University of Glasgow | School of Computing Science

Improving Touchscreen Typing Using Back-of-Device Grip Interactions

Dimitar Petrov (1005056)

April 24, 2015

ABSTRACT

Typing on touchscreen keyboards is inherently inaccurate as users tend to touch locations offset from their intended target. Offsets are user-specific and can further differ for a given user between postures (left-hand, right-hand, index-finger or two-thumb typing). On the other hand, back-of-device interaction has been used to predict screen touches on randomised abstract targets. We propose a new approach where unique offset models are learned for each posture and back-of-device is used to predict posture. Offset models are learned using linear regression while classification is achieved through SVMs and GPs. The models lead to significant improvements in typing accuracy. We show that modelling thumbs within two-thumb typing as separate postures can further improve accuracy. Posture misclassifications, however, can introduce errors in the model. This can be alleviated by weighted averaging of offsets based on probabilities from a probabilistic classifier - GP.

1. INTRODUCTION

Typing on touchscreen keyboards compared to physical ones is inherently inaccurate. There are a number of reasons for this. Firstly, keys on such keyboards occupy a screen area much smaller than the area covered by a finger. Thus, when a user tries to press a certain key, they often touch a location of the screen offset from their intended target. This leads to ambiguous results in determining the intended touch location. Thumbs are particularly problematic as they result in even larger touched areas. The problem is known as the Fat Finger Problem [13].

Another issue is concerned with the fact that mobile devices are used in a variety of contexts. Users type in different ways - using one hand or the other; or using both. Different contexts imply different behaviour so a touchscreen keyboard needs to be able to adapt to such changes in the environment. Finally, touchscreen keyboards are often used while moving. This impairs the user's ability to accurately aim for the correct key and can result in a further decrease in touch accuracy.

These features introduce difficulties in creating efficient touchscreen keyboards. We propose a new multi-modal approach to the problem. We learn offset models for each user for a number of postures and we use back-of-device interaction to determine posture. In this sense, a posture is the way the device is held by the user (e.g. left hand, two hand) and an offset model is a function mapping an actual touch location (where the user touched) to an intended touch location (where the user was aiming for). Two-thumb typing is considered a special case of the approach where we also try to infer the thumb used at a given touch and attach a unique offset model to each thumb. Learning offset models is achieved through a simple linear regression approach while posture is classified using SVMs and GPs.

The next section provides some related work while section 3 discusses the models employed for the remainder of the paper. Section 4 presents the approach proposed here in its two dimensions: posture inference (sec. 4.1) and offset modelling (sec. 4.2). The user experiments undertaken are discussed in section 5 while section 6 presents the evaluations performed and discusses the results from these. Finally, we present some conclusions and propositions for future work.

2. RELATED WORK

The work proposed here combines three distinct research areas. Firstly, posture detection provides a way for a mobile device to infer the user context, specifically how the user holds the device and operates with it. Secondly, a number of machine learning models try to solve issues with touch keyboards by learning offset functions. Finally, back-of-device interaction investigates how users interact with the back of a mobile device (in contrast to the front) and how this could be used to improve the overall user experience.

2.1 Posture Detection

Posture detection deals with inferring the posture of the hand when using the device. Recently Goel et al. proposed GripSense [6] which uses the mobile device touchscreen and gyroscope to infer posture. The postures discussed are *lying-on-a-flat-surface* - interacting with device will it is on a flat surface (table), *index-finger* - holding the device in one hand and using it with the index finger of the other, *left-thumb* and *right-thumb* use - holding the device in one hand and operating it with the thumb of the same hand.

The features used to infer posture are rotation of device, touch size and shape of swipe arcs (the arc "drawn" by a thumb while scrolling). Each of the three features produces a classification of the inferred posture and a majority voting scheme is used to make the final decision. Detecting when the device was lying on a flat surface proved to be easy to achieve and was was virtually always (99.42%) correctly classified. Classification of the other 3 postures was still reasonably accurate (87.4%).

2.2 Typing Models

Several models have been proposed to improve typing accuracy on touchscreen keyboards. Weir et al. [17] proposed using user-specific offset functions which are modelled using Gaussian Process (GP) [10] regression. The authors show

that the approach can lead to significant improvements in touch accuracy (23.47% for a $2mm$ button). They also demonstrate that user-specific models outperform models trained on a pool of users.

While this paper covers general touchscreen interaction, a second paper [16] presents an application of the model to touchscreen typing specifically. The novelty in the approach is the combination of the offset model with a language model. The offset model produces a probability distribution over keys while the language model produces a distribution over letters. Both distributions are fed into a decoder which produces the final key probabilities. Compared to a baseline method, the observed error rate was reduced by over 4.9% for typing while sitting, 5% while standing and 7.6% while walking.

2.2.1 Two-thumb typing

A limitation of most typing models is that they only consider one-thumb text entry. Two-thumb text entry was discussed in [7] and later in [3], although this is based on physical miniature QWERTY keyboards. Both papers present models for calculating a predicted text entry rate of a specific keyboard based on parameters such as size of keys and time to press a key. The assignment of keys to thumbs is particularly relevant here. In [7] it is entirely static, meaning the authors draw a virtual line through the middle of the keyboard and assume keys on each side are pressed using the respective thumb.

In [3] a dynamic assignment is introduced on a somewhat limited scale. The authors state that 4 keys in the middle of the keyboard (v, b, g and y) can be pressed by either thumb while the rest are statically assigned. The thumb used to press each of the four keys is not inferred dynamically. Instead, it is assumed the user will make a "greedy" decision to press these keys with the thumb that will take the least amount of time to move to the respective key location.

2.2.2 ContextType

Most recently Goel et al. [5] developed a typing model based on posture detection. The model called ContextType builds on the previously developed GripSense. The authors show that each posture introduces a new offset pattern for key touches, therefore a separate offset model is created for each posture. The postures observed in GripSense are the same here with one exception. The *lying-on-a-flat-surface* posture is replaced by a *two-thumb* typing one.

Once ContextType decides on a posture, the relevant offset model is loaded and used to calculate a probability for each key on the keyboard. The process also employs a 5-gram language model which provides a second probability for each key based on the previous 4 key entries. The two probabilities are then multiplied and the key with the largest final probability is chosen.

ContextType results in a 20.6% decrease in error rate compared to the control condition. The model is also said to detect postures on average within 5 user interactions with an accuracy of 89.7%.

2.3 Back-of-Device Interaction

Recently there has been a significant amount of research in back-of-device interaction with mobile devices. Noor et al. [8] propose a model for predicting screen touches using back-of-device data. The aim is to predict which area of the screen will be touched before the actual interaction taking place. The approach is based on a GP regression model which makes a touch prediction based on the back-of-device sensor values. It is shown there exists a strong correlation between back-of-device interaction and front-of-device touch target. The model can predict touch position $200ms$ before contact with an accuracy of $18mm$. What is more, contact time can be reasonably predicted about $0.5s$ before touch. Both touch position accuracy and contact time predictions are increased further with the decrease of time to touch.

In [12] Schoenleben and Oulasvirta proposed a "Sandwich keyboard". The new keyboard allows for ten finger touch typing using back-of-device interaction. It builds on previous similar keyboards utilising buttons on the back of the device by replacing these with a touch sensor. This allows for easy modifications of the keyboard including the use of different layouts, e.g. QWERTY and Dvorak Standard Keyboard (DSK). The keyboard also constantly adapts to the unique typing patterns of each user. The authors state that users can learn this technique and reach reasonable typing speeds within 8 hours of training. Using QWERTY test subjects reached a typing speed of 26.1 words per minute (wpm), while using DSK this increased to 46.2 wpm.

Buschek and the authors of the previous paper proposed a new model [2] building on this approach. The main idea is the same but more complex machine learning models are employed compared to the "naive" approach used previously. Results presented reduce the error rate of the previous approach by 40%.

