QuickSuggest: Character Prediction on Web Appliances

Ullas Gargi
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
ullas@google.com

Rich Gossweiler
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
rcg@google.com

ABSTRACT

As traditional media and information devices integrate with the web, they must suddenly support a vastly larger database of relevant items. Many devices use remote controls with on-screen keyboards which are not well suited for text entry but are difficult to displace. We introduce a text entry method which significantly improves text entry speed for on-screen keyboards using the same simple Up/Down/Left/Right/Enter interface common to remote controls and gaming devices used to enter text. The paper describes QuickSuggest's novel adaptive user interface, demonstrates quantitative improvements from simulation results on millions of user queries and shows ease of use and efficiency with no learning curve in user experiments.

Categories and Subject Descriptors

H.5.2 [**Information Systems**]: Information Interfaces and Presentation; H.3.3 [**Information Systems**]: Information Search and Retrieval; H.3.4 [**Information Systems**]: Systems and Software—*Performance evaluation (efficiency and effectiveness)*

General Terms

Algorithms, Human Factors, Measurement.

1. INTRODUCTION

With internet appliances there are many situations where a standard or touch keyboard is not available – internet-capable televisions, gaming consoles etc. Often the surrogate is an Up-Down-Left-Right (UDLR) keypad and an on-screen keyboard. The difficulty of entering text restricts the fluid dialog between the device and the person. As devices provide more services and more content (e.g. televisions with access to the WWW and online video content), the suddenly vastly larger vocabulary for search exacerbates the text-entry task.

Given a standard Up-Down-Left-Right remote control and an on-screen keyboard such as shown in Figure 6, our goal was to improve the experience and the time it takes to enter text, especially considering a large vocabulary (see Figure 1 for an example). If we move the onscreen characters that are likely to be typed to be closer each time the user types a character [1], then we improve the mechanical entry but increase the visual search cost – users don't become familiar with the layout even after repeated use. But if we do not adjust the layout, navigating the onscreen keyboard using a keypress for every letter traversal can make entry arduous.

Our solution was to implement predictive text entry at the character level and suggest four characters as a ring-like overlay inserted around the current character with minimum distortion(Figure 6). This model allows the person to rapidly get to the most likely character and, if entering an unlikely character, only cost one extra click while not significantly distorting the keyboard layout (Figure 1).

2. RELATED WORK

Improving the speed and ease of text input on constrained devices has been an active research area, mostly targeted toward mobile devices. The TNT system [2] uses a two-level grid and two keypresses select a character. The popup menu proposed by Isokoski [3] is closer to our proposed method but assumes a stylus. The popup menu does not displace the existing layout. The Dasher system [6] is philosophically similar to ours in the sense of ordering next letters in the order of probabilities. Constraining the available choices of letters on a touchscreen soft keyboard is an approach used by some automobile GPS navigation devices. Unlike some previous approaches, our adaptive interface does not require any learning on the part of the user. It evolves naturally from the existing input paradigm. In addition we have performed larger scale simulations on real user-entered text strings than most studies.

3. MODELING AND PREDICTION

QuickSuggest's language modeling is patterned after the PPM family of text compressors [5]. Given a corpus of text to train on, for every input character, we build up an array of contexts of various lengths (up to a parameter maximum, 7 in our experiments) with their associated occurrence probabilities. Optionally, we allow a popularity multiplier for every input string. We use a compact trie to store contexts and occurrence counts.

During prediction at runtime, the model loaded into memory is queried with the currently entered text. All context lengths are searched for matching contexts. We can either stop when we have sufficient predictions or aggregate probabilities for all possible next letters across all context lengths with a weight for each context. In that case the predictions we return are:

$$argmax\ P_p(L_j|X_m) = \sum P_m(L_j|X_m) * W_m,\ m = 1...M$$

where P_p is the predicted probability of letter L_j given the contexts X_m, P_m is the m-length–context probability of L_j, M is the maximum context length we use, W_m is the weight

