iLearn: The App that Learns

Rahul Rajan
CMU-SV
rahul.rajan@sv.cmu.edu

ABSTRACT

In this paper, we describe iLearn, an application that embodies the Program-by-Demonstration paradigm on the Android platform. The objective here is to be able to "demonstrate" an activity to the phone that attempts to learn it, and recognize every subsequent occurrence of this particular activity. The paper will begin with the steps involved in feature engineering the sensor data from the microphone and the accelerometer. It will then cover how a sequence of actions can be modeled using HMMs, including state discovery. It also discusses how employing user feedback in the learning process can provide for a more provocative user experience.

Author Keywords
PbD (Program-by-Demonstration), HMMs, MFCC, Android, Accelerometer.

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Algorithms, Performance, Design, Human Factors.

1. INTRODUCTION

Mobile phones, for better or worse, have woven themselves into the very fabric of our society. Their phenomenal penetration into all facets of our lives has fundamentally altered how we use them as tools and as social intermediaries. In particular, phones today have a barrage of sensors that digitize the physical world around us. On the flip side of the same coin, they keep us virtually connected to anyone on the planet. It isn't much of a stretch, then, to make the argument towards recognizing them as bona-fide social participants, if not citizens of modern society.

To buffer this line of thought it might be worthwhile to think of devices as beings. One can imagine a framework within which it would be possible to download predefined personas like Casper the friendly ghost that can be customized or adapted to the needs of the user. Some might see use in them as personal secretaries, seek them as companions or employ them as trainers. Immediately we see how voice becomes the primary mode of interaction. We also see how bringing emotional and social intelligence to these devices gives them life, so as to speak, which will be the focus of future work.

This paper describes an Android app that is a small step towards this direction. Like a lot of apps, its basic purpose would be to harness the phones capability to link the physical world with the cyber world. Only instead of being hardcoded for a particular application it can be taught to identify multiple events and respond accordingly, in essence allowing the user to intuitively script the physical world. A couple of scenarios allow us to describe it further — a phone uses its sensors to fingerprint a physical event like the door of a user's office being closed shut; it identifies the event the next time it occurs and carries out a task like blocking off the next hour on the user's calendar. If it detects multiple people in the room, it can prioritize phone calls and change the chat client status to busy. In another scenario, it fingerprints the sound of shower water running and start's a timer for 10 minutes to remind you to conserve water or can detect when you have been sitting in a single position for too long and remind you to stretch. Perhaps it can come pre-taught to detect aggravated motions and screams for help and place a call with 911 or the local police.

In section 3, we discuss the extraction of audio features from the microphone on the Android that allow us to do precisely this sort of fingerprinting. We use the framework used in popular speech recognition systems to model and learn ambient sounds (running water, whistling kettle, Starbucks). We also discuss the feature extraction for the accelerometer. For the sake of simplicity only the accelerometer and microphone sensor data are used.

In this section, we also discuss the learning algorithms used for learning the sequence of actions, namely HMMs (Hidden Markov Models) and methods to overcome the fundamental limitation of HMMs i.e. state discovery. We also compare the performance of the different methods used.

In section 4, we present the results of the system across three activities including the act of closing the door, leaving the phone on the table with random background noise, and the act of walking with the phone and clapping at the end. The purpose of the last one is to test the discriminating power of the learned model.

MobileHCI 2011, Aug 30–Sept 2, 2011, Stockholm, Sweden.
ACM 978-1-4503-0541-9/11/08-09....$10.00.

2. RELEVANT WORK

Different aspects of building such an application for the mobile phone have been touched upon by research efforts from various groups. SoundSense focuses on the use of the microphone sensor for people-centric applications [2]. The Mobile Lifelogger uses activity language to model data from the accelerometer [3]. Work by Ramos, et al. present a scheme for classification of unstructured audio based on STACS (Simultaneous Temporal and Contextual Splitting) which discovers the appropriate number of states and efficiently learns accurate HMM parameters for given data [4].

