CS 528 Mobile and Ubiquitous Computing

Lecture 6b: Machine Learning for Ubiquitous Computing

Emmanuel Agu

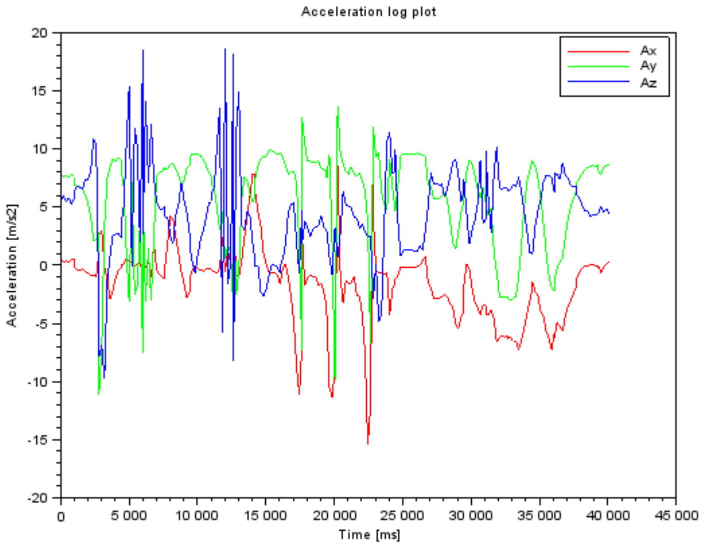
Intuitive Introduction to Machine Learning for Ubiquitous Computing

My Goals in this Section

- If you know machine learning
  - Set off light bulb
  - Projects involving ML?
- If you don't know machine learning
  - Get general idea, how it's used
- Knowledge will also make papers easier to read/understand

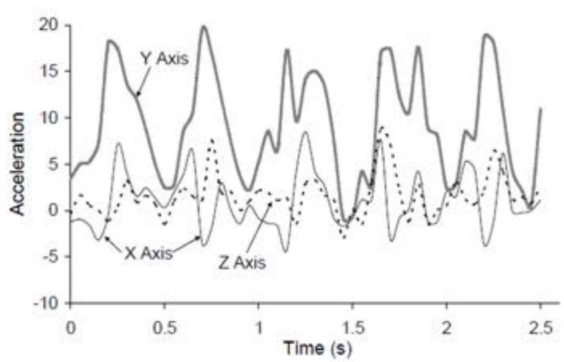
Recall: Activity Recognition

- Want app to detect when user is performing any of the following 6 activities
  - Walking,
  - Jogging,
  - Ascending stairs,
  - Descending stairs,
  - Sitting,
  - Standing



Recall: Activity Recognition Overview

Gather Accelerometer data



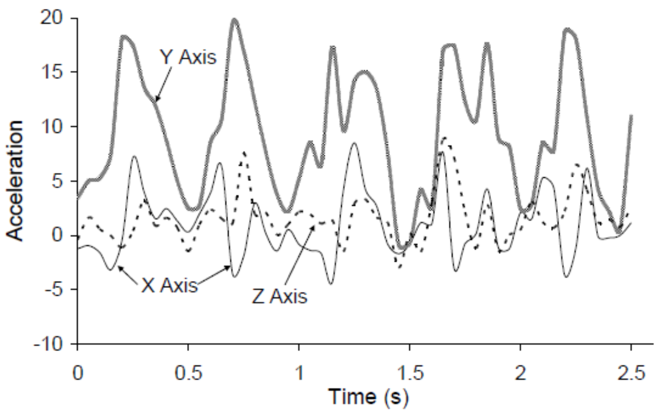
(a) Walking

→ **Machine Learning Classifier** → **Walking** / **Running** / **Climbing Stairs**

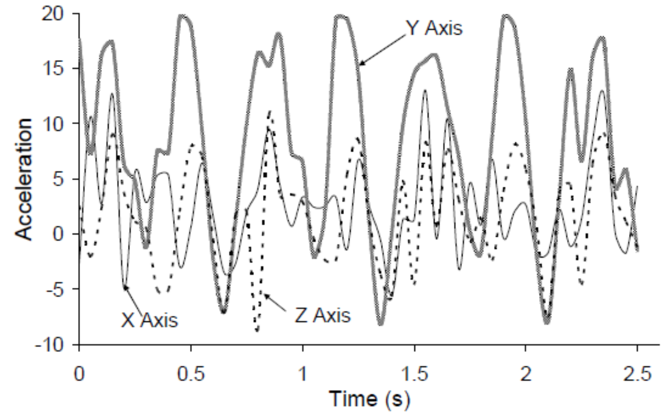
Classify Accelerometer data

Recall: Example Accelerometer Data for Activities

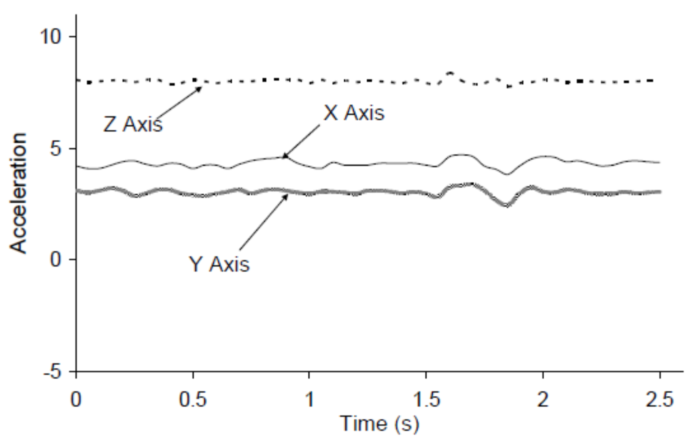
Different user activities generate different accelerometer patterns



