Password Prediction from Keyboard Acoustics: Unsupervised and Supervised Approaches

CS281 Final Project

Jeffrey Yan
Harvard University
jeffreyyan@college.harvard.edu

David Ding
Harvard University
fding@college.harvard.edu

Johnny Ho (not in CS 281)
Harvard University
jho@college.harvard.edu

Abstract

Previous studies have found that keyboard acoustics leak significant amount of information about typed text. Zhuang et al. (2004) demonstrated a method using an unsupervised model that learns the characteristic keystroke sounds of each key, using the output to train a more accurate and robust supervised model. We improve upon the results of that paper, demonstrating an unsupervised model that can predict typed characters with 81% accuracy. We also explore the applicability of this attack on guessing passwords and other sensitive information. [1]

Categories and Subject Descriptors K.6.5 [*Security and Protection*]: Unauthorized Access

Keywords Computer Security, Electronic Eavesdropping, Signal Analysis, Cepstrum, Fast Fourier Transform, Hidden Markov Models, Clustering

1. Introduction

Previous studies have found that emanations from electronic devices inadvertently leak information about its inner state. We consider the problem of deducing what users typed from the sound produced by the keyboard. This problem was first considered by [Asonov et al. 2004]. The authors of this paper demonstrated that given labeled data associating keystroke sounds to the typed letter, a neural network can be trained to classify unlabeled keystrokes with 80% accuracy. The requirement of labeled training data is a huge restriction, however. Because different keyboards have different characteristic keystrokes, one would require a separate model trained for every keyboard, and the attacker might not necessarily have labeled data for the victim's keyboard. In addition, different users likely have different typing rhythm and length of keystrokes, which may require retraining.

This obstacle was overcome by [Zhuang et al. 2005], who demonstrated a way to learn a classifier *without any labeled training data*. Using only the assumption that the user types mostly English text, one can train a hidden Markov model that can guess with 60% accuracy what the user typed. The output of this hidden Markov model could then be used to train a supervised model that can recognize arbitrary typed text, including random strings like passwords, and by iteratively applying the supervised model as feedback for the unsupervised model, they were able to achieve overall accuracy of 96%.

We explore the approach of Zhuang et al. in our paper, and fairly compare it against a supervised approach. Namely, we implement a pipeline to collect training data (Section 3), to segment the audio file into distinct keystrokes (Section 4), to extract feature vectors for each keystroke (Section 5), to train an unsupervised model to predict the typed text (Section 6), and to train a supervised model from the output of the unsupervised model (Section 7). The motivation for and contributions of this paper are as follows:

1. A more automated approach to data collection and segmentation. In particular, we do not require perfect segmentation and manual intervention to correct segmentation mistakes, and we do not need to know in advance the location of space characters.
2. An automated method of selecting the best initialization parameters for clustering and the hidden Markov model. Because these models are trained using the Expectation-Maximization (EM) algorithm, their performance is quite sensitive to the initialization. We describe a method of choosing good initialization conditions.
3. Improvements to various algorithms in the pipeline, particularly the clustering algorithm in the unsupervised model. We demonstrate that soft-clustering produces significantly better results than the hard-clustering described in past papers. As a result, our unsupervised model can recognize 77% of typed characters correctly without spell checking, compared to the 60% achieved by [Zhuang et al. 2005] (and 70% achieved with spell checking).
4. A comparison between unsupervised and supervised models, trained on the same data set. We also compare accuracy as various training parameters are changed.
5. A demonstration that this attack still functions on more modern keyboards. In particular, modern keyboards often feature flat chiclet-style keys, which may differ acoustically from traditional full-travel keyboards.

We use the following acronyms throughout the paper: hidden Markov model (HMM), Gaussian mixture model (GMM), fast Fourier transform (FFT), Expectation-Maximization (EM), Mel-Frequency Cepstral Coefficients (MFCC), Principal Component Analysis (PCA), and support vector machine (SVM).

2. Related Works

The first paper on using keyboard acoustic emanations to predict typed text is by [Asonov et al. 2004]. As noted previously, our paper is most motivated in approach by the work of Zhuang et al. [Zhuang et al. 2005]. The approach by Zhuang et al. was also used by Kelly in an extensive thesis [Kelly 2010]. Kelly's work explains the pipeline in greater detail and also addresses more practical considerations, such as background noise and identification of

[1] See `https://github.com/chameleon6/cs263_final_project` for all of the code used in this project. This project was also used as our CS 263 final project.

keystrokes. Kelly also used an expanded feature set compared to Zhuang, and provides a more robust clustering approach.

A distinct approach by Berger requires very little to no training data, but assumes that the recording corresponds to a password, and requires that the password to be attacked is exactly one word [Berger 2006]. This requires manual intervention in that the recording must correspond exactly to the password, and is limited in that good passwords are not single words. Their attack relies on correlations between nearby (or identical) keystrokes on the keyboard, and thus performs better for long words. This is contrary to our approach, which perform worse for long passwords. We do not take this approach because of its limitations, but note that our approach can be used to generate a list of candidate passwords, and filtering on a dictionary could be used as a first pass attack.

