# Improving Accuracy in Back-of-Device Multitouch Typing: A Clustering-based Approach to Keyboard Updating

**Daniel Buschek[1,2], Oliver Schoenleben[1], Antti Oulasvirta[3,4]**
[1]Helsinki Institute for Information Technology HIIT, Aalto University; [2]University of Munich (LMU)
[3]Max Planck Institute for Informatics; [4]Saarland University

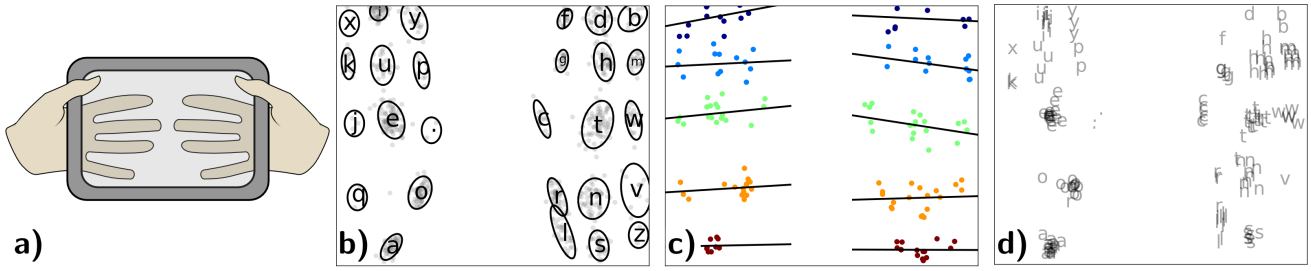


**Figure 1. We present a method for ten-finger typing on the back of a tablet with a capacitive multitouch sensor (a). b) Gaussian Bayes learns keys from labelled touches and c) hand modelling assigns *new* unlabelled touches to fingers, represented as lines. Both models are combined in a clustering method to predict characters for touches (d) and adapt keys when hand postures change during typing. We further improve predictions with language models.**

## ABSTRACT
Recent work has shown that a multitouch sensor attached to the back of a handheld device can allow rapid typing engaging all ten fingers. However, high error rates remain a problem, because the user can not see or feel key-targets on the back. We propose a machine learning approach that can significantly improve accuracy. The method considers hand anatomy and movement ranges of fingers. The key insight is a combination of keyboard and hand models in a hierarchical clustering method. This enables dynamic re-estimation of key-locations while typing to account for changes in hand postures and movement ranges of fingers. We also show that accuracy can be further improved with language models. Results from a user study show improvements of over 40% compared to the previously deployed "naïve" approach. We examine entropy as a touch precision metric with respect to typing experience. We also find that the QWERTY layout is not ideal. Finally, we conclude with ideas for further improvements.

### Author Keywords
Machine Learning; Classification; Clustering; Touch; Typing; Back-of-Device

### ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: Input devices and strategies (e.g. mouse, touchscreen)


*IUI'14,* February 24–27, 2014, Haifa, Israel.

http://dx.doi.org/10.1145/2557500.2557501

## INTRODUCTION
Low text entry rates are a recognised problem for mobile devices. Recent research has explored back-of-device interaction with multitouch sensors [3, 30, 34, 35]. Potential benefits of this concept for mobile text entry are: 1) faster typing rates engaging all ten fingers, 2) releasing display space for applications on the front. However, realising these benefits remains a challenge. Related work used physical keys on the back [18, 29]. Folding the layout in two rotated halves [29] retains the finger-to-key assignments, known from three-row keyboards like QWERTY or the Dvorak Standard Keyboard (DSK). Unfortunately, the demonstrated mean typing speeds were low (15 wpm), and the addition of physical keys breaks the familiar form factor of the device. The *Sandwich Keyboard* [28] deployed a multitouch sensor that followed this concept without extra buttons to keep the familiar form factor. A user study with training showed promising typing speeds (QWERTY: 26 wpm, DSK: 46 wpm). Users reached about 70% of their speeds for physical keyboards. However, error rates were higher than with other methods: about 12% after 7 hours of training. To make this approach a valuable alternative to existing text entry methods, further work is needed to decrease the proportion of errors.

This paper provides an extended modelling framework for typing with a capacitive multitouch sensor on the back (see Figure 1). Our approach can reduce errors by over 40%. We analyse sources of errors and propose ways to locate and remedy them. In contrast to previous work, our approach explicitly models the user's hands and adapts keyboards to changing touch behaviour during typing without collecting new labelled training data. Varying hand locations and angles relative to the device, as well as finger tremor and movement variance lead to different touch locations for the same key. These factors may change dynamically between typing sessions and while typing. Addressing them is important to make keyboards more robust with respect to mobility and individual differences in behaviour and anatomy.

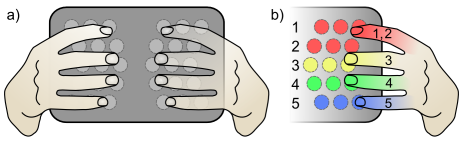


**Figure 2. A folded multitouch keyboard [28] on the back (a) with finger-to-key assignments (b). Each key is associated with one character, depending on the layout (e.g. QWERTY). Keys in this figure are not personalised; our approach adapts locations and sizes of keys to the user.**

## Approach and Related Work

The goal of this work is to improve touch classification accuracy for a folded back-of-device soft keyboard (Figure 2). Our approach utilises machine learning methods to specifically address three kinds of variability: First, we model touch location distributions around targets, the keys (Figure 1b). Second, we account for ongoing changes in hand postures and finger placement behaviour with keyboard updates, facilitated by a hand model (Figure 1c). Third, we train models on user-specific data to respect variance between users. The resulting system predicts characters for the touches (Figure 1d).

We assume a one-to-one mapping of keys to characters. Combined with the folded version of a known layout like QWERTY, this is argued to enable transfer of users' existing motor programs from physical keyboards [28]. This also defines the prediction task: each touch (at $x, y$) must be mapped to a character. Our approach models keyboard, hands and language.

**Keyboard model:** Our keyboard model personalises key-locations and sizes based on the user's touch distributions. Matching keys with the user's personal touch behaviour can be expected to improve classification. Related work on personalised key-targets for soft keyboards mostly uses a Gaussian Bayes classifier [1, 15, 25, 36]. It models keys with (bivariate) Gaussian distributions, as justified by evidence for front screen interaction [13, 32]. For typing on a tabletop, distance-based classification had been proposed based on a study with expert typists [11], before the same authors switched to decision trees [10]. For our approach, we evaluated these and other methods on the back using data from novice and expert typists. We considered different touch features and classifiers. Gaussian Bayes was confirmed as a preferable classifier with touch locations on the back as well. Only Support Vector Machines (SVMs) had slightly better accuracy, but at the price of high computational costs, because multiple SVMs need to be trained for more than two classes (keys). Without extensions, they also lack probabilistic output. Costly computations are undesirable for mobile devices and probabilistic predictions can be combined with language models. Most importantly, the Bayes model provides explicit key-locations, which can be updated with our hand model and clustering approach.

**Hand model:** Our hand model estimates finger locations from unlabelled touches and predicts which touch belongs to which finger. It supports keyboard adaptation when hand postures change during typing. This dynamic adaptation improves touch classification. In general, adapting classifiers to varying user behaviour requires new training data, ideally in each session. This may be unacceptable to the user. Related work on soft keyboards for mobile devices has proposed to collect training data from free typing, using the current model's predictions to label the touches [2]. However, this implies to trust the same model we want to update, which is inappropriate if we expect dynamically changing behaviour. Another approach trained multiple supervised models in advance, then selected the most specific one in each context to achieve adaptation [36]. Unfortunately, this method assumed discrete hand postures (e.g. one/two thumb), but typing postures on the back are continuous (see Figure 3).

