Mobile and Ubiquitous Computing
Lecture 8a: Final Project, Paper Presentations, Introduction to Machine Learning

Emmanuel Agu

Final Project Proposal

Final Project Proposal

- Final project 1 slide due, submitted on Tue (Oct 24, 10am)?
- While working on project 3, more brainstorming on final project
- November 2 (Next Thursday), Propose final project (mobile/ubicomp app or machine learning project that solves a real WPI problem)
- All teams will present next week!!
- Proposals should include:
  1. **Problem you intend to work on**
     - App that finds available study spaces (safe + available), dynamically updated
  2. **Why this problem is important**
     - E.g. 32% of WPI students living with roommates, hard to find places to study
  3. **Related Work:** What prior solutions have been proposed for this problem

Final Project Proposal

4. **Summary of envisioned solution**
   - E.g. Mobile app maintains dynamic list of available and safe study spots including Android/third party modules app will have

5. **Tally your difficulty points in your slides, summarize your tally**

- Can also do Machine learning project that classifies/detects analyzes a dataset of builds a real-time app to classify some human sensor data.
  - Can use existing smartphone datasets online, or gather your own data

- You can:
  - Bounce ideas of me (email, or in person)
  - Change idea any time

Rubric: Grading Considerations

- **Problem (10/100)**
  - How much is the problem a real problem (e.g. not contrived)
  - Is this really a good problem that is a good fit to solve with mobile/ubiquitous computing? (e.g. are there better approaches?)
  - How useful would it be if this problem is solved?
  - What is the potential impact on the community (e.g. WPI students) (e.g. how much money? Time? Productivity.. Would be saved?)
  - What is the evidence of the importance? (E.g. quote a statistic)

Rubric: Grading Considerations

- **Related Work (5/100)**
  - Prior research, other apps been previously proposed to solve this problem?

- **Proposed Solution/Classification (10/100)**
  - How good/clever/interesting is the solution?
  - How sophisticated and how are the mobile/ubiquitous computing components (high level) used? (e.g. location, geofencing, activity recognition, face recognition, machine learning, etc)

Rubric: Grading Considerations

- **Implementation Plan + Timeline (10/100)**
  - Clear plans to realize your design/methodology
  - Android modules/3rd party software used
  - Software architecture,
  - Screenshots (or sketches of UI), or study design + timeline
- **Evaluation Plan (5/100)**
  - How will you evaluate your project, metrics
  - E.g. small user studies for apps
  - Machine learning performance metrics (e.g. classification accuracy, F1 score, etc)
- **Difficulty Points (20/100)**
  - Will follow rubric handed out in class, and scale max. of 25 down to 20/100
- 40 more points allotted for your slides + oral presentation

Final Project: Proposal Vs Final Submission (Presentation + Paper)

Final Project Proposal Vs Final Submission

- Introduction
- Related Work
- Approach/methodology
- Implementation
- **Project timeline**

- Evaluation/Results
- Discussion
- Conclusion
- Future Work

Proposal Talk

Final Talk Slides
Final Paper

Note: No timeline
In final paper

Student Research Paper Presentations

Research Presentation: Mobile and UbiComp Papers

- I have put up list of research papers on Canvas
- On Nov. 16 and 30, GROUPs present 1 research paper each from my list

- Your talk should cover:
  - Motivation for problem (General)
  - Specific problem solved in paper
  - Approach used to solve the problem/how it works
  - Evaluation of solution (sample results)
  - Discussion/conclusion

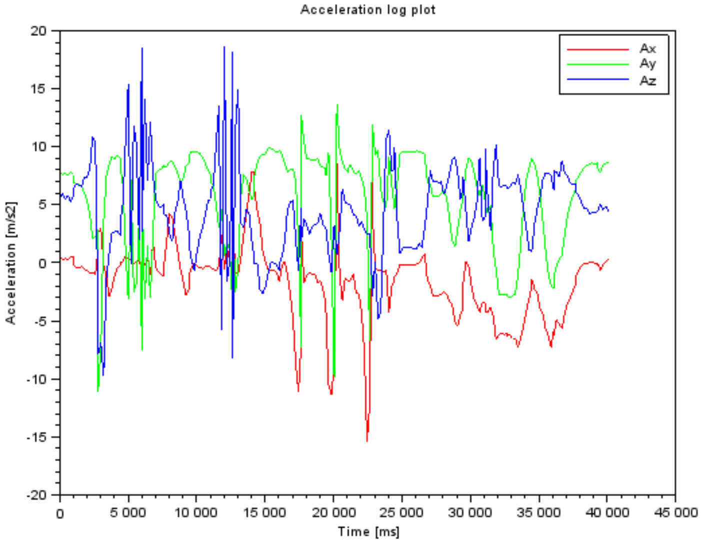
Intuitive Introduction to Machine Learning for Ubiquitous Computing

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

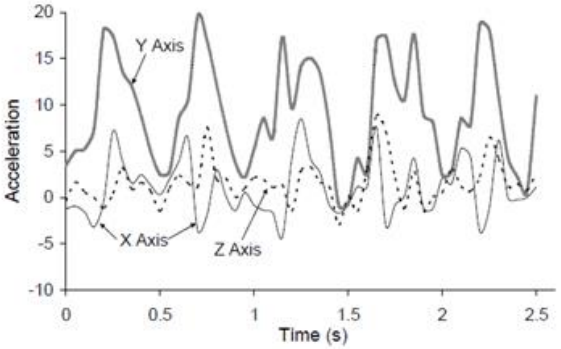
Recall: Activity Recognition

- 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

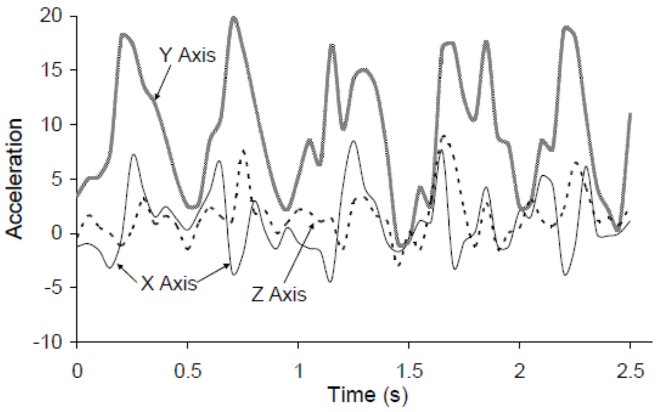
Machine Learning Classifier

Classify Accelerometer data

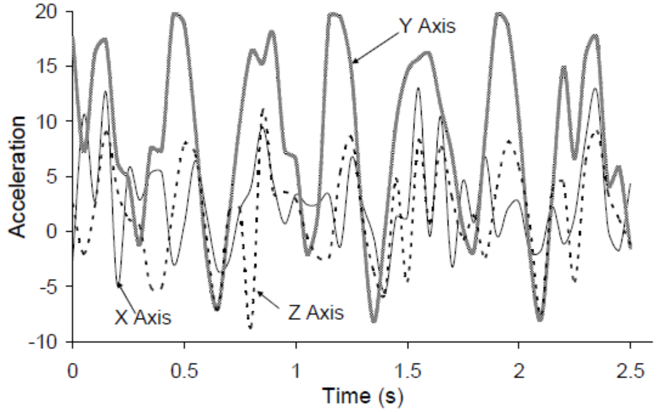
- Walking
- Running
- Climbing Stairs

Recall: Example Accelerometer Data for Activities

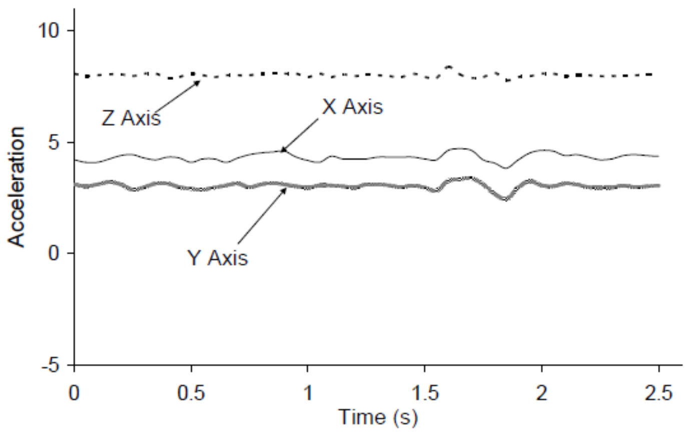
Different user activities generate different accelerometer patterns



