Detection of Substitution Errors on Mini-QWERTY Keyboards

I. Introduction

The QWERTY keyboard is the *de-facto* standard for keyboard layouts and is used in Desktops, Laptops, Mobile Phones (keypad or touchscreen) and Tablets. Each device has its own space constraints. While most humans are accustomed to this keyboard, there is considerable error while typing, especially on mobile phones and smartphones.

Substitution errors are a class of typing errors where a character is replaced by a different character in a string. There are 3 main categories of substitution errors that are being targeted: Row Substitutions, Column Substitutions, and Mirrored Substitutions. These errors account for a majority of the typing errors that occur on QWERTY keyboards, with 62.92% of the errors on a full-QWERTY keyboard and 40.2% errors on a mini-QWERTY keyboard being substitution errors [2]. In this paper, we compare standard classification techniques for the detection of substitution errors with the detection in the Automatic Whiteout algorithm [3], also by performing feature selections to reduce the dimension of features and speed the algorithms up. We will introduce the Clawson dataset and describe it's features, their relevance to current and future work in the project.

II. Related Works

Recent studies of typing error correction include Automatic Whiteout [3] and Truekeys [5]. Clawson presents a multi-feature decision tree based method [3]. In the approach of Kane and Soukoeff [6] the input words are encoded by minimum string distance (MSD) compared with a dictionary, and the nearest word is selected. The problem of these approaches are that they use a string-by-string comparison strategy which is slow when the dictionary is large.

Another approach is the character-by-character correction method. Unlike the others' string-by-string strategies, character-by-character correction means that there is no need to even show wrong characters, so the user would be bothered less. Arif [1] and Deorowciz [4] model typing errors in both string-by-string correction models and character-by-character models. Clawson presents a multi-feature decision tree based method in Automatic Whiteout [3], which we compare our results to. But these approaches may have higher requirements on algorithm speed, which is our focus in this work.

III. Approach

A. Dataset

The Clawson dataset [2] was collected from two different sizes of mini-QWERTY (RIM Blackberry style) keyboards from 14 participants typing in 20 minute sessions on the keyboard models resulting in over 400 minutes of use per participant. Combining both studies we have a dataset of 42,340 phrases that the participants typed and 1,261,791 keypresses. Every instance is a single key press. However, to prevent the cascading of errors, the part of the phrase after the first error was discarded. After this step, 973089 remained of which 6549 are substitution errors i.e 0.67% in the complete set.

The dataset contains 95 features, features detailes are listed and described in Appendix I. To correct substitution errors, we use bi-letter (prob1), tri-letter (prob) frequencies to decide which off-by-one-substitution letter was inserted and should be replaced.

B. Learning Methods

In this paper, we will present the comparison of four classification methods - Decision tree, Adaboost, and Random forest. The reason we chose these algorithms is that, unlike K-NN, these decision tree based methods can be pre-trained and are fast in the testing phase, which is more desirable for real-time typing application built into a mobile device.

1) Decision Tree: The C4.5 decision tree uses information gain criteria to automatically select which features can split the training set best at each node. Automatic Whiteout [3] also uses this classifier. The J48 implementation in Weka was used to run this algorithm.

2) Adaboost: Adaboost (Adaptive Boost) is a type of ensemble learning method which has multiple weak classifiers. The final output is calculated by the weight of all the weak classifier's output. In our evaluation, we are using an implementation of the same in Weka, which uses decision stump (depth 1 decision tree) as weak classifier, and 100 weak classifier in total.

3) Random Forest: Like Adaboost, Random Forest is also a kind of ensemble learning method, which use decision tree as weak classifier. Unlike Adaboost, Random Forest assigns equal weight to all weak classifiers. In our evaluation, we are using the Random Forest implementation in Weka with 50 trees. For each tree, 10 features are randomly selected for tree generation.

C. Data Pre-processing

By looking at the ground truth labeling in the dataset, we can clearly see that the dataset is unbalanced. The number of non-error samples form more than 99% of the dataset and substitution errors count for less than 1%. Using such an unbalanced dataset will cause the classifier to be biased toward the error-free class. The training set is therefore resampled to balance it 50-50% of the error and error-free

classes. The original dataset is used to test the resulting model.

The raw data has 95 features, not all of which will play an equal role in classification. We performed several dimensionality reductions using WEKA's Sequential Forward Selection and Principle Component Analysis (PCA) which reduced the feature set to 5 - 10 important features. Reducing the number of features improved the runtime of the algorithm.