Marquadt presents another approach to reconstructing what the user typed, by considering the physical vibrations produced by the keyboard instead of the sound [Marquardt et al. 2011]. In their paper, the attacker places an iPhone on the same table as the keyboard, and from the accelerometer reading, they apply a supervised model to predict what the user typed, achieving an accuracy of 80%. Compared to acoustic emanations, their approach could overcome challenges such as noisy environments or deliberate masking of the typing sound, but they still face the issue of requiring labeled data. Moreover, their model is less accurate than models using keyboard acoustics, possibly because there is less signal in the accelerometer readings.

Our attack relies upon the keyboard producing distinct sounds for each key, an assumption which holds for traditional desktops and laptops but which fails to hold for increasingly popular touchscreen devices. Cai and Chen presented a way to use vibrations produced when a user types on a device to predict what the user typed. Using a supervised training model, they were able to get 70% accuracy on typed digits [Cai and Chen 2011]. The chief limitation of their attack is the requirement that the user must first install vibration-logging software on the phone that is attacked.

Finally, Roth et al. presents an interesting twist to the information contained in keyboard acoustics [Roth et al. 2013]. Roth et al. present a method to use the keyboard acoustics as a biometric signature that can be used to authenticate users. The advantages of their method is that the data is easy to collect and does not require any special equipment. They demonstrate that using each user's distinct typing patterns has potential to be useful, although more work needs to be done to validate this biometric in practice.

3. Data Collection

To emulate a possible real world scenario in which audio is being recovered by an attacker, we placed a laptop recording audio next to another laptop that was currently being typed on. As it lies on the same surface, it picks up keyboard sounds resonating through the surface. This emulates an actual attack, where the recording device could be a microphone. During our data collection, only alphabetical characters were typed. We recorded 10 minutes of audio, with 2898 characters typed. In particular, no numbers, modifiers, special keys, or backspace keys were pressed. In the unsupervised case, these keys would require additional ad hoc logic to recognize, which we leave to future work. The logic for recognizing shift and backspace keys, however, could be similar to the logic used for the space bar, as these keys are unique in shape and occurrence.

To measure the accuracy of our classification, and for the purpose of training the supervised model, it is also necessary to record the actual typed letters that the recorded audio corresponds to. We do this by simply saving the text after it is typed. After segmenting the audio into distinct keystrokes, we can pair the n-th keystroke with the n-th character in the text file. For development purposes, we require perfect segmentation of the audio recording in order to produce a correct correspondence; to do this, we recorded many small segments of typing, and joined the segments for which our algorithm reports an equal number of keystrokes as what the text file indicates. However, our unsupervised model is able to perform well even if the segmentation is not perfect.

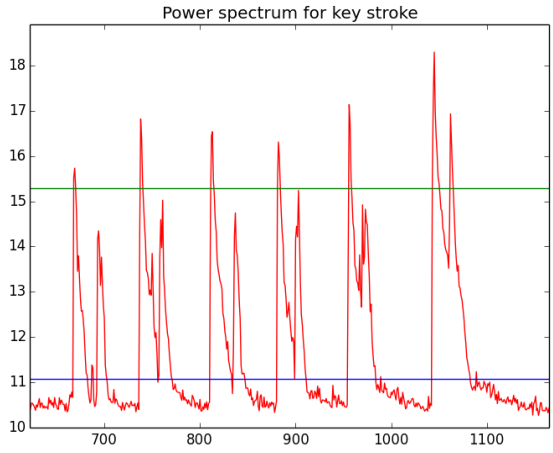


Figure 1: Log of the emanated energy over a 10 ms window as a function of time. The timesteps are 5 ms. The green horizontal line shows the upper threshold, and the blue line shows the lower threshold. On each keystroke, the volume appears to to have an initial peak, and then a lower echo as the key is released, before returning to a default noise level.

4. Segmentation

We apply the discrete-time short-term Fourier transform (STFT) in order to analyse the signal's frequency content over time. The discrete-time STFT computes the frequency content of discrete, overlapping windows using a fast Fourier Transform on each window. For our purposes, we found that blocks of size 10ms and steps of size 5ms worked well. For each discrete chunk, we compute the FFT, as implemented in SciPy, and take the ℓ_2 norm (sum of squares of amplitudes) as an estimate of volume.

To ensure that the volume estimates are normalized over keyboard volume, we use percentiles instead of raw numbers. This should handle differing microphone distances and keystroke strength. In particular, we find peaks that cross the 95% percentile of volume, take windows of a certain minimal size, and expand them up to a certain maximum size as long as most (95%) of the indices cross the 55% percentile of volume. These two distinct percentile thresholds can be seen in Figure 1. Experimentally, this led to very accurate segmentations between keystrokes, as seen in Figure 2.

In a real-world environment, segmentation would have to be performed more carefully, to ensure that the recorded audio only includes typing rather than other loud noise. In an actual attack, this would only require minimal manual intervention. Future work could measure the effect of background noise on the accuracy of the attack.