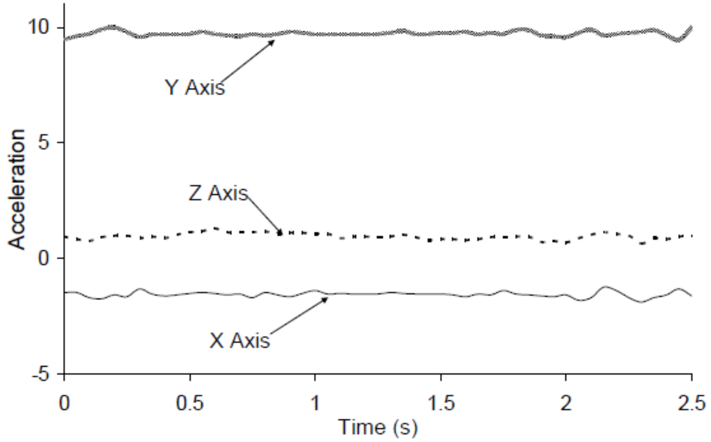
(a) Walking



(b) Jogging



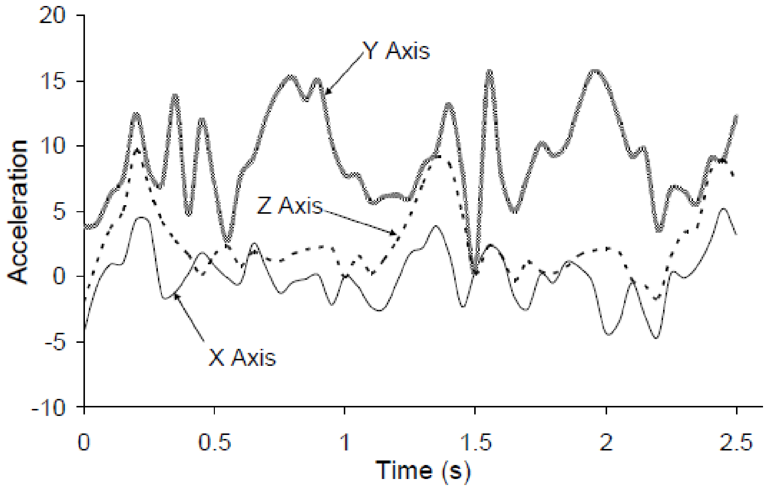
(e) Sitting



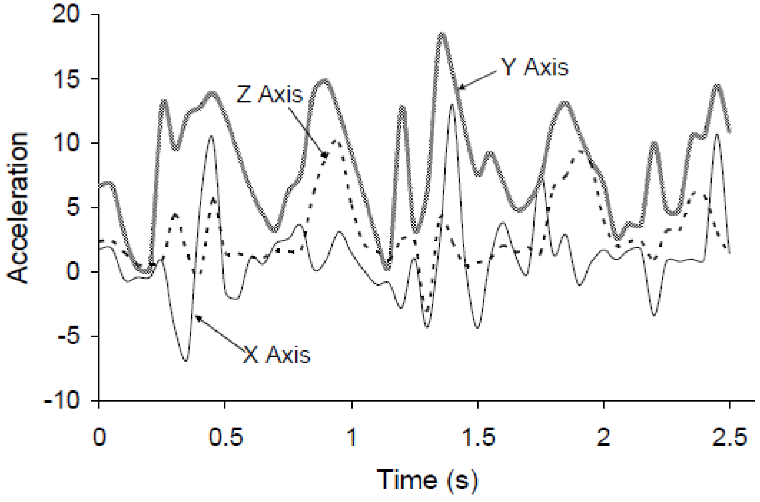
(f) Standing

Recall: Example Accelerometer Data for Activities

Different user activities generate different accelerometer patterns



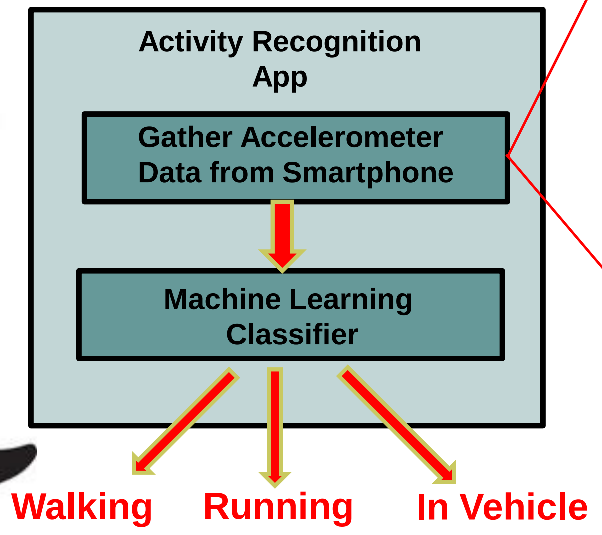
(c) Ascending Stairs



(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** (or type) accelerometer sample corresponds to



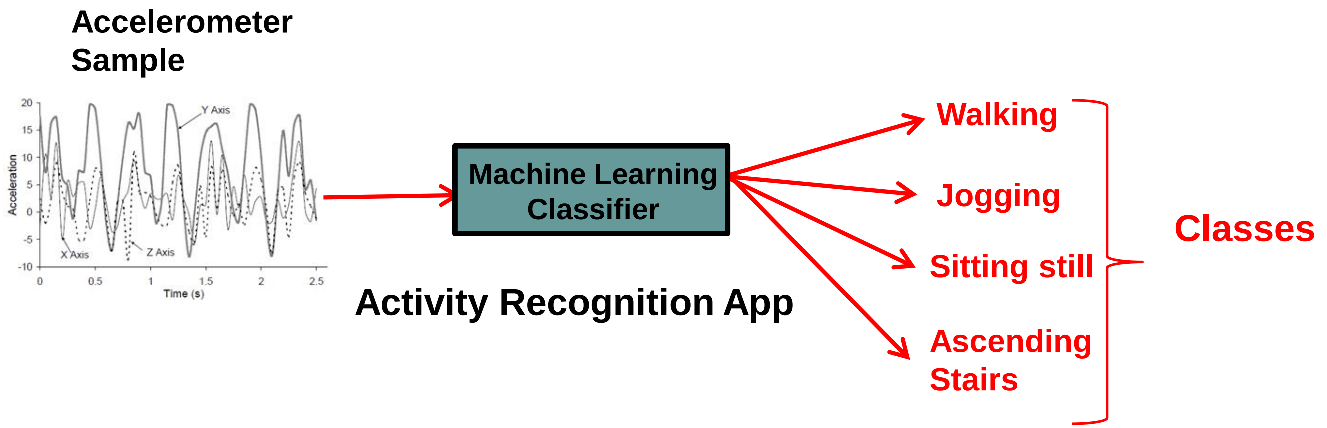

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE)
                                        as SensorManager


override fun onSensorChanged(event: SensorEvent) {

....

}
```

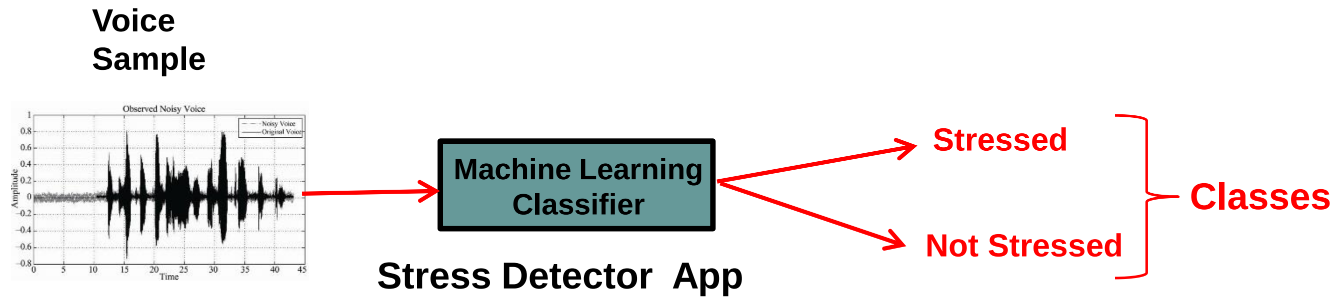
Next: Machine learning Classification

Classification for Ubiquitous Computing

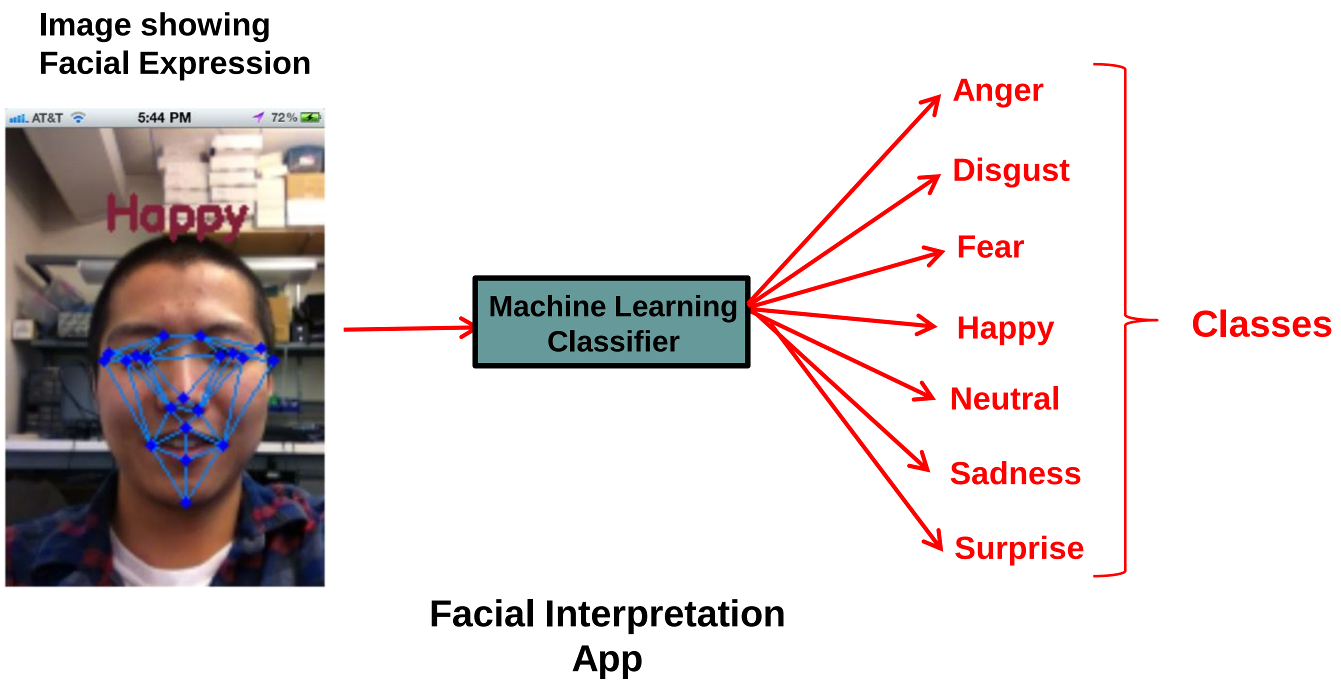
Classification

- **Classification** is type of machine learning used a lot in Ubicomp
- Classification? determine which **class** a sample (e.g. snippet of accelerometer data) belongs to. Examples:





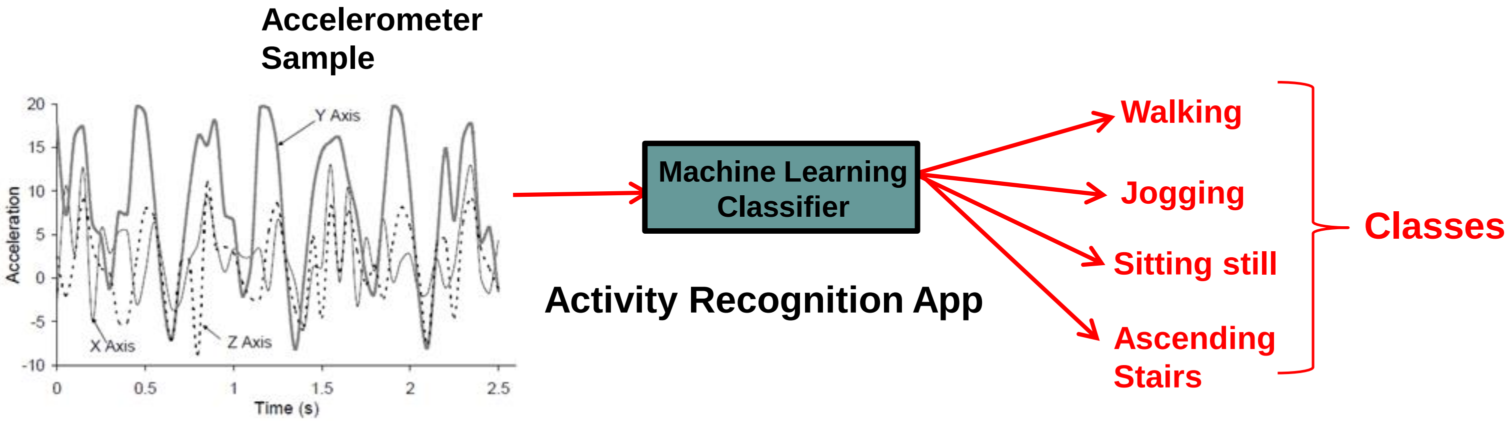
Classification



Classifier

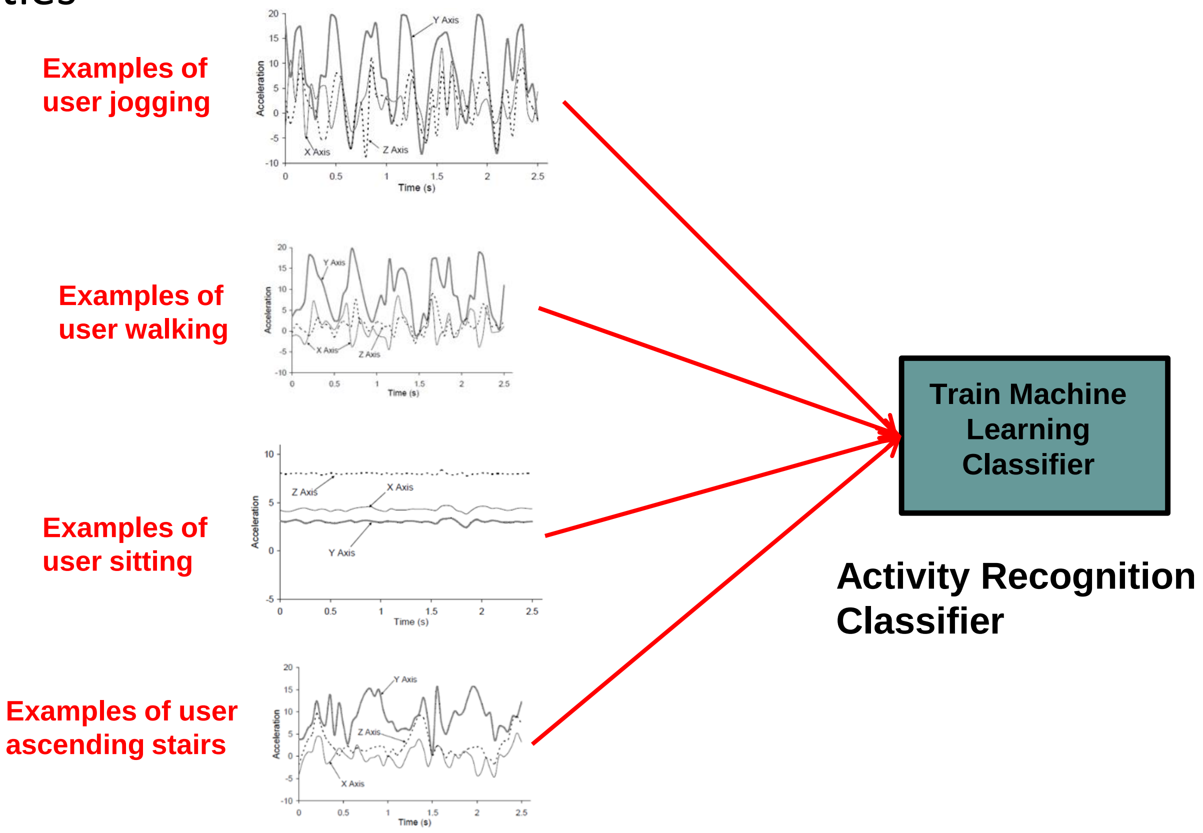
- Analyzes new sample, guesses corresponding class
- Intuitively, can think of classifier as set of rules for classification. E.g.
- Example rules for classifying accelerometer signal in Activity Recognition