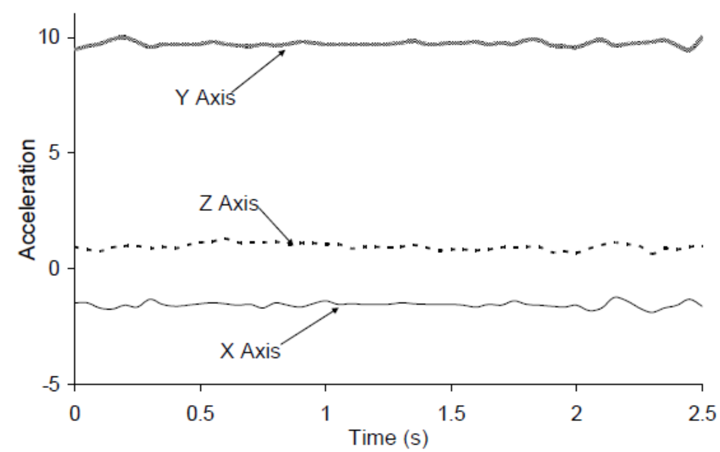
(a) Walking



(b) Jogging



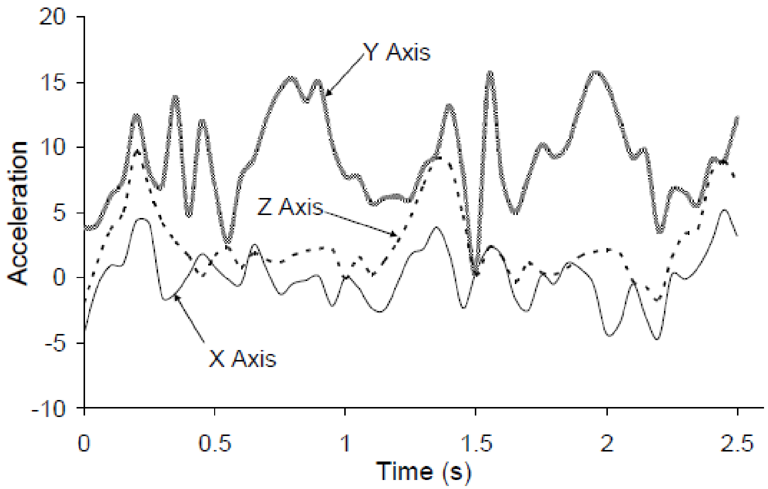
(e) Sitting



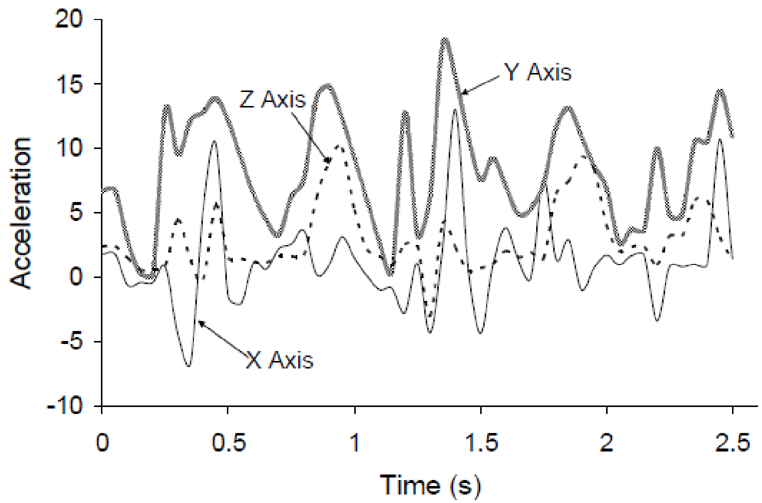
(f) Standing

Recall: Example Accelerometer Data for Activities

Different user activities generate different accelerometer patterns



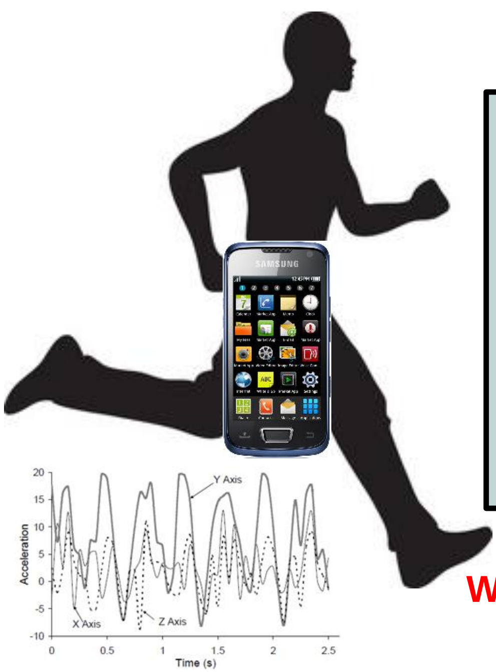
(c) Ascending Stairs

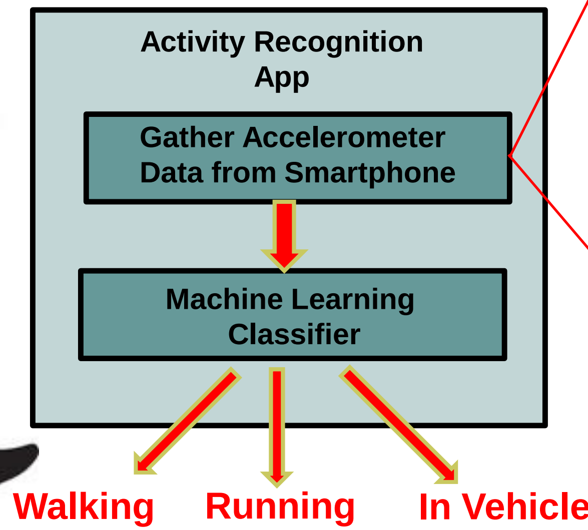


(d) Descending Stairs

DIY Activity Recognition (AR) Android App

- As user performs an activity, AR app on user's smartphone
  1. Gathers accelerometer data
  2. Uses **machine learning classifier** to determine what activity (running, jumping, etc) accelerometer pattern corresponds to
- **Classifier:** Machine learning algorithm that guesses what activity **class** accelerometer sample corresponds to





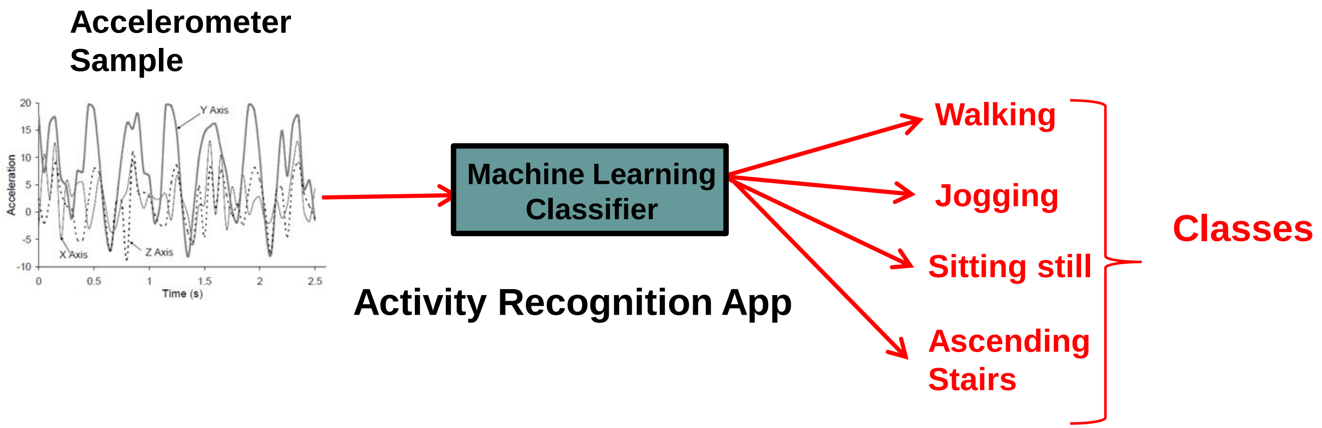

```
msensor = (mSensorManager)
   getSystemService(Context.SENSOR_SERVICE)

....

Public void onSensorChanged(SensorEvent event){

....

}
```

