Improving Automatic Whiteout++ through Feature Selection

Team 3

I. Problem Description

Mini-QWERTY keyboards are among the most common input technologies for mobile phones in the current generation. BlackBerry Limited along with other companies using mini-QWERTY on their mobile devices have produced a huge user base. These keyboards are of the size of key-pads and encompass everything from a normal QWERTY keyboard including space bar and the shift keys. While touchscreen keyboards are becoming more relevant, physical keyboards are still very much in use and are going to be the focus of this research.

Designs that deploy keyboards with key sizes of $25mm^2$ and inter-key spacing of $2mm$[2] are not uncommon. The users' thumbs being much larger, make visibility difficult and users get adapted to using the keyboard without visual assistance. This results in users making all types of errors like pressing multiple keys at once and pressing an unintended key. Fitts' Law implies that typing accuracy decreases as typing speed increases due to the relationship between target size, movement speed, and accuracy[6]. Smarter keyboards which can detect and automatically correct such human errors are important not only for a better user experience but also carry a huge business impact.

II. Related Work

The most relevant work done in this particular area was done with Automatic Whiteout++. The main focus of this algorithm was to improve the original design to better detect and correct roll-on, roll-off, key repeat, and row substitution error. Row substitution errors account for 45% of the total errors, so targeting and fixing these errors would produce a significant accuracy boost[3]. The algorithm utilizes decision trees in combination with Weka J48 to learn and classify keystrokes as errors or not. Some of the common features used in this decision tree are keys pressed, timing information, a letter's frequency in the English language, and adjacency to other keys. There were also some newer features added to the algorithm, such as bi-letter/tri-letter frequencies, probability based on large plain-text corpus, and subsequent key press evaluation. This allowed for the additional ability to handle row substitution error, and to improve the accuracy of corrections on the original Automatic Whiteout.

With all that being said, the correction accuracy can be further improved by adding some additional features to the decision tree. Physical limitations of the thumb and fingers while typing can have a great affect on the accuracy of said typing. This can be classified in a feature and utilized to better train a decision tree to more accurately predict errors while typing. This was not mentioned in the Automatic Whiteout++ implementation, and should allow for better prediction of typing errors.

III. Approach

A. Feature Selection

Automatic Whiteout++ incorporates 82 features, including bi-gram and tri-gram probabilities of letters and timing of key presses before and after the character in question.

We envisioned two types of features to improve the performance of the error detector.

1) Keyboard division: These features are based on our aim to encode the position of the key on the keyboard. Certain regions could be more conducive to errors due to difficulty of reaching that area. Hence we propose 3 different features for keyboard division:

- 3X3 rectangular grid
- Distance of the key from the bottom two corners of the phone to mimic thumb range (1)
- Distance of the key from the default position of the thumb when the phone is held (2)

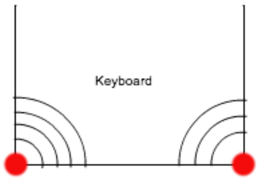

Fig. 1: Distance from the bottom corners of the keyboard

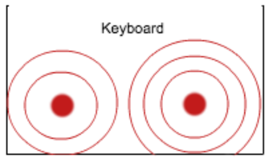

Fig. 2: Distance from the default position of the thumb

2) Insertion and Substitution probabilities: Kernighan et al[5] computed a set of confusion tables for different types of spelling errors - deletions, insertions, substitutions and transpositions. The tables were computed based on data from the Associated Press (AP) newswire. The ones that are of interest to us are insertions and substitutions. The values in the respective tables denote probability of inserting character X given that the previous character was Y and the probability of substituting character X with Y respectively.

For each character in the alphabet, we calculate the probability of it being inserted as an error, and the probability of inserting an error character after it using the following formulae:

If insertions$[X, Y]$ = count[inserting Y after X], then we can compute

$$p_{inserted}[X] = \frac{\sum_{X=a}^{z} insertions[X, Y]}{\sum_{X=a}^{z}\sum_{Y=a}^{z} insertions[X, Y]}$$

$$p_{insert-error-after}[X] = \frac{\sum_{Y=a}^{z} insertions[X, Y]}{\sum_{X=a}^{z}\sum_{Y=a}^{z} insertions[X, Y]}$$

Similarly, we can compute the probability of a character being present due to the substitution of the actual correct character as follows:

$$p_{substitution}[X] = \frac{\sum_{Y=a}^{z} substitutions[X, Y]}{\sum_{X=a}^{z}\sum_{Y=a}^{z} substitutions[X, Y]}$$

So, for each character we calculate 3 probabilities that we add as features.

