CSCI 4950/6950 Assignment 4

Note: Recommend to start this assignment early since this is considered a lengthy assignment as compared to the previous assignments.

GitHub Classroom Link: https://classroom.github.com/a/mesfvIVC

Acquiring Labelled Data (20 points)

You will use the Android/iOS app provided for Assignment 3 (Sensor Logger) 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 app to collect accelerometer data.

1. For each of the activities you choose to classify, you will toggle "Accelerometer" on the "Logger" tab and click "Start Recording". (You can turn off the "HTTP Data Push" during the data collection.) Click "Stop Recording" once finished.
2. Collect at least 5 minutes of data for each activity (the longer the better). Make sure that you have collected data for all activities for a roughly equal length of time. Each activity can be multiple recording sessions (Start -> Stop), but each session be only one activity. You should name your session in the "Recordings" tab. (Click the recording, then click "Rename" to assign "walking", "sitting" and etc.)
3. Once you have collected all the data, you can either click "Select" on the "Recordings" tab or export each recording individually. When exporting, choose "Zipped CSV" which will contain all data in CSV format.
4. Copy the zip files to the data directory, then unzip. Rename each **"Accelerometer.csv"** to something like **"Walking-Accelerometer.csv"**. Move all the renamed CSV files to the **data directory**.
5. Go to "labels.py" and change the list to include all the activities.
6. Go to "data-labeling.py" and change the "csv_files" and "activity_list" to match the order of the file name and activity. For example if you have two walking and one running csv files, the csv_files will be ["data/walking1-Accelerometer.csv", "data/walking2-Accelerometer.csv", "data/running-Accelerometer.csv"], and your activity_list will be ["walking", "walking", "running"]. Run the python script, you should get a CSV file named "all_labeled_data.csv". Your training dataset is now ready!

Training a Classifier

Part 1 - Feature Extraction. (40 points)

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://numpy.org/doc/stable/reference/generated/numpy.histogram.html#numpy.histogram) features. Here is a list of useful stats functions available in numpy that you can experiment with: https://numpy.org/doc/stable/reference/routines.statistics.html .

  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()` 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

$$\sum_{p(x)} \quad p(x)\, log(p(x))$$

where, in the case of training a decision tree, p 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()` 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 δv from the acceleration a over a time period δt:

$$\delta v \; = \; a\, \delta t$$

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

$$\delta s \; = \; \frac{1}{2} a \delta t^2 + v_o \delta t$$

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 (40 points)

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 in "labels.py" (which you should have already done during data collection. 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)
   ```

4. 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: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier .

5. Use the `fit` function to train the classifier.

6. 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.

7. 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)
   ```

8. 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.

9. 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.

10. The script is already set up to save the trained classifier to disk using pickle. Uncomment lines before running your script.

Using your Classifier for Activity Recognition in Test settings (20 points)

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

You will follow the previous instructions. 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 signal and add a column for predicted activity. 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 - the requirement is that the activities should be distinguishable.