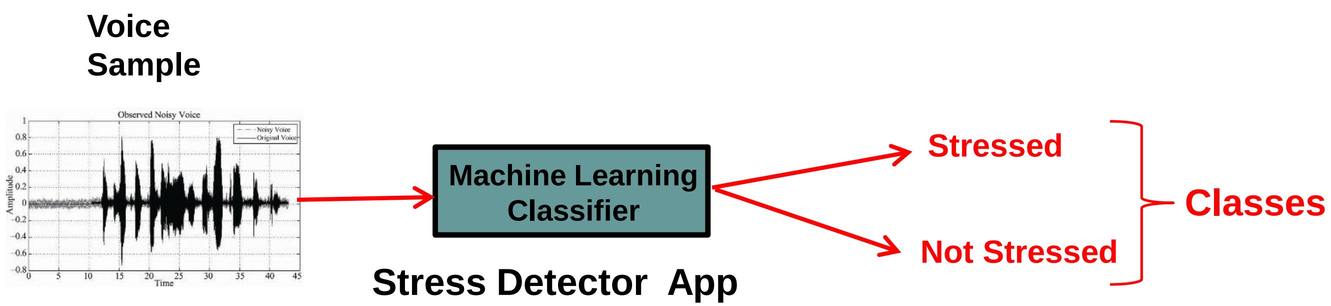
Next: Machine learning Classification

Classification for Ubiquitous Computing

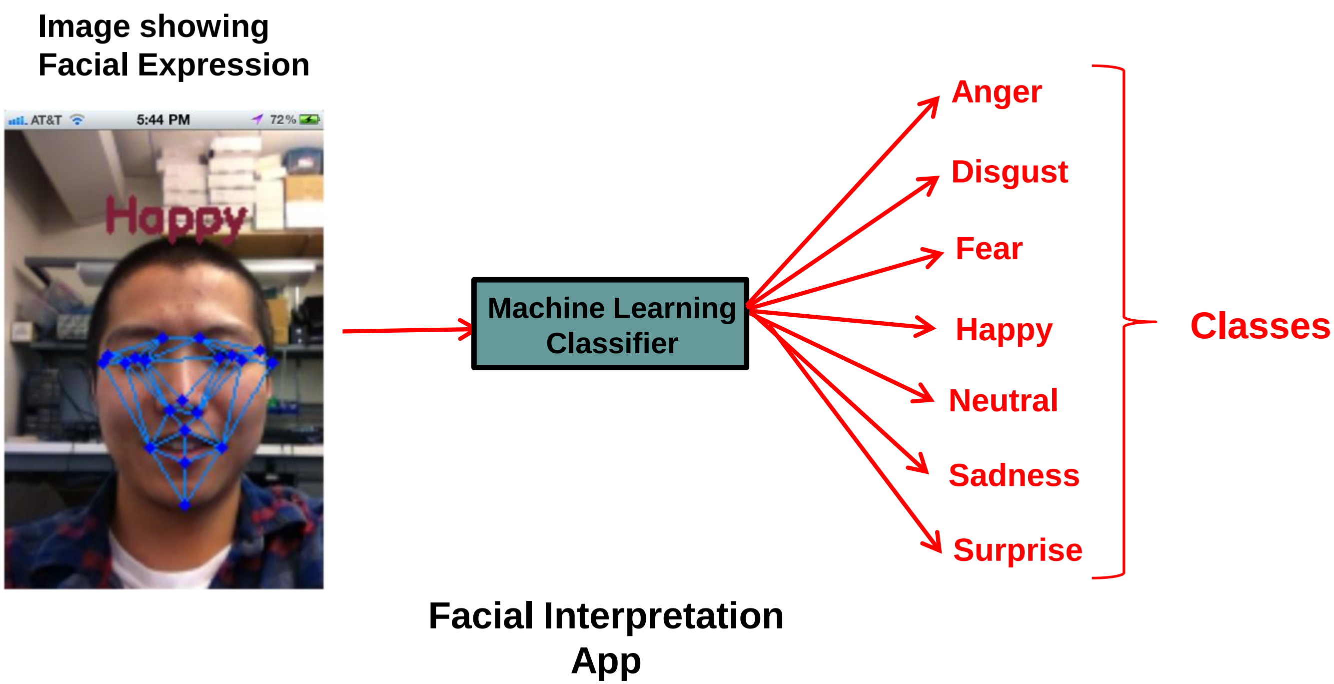
Classification

- **Classification** is type of machine learning used a lot in Ubicomp
- Classification? determine which **class** a sample belongs to. Examples:





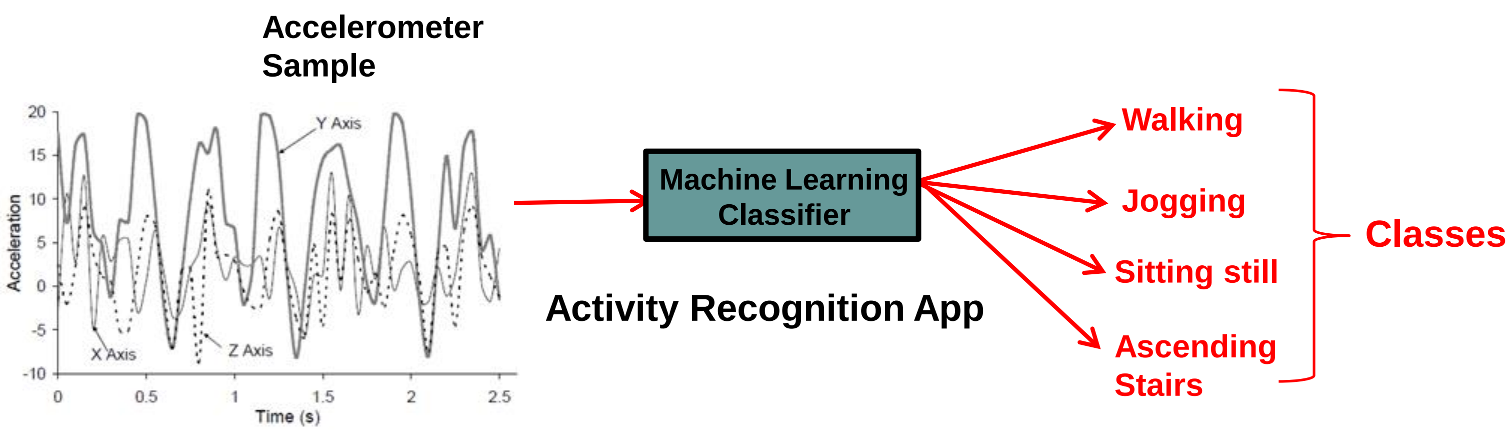
Classification



Classifier

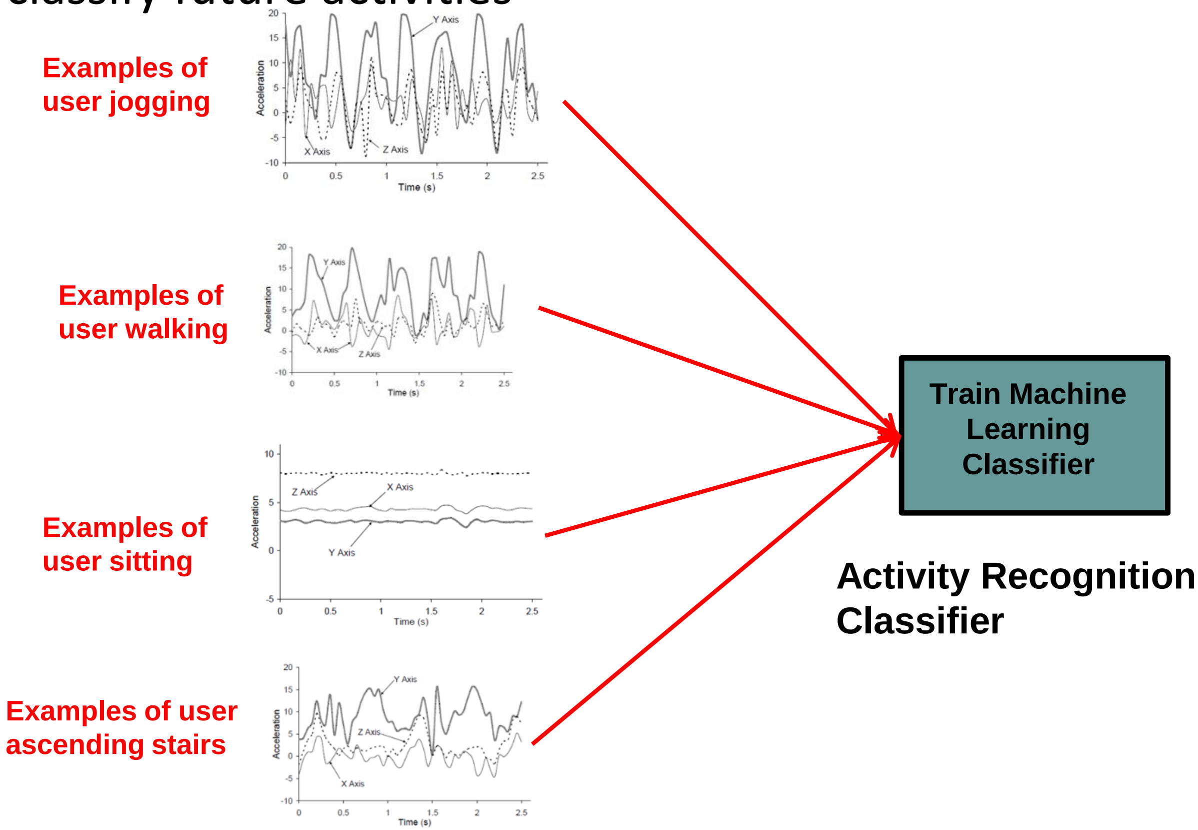
- Analyzes new sample, guesses corresponding class
- Intuitively, can think of classifier as set of rules for classification. E.g.
- Example rules for classifying accelerometer signal in Activity Recognition

```
If ((Accelerometer peak value > 12 m/s)
      and (Accelerometer average value < 6 m/s)){
             Activity = "Jogging";
}
```



Training a Classifier

- Created using example-based approach (called training)
- **Training a classifier:** Given examples of each class => generate rules to categorize new samples
- **E.g:** Analyze 30+ Examples (from 30 subjects) of accelerometer signal for each activity type (walking, jogging, sitting, ascending stairs) => generate rules (classifier) to classify future activities