PCA is a dimensionality reduction method which selects features that have the largest variance in the feature space. The problem with PCA was that the output feature was a combination of the original features and not the original features individually. Sequential Forward feature selection selected the best set of original features and discarded the others. This was more suitable to demonstrate the importances of features. In our evaluation we let sequential forward select 5 features and PCA select 10 dimensional features.

D. Weighing Against False Positives

False positives (claiming that there is an error when there is none) are an issue of concern for any classifier. Clawson [3] found that weighting 4x against false positives gave us the optimum results after experimenting with a variety of weighting schemes. We used the same scheme and added a cost of 4x to weigh heavily against false positives.

E. Testing

Classifiers modeled on unbalanced training data were tested by us by using a 90%-10% split of the original dataset. Classifiers modeled on balanced datasets were tested against the original unbalanced dataset.

IV. Evaluation

For each method, we tried 6 combinations of settings. There are 2 kinds of training sets - (ie. balanced and unbalanced) and 3 kinds of preprocessing (no preprocessing, Sequential Forward and PCA preprocessing). As outlined above, while testing the models generated by unbalanced data, we split the dataset to 90% training and 10% test. To test models generated by balanced training data, we used the original, unbalanced dataset.

Figure 1 shows the results of our experiments. Evaluation criteria includes precision, recall, f-measure and the number of false positives injected by the method. Precision indicates the number of relevant cases that are retrieved. Recall gives us the number of retrieved cases that are relevant. Since precision and recall are contradictory in most scenarios, we use the F-measure to evaluate the overall performance of the algorithms.

$$F-measure = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

A. Learning Algorithms

Here in Figure 1 we compare different classification algorithms using precision, recall and f-measure. Notice that with cost-weighting (4x) to suppress false-positives, the detection of substitution error was lowered, especially for the Adaboost.

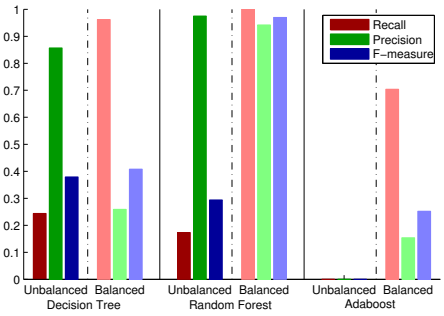


Fig. 1: Compare performances of all methods

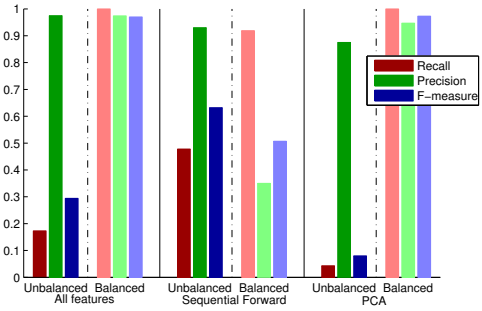


Fig. 2: Compare using all features / feature selection

B. Data Pre-processing

In Figure 2, we compare the results for different pre-processing techniques on Random Forest, including resampling and feature dimensionality reductions.

C. False Positives

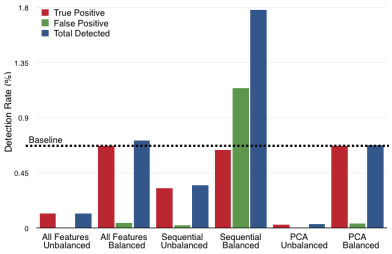
Figure 3 shows the total detection rate as well as the true positive and false positives injected by each classification method for the different pre-processing techniques. The dotted line indicates the original total detection rate (0.673%) provided by Clawson in Automatic Whiteout++. This was used as a baseline to measure the quality loss or improvement of the new models proposed by this paper.

Overall the best performance of 94.7% Precision and 100% Recall were received using Random Forest as learning method with a balanced data as a training set, and PCA as data pre-processing method. In comparison, the decision tree (used to compare our result to Automatic Whiteout++) obtained 25.9% Precision and 96.2% Recall on the data without pre-processing. The results show the significant gain in performance on reducing the number of false positives produced.

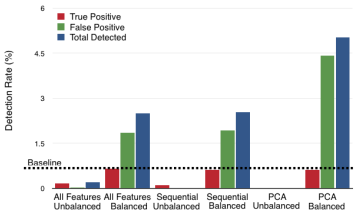
V. Discussion

A. Learning Methods

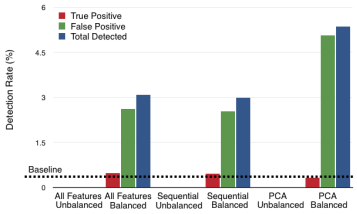
As seen in Figure 1, Random Forest trained on a balanced dataset produced the best results in terms of f-measure. This



(a) Random forest



(b) Decision Tree



(c) Adaboost

Fig. 3: False positive analysis

is expected since the model trained on unbalanced datasets would be biased toward the no-error class.

