own data, we will need to create our own training data from the same source.

In [13]:
```
import numpy as np
import scipy
import math
import xlrd
from sklearn import svm
from sklearn.model_selection import cross_val_score
from sklearn.utils import shuffle
from numpy import genfromtxt

#------------- PULLING IN DATA -----------------
workbook = xlrd.open_workbook('XYZ.xlsx') # data provided for hw
worksheet = workbook.sheet_by_index(0) # need to use this janky library to open the xlsx file

real = genfromtxt('RLdata.csv', delimiter=',') # real world data we collected
real = real[1:,:]
real = np.array(real).reshape(-1, 3)

offset = 1 # avoid the column headers

columns = [] # the matrix to build up
for j, col in enumerate(range(worksheet.ncols)):
    if (j == 0): #skip time column
        continue
```

```
    c = []
    for i, row in enumerate(range(worksheet.nrows)):
        if i < offset: #skip column headers
            continue
        c.append(worksheet.cell_value(i, j))
    columns.append(c)

#print (columns[0]) # EXAMPLE: X values for activity 1
#-----------------------------------------------

#------------- DEFINING UTILITY FUNCTIONS -----------------
def getActivity(activity): # pass in the activity you want to get a matrix for
    return columns[(activity - 1) * 4: ((activity - 1) * 4) + 3]

def getMaxWindow(data, size): # returns the highest window possible --- size is in seconds!
    return math.floor(len(data) / (size * 50))

def getWindow(activityMatrix, winNumber, size): # winNumber is count 0, size is in seconds!
    window = [] # the new matrix to populate
    for col in range(len(activityMatrix)):
        window.append(activityMatrix[col][winNumber * size * 50: (winNumber * size * 50) + (size * 50)])
    return window
#-------------------------------------------------------

#------------- FEATURE EXTRACTION FUNCTIONS -----------------
def getSignalMagnitudeVector(windowedData): # returns an array of the same length with the SMV
    SMV = []; # to populate
    for row in range(len(windowedData[0])):
        SMV.append(math.sqrt(math.pow(windowedData[0][row], 2) +  math.pow(windowedData[1][row], 2) + mat
h.pow(windowedData[2][row], 2)))
    return SMV

def getStandardDeviation(smv): # calculates the std of the smv
    return np.std(smv)
#-------------------------------------------------------

#-----------------------------------------------------------------------
# Now we're going to get into the real substance of the assignment
# The first thing we're going to do is to build of a matrix of
# feature vectors and target (label) vectors...
#-----------------------------------------------------------------------

clf = svm.SVC()
X = []
Y = []
rawXYZData = []
rawTargets = []
realDataFeatures = []

for activity in [1, 2, 3, 4, 5]:
    for window in range(getMaxWindow(columns[0], 6)):
        windowedActivityData = getWindow(getActivity(activity), window, 6)
        for rawDataIndex in range(len(windowedActivityData[0])):
            rawXYZData.append([windowedActivityData[0][rawDataIndex], windowedActivityData[1][rawDataInde
x], windowedActivityData[2][rawDataIndex]])
            rawTargets.append(activity)
        X.append(getStandardDeviation(getSignalMagnitudeVector(windowedActivityData)))
        Y.append(activity)

X = np.array(X).reshape((-1, 1))


realDataFeatures = getStandardDeviation(getSignalMagnitudeVector(real))

#-----------------------------------------------------------------------
# Now we train up a learner and get a 5-fold cross validation score
#-----------------------------------------------------------------------

X, Y = shuffle(X, Y)
rawXYZData, rawTargets = shuffle(rawXYZData, rawTargets)

scores = cross_val_score(clf, rawXYZData, rawTargets)
print('Cross Validation Accuracy using raw features only, X Y Z: {}'.format(np.mean(scores)))

scores = cross_val_score(clf, X, Y, cv=5)
print('Cross Validation Accuracy using standard deviation of signal magnitude vector only: {}'.format(np.
mean(scores)))

clf.fit(X, Y)
print('Classification for 5 seconds of walking using our own sensor\'s data (Shouldn\'t make sense due to
 different scales): {}'.format(clf.predict(realDataFeatures)[0]))
```

```
Cross Validation Accuracy using raw features only, X Y Z: 0.9378666666666667
Cross Validation Accuracy using standard deviation of signal magnitude vector only: 1.0
Classification for 5 seconds of walking using our own sensor's data (Shouldn't make sense due to different
scales): 1
```