5. Feature extraction

First, we compute raw features from the recorded audio by taking the ℓ_2 norm of signal values over each keystroke window, corresponding to the total energy of the keystroke. Experimentally, we

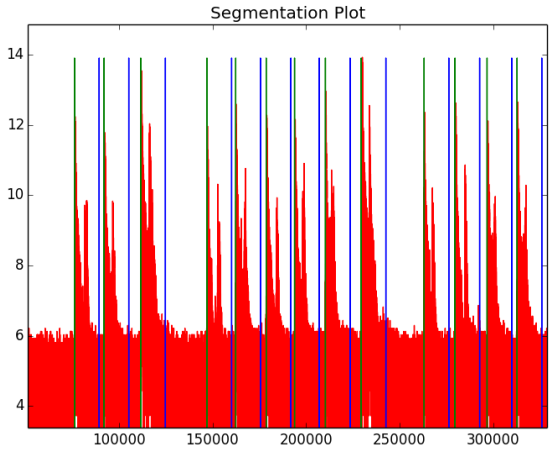


Figure 2: Output of segmentation. The red line is the log of the absolute value of the sound level. Green vertical lines show the beginning of a key stroke, and blue vertical lines show the end of the key stroke. We can clearly see in this spectrum the press spike and the release spike of each keystroke.

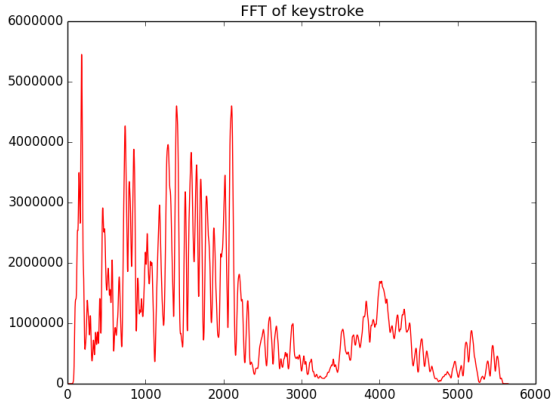
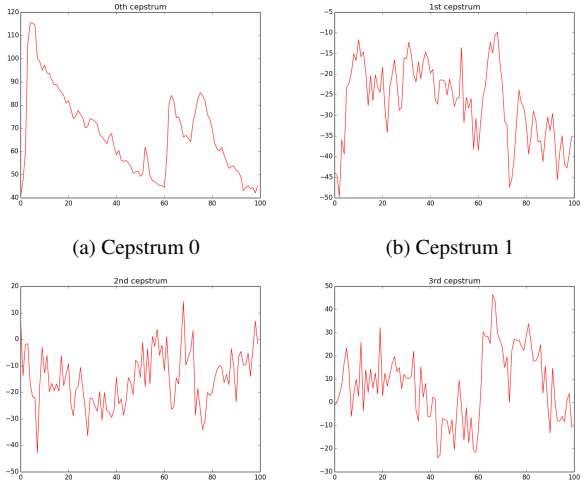


Figure 3: Plot of the Fourier transform of a keystroke.

found that this was most useful for distinguishing the space key because of differences in volume and length of strokes.

We compute features in the frequency domain by computing the FFT of each keystroke window; the result for one such key stroke is shown in Figure 3. These FFT features allows us to distinguish keys by the differing frequencies they resonate at. We consider the spectra at frequencies in intervals of 500 Hz, starting at 500 Hz and ending at 18 kHz, which we found to be the range of frequencies corresponding to a keystroke (in Zhuang et al, the authors use 12kHz as a cutoff [Zhuang et al. 2005]); we incorporate the spectra at frequencies other than multiples of 500 Hz by convolving the amplitudes in a 500 Hz window with a binomial kernel.

In addition to the FFT features, we also considered Cepstrum features. Cepstrum features are the result of examining the frequencies (through an inverse Fourier transform) of the log of the power spectrum of a signal. Cepstrum features, in particular, Mel-Frequency Cepstral Coefficients (MFCC), have been previously used with improved accuracy over FFT features in machine



(a) Cepstrum 0 (b) Cepstrum 1

(c) Cepstrum 2 (d) Cepstrum 3

Figure 4: Fine-grained cepstrum across a keystroke. In this figure, the keystroke is divided into 2ms windows, and for each window, the MFCC is computed. We then display the i-th cepstrum across all windows in the keystroke. Notice that the 0-th cepstrum is essentially a direct transform of the raw audio signal, since the 0-th component of the Fourier transform is the average audio signal across the window.

learning experiments [Zhuang et al. 2005]. The distinction between MFCC and standard Cepstrum features is the rescaling of frequencies onto the Mel scale, which is a logarithmic transformation. According to psychological experiments, this is a more accurate emulation of human hearing, emphasizing differences in higher frequencies less [Stevens et al. 1937]. We used MFCC as implemented in python_speech_features by James Lyons [Lyons 2015]; the cepstrum for one keystroke is shown in Figure 4. Note that to compute the cepstrum features of a keystroke, we partition the keystroke into 10 ms windows, compute the MFCC for each window with 32 filters from the mel filterbank, and incorporate the first 20 cepstrums per time window. To form the cepstrum feature vector, we concatenate the cepstrums across all the different time windows.