```
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 (ML model) 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



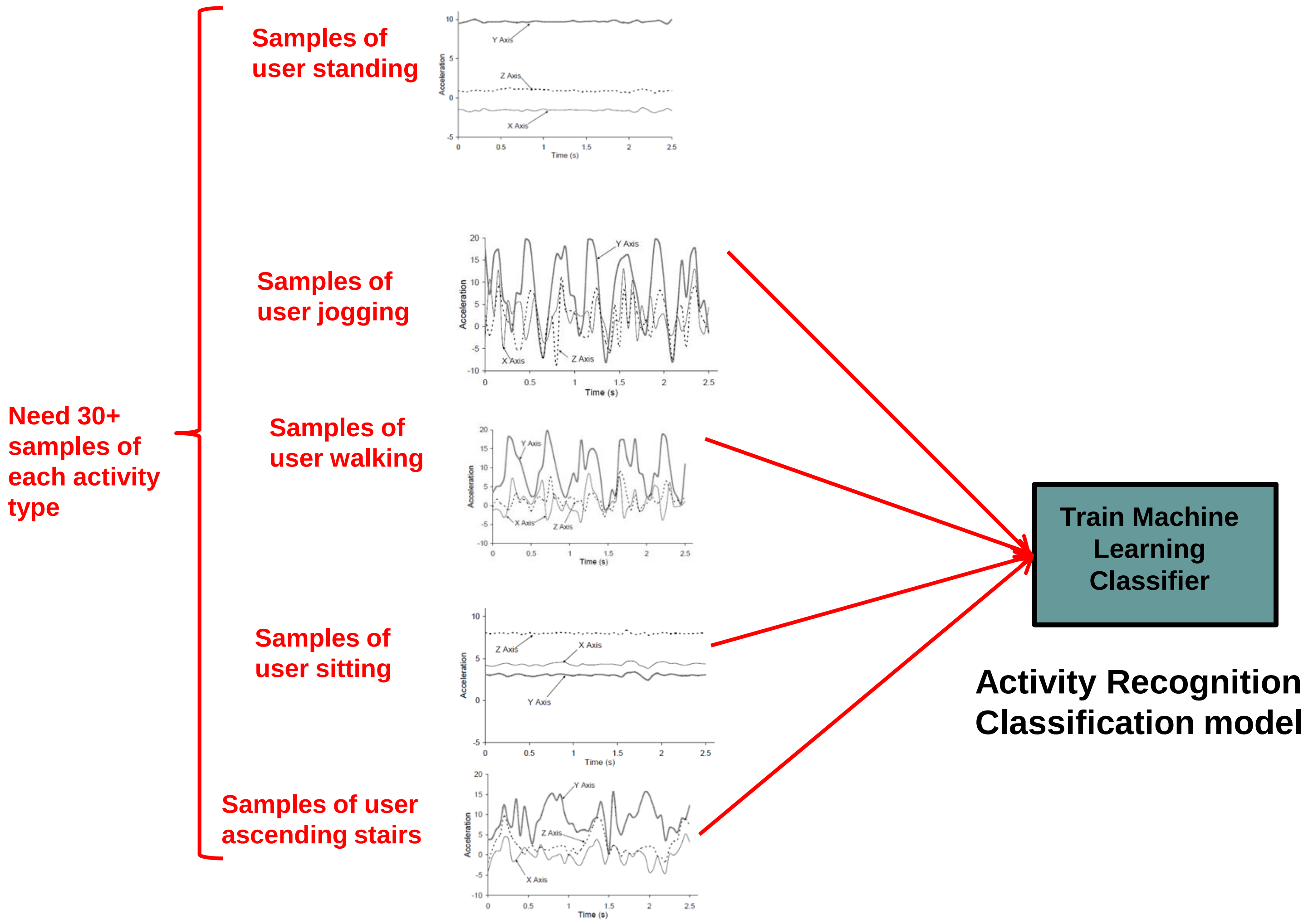
Training a Classifier: Steps

Steps for Training a Classifier

1. Gather data samples + label them
2. Import accelerometer samples into classification library (e.g. scikit-learn, MATLAB)
3. Pre-processing (segmentation, smoothing, etc)
4. Extract features
5. Train classifier
6. Deploy classifier

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

- Conduct a study to gather sample accelerometer data for each activity class
  - Recruit 30+ subjects
  - Run app 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

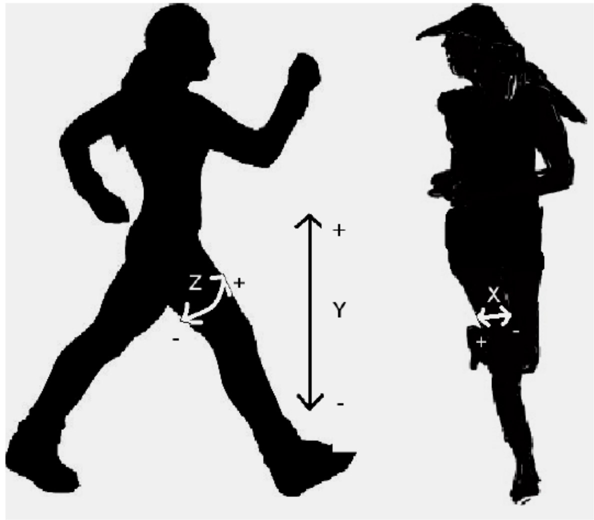
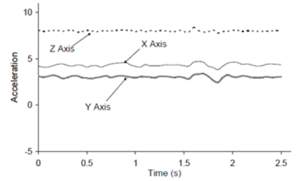
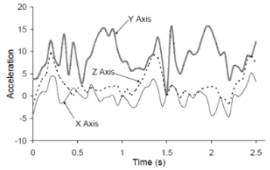


Figure 1: Axes of Motion Relative to User

30+ Samples of user sitting



30+ Samples of user ascending stairs



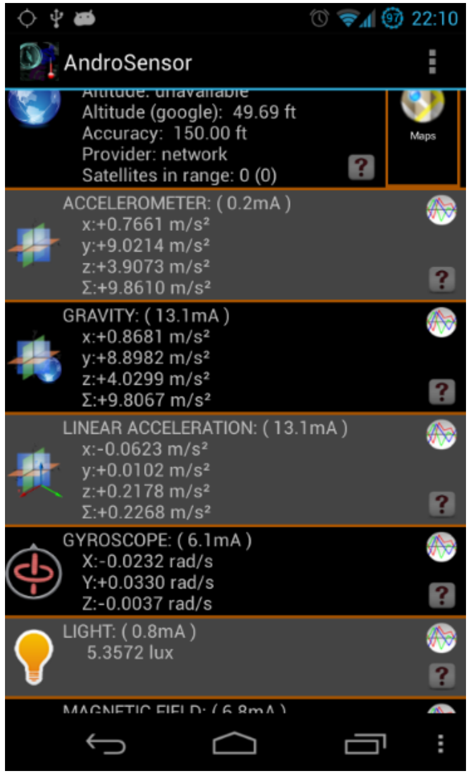
Step 1: Gather Sample data + Label them
Program to Gather Accelerometer Data

- **Option 1:** Can write "home-grown" sensor program app that gathers accelerometer data while user is doing each activity (1 at a time)

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager


override fun onSensorChanged(event: SensorEvent) {

....