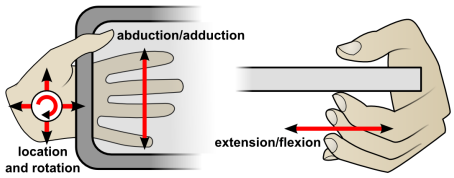


**Figure 3. Sources of variance in typing touch locations.**

We propose a *keyboard updating* approach, which does not rely on labelled touches. It has two parts. First, we utilise touch-to-finger assignments from the hand model and knowledge about the layout (e.g. QWERTY) to cluster unlabelled new touches. Second, we find the optimal pairing of existing keys and new cluster locations to update the keyboard.

We compare our approach to two simple methods and the *Sandwich Keyboard* algorithm [28]. It predicts the closest key for each touch and compares its prediction to the ground-truth key. If the prediction was correct, the key is moved towards the touch location. Incorrect predictions move it away.

**Language model:** Language context improves classification for ambiguous touch locations, because not all characters are equally likely to continue previously entered text. We build on existing work on language modelling for soft keyboards to predict characters from both touches and language context [13, 15, 25]. Our approach is two-fold. First, we use $n$-grams to improve key disambiguation during typing as in related work [13, 15]. Second, we infer words from touch sequences with Hidden Markov Models (HMMs) and use dictionaries [22, 25] to correct them. Auto-correction is a common feature of modern devices (e.g. the Android stock keyboard [14]).

We further examine touch density and entropy of probabilistic keyboard models. We discuss entropy as a metric for touch precision while typing. These methods allow us to gain insight into differences between users and layouts.

## KEYBOARD MODEL

To classify touches into keys, a formal representation of touch events is required. We evaluated these **features** (Figure 4a):

- *downX, downY, upX, upY*: Touch locations when meeting and leaving the surface.
- *travelX, travelY, travelAngle*: Distances and angle between touch down and up.
- *distanceToPrev, angleToPrev*: Distance and angle between two following touches.
- *touchDuration, timeDifference*: Time between down and up and time between two following touches.

The list is not exhaustive. Other sensors may provide new features. Our model uses *upX, upY*, based on feature evaluation.

A **touch classifier** can learn to statistically associate touch feature values with keys. It is trained on labelled touches, which means that the ground-truth key for each training touch is known. The trained classifier can then predict keys for future unlabelled touches. We use Gaussian Bayes, which models each key with a (bivariate) normal distribution, see Figure 1b). This model is defined by Bayes' Theorem:

$$p(k|t) = \frac{p(t|k)p(k)}{p(t)}$$

$p(t|k) \sim \mathcal{N}(\mu, \sum)$ is the *likelihood* of touch $t$ given key $k$. $p(k)$ models *prior* believe in $k$. $p(t)$ is a normalisation factor:

$$p(t) = \sum_{k \in K} p(t|k)p(k), \text{ for the set of all keys } K.$$

The *posterior* $p(k|t)$ gives the probability of keys after observing $t$. However, only the numerator is needed to find the most probable key $k'$, which is the classification decision:

$$k' = \underset{k \in K}{\text{argmax}} \left( \frac{p(t|k)p(k)}{p(t)} \right) = \underset{k \in K}{\text{argmax}} \left( p(t|k)p(k) \right)$$

To train the model, the likelihood $p(t|k) \sim \mathcal{N}(\mu, \sum)$ is derived from all touches with label $k$: $\mu$ is their average location, $\sum$ is their covariance. The prior $p(k)$ is given by the relative frequency of characters in the language or training text.

We propose to consider sparse training data. Keys for uncommon characters may have few training touches. This leads to poor estimates for their likelihood distributions $p(t|k)$. We solve this problem with a fallback to a default distribution. If there are less than *minPoints* training touches for a key $k$, the covariance matrix of $p(t|k)$ is not estimated from those touches. It is rather set to a default matrix with zero covariance and variance $d$. Thus, $d$ defines a default key-size.

## HAND MODEL AND KEYBOARD UPDATING

We present a simple hand model for back-of-device typing. It uses a set of lines to represent possible fingertip locations for each digit finger. Line orientations and locations are learned from touches. As a result, our model captures location and rotation of the hand relative to the device. We use this context knowledge to facilitate keyboard updates while typing with changing hand postures.

Each hand is represented by five straight lines. Each digit finger is modelled as one line, thumbs are not needed. However, the index finger gets two lines; it serves two rows of keys in the folded layout (Figure 2). The hand model $\theta$ is defined as:

$$\theta = [\theta^i], \theta^i = (\theta_1^i, \theta_2^i)^T, 1 \leq i \leq 5$$

$\theta^i$ defines the $i$-th line with intercept $\theta_1^i$ and slope $\theta_2^i$. Lines ordered vertically (Figure 4b): $\theta^1, \theta^2$ define the index finger, $\theta^3$ is the middle finger, $\theta^4$ the ring finger, $\theta^5$ the small finger.

### Learning Fingers with $k$-Lines

To learn the parameters of the presented hand model, we have to fit finger-lines to touch locations. Intuitively, our method takes a first informed guess to place initial lines. They are then refined iteratively. Each iterative step has two parts. First, each touch is assigned to its closest line. Second, slope and intercept of each line are updated to fit the assigned touches. The algorithm terminates, when no more changes occurred.

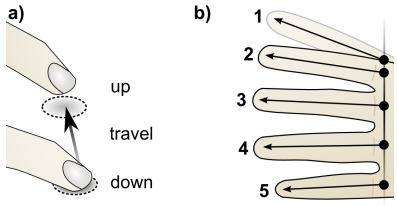


**Figure 4. A touch event (a) and the hand model (b) with lines indicating slopes and intercepts along the touch surface edge on the back. The index finger is modelled with two lines to account for the abducted stance used to reach the second row of keys (compare to Figure 2).**

The algorithm uses an iterative optimisation method similar to $k$-Means. It is referred to as $k$-Lines in related work [5]. A detailed description of our procedure is given below:

**1. Initialisation:** Initial slopes are set to 0 (horizontal fingers). Intercepts are initialised with $k$-Means using only $y$-values of the touches. We define $k$ lines $\theta^i = (c_i, 0)^T$. $c_i$ is the $i$-th cluster-mean of the $k$-Means clustering. Cluster-means are sorted to match the finger indices described for the hand model. Then, we start an optimisation loop with two steps.

**2. Fitting touches to lines:** The first step of each iteration assigns each touch $t = (t_x, t_y)^T$ to its closest line:

$$line(t) = \underset{1 \leq i \leq k}{\text{argmin}}\, distance(t, \theta^i) = \underset{1 \leq i \leq k}{\text{argmin}} \frac{|\theta_1^i + \theta_2^i t_x - t_y|}{\sqrt{\theta_2^{i\,2} + 1}}$$

**3. Fitting lines to touches:** The second iterative step fits $k$ lines to the touches, using linear regression with basis functions. We create a design matrix $X$ and target vector $y$ per line:

Let $T_i$ denote the set of all touches $t$ assigned to the $i$-th line. The $N \times M$ design matrix $X_i$ for the $i$-th line is defined as:

$$X_i = [x_{nm}], x_{nm} = \Phi_m(t_n), t_n \in T_i$$

$\Phi_m$ denotes the $m$-th of $M$ basis functions. Each of the $N = |T_i|$ rows of $X_i$ contains one touch, each column one feature. For straight finger-lines, we set $\Phi_1(t) = 1$ (bias) and $\Phi_2(t) = t_x$ (linear term). The algorithm is flexible: Other $\Phi$ and corresponding distance measures could model different assumptions, for example a quadratic component: $\Phi_3(t) = t_x^2$.