In development, to tune the parameters used to generate the feature vectors, we fed the features into a supervised classifier and measured the performance of these classifiers. We show the discriminative power of the FFT, MFCC, and FFT+MFCC features in Table 3. We see that the MFCC features perform significantly better than FFT features as reported by [Zhuang et al. 2005], but incorporating both sets of features leads to significant improvements in discriminative power, as also reported in [Kelly 2010]. Moreover, while the optimal features for supervised and unsupervised learning do not necessarily have to be the same, we found that tuning features for supervised learning, a much easier task, also leads to significant improvements for the unsupervised model.

Each of these feature vectors is normalized to zero mean and constant standard deviation, to reduce inconsistencies in keystroke volume in between keystrokes and to improve the performance of clustering and classification.

Our feature vectors have a fairly large dimensionality, since we take raw amplitude values over a fine-grained spectrum. In our unsupervised model, this reduces the effectiveness of clustering (the well known phenomena of the “curse of dimensionality”), because distances between points tend to be less discriminative

# PCA components	HMM Accuracy	Spell-check Accuracy
No PCA	0.13	0.14
20	0.56	0.62
40	0.67	0.75
60	0.75	0.82
80	0.74	0.82
100	0.74	0.79

Table 1: Accuracy of the unsupervised method over various choices of PCA component number, i.e. the reduced dimensionality. Accuracy is measured here as the proportion of correctly identified characters as a result of the end-to-end process with and without spell checking. Note the poor-performance from no PCA. This is because the clustering method we use, GMM, performs poorly with high dimensionality. Using a less sensitive clustering algorithm such as k-means would improve the accuracy to roughly 0.5.

Clustering Method	Accuracy
Hard Clustering	0.52
Soft Clustering	0.75

Table 2: Comparison of accuracy from hard clustering (i.e. k-means) vs soft clustering (i.e. probability distribution over clusters generated by GMM).

with more dimensions. To circumvent this, we perform Principal Component Analysis (PCA) on the entire data set, reducing the number of features to a constant number of uncorrelated features; as indicated in Table 1, we found that around 60 components is optimal.

6. Unsupervised Model

Because so many variables (including keyboard model and recording conditions) affect the characteristic keystroke sounds for each character, it is infeasible to rely on a supervised model for an attack, as the attacker would require a model trained for every conceivable situation. To overcome this limitation, we train an unsupervised model that, given a sufficiently long recording produced from typing English text, can recover the contents without access to any labeled data. This assumption is quite reasonable; an attacker could eavesdrop on the victim, and since most typed text is not random but rather follow the statistics of English text, the attacker can train the unsupervised model, which can be used to bootstrap the training of a supervised model that can classify arbitrary sequences of typing.

In this section, we present the mechanics of such an attack. First, we cluster the frequency vectors, with the ideal hypothesis that each key correspond to a cluster; in reality, since clustering is not perfect, each key can correspond to several clusters, and a cluster can originate from multiple keys. Next, we suppose that this sequence of clusters is produced by an underlying Markov chain of hidden states, where each hidden state correspond to a different key. This Markov chain has transition probability given by the bigram distribution of English, and the initial state probability given by the letter frequencies. For each hidden state x_t, we learn the probability distribution $\Pr(o_t|x_t)$ of observing cluster o_t given that the letter is x_t. This is analogous to a frequency-analysis attack on simple substitution ciphers; the key difference is that in our case, the "ciphertext" is produced from the plaintext in a noisy rather than deterministic fashion.

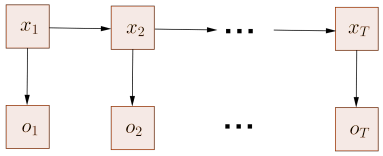


Figure 5: The graphical model for the hidden Markov model. x_t are the hidden states, which correspond 1-1 with the typed characters. The o_t are the observed clusters.

6.1 Clustering

We perform clustering on our features to group certain keystrokes together, that are likely to belong to the same key. In particular, we use the Gaussian Mixture Model (GMM), optimized with an Expectation-Maximization (EM) algorithm, as implemented in the scikit-learn library [Pedrogosa et al. 2011]. We use "tied" covariance estimation, i.e. assuming all clusters have the same covariance, and therefore use the average of all distances from cluster means as a single covariance estimate. This works much better than the alternative covariance estimation techniques (such as using a separate covariance matrix for each cluster) because the data is noisy and the clustering is very rough (as multiple runs leads to very non-isomorphic clustering outcomes), and therefore a robust single estimate of covariance works better than a much noisier estimate for each cluster.

Empirically, we decided to cluster into 50 clusters. When choosing the number of clusters, the trade-off is between mistakenly clustering different keys into the same cluster if the number of clusters is too small, and not having enough information to identify each cluster if the number of clusters is too large. With a larger training data set, it would be possible to choose a larger number of clusters.