3. BACKGROUND

The following section introduces the machine learning models used within the rest of the paper. As previously mentioned we learn offset functions using linear regression. Classifying postures is achieved with Support Vector Machines and Gaussian Processes.

3.1 Linear Regression

Linear regression is a simple regression model that allows learning a linear relationship between attributes and responses [11]. This relationship can be expressed as

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\intercal \mathbf{x},$$

where $\mathbf{x}$ is a vector we need to make predictions for and $\mathbf{w}$ is a vector describing the linear relationship in the seen data.

The task is to find the model $\mathbf{w}$ that explains this relationship best. There are a number of approaches to achieve this. The most popular and the one used here is least squares regression. This defines the best model as the one minimising a loss function. The loss function is the sum of the squared distances between the true value of a point and its predicted value. Formally, the loss function looks like:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} (t_n - f(\mathbf{x}_n; \mathbf{w}))^2,$$

where $f(\mathbf{x}_n; \mathbf{w})$ is the predicted value for $\mathbf{x}_n$ according to the model $\mathbf{w}$ and t_n is $\mathbf{x}_n$'s true value. Once $\mathbf{w}$ is calculated making a prediction for $\mathbf{x}'$ is as simple as calculating $f(\mathbf{x}'; \mathbf{w})$.

3.2 Support Vector Machines

Support Vector Machines (SVM) are nonprobabilistic binary classifiers [11]. SVMs have been successfully applied to a wide range of Machine Learning tasks and for many of these their performance is hard to beat. SVMs learn a decision function which maximises the margin. The margin is the distance between the decision boundary and the closest data points on each side. Learning the decision function is achieved by solving the following constrained optimisation problem:

$$\underset{\mathbf{w}}{\text{argmin}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} \alpha_m \alpha_n t_m t_n k(\mathbf{x}_n, \mathbf{x}_m),$$

$$\text{subject to } \sum_{n=1}^{N} \alpha_n t_n = 0 \text{ and } 0 \leq \alpha_n \leq C \text{ for all } n.$$

The set of points closest to the decision boundary are called support vectors. The decision boundary is defined with respect to its distance to the support vectors, therefore support vectors are the only points required for classification. and remaining points can be discarded. This is a main advantage of the SVM over other algorithms as only a small subset of the points is required to make predictions.

Since data is not always linearly separable, it is crucial for the SVM to be able to learn nonlinear decision boundaries. What we can do is apply transformations to the data points to lift them to a new feature space where they are linearly separable. Kernel functions (kernels) allow performing calculations in the new feature space without explicitly transforming data points. In this sense, allows us to use a linear classifier to solve nonlinear problems. Popular kernels include the Gaussian (Radial Basis Function) kernel, linear kernel and polynomial kernel.

A main advantage of SVMs over other algorithms is the fact that you can easily substitute one kernel for another and thus learn a different classifier. In the optimisation problem the kernel is represented by the function $k(\mathbf{x}_m, \mathbf{x}_n)$. Substituting this function by any available kernel is straightforward and leads to a new SVM.

The C parameter in the decision function controls the softness of the margin. A soft margin allows some of the training points to be misclassified with the aim to further increase the margin and potentially improve classification performance on unseen data. Kernels could also introduce parameters of their own. It is important to tune all these parameters. This can be achieved through cross validation.

3.3 Gaussian Processes

A Gaussian Process (GP) [10] is a collection of random variables, any finite number of which have a joint Gaussian distribution. A GP is specified by a mean function $m(\mathbf{x})$ and covariance function (kernel) $k(\mathbf{x}, \mathbf{x}')$. The Gaussian process is then defined as:

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')).$$

Without loss of generality one can set $m(\mathbf{x}) = \mathbf{0}$ and completely define the GP through its covariance function.

GPs can be used for both regression and classification tasks. Here we focus on GP classification. In the binary case we place a GP prior over the latent function $f(\mathbf{x})$ and then using a squashing function (e.g. logistic function) to squash the prior to values between 0 and 1 - a probability. The new prior becomes $\pi(\mathbf{x}) \triangleq p(y = +1|\mathbf{x}) = \sigma(f(\mathbf{x}))$. Inference is then divided into two steps. In the first step we compute the distribution of the latent variable corresponding to a test case:

$$p(f_*|X, \mathbf{y}, \mathbf{x}_*) = \int p(f_*|X, \mathbf{x}, \mathbf{f}) p(\mathbf{f}|X, \mathbf{y}) d\mathbf{f},$$

where $p(\mathbf{f}|X, \mathbf{y}) = p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|X)/p(\mathbf{y}|X)$ is the posterior over the latent variables. The second step requires using the distribution over f_* to produce a probabilistic prediction:

$$\pi(\mathbf{x}) \triangleq p(y = +1|\mathbf{x}) = \int \sigma(f_*) p(f_*|X, \mathbf{y}, \mathbf{x}_*) df_*.$$

The non-Gaussian likelihood in the first equation makes the integral analytically intractable. The second equation can similarly be intractable. This raises the need for analytic approximations of integrals. Techniques used to achieve this include Laplace Approximation and Expectation Propagation.

An advantage of GP classification over SVMs is the fact the former is a probabilistic classifier, i.e. classification results in probabilities over classes instead of hard decisions. Probabilities provide the level of certainty the classifier has on a given decision. This is very important for tasks were the misclassification cost is quite high, e.g. classifying patients as healthy when they have TB.

4. APPROACH

The approach presented here is a combination of the discussed previous work into a single typing model. From [16] we know that user-specific typing models can improve typing accuracy on mobile devices. This implies a separate offset model for each posture [5]. Assuming we have obtained offset models for each posture we need to know which posture is being used so the relevant model can be loaded. GripSense [6] provides a way to achieve this with a reasonable accuracy. However, we believe that use of back-of-device data instead, will result in improved performance.

After we have learned the regression and classifications models, we can use these to improve typing accuracy. On each touch we use back-of-device data to determine posture then load the relevant offset model and use that to calculate an intended touch location.

4.1 Inferring Posture

In contrast to previous work, the approach discussed here uses back-of-device data for posture classification. As in ContextType we discuss 4 postures - *left-thumb*, *right-thumb*, *index-finger* and *two-thumb* typing (all in portrait mode). The aim is to build a classifier able to distinguish between the four postures. Some calibration data is required from each user to train the classifier. After training it can be used to dynamically determine the current posture.

Posture classification is achieved by evaluating the capacitance values of touch sensors placed on the back of a mobile device. There are 24 such sensors and values from each are sampled on every touch. Data is represented as 24 dimensional vector (one for each sensor) of integer values in

the range $[0, 65535]$, i.e. $\mathbf{x} = [s_1\ s_2\ s_3\ ...\ s_{24}]^\intercal$. We start with N vectors $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_n$ and 4 targets $t = 0, t = 1, t = 2, t = 3$. We build an $(N, 24)$ matrix of the training data and scale it to 0 mean and unit variance. We then train the classifier using the scaled vectors and their labels $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), (\mathbf{x}_3, t_3), ..., (\mathbf{x}_n, t_n)$.

The classifier used is a multiclass one-against-one Support Vector Machine. In this sense, a one-against-one classifier is a set of binary SVMs each of which is trained on a pair of the class labels. Classification is then performed by using all SVMs. A majority voting scheme between the resulting classes is used to determine the final class label. Two SVM kernels are investigated - Radial Basis Function (RBF) kernel and linear kernel with hyper-parameters $\gamma = \{0.001, 0.01,\ 0.1, 1, 10\}$ (RBF only) and $C = \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$. Grid search with 3-fold cross validation is used to determine the optimal hyper-parameters for each SVM. The Python library for Machine Learning scikit-learn [9] was used for learning SVMs and hyper-parameter optimisation.

Making a prediction for a test point $\mathbf{x}' = [s_1\ s_2\ s_3\ ...\ s_{24}]^\intercal$ starts by scaling $\mathbf{x}$ using the same mean and variance used for scaling the training data. The new vector is then classified through the multi-class SVM which produces a class label out of the four existing classes $t' = svm(\mathbf{x}')$.