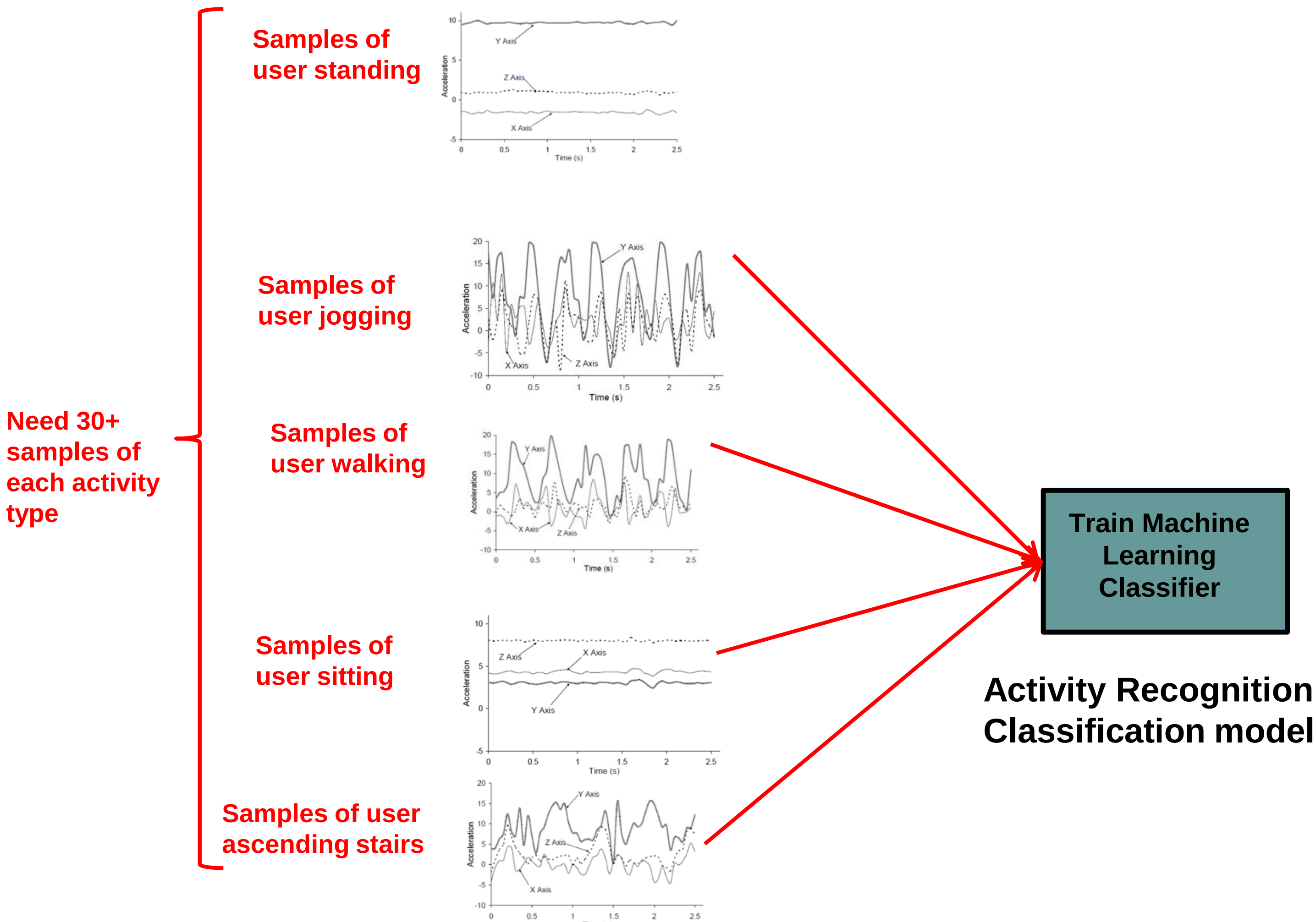
Training a Classifier: Steps

Steps for Training a Classifier

1. Gather data samples + label them
2. Import accelerometer samples into classification library (e.g. Weka, MATLAB)
3. Pre-processing (segmentation, smoothing, etc)
4. Extract features
5. Train classifier
6. Export classification model as JAR file
7. Import into Android app

Step 1: Gather Sample data + Label them

- Need many samples of accelerometer data corresponding to each activity type (jogging, walking, sitting, ascending stairs, etc)



Step 1: Gather Sample data + Label them

- Run a study to gather sample accelerometer data for each activity class
  - Recruit 30+ subjects
  - Run program that gathers accelerometer sensor data on subject's phone
  - Each subject:
    - Perform each activity (walking, jogging, sitting, etc)
    - Collect accelerometer data while they perform each activity (walking, jogging, sitting, etc)
  - Label data. i.e. tag each accelerometer sample with the corresponding activity
- Now have 30 examples of each activity

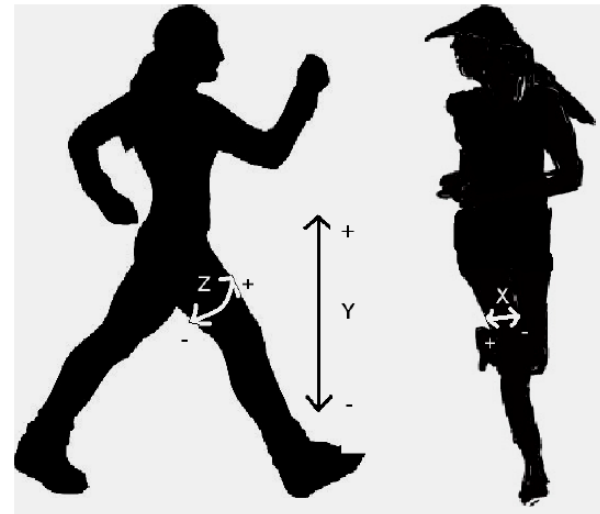
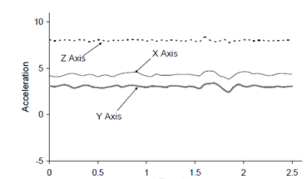
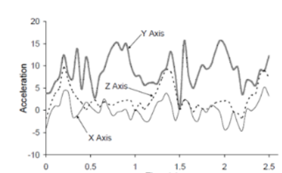


Figure 1: Axes of Motion Relative to User

30+ Samples of user sitting



30+ Samples of user ascending stairs



Step 1: Gather Sample data + Label them
Program to Gather Accelerometer Data

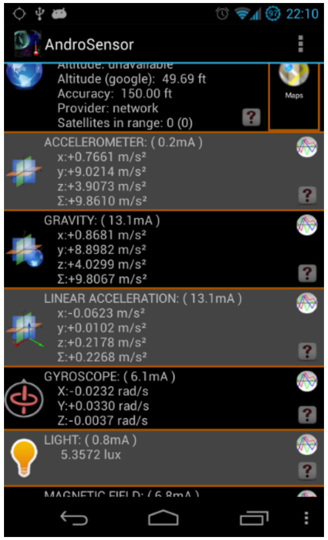
- **Option 1:** Can write sensor program app that gathers accelerometer data while user is doing each of 6 activities (1 at a time)

```
msensor = (mSensorManager)
   getSystemService(Context.SENSOR_SERVICE)

….

Public void onSensorChanged(SensorEvent event){

….

}
```

Step 1: Gather Sample data + Label them
Program to Gather Accelerometer Data

- **Option 2:** Use 3rd party app to gather accelerometer
  - 2 popular ones: **Funf** and **AndroSensor**
  - Just download app,
    - Select sensors to log (e.g. accelerometer)
    - Continuously gathers sensor data in background
- FUNF app from MIT
    - Accelerometer readings
    - Phone calls
    - SMS messages, etc
- AndroSensor

Funf



AndroSensor

Step 2: Import accelerometer samples into classification library (e.g. Weka, MATLAB)

- Import accelerometer data (labelled with corresponding activity) into Weka, MATLAB, scikit-learn (or other Machine learning Framework)