Balancing the training dataset increased the recall rate across all algorithms, which translated to better True Positive detections. The rates of total detection also significantly increased. However, the precision rates generally lowered as the False Positive rates increased, except in the case of Random Forest (all-features) and PCA. For the remaining poor detection cases (such as DT PCA and Adaboost), balancing gave us reasonable recall rates and moderately better precision rates.

We chose Adaboost to test the effect of cost weighting, as it was giving a very poor detection rate for the unbalanced all / SF / PCA tests. When unweighted, we have the best result with the unbalanced all-feature test, while we gain high recall-rate with balanced samples. When weighted the FP costs, the unbalanced tests failed to detect any error samples. However, the balanced samples gave better f-scores with higher precision rates.

B. Feature Selection

We experimented with feature selection methods since a reduced feature set would improve the time required to train the classifier and subsequently run in real-time on a device. This also enabled us to test with non-tree algorithms such as K-NN and SVM as the J48 Decision Tree algorithm has a pruning step which reduces the feature dimensionality of the training set according to the information gain of each feature.

For this reason, the results of feature reduction for the Decision Tree worsened the most of Precision and Recall rates. With Adaboost and Random Forest, which the Weka implementation does not automatically prune branches, feature selection was expected to give an improved quality. While this is partially true with the given results, we were surprised at the results of Adaboost which performed very poorly with an unbalanced training set.

C. Future Work

Most users nowadays use multiple fingers (usually two thumbs) to type. The current dataset doesn't directly tells us which finger (or thumb) was used to type the erroneous character. We think this feature would greatly impact the detection and correction of errors and also put existing low level features like hdist, vdist, dt_dd, dt_du, etc in a new perspective.

To calculate this feature, the keyboard could be divided into two zones, the first ranging from the letters QAZ to the letters TGV (as typed by the left hand), the second from the letters YHN to the letters PLM as typed by the right hand. Once the zones have been defined the time difference (dt_dd, dt_du, etc) and physical distance between the keys (hdist, vdist) now would not be between subsequent keystrokes, but subsequent keystrokes made in the same zone.

Lastly, our current hypothesis is that if we apply domain knowledges for feature selections or weighting of certain features - for instance, giving more weight on horizontal position feature over vertical position for the substitution detection, we may be able to improve the quality of all-feature decision tree which does use the vertical position feature.

References

[1] Ahmed Sabbir Arif and Wolfgang Stuerzlinger. Predicting the cost of error correction in character-based text entry technologies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 5–14. ACM, 2010.

[2] Edward Clarkson, James Clawson, Kent Lyons, and Thad Starner. An empirical study of typing rates on mini-qwerty keyboards. In *CHI'05 extended abstracts on Human factors in computing systems*, pages 1288–1291. ACM, 2005.

[3] James Clawson, Alex Rudnick, Kent Lyons, and Thad Starner. Automatic whiteout: Discovery and correction of typographical errors in mobile text input. In *MobileHCI'07: Proceedings of the 9th conference on Human-computer interaction with mobile devices and services*, 2007.

[4] Sebastian Deorowicz and Marcin G Ciura. Correcting spelling errors by modelling their causes. *International journal of applied mathematics and computer science*, 15:275–285, 2005.

[5] Shaun K Kane, Jacob O Wobbrock, Mark Harniss, and Kurt L Johnson. Truekeys: identifying and correcting typing errors for people with motor impairments. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 349–352. ACM, 2008.

[6] R William Soukoreff and I Scott MacKenzie. Measuring errors in text entry tasks: an application of the levenshtein string distance statistic. In *CHI'01 extended abstracts on Human factors in computing systems*, pages 319–320. ACM, 2001.