Next, $y_i$ is defined as the vector of the $y$-values of all touches $t = (t_x, t_y)^T \in T_i$. Finally, $X_i$ and $y_i$ are used with least-squares:

$$\theta^i = (X_i^T X_i + \lambda I)^{-1} X_i^T y_i$$

With $\Phi_1, \Phi_2$ as described above, $\theta^i \in \mathbb{R}^{2\times 1}$ represents a straight line with intercept $\theta_1^i$ and slope $\theta_2^i$. $\lambda$ penalises large $\theta$ and restricts the model to avoid steep slopes (indicate crossing fingers) and intercepts beyond the device borders.

**4. Termination:** Steps 2 and 3 are executed repeatedly, until no further changes to the line assignments occurred since the last iteration. For physical finger assignments, simply merge the two clusters of the index finger lines. This completes the desired output of the algorithm - finger assignments and the final hand model $\theta$. Figure 1c) shows an example result.

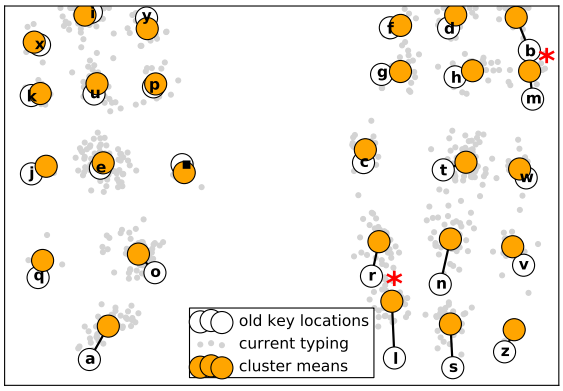


**Figure 5. Update example: An existing keyboard (white) is updated by clustering the incoming unlabelled touches (grey). Their cluster means (orange) define new key-locations. They are labelled by pairing them with the old keys, minimising the global sum of distances. Asterisks (*) point out examples for the importance of an optimal pairing per finger: Simply assigning the closest of all old keys would not have worked here.**

### Keyboard Updating

Touch behaviour can vary between or during typing sessions (see Figure 3). A personalised keyboard model has to adapt its keys to these changes or the mismatch of old key-locations and new touch locations will increase touch classification errors. Collecting new labelled data to retrain the model is not an option, because training phases interrupt the user. New touches from the user's ongoing free typing may inform the model instead, but they lack labels. Hence, it is unknown which touch should update which key.

We solve this problem with a combination of keyboard and hand models in a clustering method. This allows the system to use unlabelled touches to update the keyboard and avoids to bother the user with new training phases. Our approach has two parts. First, clustering finds unlabelled key-locations. Second, existing labelled keys are paired with the new locations. We then move each key to its new location.

Standard clustering methods (e.g. $k$-Means) can not fully utilise context knowledge about layout and hands. Hence, we propose a custom *hierarchical clustering algorithm*:

1. **Hands:** If hands do not overlap, they are trivially given by the touches' relative locations to the device center.
2. **Fingers:** Our hand model is used to cluster touches of each hand by fingers. Figure 1c) shows an example.
3. **Keys:** Touches of each finger are clustered into keys with $k$-Means, initialised with key-locations of the existing model.

Our hand model enables the important second step; the method can search for key-clusters *per finger*, not just per hand. The method needs at least one touch per key, because the number of clusters is fixed. This can be ensured by adding key-locations of the existing keyboard to the new touches.

After clustering, each existing key is associated with a new cluster mean. The solution to this set-matching problem is a pairing which minimises the sum of distances, see Figure 5. One may try all possible pairings or use the Hungarian Method [20]. Fingers can be treated separately.

Each key is then moved to its corresponding cluster mean. Updates can be scheduled regularly during typing. We chose the end of sentences in this work. The updated model is then used to classify future touches - until its next update.

### LANGUAGE MODEL

In case of ambiguous touch locations, language properties can help to infer user intention. Language models can thus complement models of touch behaviour. We use $n$-gram models, Hidden Markov Models (HMMs) and dictionaries. We do not model relationships between words here (e.g. word $n$-grams).

We use character $n$-gram models as priors like related work [12, 13] to predict $p(k_n|k_1k_2...k_{n-1})$, the probability of key $k_n$ given the last $n-1$ predictions $k_1, k_2, ..., k_{n-1}$. The Bayes model combines this with the touch likelihood $p(t|k_n)$:

$$p(k_n|t) = \frac{p(t|k_n)p(k_n|k_1k_2...k_{n-1})}{p(t)}$$

Training is the same as before. The $n$-gram model itself is trained on a large text corpus.

We also propose word evaluation, which resembles autocorrection: HMMs extend the Bayes model with state transitions to find the most likely state sequence (word), given the observation sequence (touches). We refer to related work [24] for a detailed description. Following their notation, we set up the HMM $\lambda = (A, B, \pi)$ with known information: Initial probabilities $\pi$ are character frequencies of the language (unigram model). The transition matrix $A$ is given by a bigram model. Emissions $B$ are the likelihood distributions $p(t|k)$ from the Bayes model. In addition, dictionaries can correct mistyped or misclassified words [22, 25]. They use large training text to suggest candidates, which are ranked by similarity to the input. We further rank equally similar candidates by touch probability, not by training text frequency. The input is then corrected with the best candidate.

### DATA COLLECTION

To evaluate the method, we collected three datasets. They are summarised in table 1. Dataset $D_1$ was provided by [28] from their *Sandwich Keyboard*. We refer to their work for details on apparatus and study design. In summary, two tablets were fixed back-to-back to enable touch interaction from both sides. Experienced ten-finger typists completed a training procedure using the folded versions of their layout over the course of 8 one-hour meetings. Each meeting contained multiple typing sessions. A session is defined as continuous typing without putting the tablet down. Users typed bigrams, word drills and sentences (from [31]) shown on the front screen in a sitting position. Errors were detected with an adaptive recogniser and marked with red lines under the corresponding letter in the prompt. Users reached typing speeds of 26 wpm (QWERTY) and 46 wpm (DSK). We used the 19 typing sessions without keyboard visualisation on the front, no backspacing and only sentences (about 20 per session).

We also obtained 10 typing sessions from an expert user with 70 wpm after 35 hours of training on the *Sandwich Keyboard*. This dataset $E$ was collected as a "stress test'" for keyboard updating. It has two special properties in comparison to the

other datasets and is thus only used to evaluate keyboard updating and hand models: First, the user *consciously* varied hand postures to provoke large cross-session changes. Second, sessions consisted of 40 sentences with about 40% *pangrams*. This helps to evaluate that the concepts work for all keys. However, we do not use this data to evaluate language models, since pangrams distort language properties.

