CS328 - Assignment 2 Part 1

Activity Recognition

**Group Assignment**

In this assignment, you will collect accelerometer data corresponding to various activities from your device, extract features from this data, and use them to train a classifier. You will then use this classifier to classify activities in real time using sensor data from your device.

Obtain the starter code by accepting the assignment at <https://classroom.github.com/a/ixWhp0Du>.

As part of this assignment, you will modify the given Python scripts (except util.py - do not modify it) and generate some new files. All these should be pushed back to Github. New files need to be added to Git in order to be committed or pushed: run

git add filename

to add each new file separately, or

git add .

to add all files in the folder. Following that, commit and push your code.

Acquiring Labelled Data

You will use the apps provided for A1 Part 3 (step counting) to collect activity data.

You have to classify at least 4 activities, two of which are sitting and walking. Select at least 2 more activities of your choice. You can choose anything you like, but please make sure it’s safe and doable while you have your phone on you (e.g., in your pocket).

Once you select your activities, you will use the same pipeline as that in the last assignment to collect accelerometer data.

1. You will perform each of your chosen activities in sessions.
2. After a session is completed add an extra column in your obtained csv file to indicate the label for your activity. An integer label index will help you when you start building classifiers for the data.
3. Collect at least 5 minutes of data for each activity (more is better). Make sure that you have collected data for all activities for a roughly equal length of time.
4. Once you have finished generating data for all activities, combine (concatenate) the data files into a single CSV. Name this “my-activity-data.csv” and save it in the same folder as your output files. Your training dataset is now ready!

Training a Classifier

**Part 1 - Feature Extraction**

You will first need to extract features from the raw accelerometer signal you collected in the previous step. Note that the same features are extracted for all activities - we compute various measures from the raw signal (from all classes) that we think might be useful in distinguishing *between* these classes.

For simplicity, we separate our feature extraction code into the file features.py. The extract\_features() function takes a window of accelerometer data and returns a feature vector.

* Define your own functions to extract various signal features in features.py (we have defined \_compute\_mean\_features() as an example - this returns the [x\_mean, y\_mean, z\_mean] of the raw signal). Call these functions in the extract\_features() function and append the calculated features to the feature\_vector (appending the mean features is shown as an example). Also append the names of your features to feature\_names.
* Which features should you compute?

## **Category 1 - Statistical Features**

Statistical features are some of the most commonly used and generally do fairly well in practice. These include the mean, variance and the rate of zero- or mean-crossings. The minimum and maximum may be useful, as might the median. For more comprehensive measures, try using [histogram](https://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html#numpy.histogram) features. [Here](https://docs.scipy.org/doc/numpy/reference/routines.statistics.html) is a list of useful stats functions available in numpy that you can experiment with.

**Magnitude Signal:** Note that features can be extracted from each axis as well as from the magnitude of the acceleration. Think about which signals will give you the most information to distinguish between your chosen activities. For this assignment, we ask that you extract features from all axes as well as from the magnitude.

## **Category 2 - FFT Features**

As discussed in the lectures, the dominant frequencies of the signal may be a powerful tool in distinguishing activities. For example, running may exhibit a higher dominant frequency than walking.

Use numpy’s [rfft()](http://docs.scipy.org/doc/numpy-1.11.0/reference/generated/numpy.fft.rfft.html#numpy.fft.rfft) function to compute the real-valued Discrete Fourier Transform. It returns an array of complex values. You can ignore the imaginary components by casting the array to an array of floats using numpy’s astype(float) function. You may want to take the FFT over each axis separately and/or over the magnitude signal.

**Category 3 - Other Features**

**Entropy:** The entropy of a signal may prove valuable in distinguishing between various activities. Recall from the lectures on decision trees that the entropy can be computed as

where, in the case of training a decision tree, is a probability distribution over classes at any given node in the tree. We can apply the same notion of entropy over an accelerometer signal, if we find a discrete distribution over the accelerometer values. This can be done using numpy’s [histogram()](https://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html#numpy.histogram) function over your window. You can specify how many bins you want. Then use the first return value as the distribution to the entropy equation above.

## **Integrating Acceleration:**

You might recall from Physics classes that you can compute the change in velocity from the acceleration over a time period :

If you know the initial velocity, you can then compute the distance travelled as

Applying these functions element-wise to our accelerometer signal, we can derive velocity and distance signals. However, these signals are very sensitive to **drift**. If you try computing these signals, you will notice that the values start getting extraordinarily large.