}
```

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

- **Option 2:** Use 3rd party app to gather accelerometer
  - E.g. **AndroSensor**
  - Just download app,
    - Select sensors to log (e.g. accelerometer)
    - Continuously gathers sensor data in background
    - Saves sensor data to .csv file



AndroSensor

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

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

LABELS

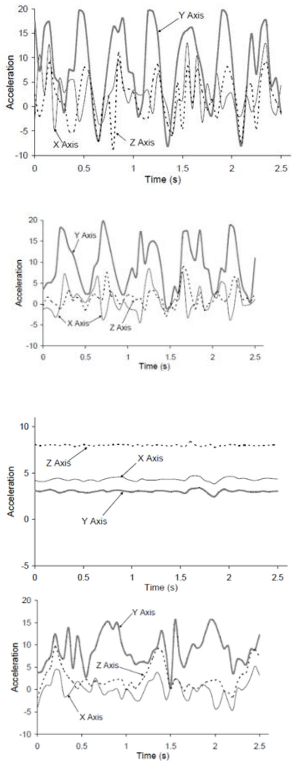
ACCELEROMETER DATA

Jogging

Walking

Sitting

Ascending stairs



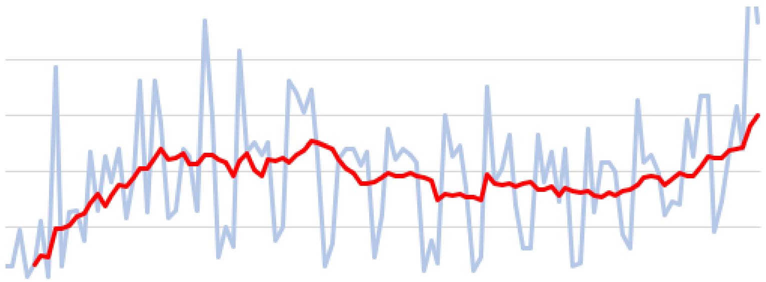
Matlab, Scikit-learn

Classifiers

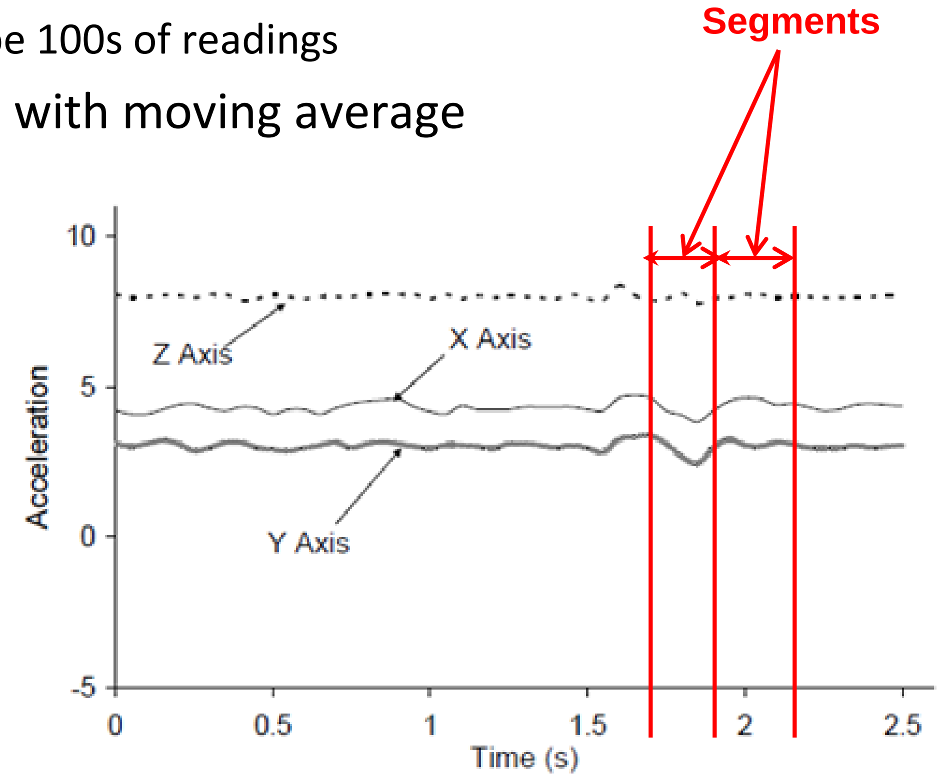
Classifier is trained offline

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

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



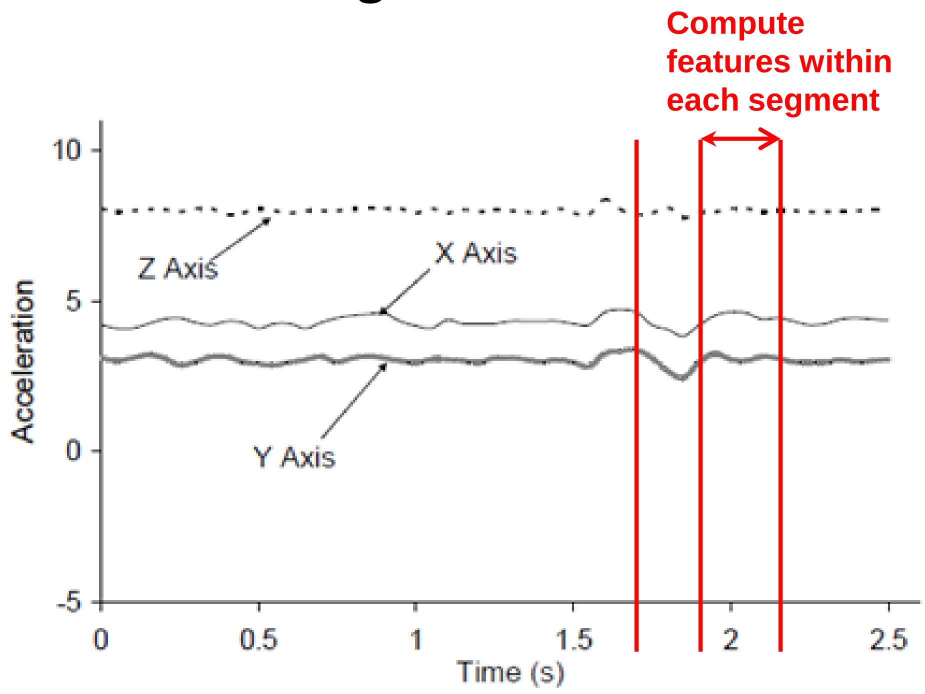
Smoothing



(e) Sitting

Step 4: Compute (Extract) Features

- For each 5-second segment (batch of accelerometer values) compute features (in scikit-learn, MATLAB, etc.)
- **Features:** Formulas to quantify attributes, characteristics of accelerometer data
- **Example features calculated using data in each segment:**
  - Minimum value
  - Maximum value
  - min-max of values
  - Largest magnitude
  - Average
  - Standard deviation
  - … various statistics



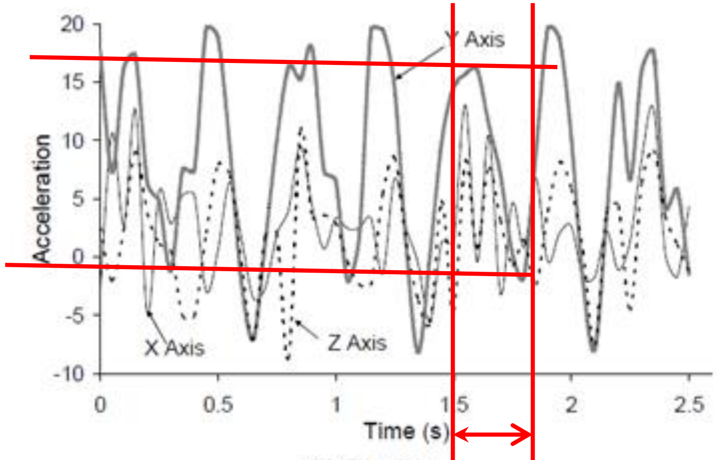
(e) Sitting

Step 4: Compute (Extract) Features

- **Important:** Ideally, feature values are different for, can distinguish each activity type (class)

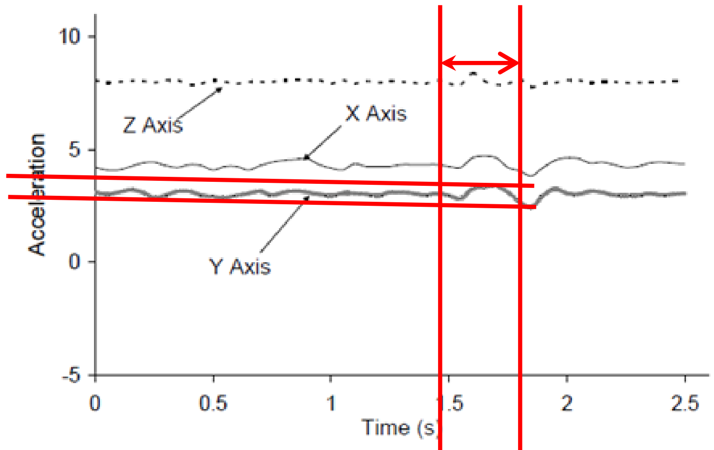
- **E.g:** Consider min-max range feature

Large min-max for jogging



(b) Jogging

Small min-max for sitting



(e) Sitting

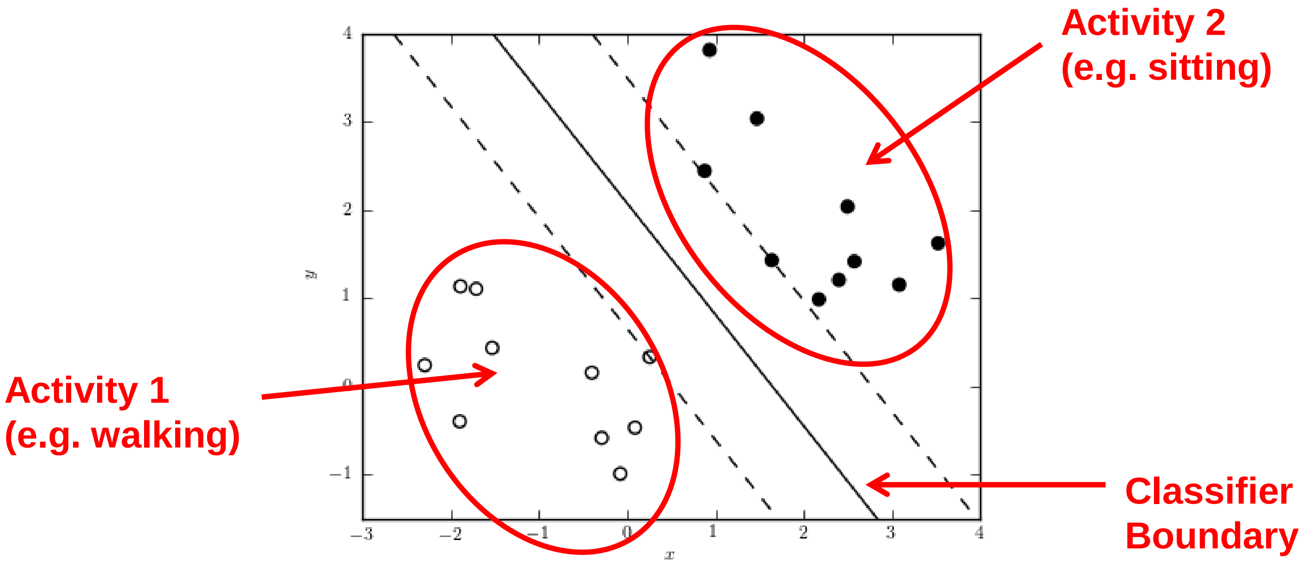
Step 4: Compute (Extract) Features

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.