Dataset $D_2$ was collected with a prototype and study design similar to $D_1$ [28]. It used a touch panel on the back, resulting in a slimmer and lighter device. In contrast to $D_1$, 5 *novice* users were recruited. They had not learned an official blind ten-finger typing method in the past. In the study, 2 used QWERTY, 3 DSK. We assigned 2 of our 5 users to QWERTY, because DSK was better in previous work [28]. A backspace key was available and users were encouraged to correct *cognitive* errors, like confusing fingers with respect to the layout. We excluded backspaced touches from evaluation. Participants completed 9 one-hour meetings similar to $D_1$. One of the users had only 7 meetings, due to time constraints on the user's side. We still use this data, since individual performances are never compared between users. Users reached 13.5 wpm on average - much slower than in $D_1$. We explain this result with the lack of prior ten-finger typing experience. We noticed that users still had to think about the relative locations of the keys during their last sessions, slowing them down. We also measured average speeds with a laptop keyboard (32.8 wpm) and the default Android keyboard on the front (21.5 wpm). As in $D_1$, we only used data from invisible keyboards (20 sessions, $\sim$ 25 sentences each). Overall, we collected about 40,000 touches from 12 users.

| Dataset | Sessions | Subjects | Typing experience |
|---|---|---|---|
| $D_1$ | 19 | 3 DSK 3 QWERTY | Experts |
| $D_2$ | 20 | 3 DSK, 2 QWERTY | Novices |
| $E$ | 10 | 1 DSK | Back-of-device expert |

**Table 1. Overview of the datasets used in this evaluation.**

| Feature | Classification accuracy (%) Dataset $D_1$ | | | Dataset $D_2$ | | |
|---|---|---|---|---|---|---|
| | *NB* | *Tree* | *kNN* | *NB* | *Tree* | *kNN* |
| downY | 54.9 | 52.2 | 51.4 | 40.2 | 37.1 | 34.5 |
| downX | 48.5 | 45.9 | 44.5 | 40.3 | 38.3 | 36.6 |
| upY | 55.2 | 52.1 | 51.3 | 40.3 | 37.4 | 33.9 |
| upX | 48.3 | 46.2 | 44.5 | 40.9 | 38.1 | 36.9 |
| angleToPrev | 23.0 | 32.0 | 32.7 | 21.4 | 26.3 | 26.2 |
| travelY | 14.0 | 14.2 | 14.3 | 11.6 | 12.1 | 12.1 |
| travelX | 13.6 | 15.1 | 14.4 | 11.7 | 12.2 | 11.9 |
| travelAngle | 12.7 | 16.1 | 15.7 | 10.3 | 12.4 | 12.4 |
| distanceToPrev | 13.9 | 17.4 | 17.1 | 13.2 | 12.8 | 12.4 |
| touchDuration | 13.9 | 12.8 | 12.2 | 10.8 | 10.8 | 10.1 |
| timeDifference | 13.2 | 12.1 | 11.4 | 11.8 | 10.0 | 10.1 |
| Majority Classifier | | 12.1 | | | 10.8 | |

**Table 2. Classification accuracy with 10-fold-cross-validation using only one feature, averaged over all sessions of each dataset. The majority classifier (baseline) predicts the most common class for each session.**

## RESULTS

All results were computed on the described datasets. We used the Weka Machine Learning environment [16] for feature evaluation and the comparison of classifiers. Significance is reported at $p \leq 0.05$. We use t-tests and pair conditions per session and also per outlier filtering tolerance or cross-validation fold, where applicable. Note that the different numbers of sessions lead to different degrees of freedoms for the datasets.

### Feature Selection

In *single feature evaluation*, classifiers were trained on one feature at a time. This indicates a feature's own explanatory power, see [10]. Results are summarised in Table 2. Location features (*downX, downY, upX, upY*) performed best. Only one other feature, *angleToPrev*, performed considerably better than the baseline across all classifiers and datasets.

*Correlation-based feature selection* (CFS, [17]) examines data to select feature subsets. It selected touch location features and *angleToPrev* for all sessions. For $D_1$, *timeDiff* appeared in subsets for 4 of 19 sessions, all other features were selected in only one session each. In $D_2$, *distanceToPrev* appeared in 2 of 20 sessions, *timeDiff* once, other features never.

Finally, we applied *wrapper feature selection* [19], which greedily adds the best feature to the existing subset, using classifiers; here Naïve Bayes, decision trees, *k*-Nearest-Neighbours (kNN, $k = 5$) and Support Vector Machines (SVM). All selected subsets had touch location features. For $D_1$ and Bayes, subsets for 10 of 19 sessions solely comprised of locations. With decision trees, *travelY* (7 sessions) and *distanceToPrev* (6) were the most common non-location features. For $D_2$ and Bayes, subsets for 14 of 20 sessions contained only locations. With decision trees, *travelY* and *angleToPrev* had 6 sessions each, followed by *distanceToPrev* (5). kNN and SVM almost exclusively selected locations.

Overall, touch locations and *angleToPrev* were favoured by single feature evaluation and CFS. The wrapper approach confirmed touch locations, but not *angleToPrev*. In conclusion, we chose only locations (*upX*, *upY*) for this work.

### Touch Classification

We tested classifiers with 10-fold cross-validation per session. Datasets were preprocessed to facilitate an evaluation of methods, not user-skill: We removed spaces, since this key was operated by the right thumb on the front and hence trivial to recognise. Touches with a *local outlier factor* (LOF, [6]) exceeding an outlier threshold *OT* were removed. We expect outliers to coincide with cognitive user-errors (e.g. confusing finger-to-key assignments for the layout). We report results in simple error rate. Values in the text are given for $OT = 2.5$, a conservative upper bound for points in a Gaussian cluster [6].

**Model comparison:** Figure 6a) shows results for $D_1$: Decision Tree (10.6% average error rate) and Random Forest (10.0%) were outperformed by kNN (9.3%) and Naïve Bayes (9.1%). SVMs ranked first (linear 8.6%, radial basis functions - RBF: 8.4%). $D_2$ confirmed this ranking, see Figure 6b). All results improved with stricter outlier threshold. For $OT = 1.1$, more than a third of the touches were outliers. This is clearly unlikely to reflect the true number of (cognitive) user-errors, but indicates expectations for more precise typing. Figure 6 shows that the classifiers' ranking was mostly independent of outlier removal. This could not have been observed with fixed outlier thresholds found in evaluations of touch behaviour in related work on soft keyboards [1, 11, 36].

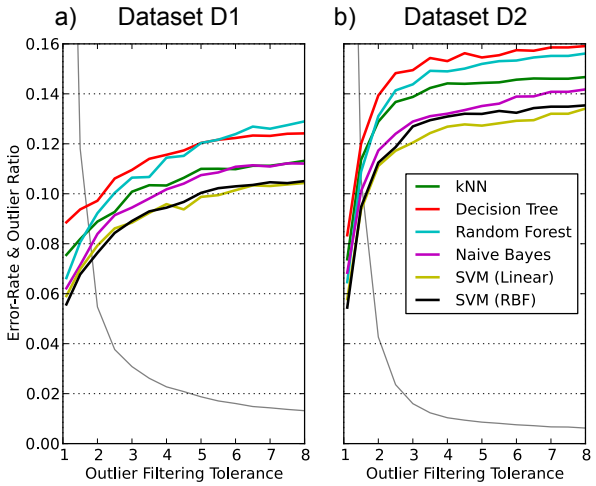


**Figure 6. Average error rates of different classifiers as a function of outlier filtering tolerance (LOF-score, [6]), computed with 10-fold cross-validation per session. Falling grey lines indicate the ratios of outliers.**

**Comparison to related work:** We evaluated key predictions from the *Sandwich Keyboard* algorithm (SWK) as *recorded* by the authors of related work [28]. We observed 10.7% ($D_1$) and 20.6% ($D_2$). These results were *optimistic*, since they had used ground-truth labels for adaptation during the study. We then slightly adjusted the SWK algorithm to be able to use a training set and perform predictions without ground-truth labels. This modification allowed for cross-validation and a direct comparison with Naïve Bayes. It significantly outperformed SWK ($D_1$: $t(284)$ = -6.16, $D_2$: $t(299)$ = -11.72). Their average error rates at $OT$ = 2.5 were 9.1% vs 9.8% ($D_1$) and 12.4% vs 14.4% ($D_2$). Note that our SWK performed better than the recorded results, since we estimated initial keys with training sets, not just with single touches by the user.