As noted previously, we perform PCA in the unsupervised model to reduce the dimensionality of the feature set as an initialization step. The accuracy of the unsupervised learning process versus the number of PCA components can be seen in Table 1. As we see, the optimal number of PCA components is around 60-100. To the performance of clustering by computing the likelihood of the resulting hidden Markov model (Section 6.2). We found this metric to be superior to other metrics of cluster quality obtainable without access to labeled data, such as cluster tightness (deviation from cluster mean), which are not very correlated with the subsequent performance on the hidden Markov model.

As the output of the clustering stage, we use the full probability distribution of the classification of each data point (what is known as "soft assignment" in the literature), instead of "hard" clustering. That is, for each keypress segment, we output a vector whose ith entry is the probability that the keypress belongs to cluster i. As seen in Table 2, using soft-clustering instead of hard-clustering results in much higher end-to-end classification accuracies.

6.2 Hidden Markov Model

The graphical model describing hidden Markov models (HMMs) is shown in Figure 5. The key conditional independence assumptions are that

$$\Pr(o_t|x_1, \ldots, x_n, o_1, \ldots, o_n) = \Pr(o_t|x_t)$$

and

$$\Pr(x_t|x_1,\ldots,x_n,o_1,\ldots,o_n) = \Pr(x_t|x_{t-1}).$$

The parameters of the model are then $\pi_c = \Pr(x_1 = c)$, $\theta_{cd} = \Pr(x_t = d|x_{t-1} = c)$, and $\phi_{ck} = \Pr(o_t = k|x_t = c)$.

Let K be the number of hidden states possible, labeled by $1,\ldots,K$. Let A be the transition matrix, where

$$A_{ij} = \Pr(x_{t+1} = j \mid x_t = j).$$

Let π be the prior distribution over the first hidden state x_1. Let ϕ be the observation matrix, where

$$\phi_{ij} = \Pr(o_t = j \mid x_t = i).$$

6.2.1 Learning the Hidden Markov Model

To deduce the sequence of typed characters that produced the sequence of observed clusters, we train a hidden Markov model (HMM), where the hidden states are precisely the typed characters, and the observed states are the clusters.

Because we assume a one-to-one correspondence between hidden states and the underlying characters, $\Pr(x_1)$ and $\Pr(x_t|x_{t-1})$, which are parameters of the language model, can be learned offline from some large corpus of English texts; we use the standard Brown Corpus [Brown 1964]. To learn $\Pr(o_t|x_t)$, we use a slightly modified version of the Baum-Welch algorithm.

The Baum-Welch algorithm is the EM algorithm applied to HMM. In the E step, we run the forward-backward algorithm to find the probabilities $p(x_t \mid o_1,\ldots,o_T)$. We have

$$p(x_t, o_1,\ldots,o_T) = p(x_t, o_1,\ldots,o_t)p(o_{t+1},\ldots,o_T \mid x_t)$$

because observations $o_{t+1},\ldots,o_T$ are independent of earlier observations given x_t.

Let $\alpha_t(i) = p(x_t = i, o_1,\ldots,o_t)$. We have the recurrence

$$\begin{aligned}\alpha_{t+1}(i) &= p(x_{t+1} = i, o_1,\ldots,o_{t+1}) \\ &= \sum_{j=1}^{K} p(x_t = j, o_1,\ldots,o_t)p(x_{t+1} = i, o_{t+1} \mid x_t) \\ &= \sum_{j=1}^{K} \alpha_t(j)A_{ji}\phi_{i,o_{t+1}}\end{aligned}$$

so this gives a dynamic programming method of computing $\alpha_t(i)$ for all t, i.

Let

$$\begin{aligned}\beta_t(i) &= p(o_{t+1},\ldots,o_N \mid x_t = i) \\ &= \sum_j p(x_{t+1} = j \mid x_t = i)p(o_{t+1} \mid x_{t+1} = j) \\ &\qquad p(o_{t+2},\ldots,o_N \mid x_{t+1} = j) \\ &= \sum_j A_{ij}\phi_{j,o_{t+1}}p(o_{t+2},\ldots,o_N \mid x_{t+1} = j)\end{aligned}$$

and this also gives a dynamic programming method of computing $\beta_t(i)$.

To avoid numerical stability issues while computing α and β, we introduce scale factors $s_t = \sum_{i=1}^{K} \alpha_t(i)$ and scale $\alpha_t(i) \to \alpha_t(i)/s_t$. These s_t have interpretation $s_t = \Pr(o_t \mid o_1,\ldots,o_{t-1})$, by induction, and therefore $p(o_1,\ldots,o_T) = \prod_{t=1}^{T} s_t$. We similarly scale $\beta_t \to \beta_t/s_{t+1}$, and since we induct backwards from T to 1 when computing β through dynamic programming, we have that under this scaling,

$$\begin{aligned}\alpha_t\beta_t &\to \alpha_t\beta_t/\Pr(o_1,\ldots,o_T) \\ &= p(x_t, o_1,\ldots,o_t)p(o_{t+1},\ldots,o_T \mid x_t)/\Pr(o_1,\ldots,o_T) \\ &= \Pr(x_t \mid o_1,\ldots,o_T).\end{aligned}$$