4.1.1 *Two-thumb Typing*

Two-thumb typing is a special case in the approach. ContextType considers this as a static posture. In other words, it does not provide for a variance in the touch of certain keys by different thumbs. It is believed most users will touch keys on far left and far right sides with the respective thumb. However, keys in the middle can be touched by both. Some users might have a tendency to use one finger more than the other. This can vary further depending on which key was touched beforehand.

Thus, we believe that typing with each thumb within two-thumb mode presents a unique offset model in itself and as a result consider these as separate postures (leading to 5 postures in total). The idea is to not only infer that a user is typing with both hands but also which thumb is used for each touch.

4.2 Offset Models

Learning offset models was achieved by a simple linear regression approach. Each offset is modelled by fitting 2 regressions - one for x and one for y where each of these is a function of both x and y. In other words, we start with M training examples $m_1, m_2, m_3, ..., m_n$ where each m is a tuple of (x, y) coordinates; M targets on x coordinate $u_1, u_2, u_3,\ ..., u_n$ and M targets on y coordinate $v_1, v_2, v_3, ..., v_n$. Targets are modelled as the distance on the given coordinate between a touch location and the centre of the key the user was asked to touch (fig. 1). Note that while x and y values are positive, u and v are real-valued numbers. User errors were filtered be removing any touches at a distance larger than 2 key widths from the given key centre.

Two different vector representations were investigated. Initially a training vector was (x, y) tuple. A second approach, similar to [1], represents a point as a vector in the form $\mathbf{p} = [x\ y\ x^2\ y^2]^\intercal$. The new model provides more flexibility to the offset functions learned. Furthermore, a slight reduction in overall mean squared error (MSE) was observed

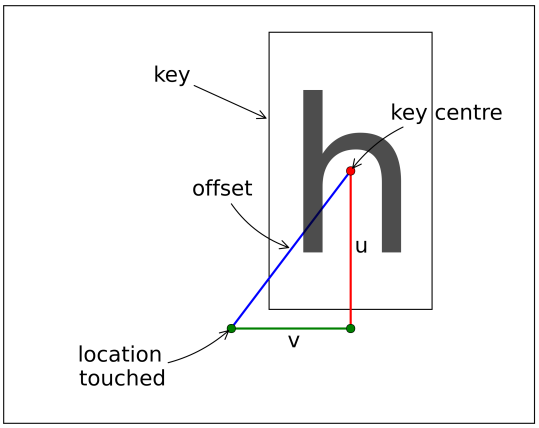


Figure 1: Modelling offsets. The intended touch location is the centre of the key but the actual location touched is offset on both x and y coordinates. The offset is modelled as the difference between the two locations on each coordinate - u and v. Note that the axes on the N9 are inverted.

compared to the simpler representation. This led to the decision to use the quadratic model for the remainder of the experiments. However, there are alternative representations which could also be suitable.

Using the least squares approach we fit 2 regression lines r_x with $(\mathbf{p}_1, u_1), (\mathbf{p}_2, u_2), (\mathbf{p}_3, u_3), ..., (\mathbf{p}_n, u_n)$ and similarly r_y with $(\mathbf{p}_1, v_1), (\mathbf{p}_2, v_2), (\mathbf{p}_3, v_3), ..., (\mathbf{p}_n, v_n)$. The scikit-learn [9] implementation of least squares regression was again used to fit the models.

To make predictions for a new touch location $a = (x, y)$, we build a new vector $\mathbf{p}' = [x\ y\ x^2\ y^2]^\intercal$ and use the regression models to predict an offset, i.e $u' = r_x(\mathbf{p}')$ and $v' = r_y(\mathbf{p}')$. The intended touch location is then $a_i = (x + u', y + v')$.

5. EVALUATION

Data for model evaluation was collected through a set of user experiments.

5.1 Hardware

An existing prototype mobile phone was used throughout the experiments. The device is a regular Nokia N9 smartphone extended with a $0.1mm$ thick flexible printed circuit board (PCB). This is placed on the back and the sides of the device and includes 24 capacitive sensors. The PCB is interfaced directly into the phone's internal bus.

5.2 Logging Application

A logging application was developed for the experiments. This was implemented using the Pygame library [1] for Python. The user interface of the application is a keyboard similar to keyboards used in touchscreen smartphones (fig. 2). The lower section of the screen contains the actual keys while the top section provides instructions and displays typed strings.

In the background the application logs data from all phone sensors. In particular on each touch of the screen, the application records:

- letter asked to type

[1] `http://pygame.org`

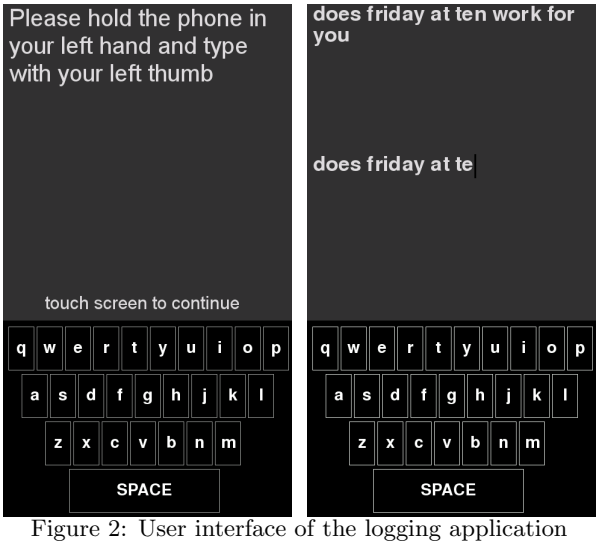


Figure 2: User interface of the logging application

- location touched as (x, y) coordinates
- accelerometer data for the past 0.5 seconds sampled at $50Hz$
- back-of-device values from all 24 sensors
- posture used for typing
- thumb used for typing (only in two-thumb posture)

All data is logged at both touchup and touchdown events so sensor data for any typed letter is recorded twice.

5.3 User Experiments

Experiments included 17 participants (12 male, 5 female), aged between 20 and 30 ($Mean = 23.4$, $SD = 2.6$). Three of the users reported as left-handed and only one of them reported they were not a regular touchscreen keyboard user.

Experiments required users to complete 24 tasks. In the first twelve of these users were asked to type a series of letters. A single task covered one posture and change of posture was required after every task. Each posture was investigated three times resulting in 12 tasks in total (4 postures, 3 times each). The aim was to obtain data from 3 separate sessions so as to investigate the magnitude of session effects in users' typing patterns.

By session effect we mean the difference in the way the user holds a device in a given posture on different occasions. For instance, after typing with right hand the user puts the phone down and then picks it back up to again type with his right hand. A significant session effect would mean than each time the user picks up the device, they hold it substantially different from previously. Furthermore, we would like to assess how susceptible our models are to such effects.

A task in the experiments required typing each of the 26 letters in the English alphabet twice, resulting in 52 touches. In two-thumb mode the number of touches doubled (52 for each thumb) and users were instructed which thumb to use for each touch. The letter sequence for each task was generated randomly.

The second set of 12 tasks followed a similar design but users were typing sentences instead. These were picked randomly from an existing sentence pool. The pool was generated from the Enron mobile dataset [15] which consists of sentences written by Enron employees on mobile devices. In this way the dataset represents a collection of real sentences typed on mobile devices which matches our requirements. The version used contained only lower case characters stripped of all punctuation. Sentences, however, contained verbalised numbers. Any sentences of length less than 10 characters or more than 100 were discarded. This resulted in a pool of 1332 sentences with average length of 38.6 characters, including space characters.