**Sparse data:** We modified the Bayes model to account for sparse data. If a key had less than *minPoints* training touches, its distribution used a default variance of 400. We considered two values (5, 15) for *minPoints*. All values are subject to further optimisation. $minPoints = 5$ significantly improved error rates ($D_1$: $t(284)$ = -8.04, $D_2$: $t(299)$ = -9.75). $minPoints = 15$ only achieved this for $D_1$ ($D_1$: $t(284)$ = -9.15, $D_2$: $t(299)$ = 2.81). There was no significant difference between the SVM and the improved Bayes model for $D_1$ ($t(284)$ = 0.08). The SVM was significantly better for $D2$ ($t(299)$ = -4.56). This indicates that Bayes suffered from sparse training data in comparison to the SVM. Our simple extension partly remedied this problem. A practical conclusion is that enough training data per key should be collected to make this unnecessary.

**Covariance:** We compared Bayes with and without covariance between $x$ and $y$. Covariance significantly improved error rates with $minPoints = 15$ ($D_1$: $t(284)$ = -4.53, $D_2$: $t(299)$ = -6.66). Their average error rates at $OT$ = 2.5 were 8.4% vs 8.6% on $D_1$ and 12.1% vs 12.5% on $D_2$. Covariance was worse than no covariance with $minPoints = 5$, but this was only significant for $D_1$ ($D_1$: $t(284)$ = 2.67, $D_2$: $t(299)$ = 0.93). The observation that covariance only improved classifiers with $minPoints = 15$ demonstrates that estimating the additional parameters for covariance requires more touches per key.

| Classifier | Error Rate MSD (%) Dataset $D_1$ | Dataset $D_2$ |
|---|---|---|
| Bigram model ($n = 2$) | 10.06 | 9.51 |
| Trigram model ($n = 3$) | 9.87 | 9.48 |
| 4-gram model | 10.08 | 10.07 |
| 5-gram model | 10.05 | 10.13 |
| Baseline: Naïve Bayes | 11.50 | 11.14 |

**Table 3. Performances of Naïve Bayes with $n$-gram language models.**

| Classifier | Error Rate MSD (%) Dataset $D_1$ | Dataset $D_2$ |
|---|---|---|
| HMM | 9.19 | 8.79 |
| HMM & Dictionary | 6.87 | 6.09 |
| Naïve Bayes & Dictionary | 7.70 | 6.86 |
| Naïve Bayes, trigram & Dictionary | 6.99 | 6.58 |
| Baseline: Naïve Bayes | 11.50 | 11.14 |

**Table 4. Performances for word by word touch classification.**

In summary, we identified Gaussian (Naïve) Bayes and Support Vector Machines as the most promising classifiers. Covariance between $x$ and $y$ can be considered if enough training touches per key are available (here $> 15$). Improvements for sparse training data can be achieved with a default key-size. For $D_1$, no significant difference was found between improved Bayes and SVM. We favour Bayes; it is computationally cheap in both training and prediction, provides probabilities and inherent handling of more than two classes. In contrast to SVMs, Bayes also offers explicit representations of key-locations, which can be updated with our clustering approach. In general, models performed better for the experienced typists ($D_1$) than the novices ($D_2$). This confirms that typing skill is transferred from a traditional keyboard to the *Sandwich Keyboard* [28].

### Language Models

Language enhanced *character prediction* was evaluated with 10-fold cross-validation. Folds consisted of whole phrases. Outliers ($OT$ = 2.5) were removed from training phrases, but not from test phrases, because removing single characters would have damaged the text for language model evaluation. Spaces remained in the data for the same reason. Results are given in $error\text{-}rate_{MSD}$, the Damerau-Levenshtein (minimal string) distance [9, 21] of predicted and expected text. $n$-gram models were built with NLTK [4]. All models were trained on the "big.txt" file from Peter Norvig [22]. It contains >1M words from public domain sources.

Table 3 shows that all models improved the baselines. Trigram models performed best with significant improvements to the baselines ($D_1$: $t(189)$ = -6.22, $D_2$: $t(199)$ = -8.27).

We implemented HMMs and dictionaries for *word prediction* in Python. Dictionaries followed Peter Norvig's concept [22]. Table 4 shows that all approaches outperformed the baseline. *HMM & Dictionary* performed best with significant improvements to the baselines ($D_1$: $t(189)$ = -11.14, $D_2$: $t(199)$ = -15.15). It also outperformed *Trigrams & Dictionary*, but only significantly for $D_2$ ($D_1$: $t(189)$ = -0.42, $D_2$: $t(199)$ = -2.38). Without dictionaries, HMMs outperformed trigrams (see Table 3) significantly ($D_1$: $t(189)$ = -3.41, $D_2$: $t(199)$ = -4.44).

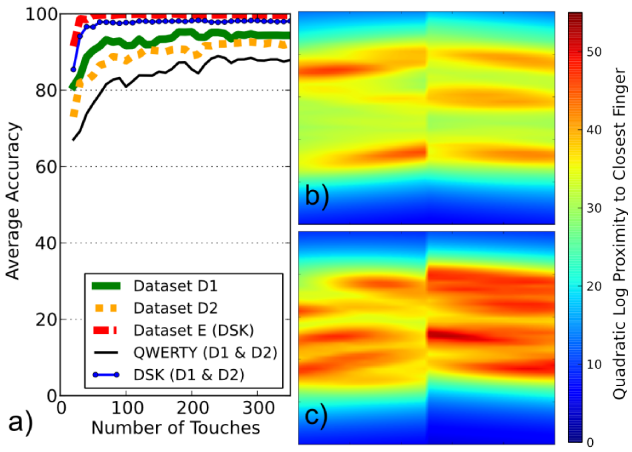


**Figure 7. Left: a) Accuracy of touch-to-finger assignments with $k$-Lines, as a function of the observed number of touches. Right: (log) average proximity of pixels to the nearest finger-line, split by layout. Red is close. These plots indicate that finger locations found with this hand model were spread further apart and varied around more distinct areas of the touch surface for b) DSK than c) QWERTY.**

In conclusion, $n$-gram models, HMMs and dictionaries significantly improved the Bayes classifier. Higher order $n$-gram models may perform better with more training text (see [8, 27]). In contrast to our model comparison (Figure 6), this evaluation included spaces and outliers to keep language properties unchanged. As a result, the ranking of the datasets reversed; we now observed lower error rates for $D_2$ than for $D_1$. This indicates that outliers had a larger impact on data without backspacing ($D_1$). Since users in $D2$ were encouraged to backspace cognitive errors, this also shows that outlier filtering with LOF [6] can indeed remove such user-errors.