SoundSense

SoundSense discusses and implements a general-purpose system for sensing sound on the iPhone. It uses a combination of supervised and unsupervised learning techniques to classify both general sounds (e.g. music, voice) and discover novel sound events unique to the individual user. Keeping in mind the resource limitations of building a system on a mobile phone the architecture uses frame admission control based on the energy level and spectral entropy of the incoming audio signal. A low energy level indicates silence or undesirable phone context. A high entropy level represents a flat spectrum (silence or white noise). Once a frame is adjudged to have interesting content, it is sent down the pipeline to a coarse category classifier (decision tree). A Markov model recognizer is used to smooth the output from the decision tree and sent to either a voice analyzer, a music analyzer or an ambient sound learner.

The ambient sound learner is an unsupervised classifier that discovers over time environmental sounds that are significant to the individual user. MFCC (Mel-Frequency Cepstral Coefficients) are used as audio features, which inherently encode details in the frequency bands to which the human ear is sensitive. A Bayes Classifier is used to bin frames into one of B multivariate Gaussians classes. These classes are determined by a ranking algorithm based on encounter frequency and summed duration of the sounds. Similar to the coarse classifier an HMM is used to smooth the output.

Mobile Lifelogger

The Mobile Lifelogger is basically a system to aid with the easy creation and retrieval of activity logs collected from the sensors on the mobile phone. The sensory data from the accelerometer and the GPS are converted to an 'activity language' so as to apply natural language processing techniques to index, recognize, segment, cluster, retrieve, and infer high-level semantic meanings of the collected lifelogs. The other advantages of such an approach are dimension reduction, and a uniform representation of heterogeneous sensor data.

The crux of the approach lies in the analogy between human activity and language (hence the term 'activity language'). They both exhibit structure and satisfy grammars. The anatomy of the body allows us to perform certain atomic movements such as "turn upper body left" where as "jump up at 10g acceleration" is not possible. Such atomic movements form the vocabulary of the activity language. Empirically, the similarity between ambulatory activity and natural languages can be evaluated using Zipf's law. While such an approach looks promising for the purposes of the iLearn app, we believe that at least for audio classification, the data wouldn't follow Zipf's law very well.

STACS

HMMs have been used extensively in speech recognition, bioinformatics, information extraction and other areas. One of the deficiencies of HMMs is that the model topology and the number of states have to be chosen in advance. Secondly there is a tendency for the model to over fit the training data if it is sparse. The authors of [1] discuss a top-down model selection and learning algorithm that constructs an HMM by alternating between parameter learning and model selection while incrementally increasing the number of states. Candidate models are generated by splitting existing states and optimizing relevant parameters (contextual and temporal), and are then evaluated for possible selection.

The Bayesian Information Criterion, or BIC score, is an asymptotic approximation of the true posterior probability (which itself is intractable to compute) and is used to compare the candidate models to each other. The BIC score effectively punishes complexity by penalizing the number of free parameters and rewards goodness-of-fit via the data log-likelihood, thus safeguarding against over-fitting.

3. DESIGN AND IMPLEMENTATION

This section discusses the design considerations and implementation approaches that were considered to develop the iLearn app on the Android. First we discuss the feature extraction process and then the machine-learning algorithm. In the rest of the paper we assume that that the data on which we can run our learning algorithms is quite sparse and consists of three repetitions of an event.

Feature Extraction

Microphone

The challenges of developing on a cellphone are defined and limited by the strain placed on the battery, CPU and memory resources. Also microphones on telephones, walkie-talkies, etc. sample the incoming audio signal at 8 KHz, which is quite adequate for human speech. But according to the Nyquist-Shannon sampling theorem, the microphone can't capture information above 4 KHz, implying that information at these higher frequencies is lost.

On the Android, the AudioRecord class manages the audio resources for applications to record audio from the hardware. Audio is sampled into the circular hardware buffer using the 16 KHz 16-bit PCM Mono format. This data is then segmented into uniform frames for feature extraction and classification. Ideally, we would like to use the same parameters used by widely used speech recognition systems. HTK, for example, uses frames of 25 ms that overlap every 10 ms. While these are heuristic figures that have been found to work well in practice, the basic requirement is that the frame width be short enough so that the audio content is stable (steady state) while being long enough to capture the characteristic signatures of the sound.