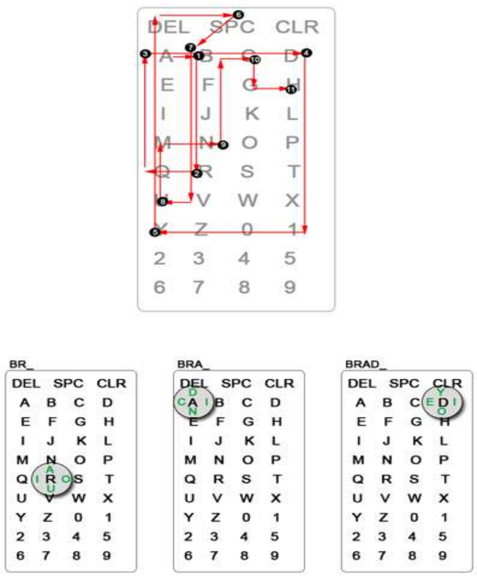

Figure 1: Navigating to enter "BRADY BUNCH" on a normal vs. QuickSuggest onscreen keyboard.

we assign to contexts of length m (longer contexts have higher weight) and $L_j \in V$, the vocabulary.

4. ADAPTIVE TEXT INPUT INTERFACE

Given a function that accepts a string prefix and returns an ordered list of next-occurring characters by probability, we developed a prediction-ring user interface. The goal was to move the most likely characters closer (reducing physical effort) without introducing significantly greater cognitive or visual search effort. Consider the navigation path required to enter the term "BRADY BUNCH" on an alphabetic onscreen keyboard. It takes 57 clicks for the 11 character term, as shown in Figure 1. With the predictive ring overlay, the click count was reduced to 25 clicks (the minimum possible is 2 clicks per character, 22 clicks). Figure 1 shows this in operation for the last 3 characters in "BRADY".

5. EXPERIMENTS AND RESULTS

We conducted two experiments to measure the utility of QuickSuggest. In both experiments we compared entering text on a non-predictive layout versus inserting the QuickSuggest predictive ring model. We chose the TiVo DVR layout as a baseline, which lays out the keys in a standard alphabetical pattern. Of course, QuickSuggest can be overlaid on any keyboard layout. The first experiment simulates millions of queries from actual users using lines from query logs or lines from entire datasets as input strings. This allowed us to test a very large vocabulary range with a very large sample set. The second study was an informal, within-subject user study with 10 people. This allowed us to see how the human element reacts to the new method. This was with a small population and over a small set of terms, so we view this as an informal, initial observation of how real people react to the new method.

5.1 Simulator Experiment and Results

5.1.1 Experimental setup

We trained a model on one of several corpora and tested the model on different sets of possible user inputs. The testing system works by passing the set of user input strings through a simulator. The simulator implements both the the baseline system (the default onscreen keyboard layout without prediction) and our prediction-augmented QuickSuggest keyboard.

Our metric is the number of keyclicks or button presses required to enter a string. For the default onscreen keyboard without prediction (the baseline) we have a static measure of the keypress distance between letters (this is the cityblock distance between them). The cost to enter a string with n letters L_i $i = 0...n-1$ is:

$$C_{nopred} = \sum_i d(L_i, L_{i+1}) \quad i = 0...n-2$$

where L_i is the current letter, L_{i+1} is the next letter, and $d(L_i, L_{i+1})$ is the number of key clicks needed to move from letter L_i to letter L_{i+1} on the onscreen keyboard. For example, in the alphabetical layout we used shown in Figure 6, $d('F','T') = 5$. For our proposed system, we predict the next letter for every letter in the input string and see if the actual next letter is in our predictions or not. The estimated cost in clicks to enter a string with n letters using QuickSuggest is the sum of costs when we are right and when we are wrong:

$$C_{pred} = C_{pred}^{right} + C_{pred}^{wrong}$$

These costs are:

$$C_{pred}^{right} = \sum_i P_r(L_{i+1}|L_0..L_i), \quad i = 0..n-1$$

$$C_{pred}^{wrong} = \sum_i P_w(L_{i+1}|L_0..L_i) \times (1 + d(L_i, L_{i+1})), \; i = 0..n-1$$

where P_r is the probability that L_{i+1} is a right prediction (i.e. one of our top (4) predictions) incurring a cost of 1, and P_w is the probablity that all our predictions are wrong, necessitating the default cost to move from L_i to $Li+1$ imposed by the keyboard layout plus a penalty of 1 to skip over our prediction ring. Note that an extra click to select the letter is always required.