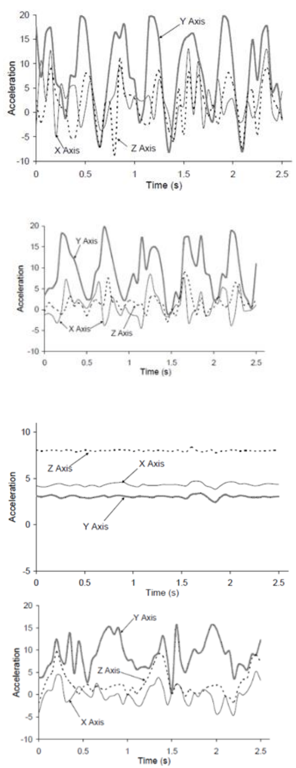
LABELS | **ACCELEROMETER DATA**

Jogging

Walking

Sitting

Ascending stairs

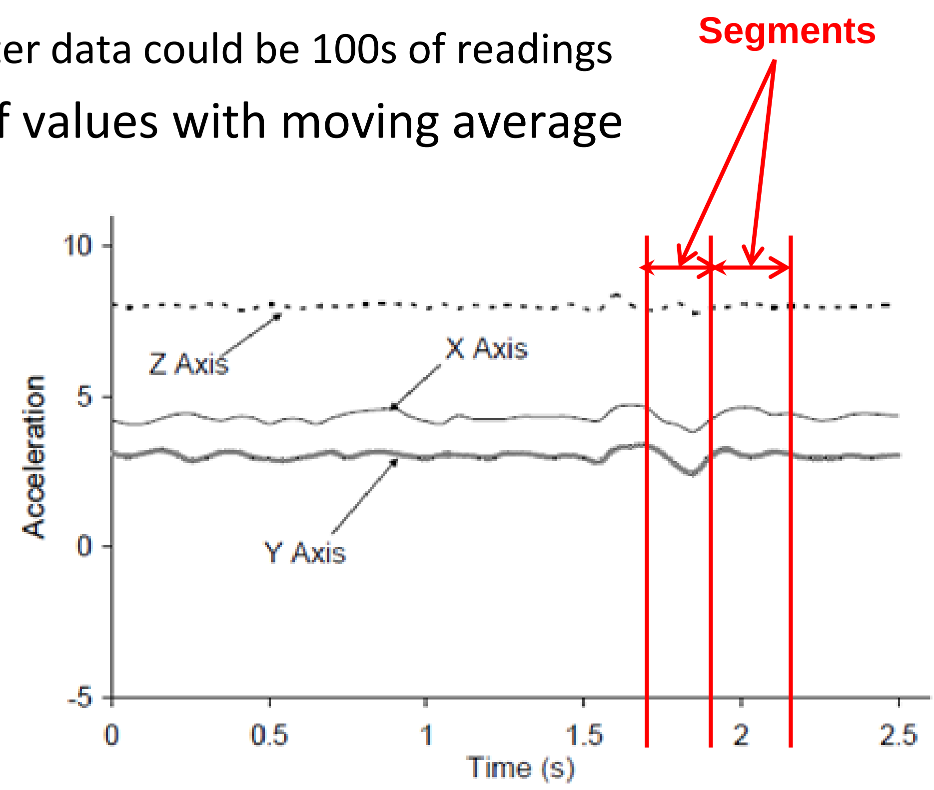


Weka, Matlab → **Classifiers**

Classifier is trained offline

Step 3: Pre-processing (segmentation, smoothing, etc)
Segment Data (Windows)

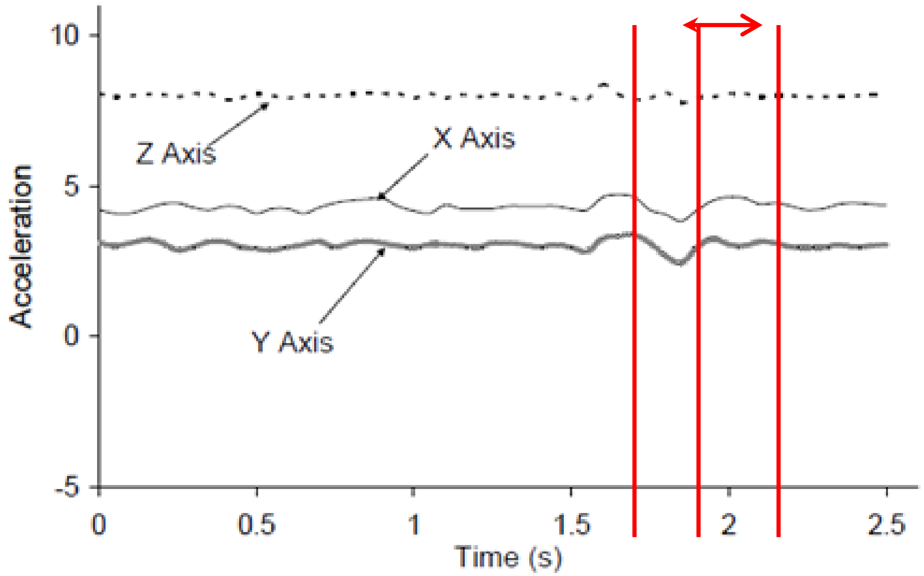
- Pre-processing data (in Weka, or MATLAB) may include segmentation, smoothing, etc
  - **Segment:** Divide 60 seconds of raw time-series data divided into chunks(e.g. 5 seconds)
    - Note: 5 seconds of accelerometer data could be 100s of readings
  - **Smoothing:** Replace groups of values with moving average



(e) Sitting

Step 4: Compute (Extract) Features

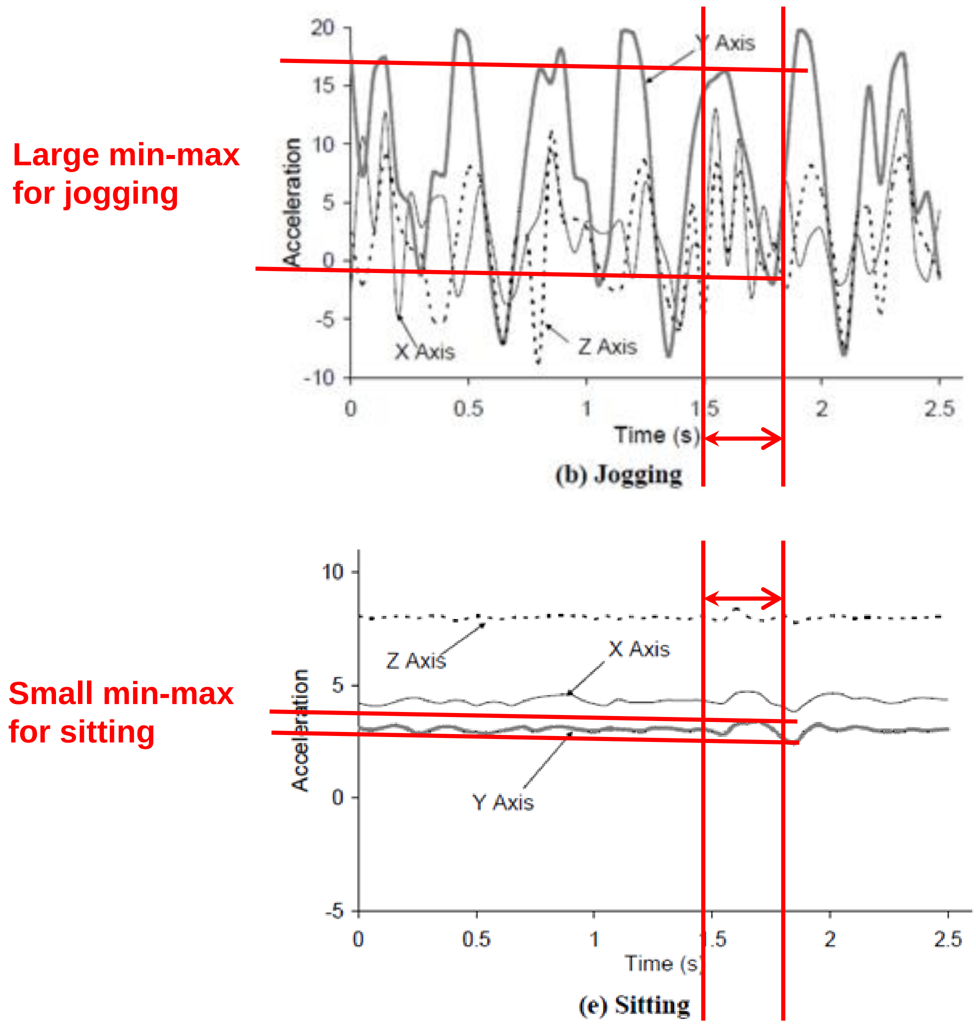
- For each 5-second segment (batch of accelerometer values) compute features (in Weka, MATLAB, etc)
- **Features:** Formulas computed to quantify attributes of accelerometer data, captures accelerometer characteristics
- **Examples:** min-max of values within each segment, largest magnitude, standard deviation



(e) Sitting

Step 4: Compute (Extract) Features

- **Important:** Ideally, values of features different for, distinguish each activity type (class)
- **E.g:** Min-max range feature