		Prediction			
		0	1	2	3
Truth	0	50	0	0	0
	1	0	36	0	0
	2	0	0	43	0
	3	0	0	0	102

(a) User 3

		Prediction			
		0	1	2	3
Truth	0	50	0	0	0
	1	0	52	0	0
	2	1	0	44	1
	3	0	0	0	83

(b) User 9

Table 1: Posture classification confusion matrices for two users. Two-thumb typing is considered as one posture. Class labels: 0 = *left-hand*, 1 = *right-hand*, 2 = *index-finger*, 3 = *two-thumb*.

Users typed 2 sentences per tasks except in two-thumb mode tasks. In these they typed 2 sentences per thumb and 4 additional sentences which did not specify a thumb. Instead users were free to use whichever thumb they liked at each touch. The aim was to collect data representative of their real typing patterns in two-thumb mode. Also, no user typed the same sentence twice.

The whole experiment took about 40 minutes to complete and users were given the option to take a break after every 8 tasks. A touch always registered the correct letter on screen and no means of error correction were available. The users did not perceive any of the errors they made. The aim was to collect data representative of the user's real typing behaviour which could be modelled as an offset function. For example, if users perceive the error and when asked to type P they tend to touch the area closer to O, they will modify their behaviour to make more accurate touches. Otherwise, we could model this offset and move any touches to the intended location in this way allowing them to type more naturally.

6. RESULTS AND DISCUSSION

This section presents the results from the undertaken evaluations. Irregular spikes were observed in one of the sensor values. This is believed to be the result of a hardware fault. Points containing such values were omitted throughout the experiments. The number of points filtered usually ranges between 1 and 2% of the number of data points for a particular user.

6.1 Inferring Posture

We started with an evaluation on the posture classification accuracy. This was based on the calibration data collected for all 3 sessions for each posture. The data was split into a training set and test set (30% of the data points). Cross validation was used to choose hyper-parameters.

Table 1 presents confusion matrices for posture classifica-

		Prediction	
		0	1
Truth	0	40	11
	1	7	35

(a) User 1

		Prediction	
		0	1
Truth	0	30	11
	1	16	35

(b) User 6

Table 2: Two-thumb classification confusion matrices for two users. Class labels: 0 = *left-thumb*, 1 = *right-thumb*.

tion for two users. Virtually all predictions here are correct. On average across all users, the model achieves a classification accuracy of 99.7% significantly better than ContextType's 89.7%. What is more, in ContextType this is "on average achieved after 5 user interactions" whereas our method makes predictions based on data for a single touch. This allows for not only more accurate predictions but also quicker ones - immediately after the user starts typing. In addition, dynamic changes of grip whilst typing can be detected instantly.

A following experiment looked into classifying thumbs within a two-thumb posture. Principal Component Analysis (PCA) was used for dimensionality reduction. Data was reduced to 2 dimensions for visualisation purposes. Figure 3 presents the PCA plots for three users. The plots do not demonstrate any clear distinction between the two classes. This is particularly true for user 6 (fig. 3b) where there does not appear to be any structure at all. What is more, we can see that there is a significant session effect in user 1's grip patterns (fig. 3a) which is something we will need to deal with. User 7 (fig. 3c) is the only user where there exists a relatively constant pattern differentiating between the two classes. In the plot, most right thumb touches are towards the top of plot while most left thumb ones are towards the bottom. This is, however, clearly not linearly separable.

We then looked into building a classifier to discriminate between the two classes. Table 2 presents the confusion matrices for two users. The significant decrease in accuracy compared to posture prediction demonstrates that this is a harder task to achieve. Predictions are still substantially better than random which suggests that there is signal in the data, although not clearly visible in 2 dimensions. Across all participants the SVM achieves an average classification accuracy of 76.8%. User 6 whose PCA plot reveals no structure still achieves 65 out of 92 points correctly classified (70.7%). Classification accuracy is much better for users 1 (80.6%) and 7 (81.7%).

6.2 Offset Models

Since typing with a thumb within two-thumb typing has not been considered before we initially looked into building offset models for this. Because of a slight change in the design of user experiments, offset models could not be calculated for the first two participants. Further results are based on the remaining 15 users.

Hyper-parameter optimisation is performed as previously. However, we also add an external 10-fold cross-validation to smooth out the results from the scripts. In other words, we run a script 10 times and each run has a unique train/test set split.

6.2.1 Two-thumb typing

User	No	Perf	Pool	SVM	GP	Aver
3	236.6	171.1	181.0	180.4	181.1	**177.7**
4	398.2	238.9	278.1	292.9	265.2	**256.3**
5	279.3	231.5	230.2	235.9	244.4	**229.3**
6	253.7	221.7	229.9	235.1	236.4	**228.4**
7	227.3	137.5	187.3	174.2	173.9	**163.7**
8	442.0	162.3	187.0	195.9	184.2	**180.4**
9	462.2	226.2	**233.2**	245.4	245.8	234.3
10	455.3	178.0	179.3	183.4	183.2	**177.8**
11	167.2	149.0	148.1	150.3	151.9	**145.7**
12	409.4	178.9	184.5	184.8	192.2	**179.3**
13	482.7	211.1	273.3	279.0	275.2	**247.0**
14	315.9	172.2	184.3	188.1	189.3	**178.4**
15	277.3	121.3	134.9	132.0	131.4	**129.8**
16	394.3	247.0	292.8	307.9	298.5	**279.4**
17	561.6	234.4	350.5	362.7	356.5	**329.0**
Mean	357.5	192.1	218.3	223.2	220.6	**209.1**

Table 3: Mean Squared Error rates for each user using each of the proposed models. Lowest values for each user in bold (not including the perfect model).

We first considered offset models within two-thumb mode as one model, similarly to previous work. We call this the *pooled* model. The first row in figure 4 presents the offsets for two participants based on the model. Figure 5 compares the *pooled* model performance against using the actual touch location (*no model*). Performance is measured as the proportion of touches within virtual buttons of different sizes. The virtual buttons are centred at the physical key centres and are shaped as circles with radius increasing from 1 to 7mm. The aim is to see how performance fluctuates on different sized buttons. A further performance metric used is MSE (table 3).

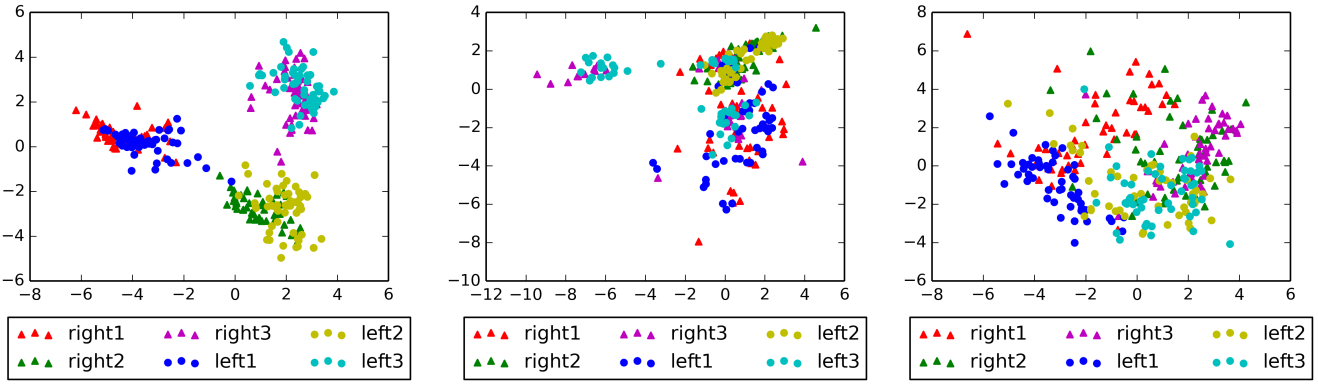
Figure 5 demonstrates a large improvement in performance of the offset model compared to the baseline. This is very apparent for both users 8 and 13. The performance gap is similar for other users. User 11 is an exception here as they observe very minimum improvement in accuracy through the model. This is probably the case as they are more accurate than other users which does not allow much room for improvement. Across users the impact of the model also decreases with the increase of the virtual button size. This is to be expected as the larger the button sizes, the easier it is to hit them. The two models usually converge at 4 or 5mm button sizes.