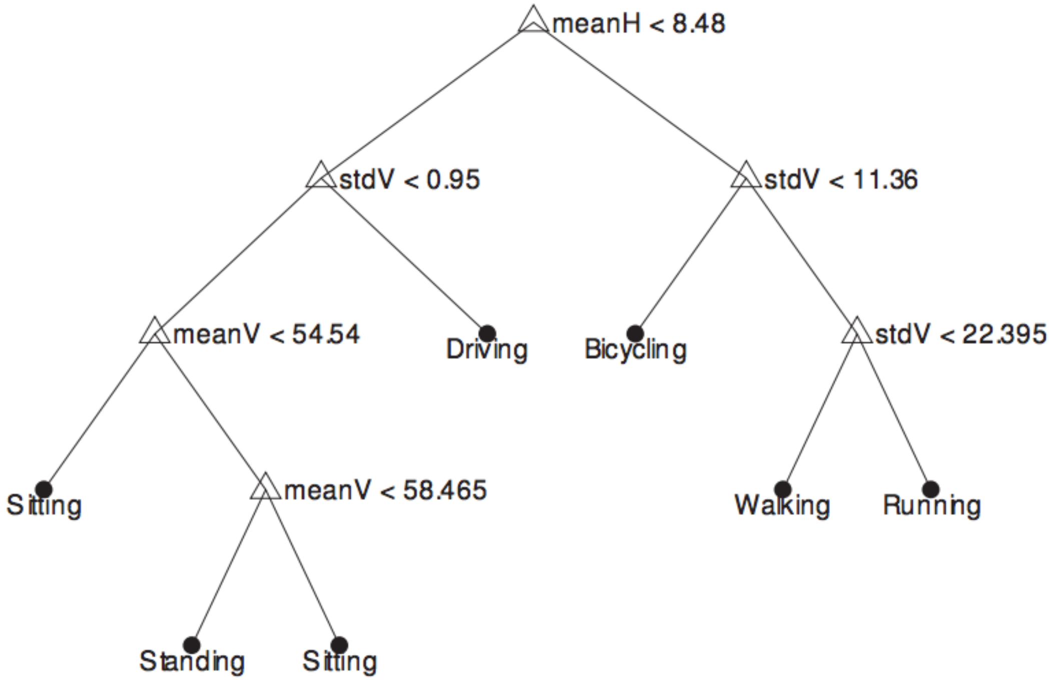
Step 5: Train classifier

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



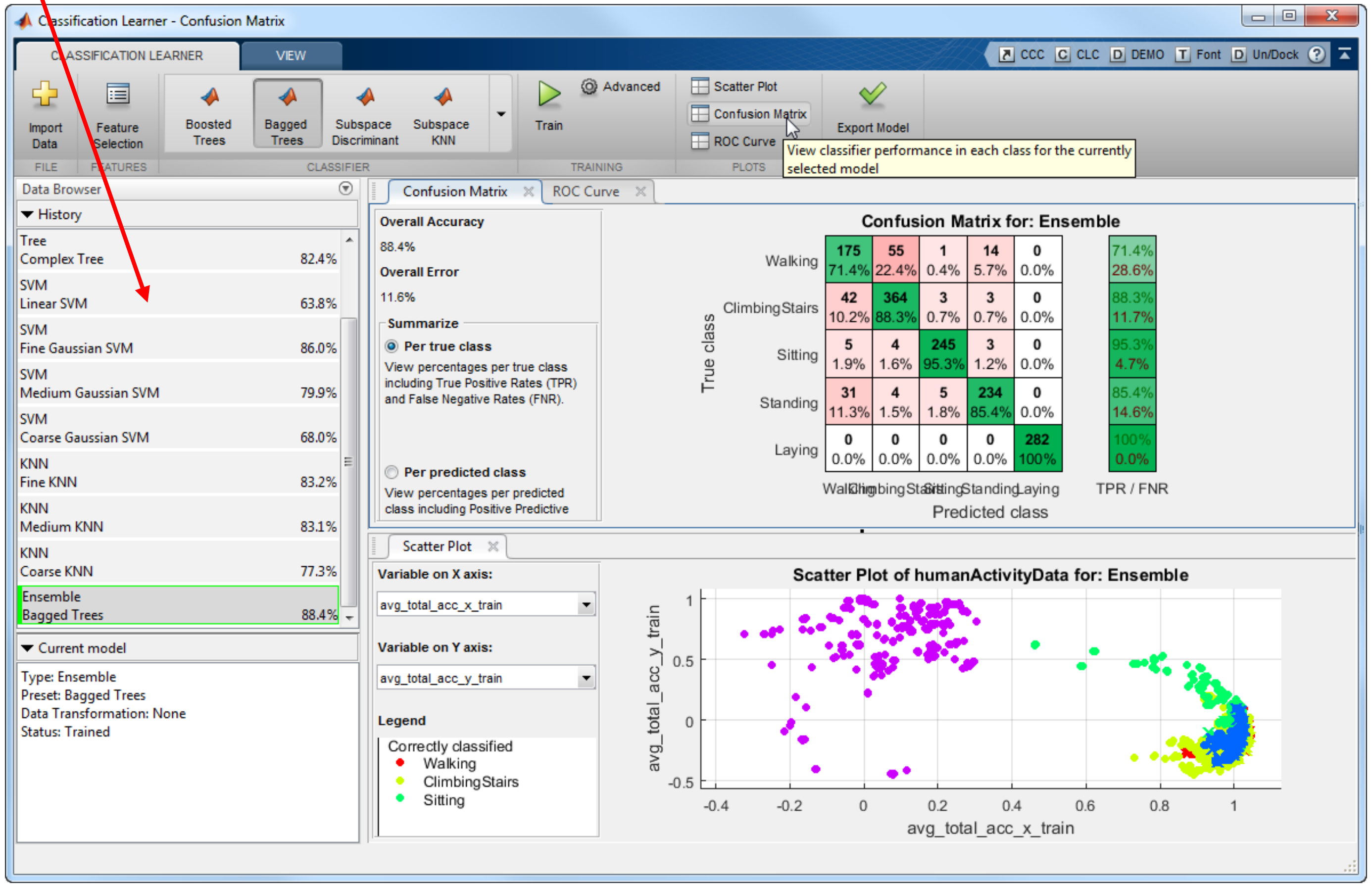
Step 5: Train classifier

- Typically split data: E.g. 80% for training classifier, 20% for testing
- **Example:** Decision Tree Classifier
- **Training:** Learns thresholds for feature values, which separate the classes
- **Testing:** How well trained model guesses labels (e.g. activity) of subjects in the test set (new examples)



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

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

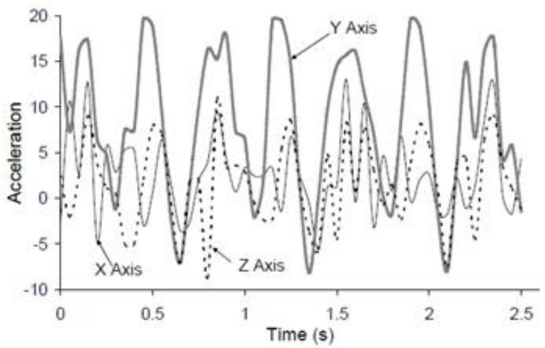
Table 2: Accuracies of Activity Recognition

	% 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: Deploy Classifier

- Export classification model (most accurate classifier type + data threshold values)
- Classifies new data live!
- Many options to deploy best classifier
- E.g. Program HTTP server, receives data, classifies, sends back results.
- In app write Android code to
  - Gather accelerometer data, segment, extract feature, send features to server

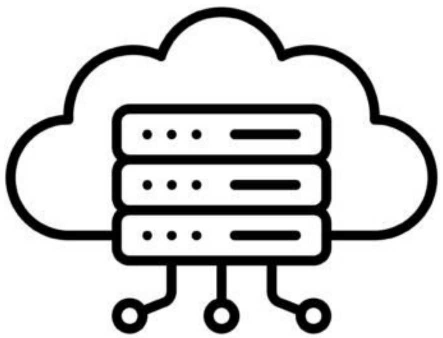


(b) Jogging

New accelerometer Sample in real time

Send data to server

Get back, display results

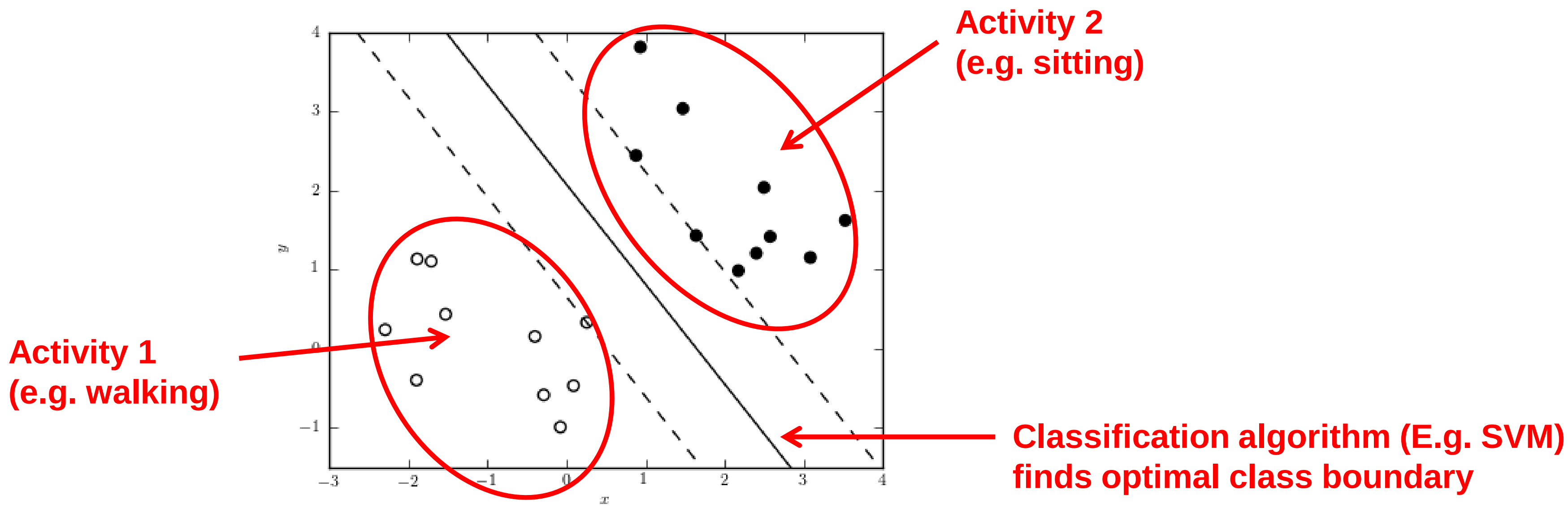


HTTP web server

Machine Learning Classification Algorithm: Support Vector Machine (SVM)

Scalable Vector Machines (SVM)

- Popular machine learning classification algorithm
- **Goal:** Determine optimal boundary between data points corresponding to different classes
- E.g. boundary between data belonging to different activities



SVM: Delineating Boundaries

- Multiple optimal boundaries exist

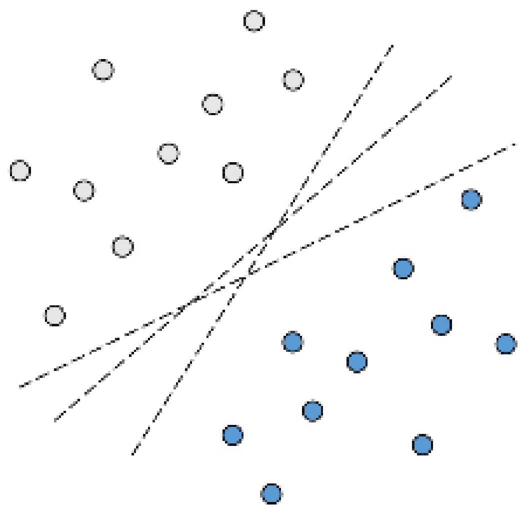


Figure 2. Multiple ways to separate two groups.

SVM: Support Vectors