You can, however, approximate the change in velocity and the distance travelled over your sliding window only.

## **Category 4: Peak Features**

Sometimes the count or location of peaks or troughs in the accelerometer signal can be an indicator of the type of activity being performed. This is basically what you did in assignment A1 to detect steps. Use the peak count over each window as a feature. Or try something like the average duration between peaks in a window.

* How many features do you need to compute?

Extract **at least 4 distinct features** (note that x\_mean, y\_mean and z\_mean all count as a single feature: mean) - you should therefore define a *minimum* of four different functions. You are required to extract **at least one feature each from the above three categories**. The other feature(s) can come from any category.

These functions should be included in features.py and the returned values should be appended to the feature vector in extract\_features().

**Part 2 - Training a Decision Tree Classifier**

Now that you have extracted features from the raw signal, you are ready to train a classifier based on these features. You need to modify activity-classification-train.py.

1. First, list the activities you want to classify between on line 58. When the classifier returns the index of the predicted activity, you’ll use class\_names[index] to get the name of the activity.
2. Since you would be working with windows of data (rather than single data points), each window must have a single class label. For this, we simply use the label of the midpoint of the window - the 10th label in our case since we have a window of size 20. There are perhaps more clever ways, such as discarding windows with ambiguity or taking the majority label, but this choice will not have a very large impact on our performance.
3. Train and evaluate a decision tree classifier on your data. Split your data into training and test sets using 10-fold cross validation:

cv = model\_selection.KFold(n\_splits=10, random\_state=None, shuffle=True)

1. Iterate over these folds as described in the script. The parameters of the decision tree are specified when the classifier is instantiated.

tree = DecisionTreeClassifier(criterion="entropy", max\_depth=3)

This creates a decision tree with a depth of no more than 3 (you can change this parameter if you want to). Make sure to specify that features should be split based on entropy. A comprehensive list of other parameters can be found [here](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier).

1. Use the fit function to train the classifier.
2. Then use the predict function to make predictions on the test data. You can then easily generate a confusion matrix by comparing the true labels with the predicted labels. The scikit-learn library has a function to do this for you:

conf = confusion\_matrix(y\_test, y\_pred)

Then, compute the accuracy, precision and recall for each fold. Output these metrics averaged over all folds.

1. Train a final classifier on **all** of the data and visualize the tree. To visualize the decision tree, use the following call:

export\_graphviz(tree, out\_file='tree.dot', feature\_names = feature\_names)

1. The out\_file parameter specifies the filename and location and the feature\_names parameter is a list of names corresponding to your features. You can then open the .dot file using [Graphviz](http://www.graphviz.org).
2. Once you have installed graphviz on your system, you can use it to generate an image file from the “.dot” file produced by the python code. Run this command:

dot -Tpng tree.dot -o tree.png

The output will be in the file “tree.png,” which you need to include in your final commit.

1. The script is already set up to save the trained classifier to disk using [pickle](https://docs.python.org/2/library/pickle.html). Uncomment lines 107-108 before running your script.

Using your Classifier for Activity Recognition in Test settings

Now that you have trained your classifier, you can use it to test it on actual activity data.

You will follow the pipeline in Assignment 1 Part 3. Create one sensor recording session with various activities in it. Unlike training data collection, you do not have to do separate sessions for each activity.

Perform predictions using the decision tree classifier created using the above training method to classify windows of the signal and add a column for the predicted activity. If you would like, you can also visualize the parts of the signal that correspond to various activities such that they can be identified easily. Color coding the activities is an option but the actual visualization is left to your imagination.

Submission and Grading

**Deadlines:**

Demo deadline: Nov 12 in class (attendance and demo required)

Code submission deadline: Nov 15

Code submission deadline: 11:59 PM (this is when Github Classroom will stop accepting submissions. The extra time after the demo deadline is provided for you to incorporate any feedback you get during the in-class demo, clean up your code if required, and commit and push to Github. Your code will be inspected later to check your implementation).

Your submitted repo must include the modified Python scripts, the CSV data files you generated, the visualization of your decision tree (.dot and .png files), and your pickled classifier (classifier.pickle).

In the demo, you will be asked to show how well your classifier is recognizing the activities you chose by showing the results of the classifier in the test setting.