B. Algorithm

Similar to Automatic Whiteout++, we decided to use decision trees to detect the errors. Given the number of features we have, decision trees are a good choice. Specifically, we used the J48 learning algorithm to build the decision tree.

IV. Evaluation

As Automatic Whiteout++, we used a set of experimental data that is an output of studies on two specific Dell and the Targus mini-QWERTY keyboards [2]. The study was carried out with 14 participants and 400 minutes of typing for each. The Twidor software was used to carry out the typing experiment. A total of 42,340 phrases were collected which accounted for 1,261,791 phrases. The data was further processed and each key press was characterized as correct or error. This set of data was used for training and testing.

We randomly separated 10% of the data to constitute the test set and the remaining 90% was used to train the model using the new features in conjunction with the old features of Automatic Whiteout++. Again similar to Automatic Whiteout++, we use the Weka J48 decision tree to train and model the data. Three models were built and tested with the same training and test data. While the first is the original Automatic Whiteout++ (we refer to it as the baseline), the second and third are the models built by adding the keyboard division feature and then the probability features incrementally. So, the following models were compared:

- Automatic Whiteout++
- Automatic Whiteout++ + keyboard division features
- Automatic Whiteout++ + keyboard division features + substitution and insertion probability features

	TP	TN	FP	TP+TN	TP+FP	Precision	Recall
Roll-on	2285	695	692	2980	2977	76.68%	76.76%
Roll-off	2959	154	442	3113	3401	95.05%	87.00%
Repeat	540	146	193	686	733	78.72%	73.67%
Substitute	3014	954	2670	3968	5684	75.96%	53.03%

TABLE I: Baseline - Training Cross Validation Metrics

	TP	TN	FP	TP+TN	TP+FP	Precision	Recall
Roll-on	240	68	71	308	311	77.92%	77.17%
Roll-off	366	13	46	379	412	96.57%	88.83%
Repeat	68	13	28	81	96	83.95%	70.83%
Substitute	363	116	290	479	653	75.78%	55.59%

TABLE II: Baseline - Test Data Metrics

	TP	TN	FP	TP+TN	TP+FP	Precision	Recall
Roll-on	2286	700	691	2986	2977	76.56%	76.79%
Roll-off	2980	162	421	3142	3401	94.84%	87.62%
Repeat	540	145	193	685	733	78.83%	73.67%
Substitute	3006	1002	2678	4008	5684	75.00%	52.89%

TABLE III: Division Features - Training Cross Validation Metrics

	TP	TN	FP	TP+TN	TP+FP	Precision	Recall
Roll-on	242	66	69	308	311	78.57%	77.81%
Roll-off	374	12	38	386	412	96.89%	90.78%
Repeat	68	13	28	81	96	83.95%	70.83%
Substitute	365	125	288	490	653	74.49%	55.90%

TABLE IV: Division Features - Test Data Metrics

For each of these models, we report their performance on the training set averaged across ten folds (10-fold cross-validation) and then on the test set using the respective trained model in the following 6 tables. Each table has metrics for the four types of the off-by-one errors and for each type of error, we report the following metrics:

- TP: The number of actual errors that were correctly classified
- FN: The number of actual errors which were incorrectly classified as some other type of error of non-error
- FP: The number of non-errors that were classified as errors
- TP + TN: Total number of actual errors
- TP + FP: Total number of errors guessed
- Precision: Percentage of total classification as errors that were actual errors
- Recall: Percentage of total actual errors that were correctly classified

V. Discussion

A. Division Feature Results

First, we look into the results of adding keyboard-division based features in Table III and Table IV.

From the perspective of cross validation metrics, we have a slight decrease (0.12%) in precision and a slight enhancement (0.03%) in recall for roll-on. For roll-off, we have a slight decrease (0.21%) in precision and a slight enhancement (0.62%) in recall. For repeat, we perform almost the same with baseline in precision and recall. And for row-substitute, we get a bit worse result (0.96% less) in precision and (0.14% less) recall.