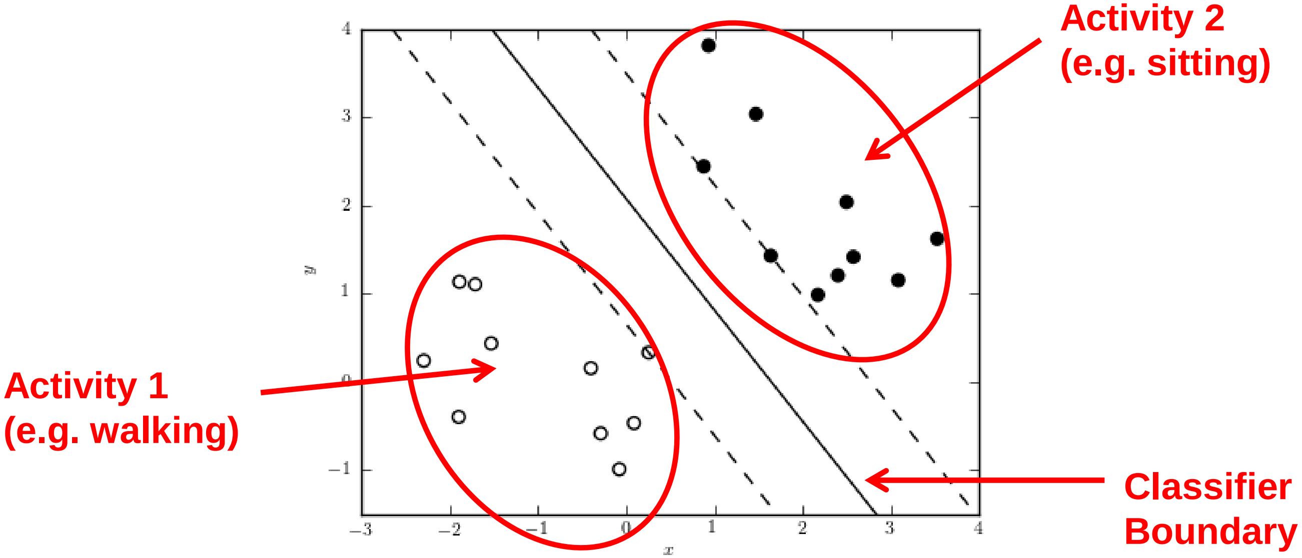
Step 4: Compute (Extract) Features

Calculate many different features

- Average[3]: Average acceleration (for each axis)
- Standard Deviation[3]: Standard deviation (for each axis)
- Average Absolute Difference[3]: Average absolute difference between the value of each of the 200 readings within the ED and the mean value over those 200 values (for each axis)
- Average Resultant Acceleration[1]: Average of the square roots of the sum of the values of each axis squared $\sqrt{(x_i^2 + y_i^2 + z_i^2)}$ over the ED
- Time Between Peaks[3]: Time in milliseconds between peaks in the sinusoidal waves associated with most activities (for each axis)
- Binned Distribution[30]: We determine the range of values for each axis (maximum – minimum), divide this range into 10 equal sized bins, and then record what fraction of the 200 values fell within each of the bins.

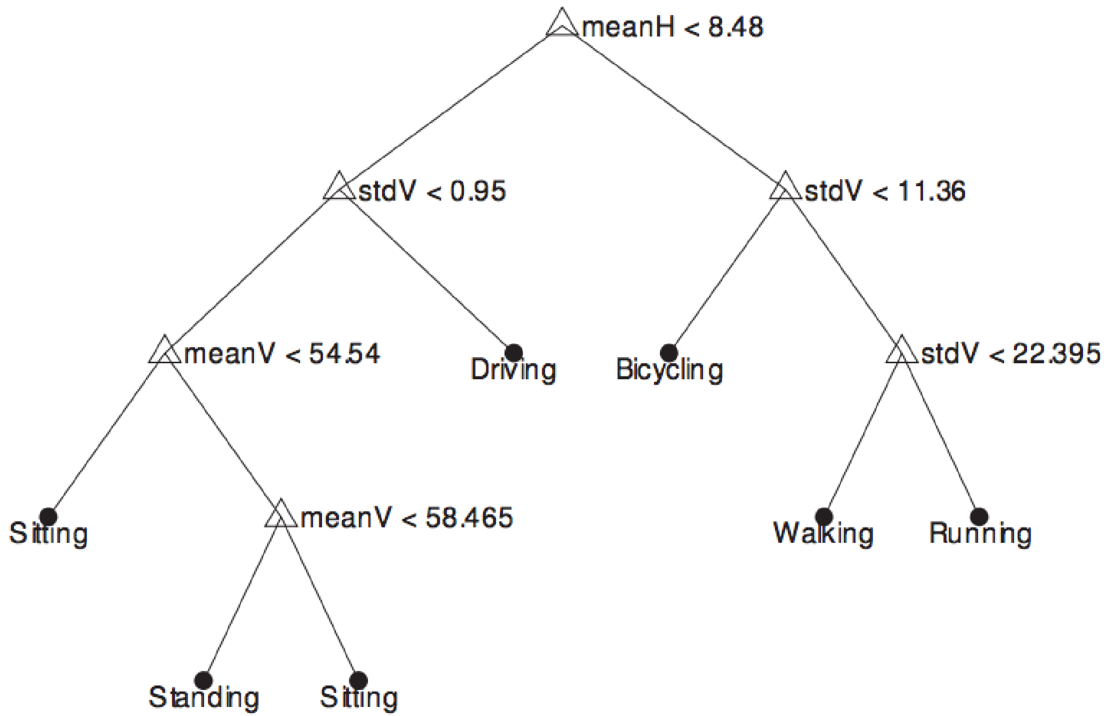
Step 5: Train classifier

- Features are just numbers
- Different values for different activities
- **Training classifier:** figures out feature values corresponding to each activity
- Weka, MATLAB already programmed with different classification algorithms (SVM, Naïve Bayes, Random Forest, J48, logistic regression, SMO, etc)
- Try different ones, compare accuracy
- SVM example



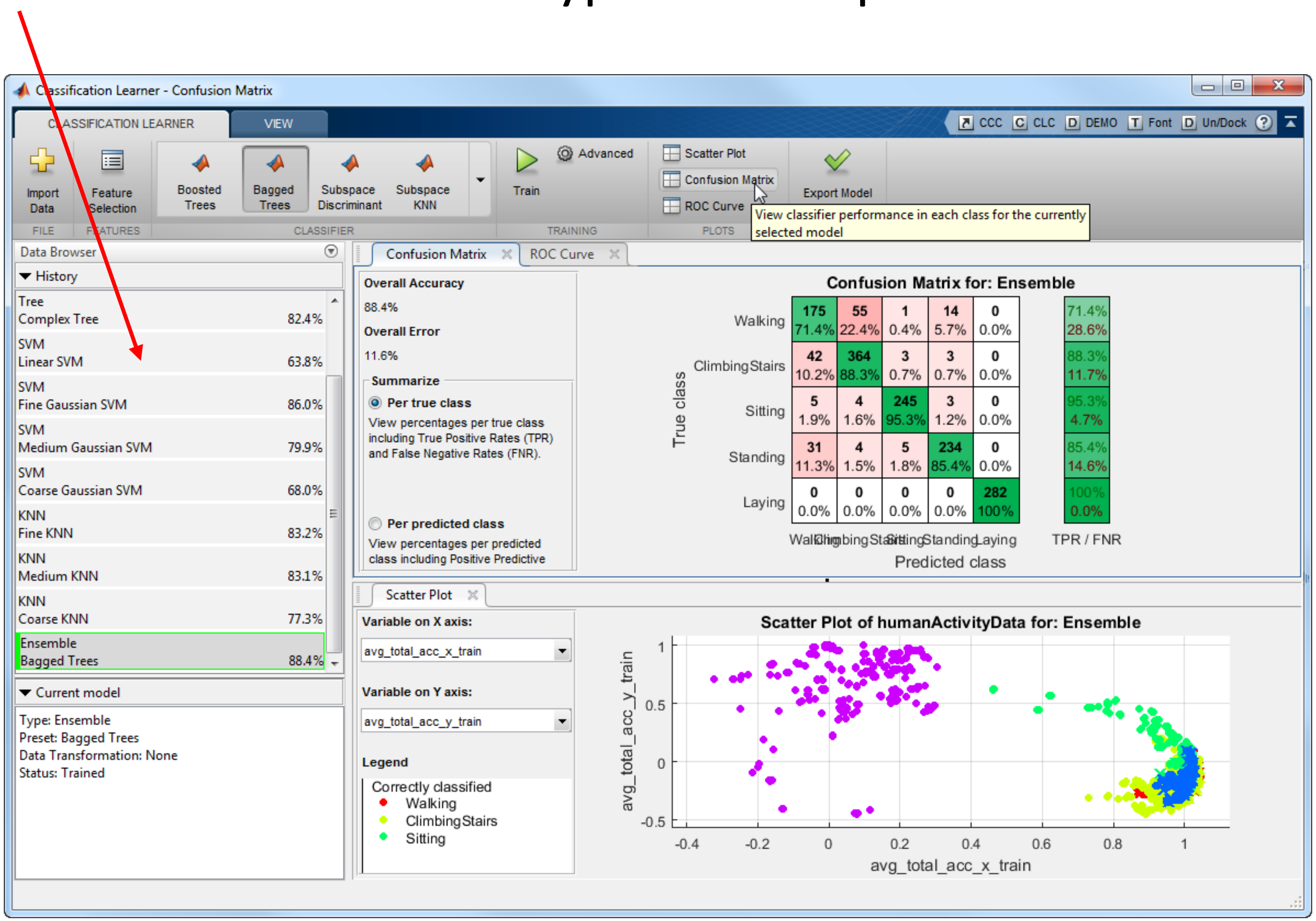
Step 5: Train classifier

- **Example:** Decision Tree Classifier
- **Training phase:** Learns thresholds for feature values extracted from examples, which separate the classes
- **Test phase:** Feature values of new sample compared against learned thresholds at each node to determine its class