This segmenting and overlap is implemented using the interface method provided by Android that implements a callback function every predefined period (set to 10 ms). This 10 ms data is copied into one of three arrays in a circular function, i.e. each of these arrays gets data once every 30 ms. These arrays then push their data into a larger array that can hold 25 ms of data in a circular fashion.

Once we have the frame of samples, we apply MFCC, which is a widely used feature set for audio processing. The steps involved are: (i) apply a hamming window to each frame to suppress the boundaries to counteract the effects of spectral leakage that occurs during the FFT, (ii) calculate the FFT spectrum of the frame, (iii) wrap onto the perceptual Mel-frequency scale, (iv) convert to dB, and (v) take the inverse discrete cosine transform (DCT-II since it's real valued). Only the first few (12) coefficients are used to represent a frame. Therefore MFCCs provide a low-dimensional, smoothed version of the log spectrum, and thus are a good and compact representation of the spectral shape. A modified version of the CoMIRVA project's MFCC class was implemented for this purpose [5].

Accelerometer

Accelerometers are devices that measure the proper acceleration, i.e. the acceleration relative to free-fall (which is zero-gravity). This is because they must work without any outside reference. This usually means that the only observable quantity is force. In particular, the gravitational pull of the Earth introduces a force that is interpreted as acceleration. Thus, the phone when kept idle on a table measures the force acted against it, which comes out to be 9.8 m/s^2 along the z-axis.

The accelerometer is sampled at the fastest setting provided by Android, which is *SENSOR_DELAY_FASTEST*, which has observed to be between 5-10 ms on the Droid. The readings along the x, y and z-axis are recorded at this frequency. They include the effect of the gravity component along the axes depending on how the phone was held. Some preprocessing of the accelerometer data is necessary to remove the effect of device orientation from the data. Otherwise the device needs to be held in exactly the same way for every repetition of an activity.

Applying a ramp filter (high pass filter) sufficiently negated the effect of gravity. It was noticed that noise occurs at the higher frequency and employing a high-frequency roll-off smoothens out the data.

Feature Set

We end up with a 15-D feature set (12 MFCC + Acceleration along x, y and z) which is written to a file every 50 ms for further offline processing on Matlab. Before any processing is done in Matlab the data is first standardized, i.e., the data is transformed to have zero mean and unit variance.

Learning Process

A number of machine-learning approaches were looked into, including DTW, heuristic thresholds and Bayes Classifier, before settling on HMMs, the de-facto method for pattern recognition. It is interesting to note that a key point that kept coming up while these methods were being discussed was the idea to employ some sort of temporal clustering on data that were similar in value, and seemed to belong to the same state, so as to speak. This, of course, is quite similar to the concept of HMMs.

Hidden Markov Models (HMMs)

HMMs are widely used in speech, handwriting and gesture recognition. HMMs assume that the system being modeled is a Markov process, i.e. the future states belong only on the current state. It makes sense to use HMMs for the iLearn app because we have a sequence of sensor data that we need to learn while the state sequence that produced this output remains hidden. We learn a model that statistically determines transition probabilities, output probabilities and priors among the states. For our purposes we assume that the features are independent of each other (diagonal covariance) and that the HMM is a forward model, i.e. the state transitions are limited to staying in the current state or moving to the next state. HMMs was implemented in Matlab using a modified version of the HMM Toolkit for Matlab [6].

State Discovery

One of the shortcomings of the HMM is that the number of states need to be defined beforehand. In the case of the iLearn app, it would not make sense to do this, as it contradicts the very purpose of the application, which is to be able to learn whatever event the user wants to teach it.

A couple of approaches were tried to discover the optimum number of states for a given activity. First, a brute force method is used wherein a new model is created by increasing the number of states of the prior model by one and its parameters learnt from scratch with random initializations. This new model is compared with the old model using BIC, which penalizes model complexity (state

size). This process continues until the prior model has a higher BIC score than the new model.