TABLE I: Appendix I - Features we used

Feature	Description
obosubstitute	(Boolean) The error type concentrated on by this paper. Off-by-one substitution error (2 keys are simultaneously pressed & the wrong letter is displayed)
errortype	What type of error was detected : Non-error / Rollon / Rolloff / Repeat / Obo-substitute / Multiple / Non-obo
ascii_nom	([A-Z], space) Current key pressed
p_nom	([A-Z], space) Key pressed before current
pp_nom	([A-Z], space) Key pressed 2 keys before current
next_nom	([A-Z], space) Next key pressed
dt	The time (in milliseconds) between the previous key and the current key dt(X-1, X). The threshold of 47ms signifies two keys were pressed together.
futdt	The time (in milliseconds) between ascii_nom & next_nom
prevdt	The dt (in milliseconds) between the current key and the previous key. dt(X,X-1)
prob	Probability of finding the ascii_nom in the dictionary given the previous two keys. P(X | X-2, X-1) Markov transition probability for previous letters
bestneighbor	Most likely neighboring key
neighborfreq	Relative frequency of most likely neighboring key, based on phrase set
neighborprob	The probability of the best neighbour occurring after X-2,X-1 Markov transition probability for previous two letters and current keys most likely neighbour
neighborprobdiff	P(X | X-2, X-1) - P(bestneighbor | X-2, X-1)
dropprobdiff	Difference in Markov transition probability scores between the strings that would result in dropping the current keystroke and dropping the upcoming one.
hdist, vdist	Physical horizontal & vertical distance between keys
hpos, vpos	Physical horizontal & vertical position of the current key
sameasprev_nom	(Boolean) True if and only if the ascii_nom is the same letter as the p_nom
futisletter_nom	(Boolean) True if and only if the next key is in [a-z]
curfutadjacent_nom	(Boolean) True if the ascii_nom is physically adjacent to next_nom
prevcuradjacent_nom	(Boolean) True if the ascii_nom is physically adjacent to p_nom key
dt_uu, dt_dd, dt_du, dt_ud	The time (in milliseconds) between two keydowns (dd), keyups (uu), a keydown & keyup (du) and a keyup & keydown (ud)
dur	The duration (in milliseconds) of a keypress
dur_avg_minus_dur	Duration variance (in milliseconds)
average_uu /dd/du/ud	Average the first three features (in milliseconds) and subtract the final one. Used for average_uu, average_dd, average_du, average_ud
du_sub	Subtract the first feature from the second (in milliseconds)

TABLE II: Appendix II - Detailed Results

(a) Decision Tree

	Training	Precision	Recall	F-measure	True-Positive Rate (%)	False-Positive Rate (%)	Total Detect Rate (%)
No Data Pre-processing	Unbalanced	0.857	0.244	0.379	0.1664	0.0277	0.1942
	Balanced	0.259	0.962	0.408	0.6472	1.851	2.498
Sequential Forward	Unbalanced	0.893	0.15	0.257	0.1027	0.0123	0.0115
	Balanced	0.239	0.9	0.377	0.606	1.935	2.54
PCA	Unbalanced	0	0	0	0	0	0
	Balanced	0.12	0.896	0.212	0.6027	4.442	5.024

(b) Random Forest

	Training	Precision	Recall	F-measure	True-Positive Rate (%)	False-Positive Rate (%)	Total Detect Rate (%)
No Data Pre-processing	Unbalanced	0.975	0.173	0.294	0.1181	0.003	0.121
	Balanced	0.942	1.0	0.97	0.673	0.041	0.714
Sequential Forward	Unbalanced	0.93	0.478	0.634	0.3267	0.024	0.3513
	Balanced	0.35	0.919	0.507	0.6372	1.147	1.7849
PCA	Unbalanced	0.875	0.042	0.08	0.0287	0.0041	0.032
	Balanced	0.947	1.0	0.973	0.673	0.037	0.674

(c) AdaBoost

	Training	Precision	Recall	F-measure	True-Positive Rate (%)	False-Positive Rate (%)	Total Detect Rate (%)
No Data Pre-processing	Unbalanced	0	0	-	0	0	0
	Balanced	0.154	0.704	0.252	0.473	2.611	3.084
Sequential Forward	Unbalanced	0	0	-	0	0	0
	Balanced	0.154	0.682	0.252	0.461	2.530	2.991
PCA	Unbalanced	0	0	-	0	0	0
	Balanced	0.058	0.462	0.103	0.310	5.056	5.366

(d) AdaBoost without weight cost

	Training	Precision	Recall	F-measure	True-Positive Rate (%)	False-Positive Rate (%)	Total Detect Rate (%)
No Data Pre-processing	Unbalanced	0.584	0.313	0.407	0.2137	0.1520	0.3658
	Balanced	0.058	0.884	0.109	0.5952	9.6556	10.2508
Sequential Forward	Unbalanced	0	0	-	0	0	0
	Balanced	0.032	0.964	0.062	0.6488	19.47	20.117
PCA	Unbalanced	0	0	-	0	0	0
	Balanced	0.02	0.809	0.04	0.5441	26.04	26.58