In the test data metrics, precision and recall increase slightly both for roll-on and roll-off. For repeat, precision and recall is on the same level. For row-substitute, our performance is worse in precision (1.29% less) and better (0.31% more) in recall.

Considering that we focus more on row-substitute than other division features and the fact that we care more about precision than recall, the result is not as exciting as hoped, but some improvements were still made. The reason is that those three features are not important enough comparing with those original features. We added insert and substitution probability features hoping to improve the performance a bit further.

B. Division + Probability Feature Results

Comparing the training cross validation metrics after the addition of both the new features in Table V with the baseline in Table I, the precision increases by 1.5% for repeats while it is a bit worse for the other three types. Recall is slightly better for the roll-off and substitution errors and slightly worse for roll-on and repeats.

Considering the test data metrics in Table VI vs the test data metrics of the baseline in Table II, precision is improved by 1.4% for roll-on but decreases for the other types. Recall performs consistently better for all the four types of errors and is better by 3.1% for the repeats.

Comparing this test data against the test data metrics with only the division features, all the precisions suffer a hit and all the recalls except for repeats also perform worse. It's worthwhile to note that repeats was the only error type which showed no performance gain with the division features as compared to the baseline.

Although the addition of the new features gave a better performance on the recall consistently, it negatively effected the precision and thus calls for a trade-off. Overall, the addition of the probability features improve performance over the baseline, but do not add any substantial improvement over the original division-based features and actually decrease some of the original improvements over the baseline.

C. Conclusion

This project's focus started out with the intention of looking solely at the division of the keyboard to increase error detection and in-turn error correction. The keyboard division based features were expected to improve the substitution error detection, but actually improved the roll-on and roll-off error detection instead. The amount of improvement was not as high as originally expected, so exploration was done on the topic to find other suitable features to test for improving error detection.

This is when the project's focus expanded to include the insertion and substitution probabilities to see if this improved the error detection. The results with these features were also not nearly as high as originally anticipated, but some improvements were still made. This allowed us to really learn how to iterate over a list of viable hypotheses while working on projects and adapting to see which work better or worse than our original hypothesis. That being said, there are other areas that should also be explored in order to further improve the accuracy and predictability of Automatic Whiteout++, and those will be covered in the next section.

	TP	TN	FP	TP+TN	TP+FP	Precision	Recall
Roll-on	2282	707	695	2989	2977	76.35%	76.65%
Roll-off	2977	166	424	3143	3401	94.72%	87.53%
Repeat	523	129	210	652	733	80.21%	71.35%
Substitute	3018	1002	2666	4020	5684	75.07%	53.10%

TABLE V: Division + Probability Features - Training Cross Validation Metrics

	TP	TN	FP	TP+TN	TP+FP	Precision	Recall
Roll-on	242	67	69	309	311	78.32%	77.81%
Roll-off	371	14	41	385	412	96.36%	90.05%
Repeat	71	16	25	87	96	81.61%	73.96%
Substitute	364	127	289	491	653	74.13%	55.74%

TABLE VI: Division + Probability Features - Test Data Metrics

D. Future work

In this project, our efforts towards improving Automatic Whiteout++ were directed towards better feature selection, specifically, adding features based on keyboard division and substitution/insertion probabilities. We used the same classification algorithm, namely the J48 decision tree algorithm. Future work could compare the performance of other algorithms on the same set of features. One possible example that could be evaluated is random forests.[1]

References

[1] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[2] JAMES Clawson, Kent Lyons, Edward Clarkson, and Thad Starner. Mobile text entry: An empirical study and analysis of mini–qwerty keyboards. *Submitted to the Transaction on Computer Human Interaction Journal*, 2006.

[3] James Clawson, Kent Lyons, Alex Rudnick, Robert A Iannucci Jr, and Thad Starner. Automatic whiteout++: correcting mini-qwerty typing errors using keypress timing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 573–582. ACM, 2008.

[4] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[5] Mark D Kernighan, Kenneth W Church, and William A Gale. A spelling correction program based on a noisy channel model. In *Proceedings of the 13th conference on Computational linguistics-Volume 2*, pages 205–210. Association for Computational Linguistics, 1990.

[6] R William Soukoreff and I Scott MacKenzie. Towards a standard for pointing device evaluation, perspectives on 27 years of fitts' law research in hci. *International journal of human-computer studies*, 61(6):751–789, 2004.