The second approach used the STACS method to generate N candidate models at each step, where N is the number of states in the prior model. Each split state generates one particular candidate. The initialization of the new states depends on the state that was split. The priors of the new states are half the priors of the parent state. The transitions from one state to the other are half the self-transition probability of the parent state. These candidates are compared against each other and the prior state using BIC. The state splitting continues until the prior model carries a higher score than the other candidate models.

As an addition to the second approach, the models were trained on two sequences of data (first two repetitions) and the BIC score was calculated on the held out third sequence of data to check for and prevent overfitting of the sparse dataset.

4. EXPERIMENTAL RESULTS

Feature Extraction

Initially the feature extraction process on the Android was taking upwards of 250 ms. It was realized that Java's garbage collection in Android was taking over 200 ms and that this phenomenon was occurring frequently within the MFCC extraction process. Further analysis revealed that it was the instantiation of arrays every time the MFCC was computed that was causing the garbage collection to kick in. To circumvent this, the code was modified and all the arrays were made static. This included making functions in-line and having certain variables be made public so that passing and copying of arrays could be made minimal. This brought down the time taken to compute the MFCC to a more palatable 15 ms.

The bottleneck it appeared was the wrapping of the FFT to the Mel-frequency which takes 12 ms. This step is performed by multiplying a 24x513 Mel-filter bank with the 513x1 real FFT. Normal matrix multiplication is $O(N^3)$. The fastest algorithm for multiplying matrices is the Strassen Algorithm with $O(N^{2.8})$. But the speed up is noticed only for very large matrices. Various other methods were looked into including performing the matrix multiplication in C using JNI. In Java, however, 2-d arrays are objects that need to be passed and copied from one environment to the other. The improvement in time would be negligible if at all.

The push to lower the time taken by the MFCC is to be able to extract features on 25 ms frames that overlap every 10 ms. However, in [2] Lu, et al. show that a larger frame length tends to increase recall but decrease precision for ambient sounds. 64 ms it appears is a good trade-off point considering both precision and recall. Thus we see that the length of the MFCC frame is critical as it impacts the computational costs and classification accuracy. It would be an important parameter to tweak once an end-to-end system is deployed and there is sufficient data to run tests on.

Learning Process

We evaluate the three different state discovery approached independently, and then use a test matrix to check their performance across different activities.

Brute Force

Using this approach, we find that because of the random initializations, the fact that the EM algorithm is prone to find local minima, and the sparseness of the data, the state size varies a lot. To learn the door close activity over three repetitions, the state size varies from 2 to 8 states. The log likelihood on the training data falls in between -2500 to -1500.

STACS

Here we find this approach overfitting the training data very strongly. Upwards of 20 states is usually learnt on the three sequences of training data. The algorithm seems to be assigning states to single data points as the log likelihood becomes positive (+250) for some of the candidate models. This is characteristic of overfitting in Gaussians. Because there is only one data point, the variance goes to zero and the likelihood goes to infinity.

STACS using Held-out data

To overcome the strong overfitting tendencies of the STACS method on our sparse dataset we decided to try training on just two sequences, while computing the BIC score of the model using the log likelihood over the third training sequence. With this approach the number of states learnt varies from 1 to 3 states (mostly 2). The log likelihood on the training sequence is a lot more consistent too at around -1550.

Test Matrix

Three datasets were collected and used for the initial experimental analysis. The first one consists of the activity of closing the door (Activity 1). The second one consists of the phone being idle on the table with random background noise (Activity 2). The third consists of walking with the phone and clapping at the end (Activity 3). This is purely to test the basic discriminative powers of the model as opposed to testing real user-scenarios. Each data set consists of 10 repetitions or 10 sequences of the same activity. The results are presented in Table 1.