- SVM steps:
  1. Finds **support vectors** in each group: peripheral data points in group 1 that are closest to points in group 2
  2. Find **optimal boundary** between support vectors of both groups
- SVM computationally efficient since it relatively few data points (support vectors)

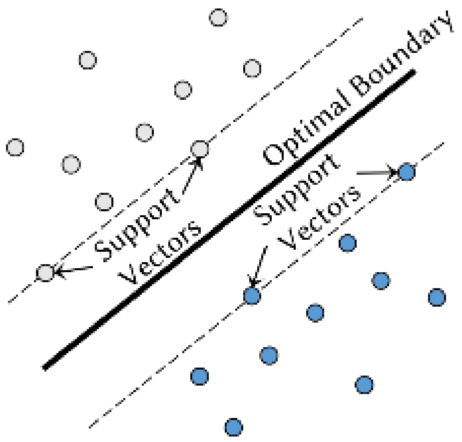
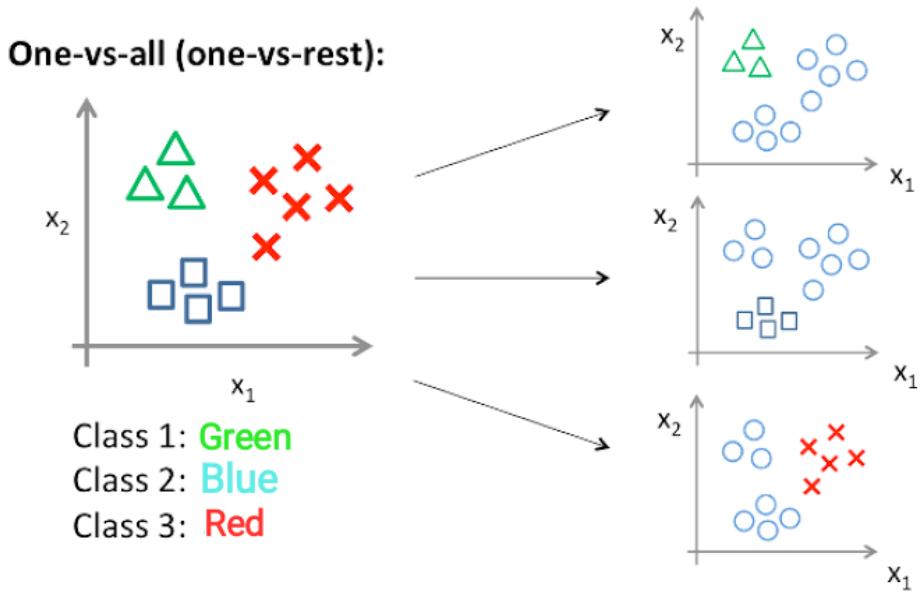


Figure 3. Optimal boundary is located in the middle of peripheral data points from opposing groups.

SVM Limitations

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

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



k-Nearest Neighbors Classifier

K-Nearest Neighbors

- Assign each point same class as majority of its *k* nearest neighbors
- E.g. if $k = 5$ (below), unknown point (?) classified as red
- Why? 4 red neighbors, 1 black

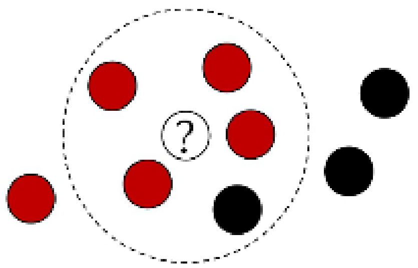


Figure 1. The center data point would be classified as red by a majority vote from its five nearest neighbors.

- *k* is the number of neighbors to consider for voting

K-Nearest Neighbors

- *k* is a parameter, which affects accuracy
  - *k* too small, algorithm considers only immediate neighbors => overfitting
  - *k* too large, tries to fit data points too far, not relevant => underfit
- **Overfitting:** algorithm determines boundaries that fits specific dataset but boundary may not hold for new data points

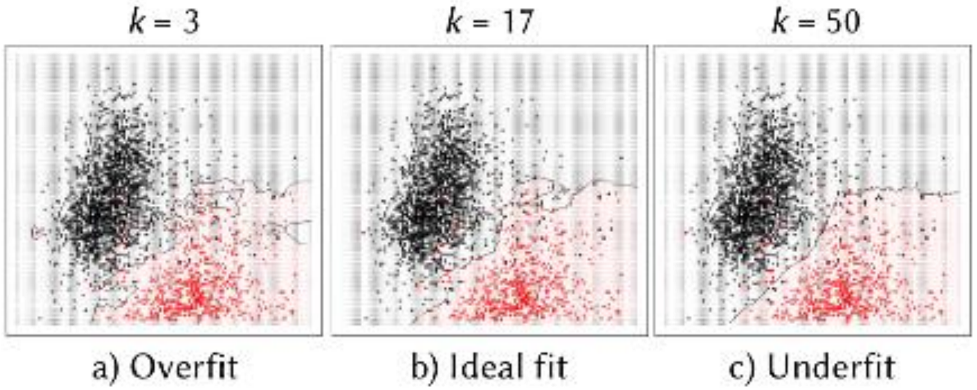


Figure 2. Comparison of model fit using varying values of k. Points in the black region are predicted to be white wines, while those in the red region are predicted to be red wines.

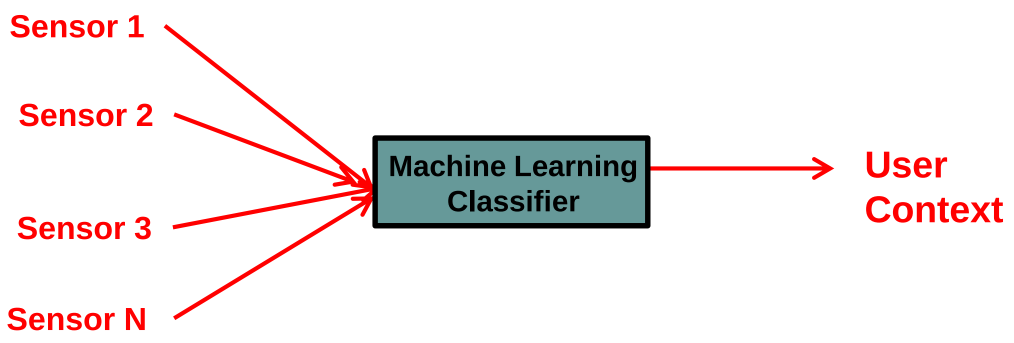
Context Sensing

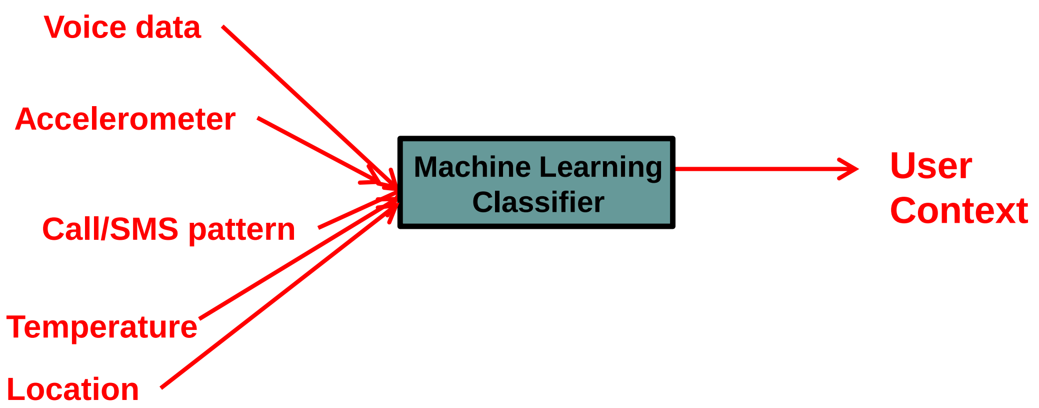
Recall: Ubicomp Senses User's Context

- 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 typically uses data from 2 sensors: accelerometer and gyroscope