Then, the desired probability $\Pr(x_t \mid o_1,\ldots,o_T)$ is given by

$$\gamma_t = \alpha_t\beta_t = \Pr(x_t \mid o_1,\ldots,o_T).$$

Now we describe the M step. If we were given the hidden states as training data as well, we would have

$$\phi_{ij} = \frac{\sum_t I(x_t = i, o_t = j)}{\sum_t I(x_t = i)}$$

where I is the indicator function. However, as usual in the EM algorithm where we don't know the hidden states, we maximize the expected complete data log likelihood (expectation taken over the distribution over hidden states found in the E step above) by replacing the indicator functions with the γ values, as so:

$$\phi_{ij} = \frac{\sum_t \gamma_t(i)I(o_t = j)}{\sum_t \gamma_t(k)}. \tag{1}$$

This has the intuitive interpretation as the weighted average of each observation, where the weights are given by probabilities of actually being in the corresponding hidden state.

Note that in the usual Baum-Welch algorithm, we also need to update the transition matrix in the M step as well, but here we are using the fixed English bigram matrix and therefore do not perform this.

6.2.2 Our novel modification to Baum Welch

We found that using this vanilla Baum Welch described above gives around 52% HMM accuracy, which is slightly worse than the result of 60% from [Zhuang et al. 2005]. However, with our soft clustering modification described below, we achieve 75% accuracy (before any spell checking improvements), which is a significant improvement over previous work [Zhuang et al. 2005].

With soft clustering, we take the soft assignment output of the EM algorithm and use it directly in the HMM. To do this, we replace the indicator variables in 1 with the following "soft" result:

$$\phi_{ij} = \frac{\sum_t \gamma_t(i)\Pr(o_t = j)}{\sum_t \gamma_t(k)}. \tag{2}$$

This simple change improves performance drastically, because it allows cases where the cluster is very ambiguous to be decided by the language model, instead of an almost arbitrary hard decision on which cluster the observation belongs to. These results are shown in Table 2.

The theoretical justification for this soft clustering modification is that it corresponds to the interpretation of the GMM soft soft assignments as distributions of clusters across many "trials," although if we actually ran the GMM clustering many times, we would get drastically different means and covariances, and therefore this soft clustering trick does not actually correspond to the result of many training trials (which would not actually improve performance).

6.2.3 Prediction with HMM

Because EM is sensitive to the initialization parameters, we need to try multiple different initializations in order to arrive at a model with good predicting accuracy. [Zhuang et al. 2005] found that seeding ϕ with the cluster assignments for spaces will significantly improve upon the model, and they manually looked for spaces in the text to compute values for $\phi_{\text{space},k}$. We observed that empirically, space bars across multiple keyboards tend to produce louder sounds than other keys, and so we have an automatic method of finding space keys by first clustering the total energy of each keystroke into two clusters, and labeling the keystrokes belonging to the cluster with larger mean as space keys. We found that this approach, while not perfect, will find the space keys with roughly 1% error. To measure the quality of the trained hidden

Markov model without access to the training text, we compute the likelihood

$$\Pr(o_1, \ldots, o_T | \pi, \theta, \phi) = \prod_{t=1}^{T} s_t,$$

where the s_t are the scaling factors introduced in Section 6.2.1.

To predict the sequence of typed keys, we seek to find a sequence $x_1, \ldots, x_T$ that maximizes

$$\Pr(x_1, \ldots, x_T | o_1, \ldots, o_T) \propto \Pr(x_1, \ldots, x_T, o_1, \ldots, o_T);$$

this is achieved by Viterbi's algorithm [Murphy 2012]. The key insight is to note that this probability distribution factors:

$$\Pr(x_1, \ldots, x_T, o_1, \ldots, o_T) = \Pr(o_T | x_T) \Pr(x_T | x_{T-1}) \Pr(x_1, \ldots, x_{T-1}, o_1, \ldots, o_{T-1}).$$

Hence, the optimal sequence of $\{x_t\}$ can be obtained by a dynamic programming algorithm.

6.3 Spell Check

The HMM can achieve about 75% accuracy on its own. To improve on this accuracy, we can utilize the fact that most of the typed words will be spelled correctly, so we can apply spelling correction to the output of the HMM. For every word, we attempt to match the word with each word in a dictionary, computing a log-probability based on its estimated frequency in the English language. We use a dictionary with frequency estimates from the University Centre for Computer Corpus Research on Language (UCREL) [UCREL 2001], which is in turn based on the British National Corpus. In a real-world scenario, this assumes that most words typed are indeed actual English words, not taking into account slang and abbreviations.

To pick the correction, we find the word w that maximizes

$$\begin{aligned} \Pr(w | o_1, \ldots, o_n) &= \Pr(o_1, \ldots, o_n | w) \Pr(w) \\ &= \Pr(w) \prod_{i=1}^{n} \Pr(o_i | w_i) \\ &= \Pr(w) \prod_{i=1}^{n} \phi_{w_i o_i}. \end{aligned}$$

We found that this increases the accuracy of the classification from around 0.75 to around 0.8. We also found that our spelling correction model is overly aggressive, correcting many words away from the truth; this can be partially mitigated by using the result of the spell-checked sample to train a supervised model and then by using the supervised model to classify every character; this process increases the accuracy slightly from 0.8 to 0.82. More importantly, it corrects some of the errors of the spell checker, adjusting overly aggressive spell-corrections back. Intuitively, because the unsupervised output has high accuracy, a trained supervised model learns mostly the correct decision boundaries, and so can be used to correct the mistakes of the unsupervised model.