To compare the performance of models across all users we employ two metrics:

- V_2 - virtual button accuracy on buttons with 2mm radius (higher is better)
- MSE - mean squared error in pixels - measured as the sum of MSE values on x and y coordinates (lower is better)

With respect to these metrics across all users the *pooled* model achieves $V_2 = 83.1\%$, $MSE = 218.3$ while the baseline achieves $V_2 = 64.7\%$, $MSE = 357.5$.

We then considered thumbs as separate models and build offset plots for these (second row of fig. 4), we call this the *perfect* model as we know exactly which thumb was used for each touch. It can be observed that these two particular



(a) User 1 (b) User 6 (c) User 7

Figure 3: PCA of back-of-device data on two thumb typing for three users. Circles represent touches with left hand and triangles represent touches with right hand. Each of the three sessions is in a different colour.

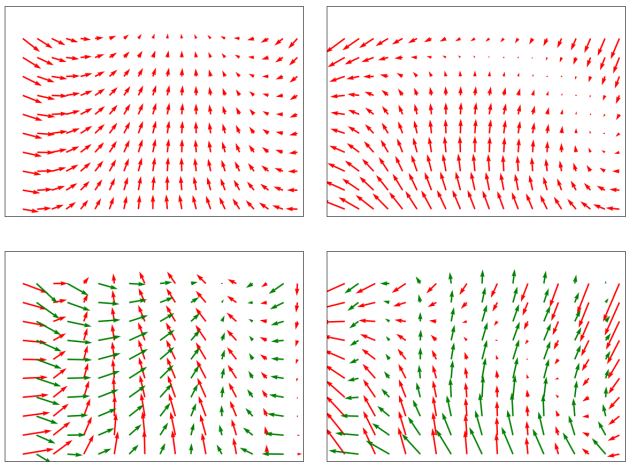


Figure 4: Offsets models for two users. Top row - *pooled* model and bottom row - *perfect* model (left thumb offsets in red and right thumb ones in green). Arrows are scaled to better reveal the pattern.

users have unique typing patterns for each of their thumbs. The *pooled* model in the top row looks like the average of the two thumb models in the bottom row. This limits the flexibility of the learned model which we believe would reduce typing accuracy. Modelling thumbs as separate models should be worthwhile.

As a following experiment we looked into modelling each thumb as a separate posture and based on back-of-device data, dynamically predicting the thumb used - *predictSVM* model (we use an SVM for classification). The *perfect* model has ideal knowledge on the thumb used, i.e. the model presents the ideal performance we can reach if make perfect predictions. Our goal is to build a thumb prediction model as close as possible to the *perfect* one.

We then compared the performance of all three models- *pooled*, *perfect* and *predictSVM* (fig. 6). As expected the *perfect* model outperforms all others for all three users. The *predictSVM* model, however, does not necessarily lead to an improvement over the *pooled* one. Across users the *perfect* model achieves $V_2 = 86.5\%$, $MSE = 192.1$ over $V_2 = 82.7\%$, $MSE = 223.2$ for the *predictSVM* model.

The problem with the *predictSVM* model is the cost of misclassification. From section 6.1 we know that about 25% of the time our classifier makes the wrong decision. Although most points are classified correctly, the cost of misclassification is high. Using the wrong regression model can lead to offsetting a touch to a new location further away from the centre of the key. This raises the need for new classification methods. In particular we require a probabilistic classifier to gain some confidence on the certainty of the prediction.

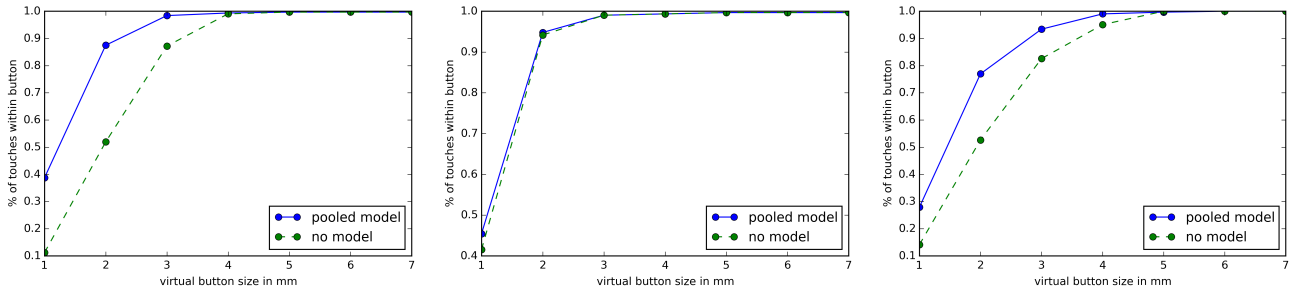
This led to the introduction of GP classification. Two new models were built here: *predictGP* and *averageGP*. The former is identical to the previous *predictSVM* with the only exception being replacing the classification algorithm with a GP. The class with highest probability was chosen as the class label. GP classification was implemented using the GPy[2] library for Python. The likelihood used is Bernoulli and inference was performed using the Expectation Propagation algorithm. The covariance function is RBF. The difference in the formulation of the RBF function in GPy and scikit-learn required a mapping from one to the other to allow the search through the same hyper-parameter values. GPy exposes two parameters for their RBF kernel implementation - variance σ and lengthscale l. By fixing $\sigma = 1$, one can express the relationship between l and scikit-learn's γ as:

$$l = \sqrt{\frac{1}{2\gamma}}$$

The equivalent range of values for l is now $l = \{0.22, 0.71, 2.24, 7.07, 22.36\}$. This is searched in the same fashion as previously.

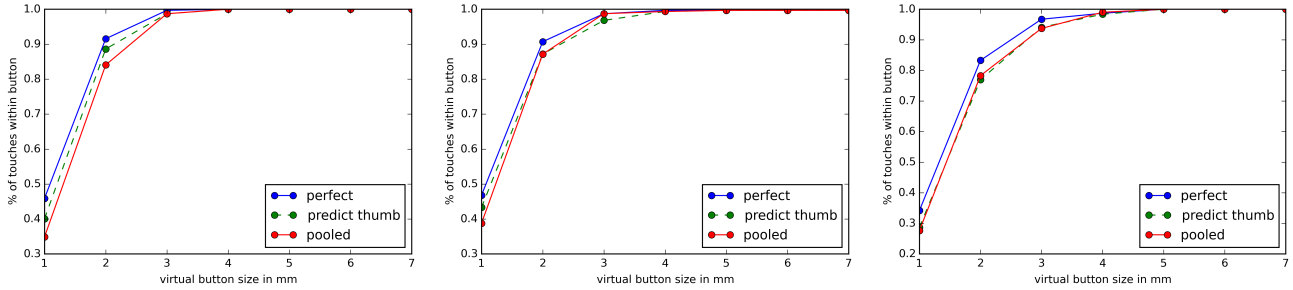
The *averageGP* model leverages the probabilities from the GP classification approach. Instead of making a decision on the thumb used, this model calculates offsets using regressions for both thumbs. A weighted average of the resulting offsets, based on the probability for each class, is taken as the final offset. With regards to out offset model definition, for each touch $a = (x, y)$ we calculate u_l, v_l, u_r and v_r, where u_l is the offset on x using the left thumb regression

[2] `http://sheffieldml.github.io/GPy/`



(a) User 8 (b) User 11 (c) User 13

Figure 5: Proportion of touches within different virtual button sizes for 3 users. The *pooled* model is compared to the *no model*.