To evaluate the amount of improvement or worsening we used the gain (loss), defined as the ratio of the decrease (increase) in the number of clicks needed with prediction for a particular string, to that without prediction for that same string. Another widely-used metric is Key Strokes Per Character metric (KSPC) [4] which we also measured in our large-scale experiments. We did not measure words-per-minute rate. The number of keypresses required is a good indicator of the motor load. The experiments varied in the training and testing data sets used. The datasets we used were: TV corpus (US TV show names and metadata), YouTube Titles (we sampled from approximately 3.7 million popular YouTube video titles), YouTube Queries (a set of completely anonymized user search queries on YouTube.com), and Google Queries (a set of completely anonymized user search queries on google.com.

5.1.2 TV corpus train, TV corpus test

In this experiment we trained a language prediction model on the TV corpus and also tested it on input strings (2–14 characters) from the same corpus. This was useful to measure the system performance on a somewhat narrow target domain where user queries are likely to be drawn from the same set.

Table 1: Prediction Impact By Corpus.

Impact	TV		YouTube		Google	
	Count	%	Count	%	Count	%
Positive	62283	82%	67M	78%	536K	53%
Negative	9036	11%	16M	19%	400K	39%
Neutral	4375	7%	3M	3%	77K	8%
Total	75694	100%	86M	100%	1015K	100%

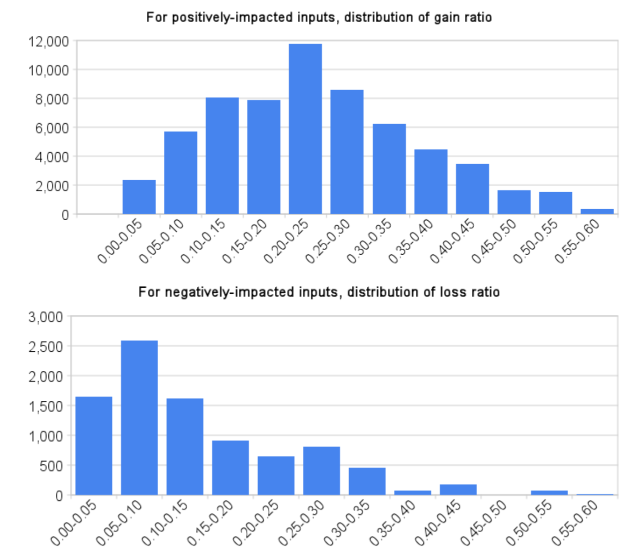

Figure 2: Distribution of gain and loss for positively– and negatively–impacted TV corpus inputs.

Table 1 lists the proportions of input strings which were improved, worsened or unchanged by using QuickSuggest. For the inputs which were positively (negatively) impacted, the gain (loss) had the distribution shown in Figure 2. It can be seen that the gain for positively–impacted inputs is substantially greater than the loss for negatively–impacted inputs. Choosing a sample of 3,072 8-character strings, we counted the cumulative cost to enter letters at each of the 8 positions (Figure 3). The graph without prediction shows a linear relationship as expected. With prediction however, we pay an initial penalty when our predictions are wrong for short contexts and the curve is above the baseline; then as we have greater context, the curve improves to below the baseline; finally, it saturates up to a length of 7 and then becomes linear again. This is because our maximum context length is 7 and for strings longer than that we lose some predictive power. The overall KSPC without prediction was 4.438. Using prediction, it dropped to 4.157. The minimum

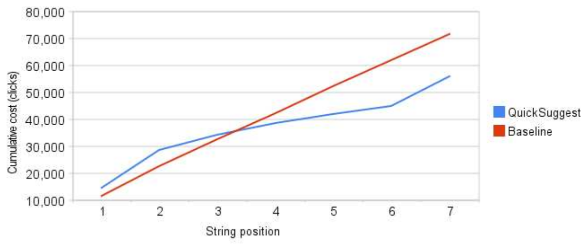

Figure 3: Cumulative cost (clicks) for a sample of 8-character strings in the TV corpus.