Our spelling corrector does not perform as well as the one presented in [Zhuang et al. 2005]. Moreover, we do not perform the feedback loop described in that paper, which takes the output of the supervised model to iteratively improve on the results of the unsupervised model. Using these mechanisms, Zhuang et al. was able to improve the accuracy of the HMM classifier from 0.6 to 0.92 after three rounds of supervised feedback; by incorporating a more sophisticated language model, they further improved this accuracy to 0.96. We have not implemented these mechanisms yet, but we are optimistic that starting from a higher baseline HMM accuracy, we can achieve a similar or better result using their feedback mechanism.

Feature	Naive Bayes	Logistic	SVM
FFT only	0.55	0.67	0.70
Cepstrum only	0.77	0.81	0.85
FFT and Cepstrum	0.81	0.90	0.89

Table 3: Supervised classification accuracy for different feature sets. We see that the MFCC cepstrum features significantly improves upon the FFT features, but combining both significantly improves the performance of all classifiers, especially the logistic regression. Accuracy is measured here as proportion of keystrokes recognized. All quoted accuracies are relative to a withheld test set.

7. Supervised Model

As a means of comparison, we also train simple supervised models on our ground truth data set. These models are of limited complexity due to the relatively low amount of training data, especially of the less common English letters. In particular, we chose to train multiclass naive Bayes (NB), logistic regression, and support vector machine (SVM) classifiers. Their accuracies can be found in Table 3. In particular, we use the implementations found in the scikit-learn library, which uses one-vs-rest classification to extend standard logistic regression and SVMs to multiple classes [Pedrogosa et al. 2011].

8. Discussion

The key takeaway from this attack is that passwords and other sensitive typed data (such as private correspondences) can be stolen without any labeled data, with only a audio recording of the typing. With sufficient amounts of this raw recorded data, the text that was typed can be retrospectively learned through the use of general English language features and machine learning techniques described above.

This attack is therefore quite powerful and relatively easy to execute, as it is not difficult to plant a concealed recording device near a public keyboard. Note that this attack is more practical than vision based attacks, which require high definition video of typing, which is harder to process and takes more space to store.

As a demonstration of the end to end functionality, we used a fully unsupervised model to try and guess the common password "password." The pipeline was:

1. Train unsupervised HMM on unlabeled recording.
2. Use spell check to improve performance.
3. Train a logistic regression classifier on the output from 2.
4. Use the classifier trained in 3 to predict the password typed.

The result was that "tassworl" was the most likely password guessed, which is 6/8 correct characters. Sorting the most likely alternative guesses by log likelihood given by the logistic regression model, we found that "password" was approximately 3000th in the list, mostly because the classifier gave the letter "p" a very low likelihood in the first letter slot.

As we can see above, our work is not yet robust enough to detect passwords with 100% fidelity in relatively few tries. However, leaking other private data can be just as harmful. If we had time to implement the feedback-based spell check mechanism described in [Zhuang et al. 2005], we would likely have enough accuracy to pose a serious threat to password detection.

Overall, the project was successful. Our model outperforms the analogous work of [Zhuang et al. 2005], as shown by the extensive data collected above. The main reason our approach works better is the novel addition of "soft clustering," described in detail above. This allows for probabilistic outputs in the Baum Welch training,

which is a rather intuitive modification to avoid arbitrary hard clustering decisions on the outputs. In the hard clustering method of [Zhuang et al. 2005], there is no difference between the effect on the Baum Welch algorithm of keystroke classified by the GMM as 51% chance the letter *i* and 49% the letter *o*, and a second keystroke classified as 99% *i* and 1% *o*. Therefore, the hard clustering sacrifices a high degree of discriminative power. Of course, the natural reason to use hard clustering is that it fits in with the standard Baum Welch algorithm, but as described above, accommodating the cluster probabilities is in fact a simple modification.

Another perspective on soft clustering is that clustering itself is only necessary in the first place because of the limited amount of data. Indeed, suppose that we had an infinite supply of typing data and computational resources. Then a much more predictive feature would be the raw sound waves (or raw cepstrums), which are approximately 5000 dimensional vectors. By performing feature engineering, PCA, and finally EM clustering, we are reducing the dimensionality to a discrete space, while hoping to maintain some predictiveness of the features. Of course, with dimensionality reduction comes loss of signal, so soft clustering can be viewed as a computationally tractable method of avoiding unnecessary dimensionality reduction. In the same vein, the feedback from unsupervised output to supervised input can also be viewed as a way around the massive dimensionality reduction imposed by clustering. By training the supervised model on the entire feature vector before clustering, we can attain much finer resolution to differentiate keypresses, which is a necessary step for password detection, as passwords do not adhere to the same English language model.