(a) User 7 (b) User 8 (c) User 13

Figure 6: Comparison of *perfect*, *pooled* and predict thumb (*predictSVM*) models for three users.

model. The overall offset is then $u = p(l)u_l + p(r)u_r$ and $v = p(l)v_l + p(r)v_r$, where $p(l)$ is the probability of class *left-thumb* and $p(r)$ is the probability of class *right-thumb*, resulting from the thumb classification with the GP.

We compared the performance of the new models on the V_2 metric (figure 7). There is a notable improvement on all models against the *no model* approach and again the *perfect* model seems to perform best. More interesting here is the comparison between the thumb predicting models. It is clear that the *averageGP* model ($V_2 = 84.9\%$, $MSE = 209.1$) outperforms the others across most users. The improvement is especially prominent for users 6, 7 and 13. The only user that does not benefit from the weighted averaging is user 16.

To evaluate the strength of the weighted averaging approach, we also built a simple alternative model where we do not try to predict thumbs. Instead we calculate offsets for each thumb and simply take the mean of the two values. The aim was to see if simpler methods can lead to similar performance to the *averageGP* model. We compared the two models for users which observed large performance improvements by weighted averaging. With the simple averaging model user 13 achieved an $MSE = 271.6$ while user 17 achieved $MSE = 349.9$. These values are much higher than *averageGP*'s $MSE = 247.0$ and $MSE = 329.0$, respectively (tab. 3), and approach the respective *pooled* model values. This also supports are hypothesis that a *pooled* offset model represents an average of the two thumb offset models. Overall, there is a notable advantage of weighted averaging compared to a simple mean approach.

Comparing the GP to SVM reveals no significant difference in performance. The two models seem to slightly outperform each other per user with a significant difference in performance in favour of the GP model for participants 4 and 17. Overall, *predictGP* achieves $V_2 = 83.1\%$, $MSE = 220.6$.

The performance gap between the *perfect* model and others demonstrates that building offset models for separate thumbs can lead to significant performance improvements in two-thumb typing tasks. Perfect classification, however, is not realistically achievable and this hinders performance of thumb-predictive models.

The use of a probabilistic classifier along with a weighted averaging scheme can alleviate the problem. The *averageGP* model clearly outperforms other predictive models in the thumb classification task. The *averageGP* is the model with highest V_2 value and lowest MSE value across all users.

6.2.2 *All Postures*

A similar evaluation was performed for models covering all postures. As we have shown that modelling thumbs as separate postures can improve typing accuracy (sec. 6.2.1), here we consider these separate. This leads to 5 postures in total. We start with a *predictSVM* model, then build a *predictGP* one and finally an *averageGP* one. These are identical as previously except that: (1) instead of using binary classifiers we use multiclass ones (one-against-one SVM and a multinomial GP) and (2) we build 5 separate offset models for each user (1 per posture). Also, the *averageGP* model performs the weighted averaging on 5 offset values instead of 2.

Since multiclass Gaussian Process classification is not available for Python, a new implementation was required. This was achieved by translating MATLAB code for the multiclass GP approach proposed by Girolami and Rogers [4]

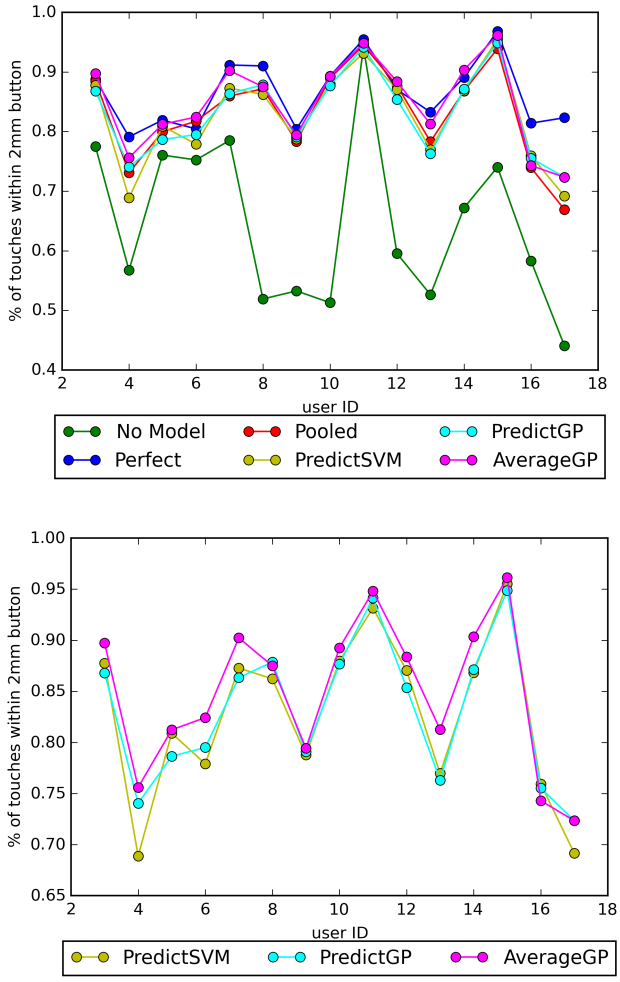


Figure 7: Percent of touches within 2mm button for each user: (top) across all models and (bottom) across thumb prediction models only.

(available online[3]). The authors propose a multinomial probit regression with a GP prior. A variational Bayesian approach is used for inference. We run the algorithm with a maximum of 50 iterations and monitor the percent change in the lower bound of the marginal likelihood. If the difference between iterations is less than 0.1% we terminate the algorithm. These limitations were introduced to limit the runtime of the experiments. Experiments here use the same covariance function (RBF) and hyper-parameters as previously. Again we measure V_2 and MSE values (table 4).

As a first experiment here we compare the performance of the baseline to the *predictSVM* model. Figure 8 presents accuracy results for virtual buttons of different sizes for three participants. All three users achieve a substantial improvement in virtual button accuracy. This is especially true for users 8 and 13. For user 11, who did not achieve significant performance improvements in two-thumb mode, we can still see an increase in accuracy. Although the user is quite accurate when typing with two thumbs, they are not so in other postures and this is where the models can lead to improvements. This can also be observed in other users where they would perform better in postures they usually use for typing and not so well in others, they are not too familiar with. Overall, the model achieves $V_2 = 82.5\%$ and $MSE = 222.2$ over $V_2 = 63.1\%$ and $MSE = 368.8$ for the baseline.

[3] `http://www.dcs.gla.ac.uk/people/personal/girolami/pubs_2005/VBGP/`

User	NoMod	SVM	GP	Average
3	293.5	230.7	230.1	**228.2**
4	457.2	255.9	255.5	**254.6**
5	324.4	264.7	265.8	**264.3**
6	279.5	227.1	227.8	**223.8**
7	228.9	179.9	181.1	**176.8**
8	433.5	190.9	**188.3**	188.6
9	422.7	238.7	238.3	**233.6**
10	398.4	188.4	188.4	**187.8**
11	198.5	157.1	**153.8**	156.7
12	433.3	207.9	205.3	**205.**
13	477.6	289.3	287.	**279.8**
14	330.3	206.6	208.1	**204.1**
15	316.8	141.3	**138.5**	138.8
16	452.6	287.7	**284.2**	287.6
17	485.1	267.1	**259.5**	274.7
Mean	368.8	222.2	220.8	**220.3**

Table 4: Mean Squared Error rates for each user using each of the proposed models across all postures. Values represent the sum of MSE values on x and y coordinate. Lowest values for each user in bold.