Step 5: MATLAB Classification Learner App

- Import accelerometer data into MATLAB
- Click and select Classifier types to compare



Step 5: Train classifier
Compare Accuracy of Classifier Algorithms

- Weka, MATLAB also reports accuracy of each classifier type
- **Accuracy:** Percentage of test cases that classifier guessed correctly

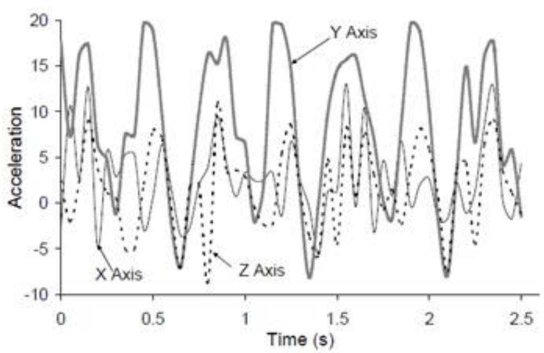
Table 2: Accuracies of Activity Recognition

	% of Records Correctly Predicted			
	J48	**Logistic Regression**	**Multilayer Perceptron**	**Straw Man**
Walking	89.9	**93.6**	91.7	37.2
Jogging	96.5	98.0	**98.3**	29.2
Upstairs	59.3	27.5	**61.5**	12.2
Downstairs	**55.5**	12.3	44.3	10.0
Sitting	**95.7**	92.2	95.0	6.4
Standing	**93.3**	87.0	91.9	5.0
Overall	85.1	78.1	**91.7**	37.2

Compare, pick most accurate classification algorithm

Step 6: Export Classification model as JAR file
Step 7: Import into Android app

- Export classification model (most accurate classifier type + data threshold values) as Java JAR file
- Import JAR file into Android app
- In app write Android code to
  - Gather accelerometer data, segment, extract feature, classify using classifier in JAR file
- Classifies new accelerometer patterns while user is performing activity => Guess (infer) what activity



(b) Jogging

New accelerometer Sample in real time

Classifier in Android app

Activity (e.g. Jogging)

Support Vector Machine (SVM)

Scalable Vector Machines (SVM)

- One of the most popular classification algorithms
- If plot example points with features as axes
- Classification problem: Find boundary between classes
- E.g Classify healthy vs unhealthy patients
- 2 Features are strongest predictors
  - Age
  - Maximum exercise rate

Classification algorithm (e.g. SVM) finds this boundary

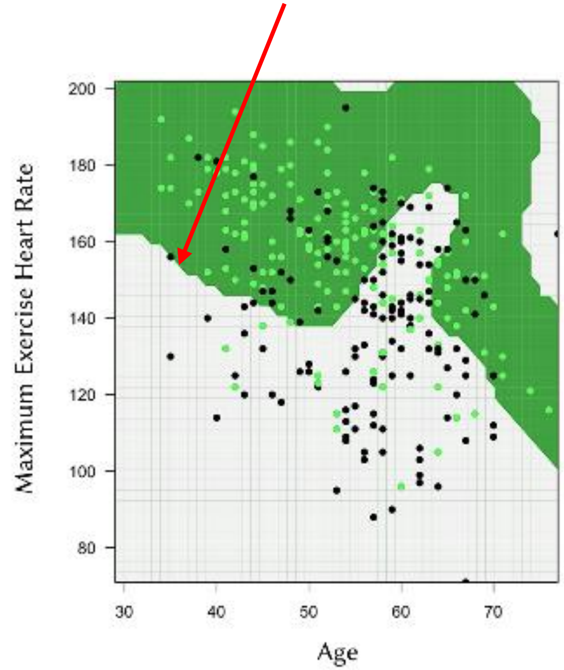


Figure 1. Using SVM to predict the presence of heart disease. The dark green region represents the profile of healthy adults, while the gray region represents the profile of heart disease patients. The light green and black points represent healthy adults and heart disease patients respectively.

SVM: Delineating Boundaries

- Multiple ways to delineate optimal boundary

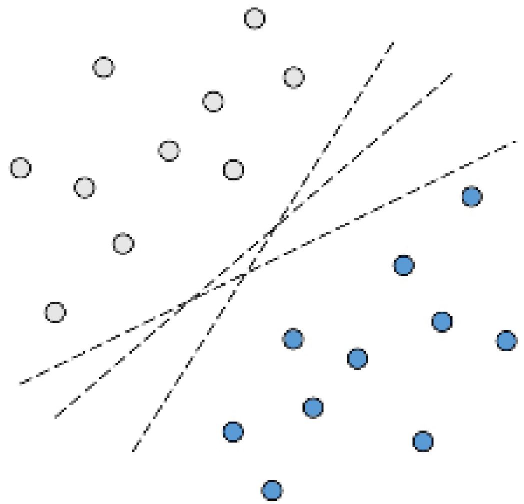


Figure 2. Multiple ways to separate two groups.

SVM: Support Vectors

- SVM first finds peripheral data points in group 1 that are closest to the points in group 2 (called **support vectors**)
- Then draw **optimal boundary** between support vectors of both groups
- Since SVM uses only relatively few data points (support vectors), it is computationally efficient

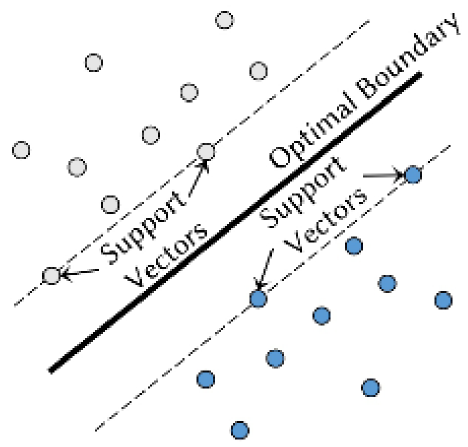


Figure 3. Optimal boundary is located in the middle of peripheral data points from opposing groups.

SVM Limitations

- **Inaccurate for small datasets:** Smaller dataset would have fewer points, less likely to find good support vectors

- **Classifying multiple groups:**
  - SVM classifies 2 groups at a time.
  - Multiple groups handled by making multiple 2-group classifications
  - Multi-group SVM: On each iteration, classify 1 group from the rest

- **Overlapping groups:**
  - Since SVM classifies points based on what side of boundary it lies, overlapping groups present a challenge
  - If classes overlap, points close to boundary may be mis-classified