- **User context recognition:** Use machine learning to analyze combined data from multiple sensors (all smartphone sensors?)





Regression

Linear Regression

- Strongest predictors of home prices are:
  1. Number of rooms in house
  2. Number of low income neighbors in area
- Linear Regression:
  1. Plot these variables for actual example homes
  2. Fit straight line of best fit
  3. Can use this best fit line to guess price of new homes

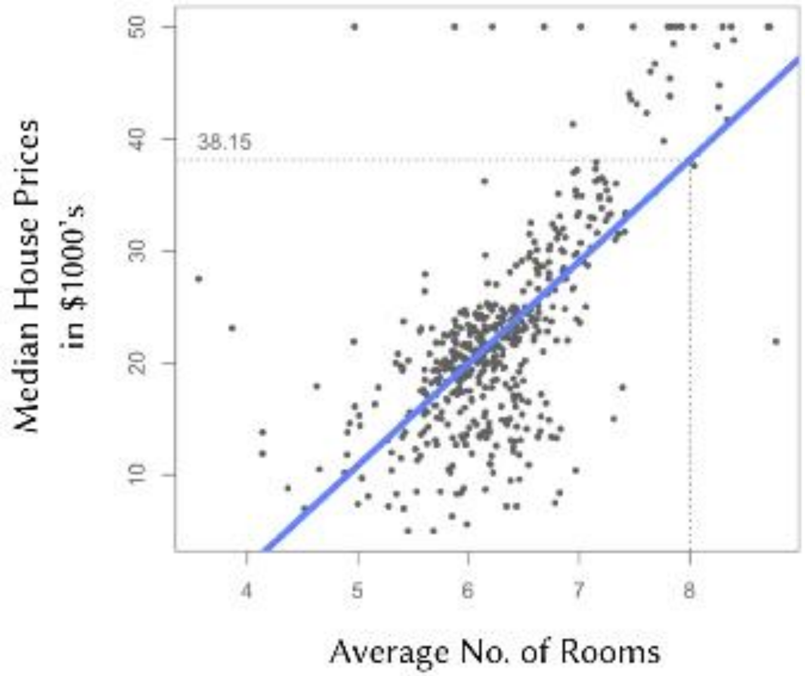
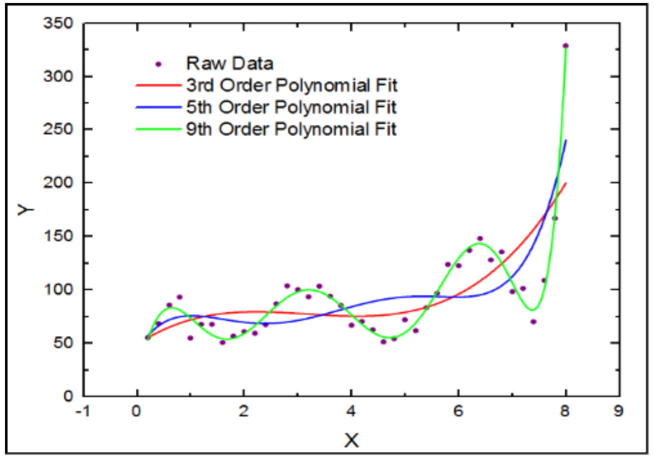


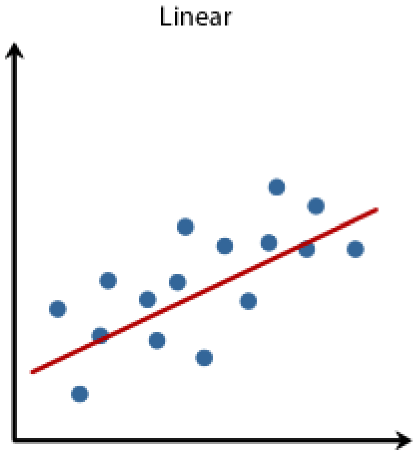
Figure 1. House price against number of rooms.

Different Types of Regression

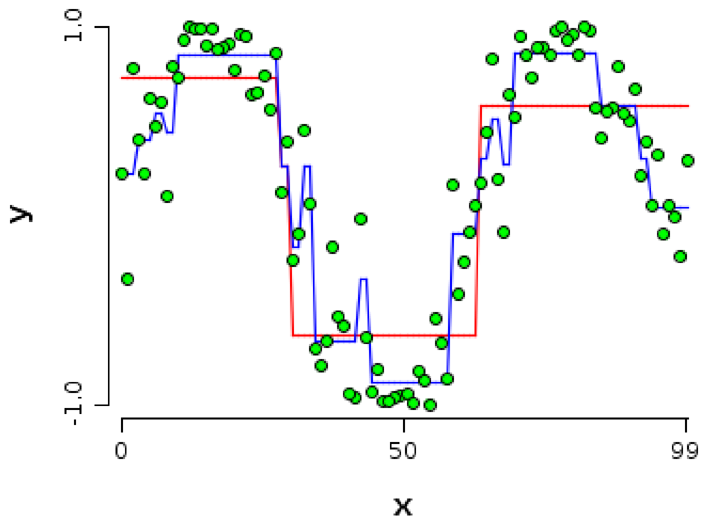
- Different regression functions to fit data to
  - Linear
  - Polynomial
  - Decision tree
  - Etc
- Determine which function has best fit, lowest error (difference)



Polynomial



Linear

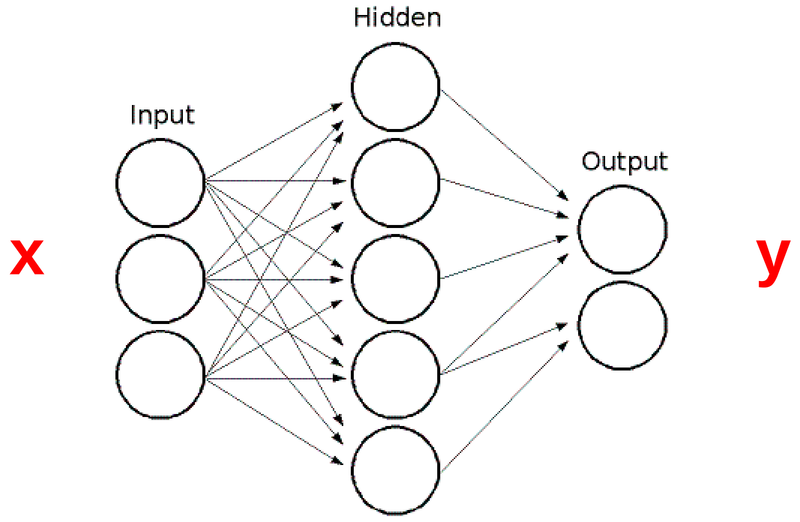


Decision Tree

Deep Learning

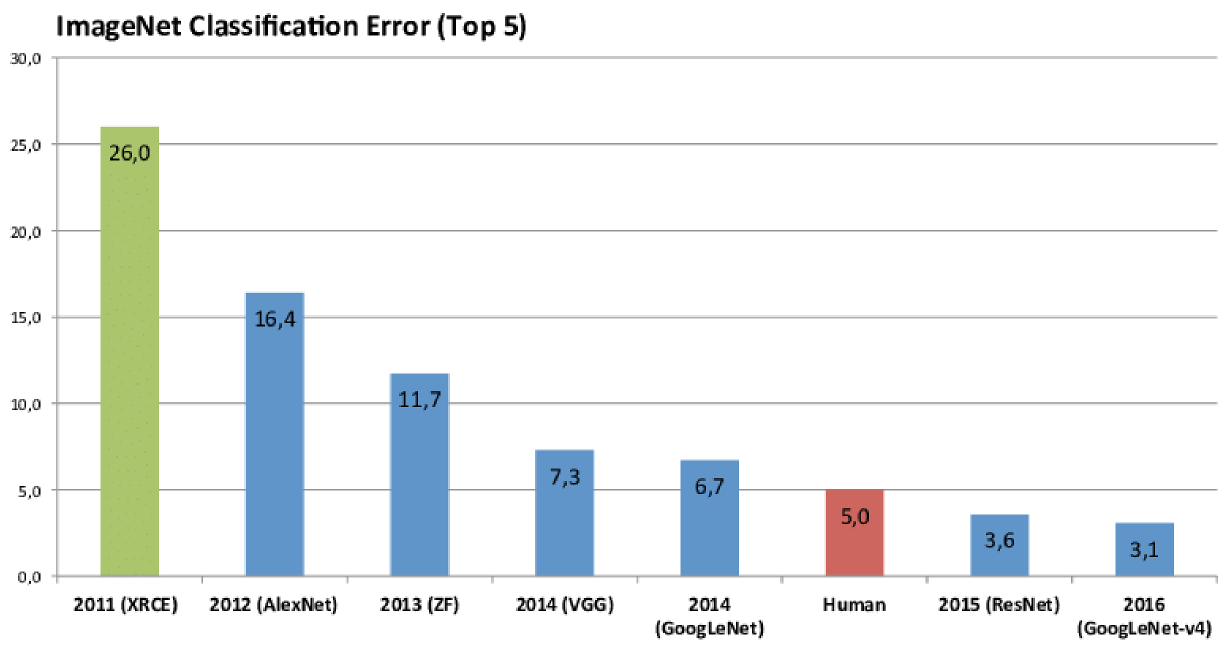
Deep Learning/Neural Networks

- Machine learning models described so far are traditional/classic ML
  - Involves converting raw data to features (extraction)
- Newer approaches use neural networks or deep learning
  - Learns directly from features or raw data (no need for feature extraction)
- **Neural Networks (NN):** Network of nodes, connectivity weights learned from data
- Learns from data, best weights of edges to classify inputs (x) into outputs y

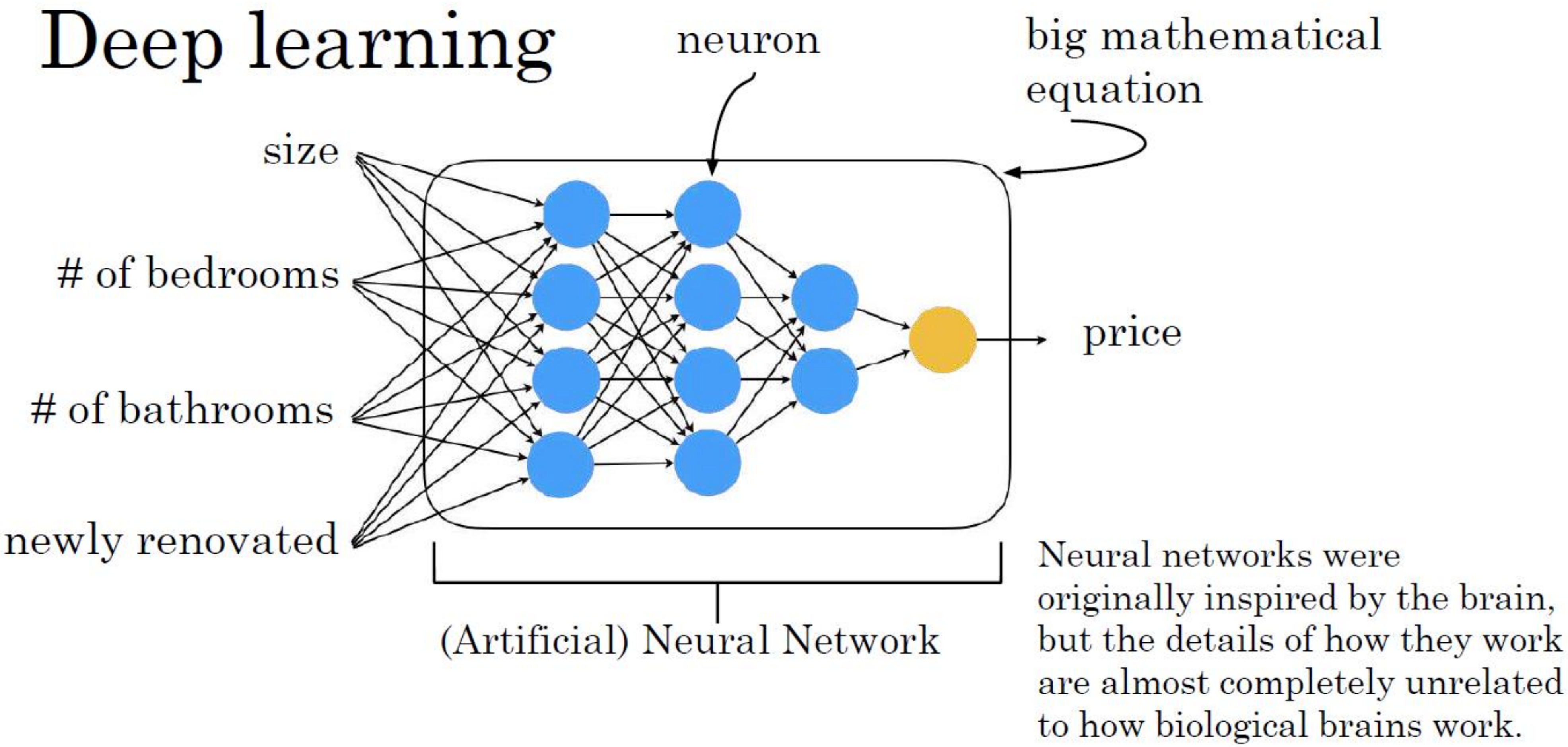


Deep Learning/Neural Networks

- NN generally more accurate if adequate data is available
- Requires lots of computational power to train
- NN first outperformed traditional ML on image classification in 2012 (AlexNet)