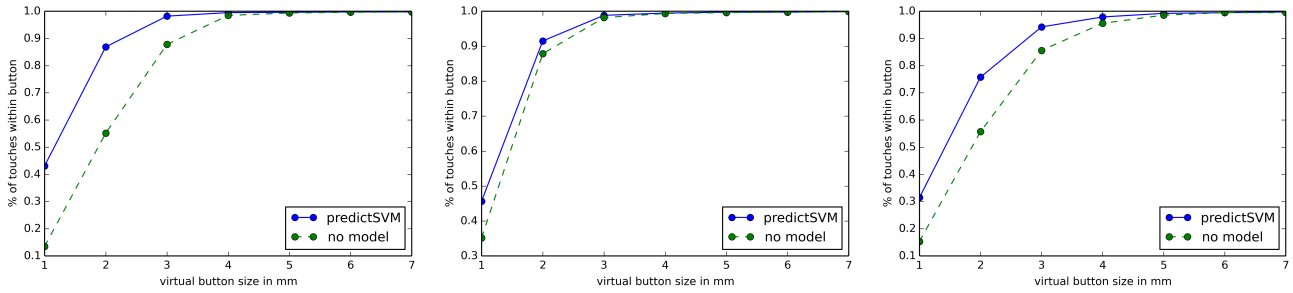
As previously we also compared the performance across models (fig. 9 and table 4). All of these lead to notable improvements across all users over the *no model* approach. The *predictGP* model achieves $V_2 = 82.7\%$ and $MSE = 220.8$, while the *averageGP* attains $V_2 = 82.8\%$ and $MSE = 220.3$. In figure 9 we can see that the *averageGP* models still seems to perform best for most users. This is not the case, however, for users 16 and 17 where the *predictGP* model is the top performer.

Overall, there does not seem to be a large performance gap between the three models except for a slight edge of the GP models over the SVM one. This is probably due to the fact of the almost perfect classification we get from the back-of-device sensor (sec. 6.1). The low misclassification rate does not allow for much improvement as we are almost always using the correct model. A further reason is the limitations we impose on the Variational Bayes iterations. Examining the resulting probabilities from the GP we noticed that even for postures easy to classify such as *index-finger* typing the model results in probabilities no higher than 0.6 for the respective class. Fitting a better GP could result in further improvements. Still, for some users we can gain a performance increase by using a GP over an SVM.

6.3 Text Entry Error Rate

As a final experiment we evaluated the performance of the typing models using the error rate metric proposed in [14]. The error rate is calculated as a function of the minimum string distance between the presented text (what we asked the user to type) and the transcribed text (what they actually typed).

Previous experiments considered letter typing data only while here we consider both letters and sentences. We use the former for learning the classification and regression mod-



(a) User 8 (b) User 11 (c) User 13

Figure 8: Proportion of touches within different virtual button sizes for 3 users across all posture. The *predictSVM* is compared to actual touch location

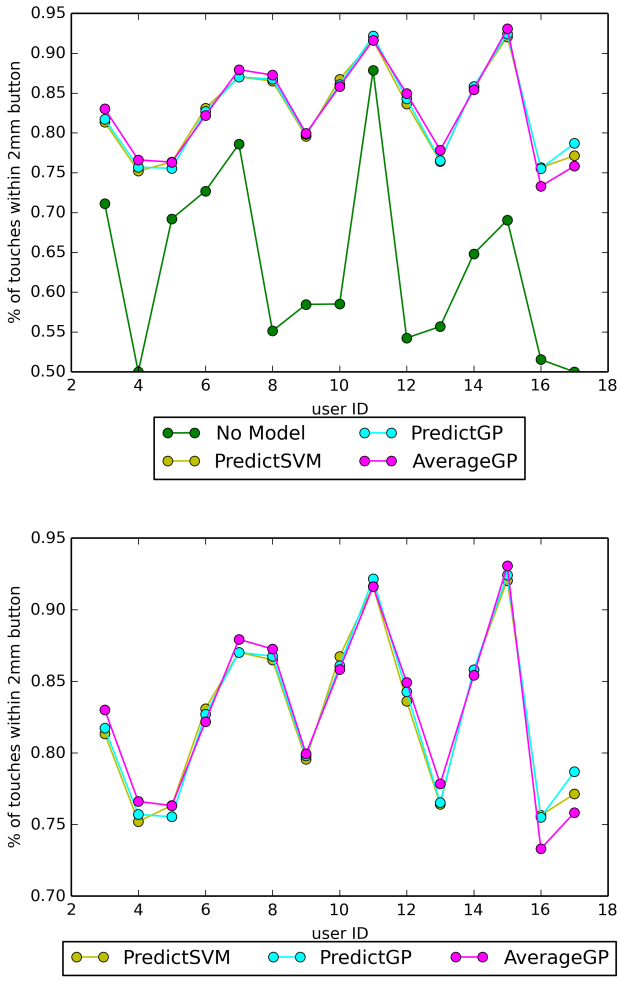


Figure 9: Percent of touches within 2mm button for each user across all postures: (top) all models and (bottom) prediction models only.

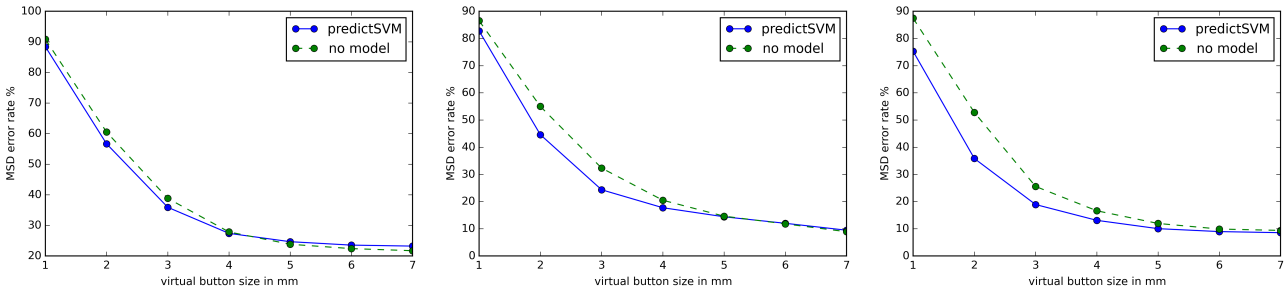
els (calibration data) and the latter as test data. As the test data is now in the form of sentences, this evaluation is more representative of the model performance on real typing scenarios. Moreover, since test data comes from sessions separate from the calibration ones, we also evaluate the generalisation performance of the models and their susceptibility to session effects. The experiments were undertaken on both virtual buttons (as previously) and physical ones (the actual key dimensions as used in the logging application keyboard). To gain an understanding of the overall performance per user and across users, the only models evaluated are the all posture ones.

It needs to be noted that all error rates reported in this section are slightly higher that actual. The reason for this is the filtering of touches which observe spikes in the back-of-device sensor. By omitting these touches we remove actual characters from the transcribed text which in turn artificially increases the error rates. This is, however, a rare event and is also observed on all models so if a touch is filtered out on one model, it will also be filtered on all of the others. Although this leads to an increase in error rates in absolute terms, the relative model performance should still be comparable.

Using physical buttons and no offset modelling users achieve an error rate of 18.45%. This is reduced to 17.22% for *predictSVM* and 17.28% for *predictGP* (lower values are better). As expected *averageGP* is the overall best performer at 16.29%. Improvement of *averageGP* over the baseline is statistically significant. This was determined through a paired t-test. We calculated this on both per-user basis ($p<0.02$) and per-sentence basis($p<0.001$). Again *predictSVM* and *predictGP* have very similar performance.