### User- and Session-Specificity

We trained classifiers on touches of one session (i.e. typing without putting the tablet down) and applied them to touches of another one. Spaces and outliers ($OT$ = 2.5) were removed. The observed high error rates (Table 5) show that typing behaviour was *user-specific*, supporting personalised keyboards. It was also *session-specific*. In practice, the cross-session case is the default, since it is impractical for users to retrain their keyboard before each typing session. We conclude that updating keyboard models across sessions is vital in this context.

### Finding Fingers

We evaluated touch-to-finger assignments from our hand model. Expected text and layout defined the ground-truth. For example, "f" is the left index finger for QWERTY. Outliers were removed from the data ($OT$ = 2.5).

Figure 7a) shows that accuracy improved with more touches. For $D_1$, up to 95.3% average accuracy was observed after 200 touches. For $D_2$, 92.6% was reached at 290 touches. With 20 touches, the algorithm achieved 81.1% ($D_1$) and 73.0% ($D_2$). With 30 touches, results for $D_2$ already reached 82.0%. For the expert user ($E$), it achieved 91.5% with 20 touches and consistently more than 99% after 50 touches. We observed significantly higher accuracy for DSK than QWERTY users

| Classifier | Error Rate (%) same session $D_1$ | same session $D_2$ | cross-session $D_1$ | cross-session $D_2$ | cross-user $D_1$ | cross-user $D_2$ |
|---|---|---|---|---|---|---|
| Naïve Bayes | 9.14 | 12.40 | 33.96 | 25.49 | 65.40 | 51.80 |
| Decision Tree | 10.61 | 14.83 | 33.18 | 27.86 | 60.40 | 52.70 |
| kNN ($k = 5$) | 9.27 | 13.67 | 30.90 | 26.27 | 54.24 | 48.79 |
| SVM | 8.43 | 11.87 | 30.85 | 26.40 | 59.21 | 51.15 |

**Table 5. Cross-user (same layout) and cross-session applications.**

| Method | Error Rate MSD (%) $D_1$ | $D_2$ | $E$ |
|---|---|---|---|
| Baseline: No Updates (Naïve Bayes) | 28.92 | 20.94 | 39.37 |
| Baseline: *Sandwich Keyboard* algorithm | 24.54 | 19.05 | 28.90 |
| Distribution Updates | 28.03 | 19.75 | 38.23 |
| Fast Location Updates | 26.82 | 17.77 | 31.01 |
| Clustering Updates | 20.12 | 16.68 | 16.82 |

**Table 6. Performance of (updated) Naïve Bayes across sessions, averaged over all possible cross-session combinations for all users.**

($t(33) = 19.60$). This could be seen as a limitation of the algorithm, but also as one of the layout: Finger modelling revealed that DSK facilitates more precise movements and therefore clearer finger separation in this context. Figures 7b) and c) summarise fingers by layout. Derived finger locations showed showed more cluttered fingers for QWERTY, and more distinctive regions for DSK.

### Keyboard Model Updating

We evaluated our clustering approach in comparison to the *Sandwich Keyboard* algorithm and two naïve methods: The first one, *Distribution Updates*, adds each touch to the distribution of its predicted key. The second one, *Fast Location Updates*, adjusts the predicted key's location, based on the average of its existing mean and new touches. Hence, it weights each touch of the current session as much as all touches of the training session, leading to faster adaptation. We consider these methods naïve, because they use new data *touch by touch*. In contrast, clustering uses all new touches in each update. Figure 8a) shows intermediate results, Table 6 final results over the full sessions. Figure 8b) visualises key-movements.

*Clustering Updates* significantly improved error rates for both baselines: *No Updates* ($D_1$: $t(47) = -4.95$, $D_2$: $t(59) = -16.68$, $E$: $t(89) = -16.76$) and the *Sandwich Keyboard* algorithm ($D_1$: $t(47) = -2.55$, $D_2$: $t(59) = -4.54$, $E$: $t(89) = -8.51$). After 150 touches, a typical short message, clustering had already decreased error rates of Naïve Bayes by 4.93% on $D_1$, 2.19% on $D_2$ and 10.53% on $E$.

In contrast, overall improvements with *Distribution Updates* were small (Table 6). *Fast Location Updates* performed better, but was outperformed by *Clustering Updates* (on all datasets) and the *Sandwich Keyboard* algorithm (on $D_1$ and $E$).

The expert user tried different hand postures as a "stress test" for updating. Hence, cross-session variance is the highest for $E$. Clustering handled it well, in contrast to the other approaches (see Table 6). Experienced typists ($D_1$) showed higher session-specificity than novices ($D_2$). Novices placed fingers more slowly and consciously, because they were still learning movements and memorising the keys. Thus, their typing behaviour appeared more stable across sessions. This indica-

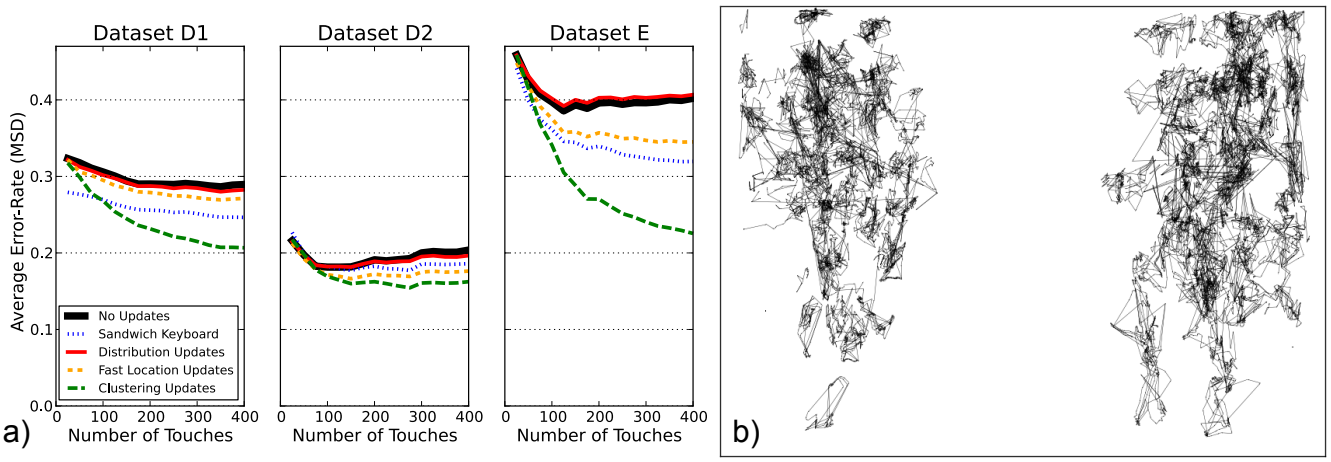


**Figure 8. Left (a): Error rates after $x$ touches, averaged over all cross-session combinations. Plots are limited to the first 400 touches to ensure that they always represent *all* sessions. Table 6 shows results over the full lengths. For $D_1$, the *Sandwich Keyboard* algorithm started noticeably better than the other methods after only $x = 25$ touches: It was used to detect and mark errors in this study. Hence, users initially adapted to this model. Clustering updates still outperformed it after about 100 touches. Right (b): All key-movement paths from clustering updates on dataset $D_2$, revealing that back-of-device typing is highly dynamic. The long and overlapping traces show that touch behaviour greatly varies during typing and between users.**

tes that dealing with cross-session variance may become even more important with increasing experience, at least in this unrestricted setting with no visual or tactile posture cues.