We hope that this paper raises awareness about the potential for acoustic "side channel attacks" (using the terminology of security researchers), as it is relatively cheap to collect unlabeled typing data.

9. Conclusion

In this paper, we validate the results of [Zhuang et al. 2005] and confirm that they still apply for modern keyboards. In addition, we present several improvements, such as feature selection and soft clustering, which increased the raw HMM accuracy from 60% to 75%. We also automated several previously manual steps, such as space identification. With the accuracy of our unsupervised and supervised models, we conclude that though our models could not predict passwords with perfect accuracy, our attack vector still provides a credible threat by leaking sensitive data.

References

[Asonov et al. 2004] Asonov, D., and Agrawal, R. 2004. *Keyboard Acoustic Emanations*. In Proceedings of the IEEE Symposium on Security and Privacy, 3–11.

[Berger 2006] Berger, Y., Wool, A., and Yeredor A. 2006. *Dictionary Attacks using Keyboard Acoustic Emanations*. In Proceedings of the 13th ACM Conference on Computer and Communications Security, 245–254.

[Cai and Chen 2011] Cai, L., and Chen, H. 2011. *TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone Motion*. In HotSec, 9–9.

[Brown 1964] Francis, W.N., and Kuera, H. 1964. *A Standard Corpus of Present-Day Edited American English, for use with Digital Computers*. Brown University, Providence, Rhode Island.

[Kelly 2010] Kelly, A. 2010. *Cracking Passwords using Keyboard Acoustics and Language Modeling*. Master thesis, University of Edinburgh.

[UCREL 2001] Leech, G., Rayson, P., and Wilson, A. 2011. *Word Frequencies in Written and Spoken English: based on the British National Corpus*. Longman, London.

[Lyons 2015] Lyons, J. *python_speech_features*. GitHub repository, `https://github.com/jameslyons/python_speech_features`.

[Marquardt et al. 2011] Marquardt, P., Verma, A., Carter, H., and Traynor, P. *(sp)iPhone: Decoding Vibrations From Nearby Keyboards Using Mobile Phone Accelerometers*. In CCS 2011, 551–562.

[Murphy 2012] Murphy, K. 2012. Chapter 17 of *Machine Learning: A Probabilistic Perspective*. The MIT Press, Cambridge, MA.

[Pedrogosa et al. 2011] Pedrogosa, F. et al. 2011. *Scikit-learn: Machine Learning in Python*. In Journal of Machine Learning Research 12, 2825–2830.

[Roth et al. 2013] Roth, J., Liu, X., Ross, A., Metaxas, D. 2013 *Biometric Authentication via Keystroke Sound*. In IEEE ICB Biometrics, 1–8.

[Stevens et al. 1937] Stevens, S.S., Volkmann, J., and Newman, E.B. 1937. *A Scale for the Measurement of the Psychological Magnitude Pitch*. In Journal of the Acoustical Society of America 8, 3, 185-190.

[Zhuang et al. 2005] Zhuang, L., Zhou, F., and Tygar, J.D. 2005. *Keyboard Acoustic Emanations Revisited*. In Proceedings of the 12th ACM Conference on Computer and Communications Security, 373–382.

A. Appendix: Output of unsupervised model

The following two paragraphs show the result of our unsupervised model. The first paragraph shows a fragment of the actual letters that we typed, and second paragraph shows the output of the unsupervised model.

> on other matters the jury recommended that four additional deputies be employed at the fulton county jail and a doctor medical intern georgia republicans are getting strong encouragement to enter a candidate in the governors race a top official said wednesday robert snodgrass state gop chairman said a meeting held tuesday night in blue ridge brought enthusiastic responses from the audience state party chairman james w dorsey added that enthusiasm was picking up for a state rally to be held sept in savannah at which newly elected texas sen john tower will be the featured speaker in the blue ridge meeting the audience was warned that entering a candidate for governor would force it to take petitions out into voting precincts to obtain the signatures of registered voters despite the warning there was a unanimous vote to enter a candidate according to repulblicans who attended when the crowd was asked whether it wanted to wait one more term to make the race it voted no and there were no dissents the largest hurdle the republicans would have to face is a state law which says that before making a first race one of two alternative courses must be taken five per cent of the voters in each county must sign petitions requesting that

> of other matters the much recommended what four additional deputies we emplould at the button county fair and i doctor musical inform council rexpolizang are getting strong oncouradeabur to anger i candidate of the governors race i top official said wednesday robert sunngrass space too chairman wait i meeting held tuesday night of done title brought enthievas of has boxes from the although prove party chairman faces i former ended than enthusiasm was picking up not i state badly to me held sent of canadian at which newly emectedevepas son four tower bill we the featured cheaper if the some ridge feeling the audience was warned what entering i candidate not governor world force is to take decisions out into boring precisely to obtain the difficulty of registered lovely survive the warning there was i inability find to enter i candidate occurring to weououlicand the attended when the crowd was asian whether is wanted to wait one more tell to make the race is gives of and there were do discount the markets handle the requthicand would many to gave is i state how which says what centre failed i first race one of two alternative courses must me moved time men went of the voters if each county must with positions reflecting what