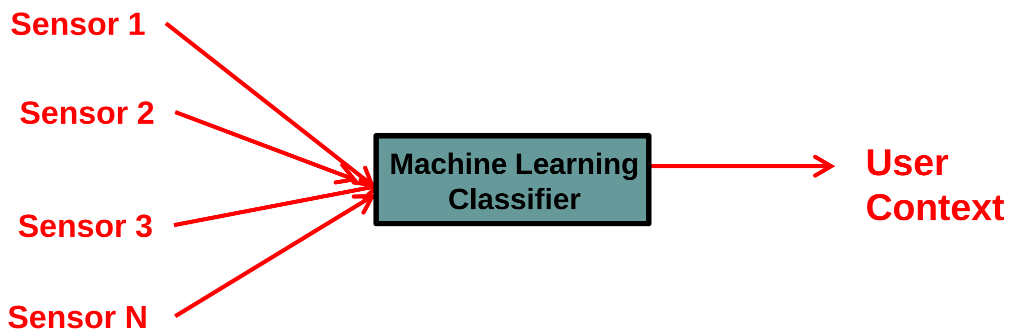
Context Sensing

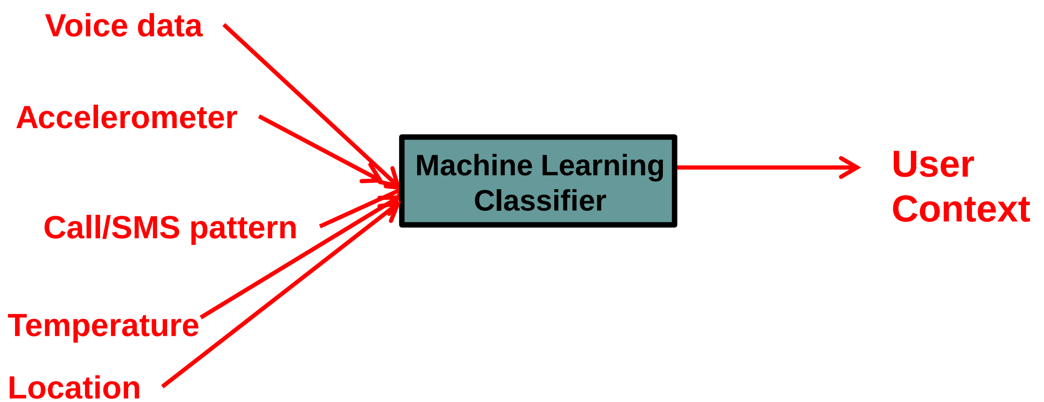
Recall: Ubicomp Senses User's Context

- Context?
  - *Human:* motion, mood, identity, gesture
  - *Environment:* temperature, sound, humidity, location
  - *Computing Resources:* Hard disk space, memory, bandwidth
  - *Ubicomp example:*
    - *Assistant senses:* Temperature outside is 10F (environment sensing) + Human plans to go work (schedule)
    - *Ubicomp assistant advises:* Dress warm!
- Sensed **environment + Human + Computer resources** = ***Context***
- *Context-Aware* applications adapt their behavior to context

Context Sensing

- Activity Recognition uses data from accelerometer and gyroscope (2 sensors)
- Can combine multiple sensors, use machine learning to learn **user context** that occur to various outcomes (e.g. user's emotion)
- More later

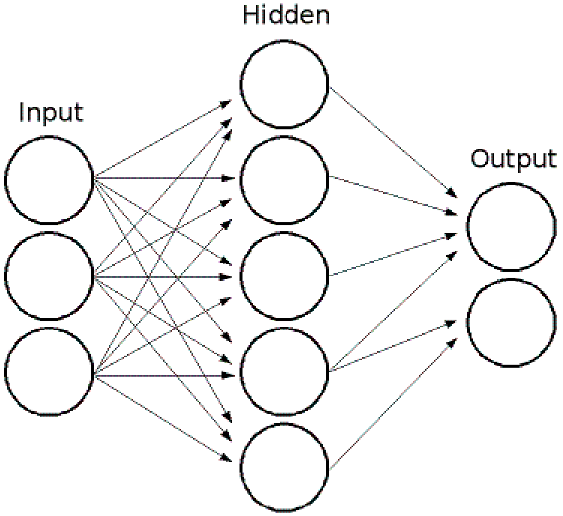




Deep Learning

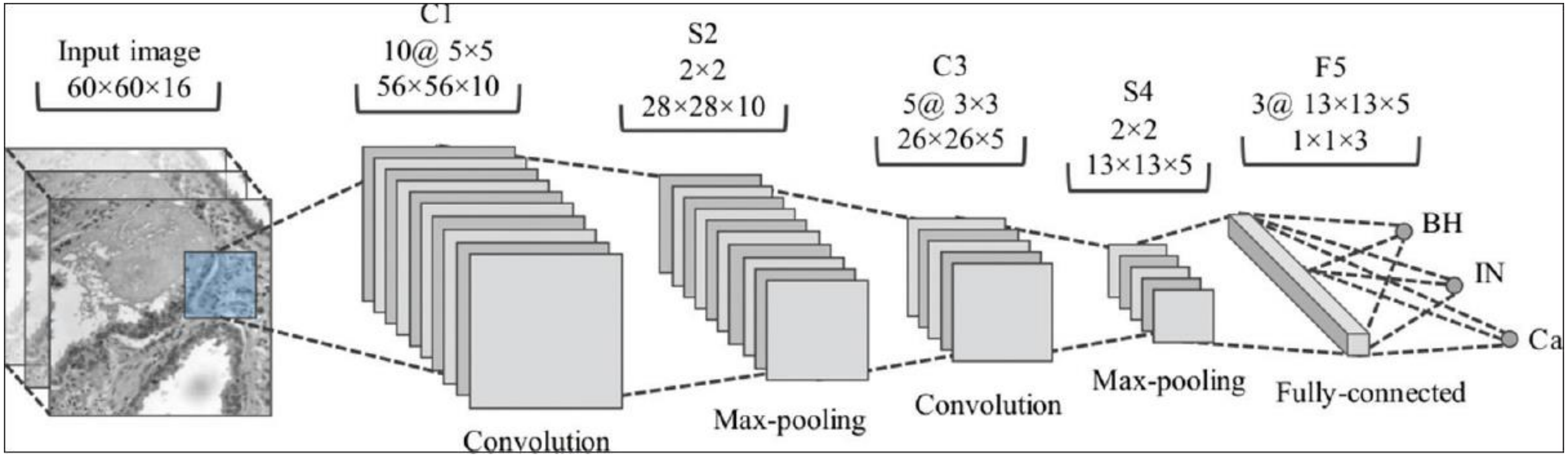
Deep Learning

- Network of nodes, connectivity weights learned from data
- Learns best weights to classify inputs (x) into outputs y
- Can think about it as curve fitting
- Generally more accurate if more data is available
- Requires lots of computational power to train



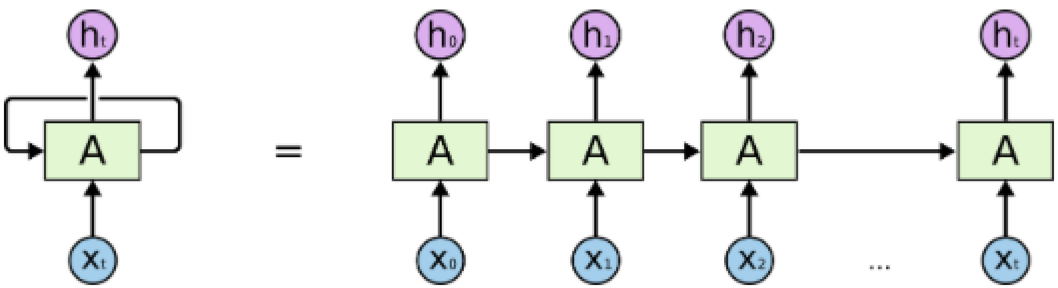
Convolutional Neural Networks (CNNs)

- Different types of neural networks good for different things
- Convolutional Neural Networks good for classifying images
- E.g. Is there a cat in an input picture?



Recurrent Neural Networks (RNNs)

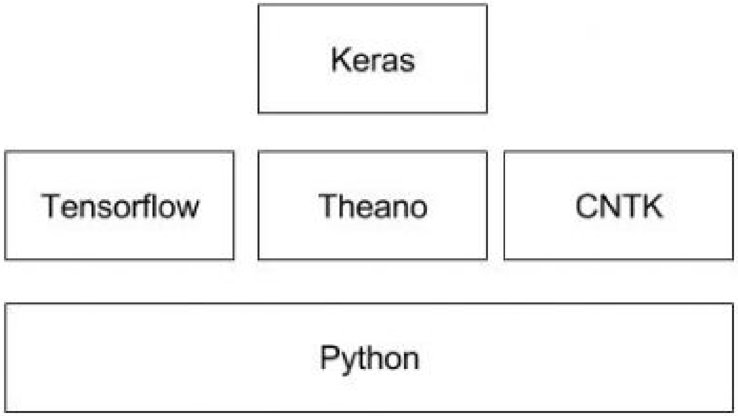
- Good at classifying sequential data
- E.g. Speech translation: sequence of words
- E.g. translate german sentence to English



Programming/Mobile Support for Neural Networks

https://developer.android.com/ndk/guides/neuralnetworks/index.html

- Many python libraries for neural networks/deep learning
- Enable training neural networks in a few lines of code
  - Keras
  - PyTorch
  - ScikitLearn
- Training neural networks on Smartphone still tough
- New in Android 8.1: Android Neural Networks API (NNAPI) allows inference (test) of pre-trained neural networks on smartphone
  - Minimally supports several machine learning frameworks (e.g. Tensorflow lite, caffe2)
- Keras also has some mobile support



References

- Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore, Activity recognition using cell phone accelerometers, SIGKDD Explor. Newsl. 12, 2 (March 2011), 74-82.
- Deepak Ganesan, Activity Recognition, Physiological Sensing Class, UMASS Amherst