Regarding computational demands, an update with about 500 touches took less than 300 ms with our unoptimised Python script on a modern laptop. We expect faster processing with optimised and precompiled code. Clustering with our concept can also be performed in its own thread. Keys can then be updated almost instantaneously once the results are available.

### Discussion

In our user study, Gaussian Bayes was identified as a preferred *keyboard model* on the back. Touch locations were the only useful features here, but the model is flexible and could be used with other features, too. It currently handles sparse training data with a simple fallback threshold (*minPoints*). A more complete Bayesian approach could employ conjugate priors instead [15]. Future model comparisons could also rigorously optimise hyperparameters. New sensors or raw touch data present further interesting opportunities [26].

We observed accurate touch-to-finger assignments with our *hand model*, especially for DSK users ($> 95\%$). The current model neglects more complex hand properties like finger joints, but can be learned solely from touch data. We used it to adapt key-locations to changing hand postures during typing. Updating the full key-distributions (i.e. shape, size) is left to further investigation.

*Language models* greatly improved classification accuracy, despite their limitations. We did not consider relationships between words and used only character $n$-grams. We expect further improvements with extended language modelling techniques and more training text.

All methods were evaluated offline on collected user data. Our dataset contained only 12 users, but we observed an interesting range of typing experience and up to 9 hours of training per participant. Next, we plan to test our approach online to explore how users and adaptive keyboards interact.

## ANALYSING PERSONALISED KEYBOARD MODELS

The described Bayes model was used to predict keys for touches. We present two simple methods to further analyse the information captured in such a probabilistic model and facilitate a keyboard-wide analysis of typing touch behaviour.

### Keyboard Touch Likelihood

Intuitively, touches are more likely to belong to the typing process if they appear in regions where many typing touches have been observed so far. We refer to this touch density as *keyboard touch likelihood*, because it is high if the touch is close to *any* key. Hence, this models a keyboard's *surface*. Figure 9 shows examples.

For a touch location $t \in \mathbb{R}^{2\times 1}$, and distributions $p(t|k)$ for the set of keys $K$, the *keyboard touch likelihood* $L_t$ is defined as:

$$L_t = \frac{1}{|K|} \sum_{k \in K} p(t|k)$$

Formally, this describes a Gaussian Mixture Model. Keyboards could filter touches with low $L$, since they are unlikely to be aimed at keys. Unintended touches on the back may occur from carrying the device between typing sessions.

### Keyboard Entropy

Observing entropy of probabilistic keyboard models can reveal areas of ambiguous touch behaviour. Entropy is high for ambiguous touches and low for clear key-presses. Hence, high entropy is found where keys "overlap" and indicates *edges* of soft keyboards. Figure 10 shows examples.

Formally, for touch $t \in \mathbb{R}^{2\times 1}$ and model $p(k|t)$, entropy $S_t$ is:

$$S_t = -\sum_{k \in K} p(k|t) \ln p(k|t), \text{ for the set of all keys } K.$$

We can also compute an average, for example over all pixels, which reflects overall ambiguity: Precise and consistent typing touch behaviour produces smaller values, sloppy typing

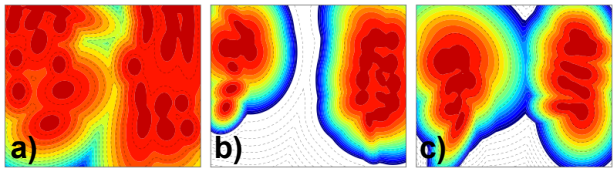


**Figure 9. Keyboard touch log-likelihood from different sessions of: a) a novice user, whose hands drifted upwards over the course of the session; b) a more experienced typist; c) the expert user. Red indicates high touch density, white marks regions below the chosen colour scale.**

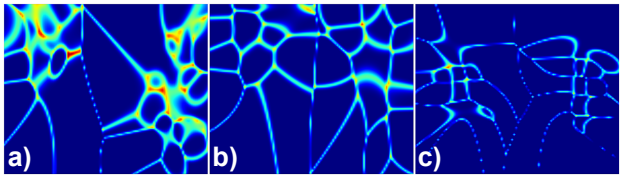


**Figure 10. Posterior entropy of Gaussian Bayes keyboard models of: a) a novice user's first try (average entropy 0.18), b) the same user after more training (0.11), and c) the expert user (0.05). Red indicates high entropy and marks regions of "overlapping" keys - the keyboard's edges.**

larger ones. Hence, this yields a metric for typing touch precision. In comparison to error rates, cognitively confusing two keys does not matter here, as long as the key-press itself is accurate. We propose to examine entropy to complement existing measures, like error rate and speed.

### Observations on Touch Distributions

Modelling typing touch density was presented to examine soft keyboard surfaces. We studied individual sessions (see Figure 9), and observed great variations between users, which can not only be attributed to different hand sizes. Experienced typists showed a tendency to place keys closer together than novices. We propose a simple filter-application: Touches in regions of low density could be ignored, since they are unlikely to be intended typing behaviour. Back-of-device interaction seems prone to generate non-typing touches, for example when carrying the device or putting it down.

We further proposed posterior entropy to examine soft keyboard edges and to define a metric for typing precision. We evaluated average entropies for all sessions: The expert *Sandwich Keyboard* user ($E$) had a mean of 0.06. The experienced ten-finger typists from $D_1$ scored second with mean 0.09, followed by the novice typists from $D_2$ with mean 0.16. One-way ANOVA showed a significant effect of typing experience ($F_{2,46} = 24.46$, $p < 0.001$). These results show that posterior entropy can reflect touch precision, related to typing experience. Figure 10 visualises examples.

## CONCLUSIONS

We presented a multi-model approach with keyboard, hand and language models to improve error rates for back-of-device typing. Our evaluation revealed insights into the contributions of the individual modelling assumptions. Considering touch variance per key improved the previous distance-based approach. $n$-grams provided useful character context, but dictionaries achieved even larger improvements. Word prediction was slightly better if touch sequences were first processed by a HMM. Hence, language strongly supports inference for this back-of-device soft keyboard, and a close combination of touch and language modelling is desirable.

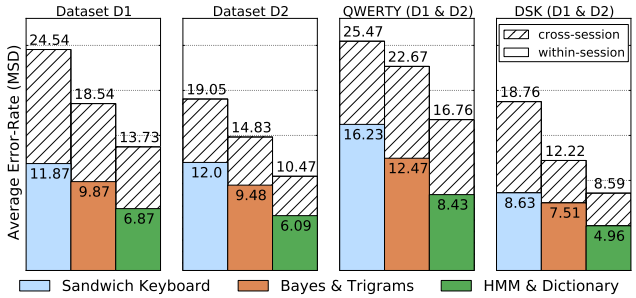


**Figure 11. Best practice approaches across sessions and within (cross-validation). Our methods used clustering updates for the cross-session cases. The baseline is given by the *Sandwich Keyboard* algorithm. Splitting the data by layout reveals a superiority of DSK users.**

Finally, our hierarchical clustering method updated keyboards while typing. The clustering approach reconsidered all previous touches in each update. It handled unlabelled free typing data without trusting the possibly outdated current keyboard model when deriving the new key-locations. This significantly outperformed previous adaptation methods, which only considered new information touch by touch, and had to trust the current model to label these new touches first. Hence, we conclude that clustering methods can significantly improve accuracy for dynamic typing with touch, compared to only addressing it as a static classification problem.