For the purposes of the initial tests, the models obtained from the different state discovery approaches were trained on only Activity 1 and tested across unseen data of the three activities. Furthermore, the models were trained on only three sequences of the Activity 1. A simple 1-state HMM,

	1-state	Brute	STACS	STACS-2
Activity 1	-7310	-7814	-Inf	-10352
Activity 2	-14639	-18205	-Inf	-16616
Activity 3	-7131	-9155	-Inf	-8298

Table 1. Performance of models trained based on the four methods described above across three different activities. Values are in log likelihoods.

which effectively becomes a multivariate Bayes Classifier is also included to compare effectiveness.

5. ANALYSIS

The performance evaluation shows that the 1-state model performs well in discriminating Activity 1 from Activity 2, but is not able to demonstrate the same prowess with regards to Activity 3. We can clearly see the advantage of the HMM looking at the results from the Brute model wherein it was able to pick the right activity. The discrimination from Activity 2 is also higher.

STACS did not perform well and this was expected as the algorithm was severely overfitting the training data. In fact, on the training data we were achieving a log likelihood of 39.44, but it was not able to classify the unseen data.

The modified STACS where the BIC is calculated over unseen data shows an improvement but is not able to perform as strongly as the Brute because it's trained on less data. The difference between learning two and three samples is plain to see.

Hence, while the brute force method might seem inefficient, we are not as affected by this because we have a small dataset, and it makes sense to use this approach in the future.

6. FUTURE DIRECTIONS

There are a couple of steps that will be attended to in the immediate future. First, would be to build a WFST to achieve real-time decoding that allows us to identify an event if and when it occurs. The second is to stress test the system once an end-to-end system is built, and learn the optimum parameters like frame size and BIC criterion. Third, we would like to allow for a more provocative user experience by encouraging user-feedback and using that to perform online learning.

A learning system such as this is inherently going to have a lot of false positives and is highly dependent on user feedback to improve prediction and usability. The most natural way to go about this would be to use vocal feedback empowering the app to 'listen' to what the user is saying without requiring them to become the users main focus. This brings up a couple of research questions — can we go beyond a simple "yes" and "no" response model to a more natural "cool, that is right!" where we are able to classify any response as positive or negative? What methods (character, tone of voice, when to ask) can be employed so as to be most pleasing to the user? For instance, certain traits in students make teachers more patient with some students than with others. Identifying a couple and including them in our system could encourage constructive user feedback and hence improve the system.

Further down the road, we need to figure out how to use the data available from the rest of the sensors on the phone, and methods for homogenous representation. We also need to figure out how and when to select sensors as appropriated by the activity.

7. CONCLUSIONS

This paper presents a unique user experience scenario wherein the user is able to teach the phone to recognize an event or a series of actions by demonstrating the same to the phone. The vision is that as the modern cellphone matures to become a personal companion, it would need to learn more and more from and about the user, with voice being the main mode of communication. This work represents a step in that direction.

Overall we built a system that is able to record sensor data (microphone and accelerometer in particular) over a sequence of actions and machine-learn it in order to recognize them the next time it occurs. Different machine-learning approaches were considered and evaluated, with the brute force method coming out on top. What remains is being able to perform decoding in real time, and be able to employ user feedback for online learning.

8. REFERENCES

1 Siddiqi, Sajid, Gordon, Geoffrey J., and Moore, Andrew W. *Fast State Discovery for HMM Model Selection and Learning*.

2 Lu, Hong, Pan, Wei, Lane, Nicholas D., Choudhury, Tazneem, and Campbell, Andrew T. SoundSense: scalable sound sensing for people-centric applications on mobile phones. In *Proceedings of the 7th international conference on Mobile systems, applications, and services* (2009), ACM, 165-178.

3 Chennuru, Snehal, Chen, Peng-Wen, Zhu, Jiang, and Zhang, Ying. *Mobile Lifelogger - recording, indexing, and understanding a mobile user's life*.

4 Ramos, Julian, Siddiqi, Sajid, Dubrawski, Artur, Gordon, Geoffrey, and Sharma, Abhishek. *Automatic State Discovery for Unstructured Audio Scene Classification*.

5 Schedl, Markus: CoMIRVA

6 Murphy, Kevin: HMM Toolkit for Matlab