- For most tasks today (image, speech, text, sensor, etc.) NN solution is most accurate



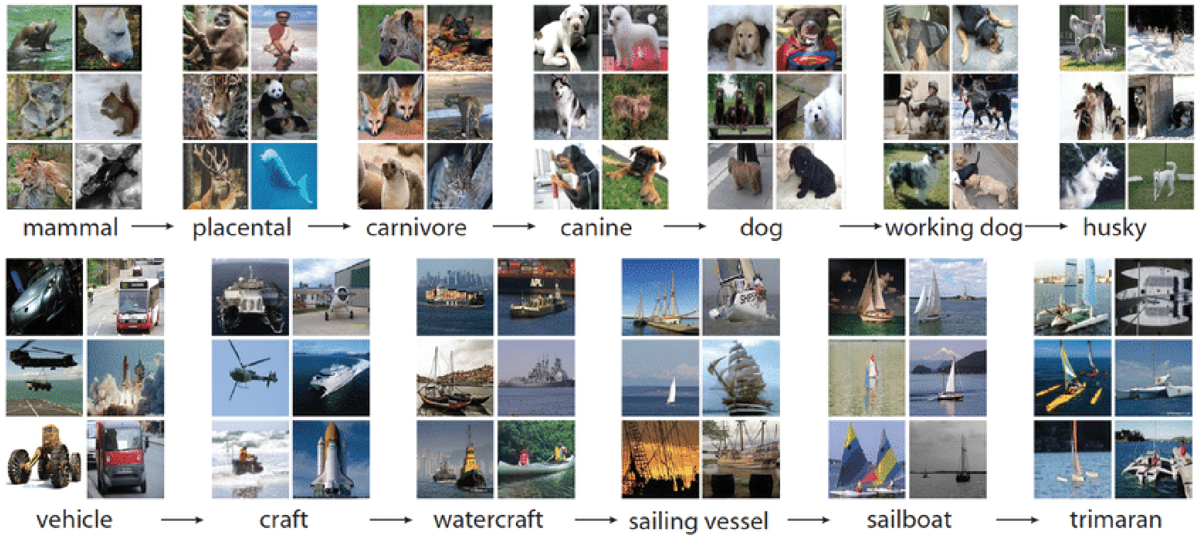
Can conceptualize as NN generates a curve/fitting function to fit data (massive equation)



Courtesy: AI for everyone by deeplearning.ai

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?
- **ImageNet:** Popular image dataset, 14 million images in 1000 classes (dogs, sailboat, etc)



ImageNet

CNN

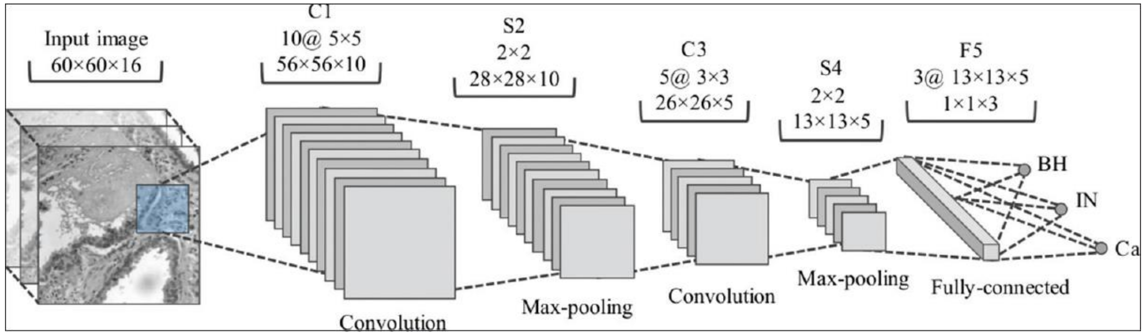


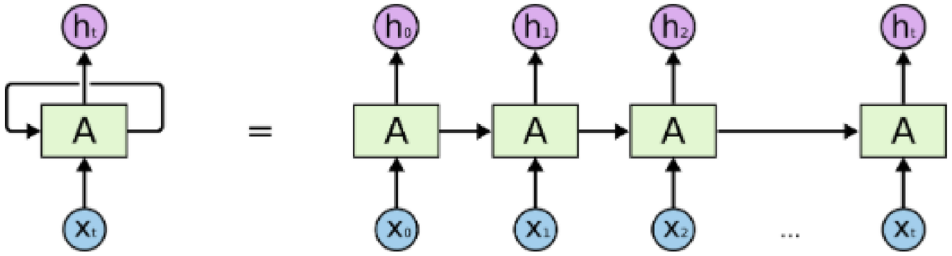
Image feature extraction (automatic)

Image feature classification

Recurrent Neural Networks (RNNs)

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

```
English: he loves soccer .
Gegman: <start> er ist ein grosser fussballfreund . <end>
Tokenized german: [1, 14, 6, 19, 571, 4200, 3, 2]

English: school's out .
Gegman: <start> die schule ist vorbei . <end>
Tokenized german: [1, 26, 304, 6, 300, 3, 2]

English: i like cookies .
Gegman: <start> ich esse gerne kekse . <end>
Tokenized german: [1, 4, 261, 149, 1656, 3, 2]

English: tom is early .
Gegman: <start> tom ist frueh dran . <end>
Tokenized german: [1, 5, 6, 391, 338, 3, 2]

English: where's tom from ?
Gegman: <start> woher kommt tom ? <end>
Tokenized german: [1, 971, 120, 5, 7, 2]
```



RNN

Programming/Mobile Support for Neural Networks

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

- Python libraries for neural networks/deep learning, train models in few lines of code
  - Keras
  - PyTorch
  - ScikitLearn
- Training neural networks on Smartphone still tough, currently only testing
- From Android 8.1: Android Neural Networks API (NNAPI) allows inference (test) of pre-trained neural networks on smartphone
  - Supports several machine learning frameworks (e.g. Tensorflow lite)

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