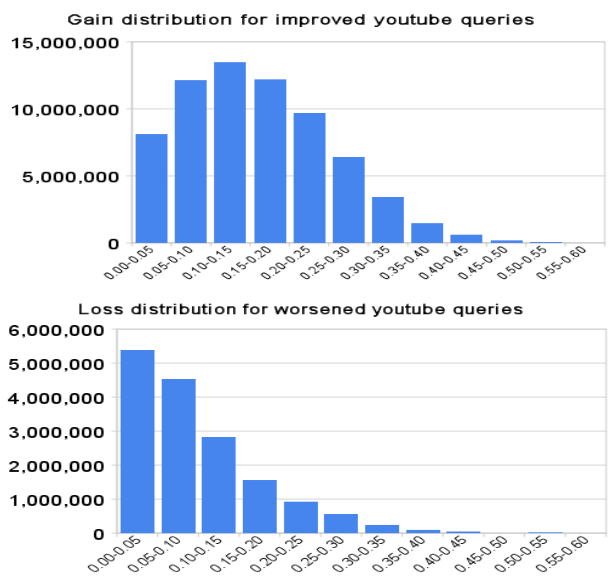

Figure 4: Distribution of gain for positively– and negatively–impacted YouTube queries.

theoretically possible KSPC is 2. Automatically moving the cursor to the top prediction dropped the KSPC to 1.605.

5.1.3 *YouTube Titles train, YouTube Queries test*

We used a model trained on 1 million popular YouTube video titles. We improved 78% of the queries and made 19% of the queries worse (Table 1). Gains on positively–impacted queries were larger than losses on negatively–impacted queries, as before (Figure 4) The KSPC metrics were 4.49, 4.10, and 1.58, for baseline, QuickSuggest, and QuickSuggest's top prediction auto-selection, respectively.

5.1.4 *TV corpus train, Google Queries test*

Even with a model not optimized for the domain, QuickSuggest shows significant improvement although not as marked as in our other experiments. The KSPC improved from 4.46 for the baseline system to 4.43 for QuickSuggest and 1.68 when auto-selecting the top prediction.

5.1.5 *Discussion of simulator experiment results*

In all three experiments, over large sets of user input text, the number of key presses required was decreased by using QuickSuggest. Even when text entry was made worse, the loss was less than the gain in those inputs that were improved. QuickSuggest initially incurs a penalty for short strings where there are too many possibilities for the short context; but then offers an improvement for longer strings. One possible modification is to turn on QuickSuggest only when sufficient text has been input. Automatically moving the cursor to the top predicted letter also lowered the KSPC substantially. Training the language model on the same domain as the expected user input was also shown to have an impact on performance.

5.2 Informal User Study Method and Results

The goal of this experiment was to obtain initial, informal results from people using the system. Ten subjects, five males and five females were asked to participate in a simple within-subject experiment. Half of the subjects owned a

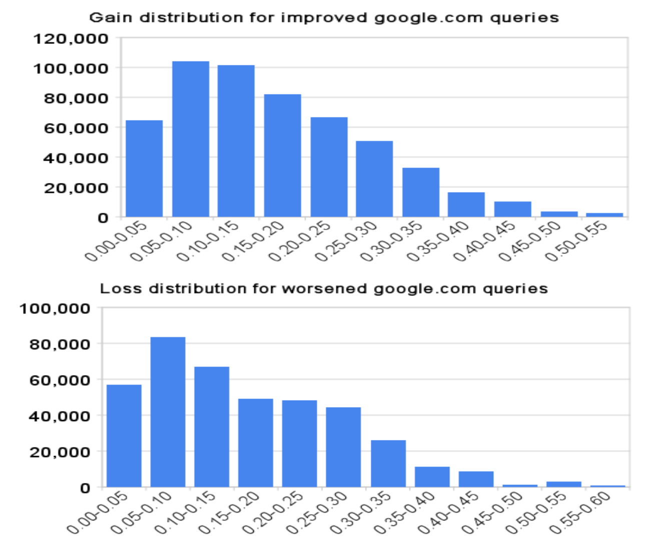

Figure 5: Distribution of gain and loss for positively– and negatively–impacted Google queries.