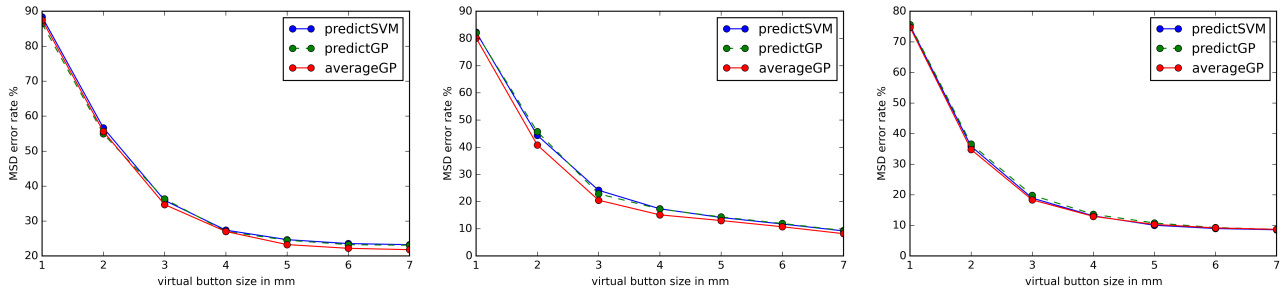
There are also substantial improvements for particular users, for example user 4 (*no model* = 18.08%, *predictSVM* = 13.44%, *predictGP* = 13.13%, *averageGP* = 10.37%) and user 8 (*no model* = 13.65%, *predictSVM* = 8.75%, *predictGP* = 8.39%, *averageGP* = 7.96%). There are, however, others who do not necessarily benefit from the more complex models, e.g. user 10 (*no model* = 15.77%, *predictSVM* = 10.96%, *predictGP* = 11.12%, *averageGP* = 10.89%).

Things are similar on virtual button sizes. Figure 10 presents the improvement of the *predictSVM* model over the *no model* approach. There is again a notable reduction in the *predictSVM* error rate against the baseline. This is more pronounced for users 4 and 10 and less so for user 3. We can also state that the model generalises well on unseen data and is not for most users susceptible to session effects.



(a) User 3 (b) User 4 (c) User 10

Figure 10: Error rates at different virtual button sizes for three users using *predictSVM* and no model. Note that lower values are better.



(a) User 3 (b) User 4 (c) User 10

Figure 11: Error rates at different virtual button sizes for three users, comparing predictive models.

We compare the error rate performance of the predictive models in figure 11. Improvements of the *averageGP* model over the others are not as pronounced here but still existent. This is especially true for user 4. Across users, at 2mm virtual buttons, the models achieve the following error rates: *no model* = 53.07%, *predictSVM* = 44.30%, *predictGP*=44.61%, *averageGP* = 43.07%.

Although in section 6.2.2 we did not observe a significant performance difference between the *predictSVM* and the *averageGP* approach, this is more pronounced here. We believe the reason for this is a more noticeable session effect for some users (e.g. user 4). Such effects increase the misclassification rate of the posture classification algorithm which limits the impact from the offset models. The weighted average approach, however, counterbalances the error to some extent, thus leading to better results.

However, we still believe that the performance is limited by the almost perfect posture classification. To investigate this hypothesis we looked into the error rate improvements for users who are known to benefit from the *averageGP* approach on two-thumb typing. For instance, using *predictSVM* across all postures user 13 achieves error rates of: virtual = 44.38% and physical = 15.92%. Switching to *averageGP* reduces these to virtual = 43.36% and physical = 15.44%. Looking at the same results for two-thumb typing only, we see that the user achieves the following error rates: virtual = 51.26% and physical = 19.75% for *predictSVM*, and virtual = 48.32% and physical = 16.23% for *averageGP*. Things look similar for user 17. For these users, the improvement in the two-thumb postures is more substantial than overall. However, as the improvement of the model in the remaining three postures is limited, this dominates the average across all 5 postures and does not allow for large overall improvements.

7. CONCLUSIONS

We proposed a new multi-modal approach to touchscreen typing based on back-of-device interaction and user-specific offset models. We showed that using back-of-device to determine posture is significantly more accurate than previously proposed approaches. Furthermore, our approach makes predictions instantly as opposed others which require a number of interactions.

We also demonstrated that thumbs should be modelled using separate offset models in two-thumb typing tasks. However, thumb inference is harder than the prediction of other postures. Misclassifications are expensive and can lead to a decrease in performance. Using a probabilistic classifier (GP) and a weighted averaging approach based on probabilities can help counterbalance these errors. The *averageGP* model substantially outperforms other thumb predictive models in two-thumb typing tasks. Across all postures the approach is also notably better than the other investigated models.

Future work could investigate better predictive models (GP or other) for the thumb used in two-thumb mode. The *perfect* model demonstrated that there is a lot of room for improvement over other models and the *averageGP* model is only on a partial solution to the problem. Another avenue for future work is exploring alternative offset functions, e.g. GP regression, similarly to [17].

Finally, the combination of the proposed models with a language model could be investigated. This is something

that was considered here but was not implemented due to time constraints. We would use the probabilities from the classification model and combine these with probabilities from the offset models which results in a probability distribution over keys. The language model then produces a similar distribution over letters. We use a decoder to combine these and produce the final key probabilities. The probabilities would provide more information to the decoder which would allow it to make better informed decisions.

8. REFERENCES

[1] D. Buschek, S. Rogers, and R. Murray-Smith. User-specific touch models in a cross-device context. In M. Rohs, A. S. 0001, D. Ashbrook, and E. Rukzio, editors, *Mobile HCI*, pages 382–391. ACM, 2013.

[2] D. Buschek, O. Schoenleben, and A. Oulasvirta. Improving accuracy in back-of-device multitouch typing: A clustering-based approach to keyboard updating. In *Proceedings of the 19th International Conference on Intelligent User Interfaces*, IUI '14, pages 57–66, New York, NY, USA, 2014. ACM.

[3] E. Clarkson, K. Lyons, J. Clawson, and T. Starner. Revisiting and validating a model of two-thumb text entry. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*, page 163, New York, New York, USA, Apr. 2007. ACM Press.

[4] M. Girolami and S. Rogers. Variational bayesian multinomial probit regression with gaussian process priors. *Neural Computation*, 18(8):1790–1817, 2006.

[5] M. Goel, A. Jansen, T. Mandel, S. N. Patel, and J. O. Wobbrock. ContextType. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*, page 2795, New York, New York, USA, Apr. 2013. ACM Press.

[6] M. Goel, J. Wobbrock, and S. Patel. Gripsense: Using built-in sensors to detect hand posture and pressure on commodity mobile phones. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 545–554, New York, NY, USA, 2012. ACM.

[7] I. S. MacKenzie and R. W. Soukoreff. A model of two-thumb text entry. In *Proc. Graphics Interface*, pages 117–124, May 2002.

[8] M. F. Mohd Noor, A. Ramsay, S. Hughes, S. Rogers, J. Williamson, and R. Murray-Smith. 28 frames later: Predicting screen touches from back-of-device grip changes. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 2005–2008, New York, NY, USA, 2014. ACM.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[11] S. Rogers and M. Girolami. *A First Course in Machine Learning*. Chapman & Hall / CRC, Nov. 2011.

[12] O. Schoenleben and A. Oulasvirta. Sandwich keyboard: 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*, MobileHCI '13, pages 175–178, New York, NY, USA, 2013. ACM.

[13] K. Siek, Y. Rogers, and K. Connelly. Fat finger worries: How older and younger users physically interact with pdas. In M. Costabile and F. PaternÃš, editors, *Human-Computer Interaction - INTERACT 2005*, volume 3585 of *Lecture Notes in Computer Science*, pages 267–280. Springer Berlin Heidelberg, 2005.

[14] Soukoreff, R. William, MacKenzie, I, and . Scott. Measuring errors in text entry tasks: an application of the levenshtein string distance statistic. In *Proceedings of ACM CHI 2001 Conference on Human Factors in Computing Systems*, volume 2 of *Short talks: interaction techniques*, pages 319–320, 2001.

[15] K. Vertanen and P. O. Kristensson. A versatile dataset for text entry evaluations based on genuine mobile emails. In *MobileHCI '11: Proceedings of the 13th International Conference on Human-Computer Interaction with Mobile Devices and Services*, August 2011.

[16] D. Weir, H. Pohl, S. Rogers, K. Vertanen, and P. O. Kristensson. Uncertain text entry on mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2307–2316, New York, NY, USA, 2014. ACM.

[17] D. Weir, S. Rogers, R. Murray-Smith, and M. Löchtefeld. 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*, UIST '12, pages 465–476, New York, NY, USA, 2012. ACM.