### Best Practice Approaches

Putting the pieces together, we arrive at the following best practices: Bayes with trigram models *per touch*, HMMs and dictionary for *word predictions*, clustering for keyboard updating *across sessions*. Figure 11 shows the results.

Including word correction, we reduced errors by $> 40\%$ compared to previous work [28]. Using the DSK layout, this constitutes the best and recommended case (4.96% within sessions, 8.59% across). In comparison, previous work [28] reported 9.8% for DSK users (within their final sessions), using ground-truth labels of the touches to update the keyboard.

Outliers were not removed from the test sets here to keep the text unchanged for language model evaluation. Thus, user-errors likely remained in the data and these results could be considered pessimistic from a modelling point of view.

### Future Work

We expect that the lessons learned here can be useful on a broader scale. In particular, we found that normal distributions can also model touches aiming at occluded key-targets on the back, complementing previous work for the front [32]. Our hand model and updating method could prove interesting for other soft keyboards, for example on tabletops. Touch density and entropy discussed here can reveal areas of activity and potential mistakes for interfaces in which targets (e.g. buttons) can be described with touch distributions.

We conclude with ideas for further improvements: Touch classification and finger modelling results suggested a superiority of DSK. Better layouts could be derived with optimisation methods [23]. Touch offset models [7, 33] might help to account for targeting errors. Besides machine learning, refined user training and tactile (back-of-device) markers could

improve speed and precision. Visualisations on the front screen can help the user, too, but would cover screen space. A static keyboard should be compared to the personalised version to investigate more restricted typing behaviour as well.

## ACKNOWLEDGEMENTS

This work was supported by the Academy of Finland, the EIT ICT Labs, and the Cluster of Excellence for Multimodal Computing and Interaction at Saarland University.

## REFERENCES

1. Azenkot, S., and Zhai, S. Touch behavior with different postures on soft smartphone keyboards. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services* (2012), 251–260.
2. Baldwin, T., and Chai, J. Towards online adaptation and personalization of key-target resizing for mobile devices. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces* (2012), 11–20.
3. Baudisch, P., and Chu, G. Back-of-device interaction allows creating very small touch devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2009), 1923–1932.
4. Bird, S., Klein, E., and Loper, E. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.
5. Bobrowski, L. K-Lines Clustering with Convex and Piecewise Linear (CPL) Functions. *Mathematical Modelling 7*, 1 (2012), 108–111.
6. Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. LOF: identifying density-based local outliers. *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* (2000), 93–104.
7. Buschek, D., Rogers, S., and Murray-Smith, R. User-Specific Touch Models in a Cross-Device Context. In *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services* (2013), 382–391.
8. Chen, S. F., and Goodman, J. An empirical study of smoothing techniques for language modeling. *Proceedings of the 34th annual meeting on Association for Computational Linguistics* (1996), 310–318.
9. Damerau, F. J. A technique for computer detection and correction of spelling errors. *Communications of the ACM 7*, 3 (1964), 171–176.
10. Findlater, L., and Wobbrock, J. O. Personalized input: improving ten-finger touchscreen typing through automatic adaptation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2012), 815–824.
11. Findlater, L., Wobbrock, J. O., and Wigdor, D. Typing on flat glass. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2011), 2453–2462.
12. Goel, M., Jansen, A., and Mandel, T. ContextType: Using Hand Posture Information to Improve Mobile Touch Screen Text Entry. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2013), 2795–2798.
13. Goodman, J. T., Venolia, G., Steury, K., and Parker, C. Language modeling for soft keyboards. In *Proceedings of the 7th international conference on Intelligent user interfaces*, ACM Press (New York, New York, USA, Jan. 2002), 194–195.
14. Google. Google keyboard. https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin (date accessed 08.10.2013).
15. Gunawardana, A., Paek, T., and Meek, C. Usability guided key-target resizing for soft keyboards. In *Proceedings of the 15th international conference on Intelligent user interfaces* (Feb. 2010), 111–118.
16. Hall, M., National, H., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The WEKA Data Mining Software: An Update. *SIGKDD Explorations 11*, 1 (2009), 10–18.
17. Hall, M. A. *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato, 1999.
18. Kim, H., Row, Y., and Lee, G. Back Keyboard: A Physical Keyboard on Backside of Mobile Phone using QWERTY. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems* (2012), 1583–1588.
19. Kohavi, Ron and John, G. H. Wrappers for feature subset selection. *Artificial Intelligence 97*, 1 (1997), 273–324.
20. Kuhn, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2*, 1-2 (1955), 83–97.
21. Levenshtein, V. I. Binary codes capable of correcting deletions, insertions and reversals. *Soviet physics doklady 10*, 8 (1966), 845–848.
22. Norvig, P. How to write a spelling corrector. http://norvig.com/spell-correct.html (date accessed 16.09.2013).
23. Oulasvirta, A., Reichel, A., Li, W., Zhang, Y., Bachynskyi, M., Vertanen, K., and Kristensson, P. O. Improving two-thumb text entry on touchscreen devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2013), 2765–2774.
24. Rabiner, L. R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE 77*, 2 (1989), 257–286.
25. Rashid, D. R., and Smith, N. A. Relative keyboard input system. In *Proceedings of the 13th international conference on Intelligent user interfaces* (Jan. 2008), 397–400.
26. Rogers, S., Williamson, J., Stewart, C., and Murray-Smith, R. AnglePose: Robust, Precise Capacitive Touch Tracking via 3D Orientation Estimation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2011), 2575–2584.
27. Rosenfeld, R. Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE 88*, 8 (2000), 1270–1278.
28. Schoenleben, O., and Oulasvirta, A. SandwichKeyboard: Fast Ten-Finger Typing on a Mobile Device with Adaptive Touch Sensing on the Back Side. In *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services* (2013), 175–178.
29. Scott, J., Izadi, S., Rezai, L. S., Ruszkowski, D., Bi, X., and Balakrishnan, R. RearType: Text Entry Using Keys on the Back of a Device. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services* (2010), 171–180.
30. Shen, E.-l., Tsai, S.-s., Chu, H.-h., Hsu, J. Y.-j., and Chen, C.-w. Double-side multi-touch input for mobile devices. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems* (2009), 4339–4344.
31. Vertanen, K., and Kristensson, P. O. A versatile dataset for text entry evaluations based on genuine mobile emails. *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services* (2011), 295–298.
32. Wang, F., and Ren, X. Empirical evaluation for finger input properties in multi-touch interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2009), 1063–1072.
33. Weir, D., Rogers, S., Murray-Smith, R., and Löchtefeld, M. A user-specific machine learning approach for improving touch accuracy on mobile devices. In *Proceedings of the 25th annual ACM symposium on User interface software and technology* (2012), 465–476.
34. Wobbrock, J. O., Myers, B. A., and Aung, H. H. The performance of hand postures in front- and back-of-device interaction for mobile computing. *International Journal of Human-Computer Studies 66*, 12 (2008), 857–875.
35. Wolf, K., Müller-Tomfelde, C., Cheng, K., and Wechsung, I. Does proprioception guide back-of-device pointing as well as vision? In *CHI '12 Extended Abstracts on Human Factors in Computing Systems* (2012), 1739–1744.
36. Yin, Y., Ouyang, T. Y., Partridge, K., and Zhai, S. Making touchscreen keyboards adaptive to keys, hand postures, and individuals: a hierarchical spatial backoff model approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2013), 2775–2784.