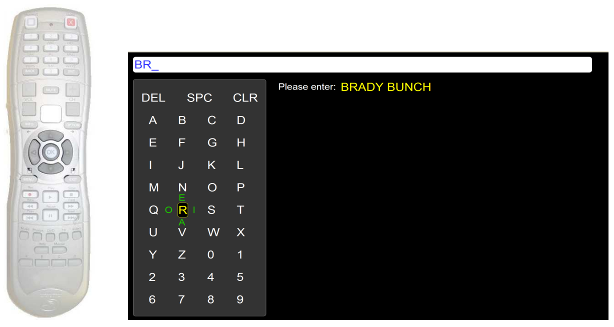

Figure 6: Remote control and on screen display used in the user study.

DVR. The subjects were given a remote control and sat in front of a 23-inch LCD screen. Half were asked to enter five shows using the standard TiVo layout and then the same five terms using the predictive ring layout and the other half did the predictive model first. We timed their button click rate, overall term entry speed and interviewed them about the two methods. The experiment presented a show term (e.g. "LOST") and an onscreen keypad that they could navigate with the remote control. They were asked to enter the show. When they completed the show, they pressed "OK" on the remote and then went on to the next term. We presented the following shows: LOST, BRADY BUNCH, ENTOURAGE, FAMILY GUY, and HOUSE. The first show, LOST, was a practice term and was discarded.

When the person was done, we asked them three questions to elicit feedback: Did they own a DVR? What were their thoughts on the two methods? Which would they prefer to have?

5.2.1 *Results and Observations*

As Figure 7 implies, the predictive model generally reduces the term-entry time. Our results also revealed that while the predictive ring reduced the number of clicks to get to a letter, it did increase the time per click since the person had to scan the ring. The non-predictive click time was approximatey 0.5 seconds while the predictive time was

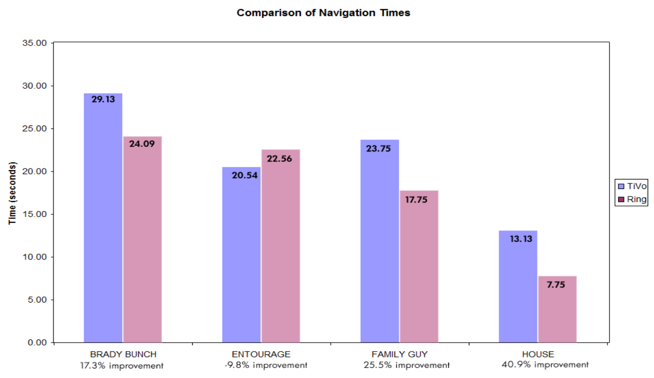

Figure 7: Comparison of navigation times for novice users.

approximately 0.9 seconds. Nine of the ten people strongly prefered the predictive model while one preferred the non-predictive layout. One person commented that "It seemed like it predicted the next letter as the 'up' [position on the ring] and so it was really easy to just go up up up." One person did not like it and commented that the ring was distracting and got in the way.

Several people commented that when they selected an item on the ring, they really liked that it automatically jumped to that letter's position on the keypad. We believe this helped ground them, making the keypad consistent and the ring feel as if it were a short-cut.

6. CONCLUSIONS

Based on both the informal user studies and large-scale statistical experiments on real user queries, QuickSuggest's predictive model shows merit as a light-weight "short cut" mechanism for character based entry when using web-enabled devices. Many appliances may want to have a simple input device rather than a full keyboard but still provide access to a large corpus of content. The predictive ring model helps balance visual search costs while reducing the distance to the target to reduce physical effort (Fitt's law).

7. REFERENCES

[1] T. Bellman and I. S. MacKenzie. A probabilistic character layout strategy for mobile text entry. In *Graphics Interface '98*, 1998.

[2] M. Ingmarsson, D. Dinka, , and S. Zhai. TNT - a numeric keypad based text input method. In *Human Factors in Computing Systems (CHI)*, pages 639–646. ACM, 2004.

[3] P. Isokoski. Performance of menu-augmented soft keyboards. In *Human Factors in Computing Systems (CHI)*, pages 423–430. ACM, 2004.

[4] I. S. MacKenzie. KSPC (keystrokes per character) as a characteristic of text entry techniques. In *Fourth International Symposium on Human-Computer Interaction with Mobile Devices*, pages 195–210, 2002.

[5] A. Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on communications*, 38(11), Nov 1990.

[6] D. J. Ward, A. F. Blackwell, and D. J. C. MacKay. Dasher - a data entry interface using continuous gestures and language models. In *UIST*, 2000.