TouchMiner – Mining for Usability

User Manual for v. 0.03 Alpha

February 16, 2012

Abstract

TouchMiner is a command-line data mining tool for analyzing sequential patterns of characters in a preprocessed text data file. Specifically, we call this area of research **Touch Mining**, which stands for "*data mining for better keyboard usability.*" TouchMiner finds frequencies of patterns, calculates their probabilities and also writes information about the distributions of pattern frequencies. It is still in experimental alpha stage, missing many of planned features such as pattern filtering tools and an optimal keyboard creation algorithm. These things, among with others, are implemented during spring 2012. For now, feel free to try out the software.

Introduction

TouchMiner is a command-line tool for analyzing patterns in text data. Programmed in Java with Eclipse, it includes only a small fraction of object-oriented programming features. Therefore, it performs its duties quickly. Thanks to the computational power of modern personal computers, a standard dual-core PC can manage the calculations in a matter of minutes, provided the dataset is not exceptionally large. So far, TouchMiner has been tested with a sample data of 160 000 lines (or about 8000 pages) from Gutenberg free e-book library.

The driving force behind touch mining is the need to develop a better way to learn touch typing and to improve usability. As we continue using qwerty keyboards for no apparent reason, we are at the same time creating an unnecessary overhead by non-optimal key placing.

Anyways, it's a miracle if qwerty will be replaced in near future. Still, we can benefit from touch mining by charting the most frequently used patterns of text and focusing on their practice. A simple analysis of individual characters is not enough, since we rarely type letters separately. Rather, identifying patterns of 2, 3 and 4 letters makes more sense, and longer expressions are composed of these parts.

If you choose TouchMiner for your analysis needs, note that

- you should already have your data as *a preprocessed text file*. In some distant future, TouchMiner may be modified to accept multiple files as input, but not for some time
- you need *at least 200 MB of disk space on C:* for analysis files
- so far, TouchMiner has only been tested on a PC running Windows 7. Linux testing is underway.

Note that **TouchMiner is not Open Source or "Free Software"** since we do not, for the time being, want other programmers to create forks that may or may not come out right. We want to ensure the mathematical and statistical treatment of data remains correct, untampered and of high quality, mainly because the distributions analyzed are not normal (i.e. Gaussian statistics are not valid). The source may or may not be released at a later stage, however, if math or statistics people are interested in developing it further.

Jyväskylä, February 16, 2012,

Riku Järvinen

Contents

1 A Short Tutorial

This section tells you how to get started with the analysis tools of TouchMiner. The program takes less than five minutes to set up, provided you have `Java Runtime Environment` (JRE) installed. If not, download it now.

1.1 Download and set up TouchMiner

Download the latest version from TouchMiner homepage. The software, user manual and an example text are packaged inside a `.zip` file which you need to extract (double-click) to some folder. TouchMiner is used from the `command line`, which can be opened by holding the shift key and right-clicking somewhere inside the folder where you extracted the files. A menu appears from which you can choose to open a command line window.

1.2 Perform example analysis

Once the command window opens, you start TouchMiner by typing

```
java -jar TouchMiner
```

You can browse the built-in help if you like. Whenever ready, go back to main menu and select option 2, which starts the *Small characters raw data analysis*.

TouchMiner asks for a directory name and creates the directory under `C:\TouchMiner\`. The analysis files are stored inside that folder. Next you need to specify the relative path to the text file to be analyzed, including the filename and extension. If you place the file inside the same folder in which TouchMiner resides, you only need to write the filename, e.g. `example.txt`.

Try out the program. It takes a few minutes to perform the analysis, and you should find the result under `C:\TouchMiner\` in the folder you specified. In the next section, we tell a bit more about the files TouchMiner creates.

2 Features

TouchMiner v. 0.03 alpha has three features: *"Naive" frequency analysis of characters*, *small characters raw data analysis* and *creation of numerical data for text pattern frequency distributions*. Let's take a brief look at these.

2.1 Naive frequency analysis of letters

As a comparative feature, we provide a character frequency analysis. When more advanced analysis features are developed, it will be interesting to compare by how much the results deviate from those obtained in this trivial way.

Below we have a sample figure 1 from the analysis of approximately 160 000 lines (about half Finnish and half English). Over six million letters were analyzed.

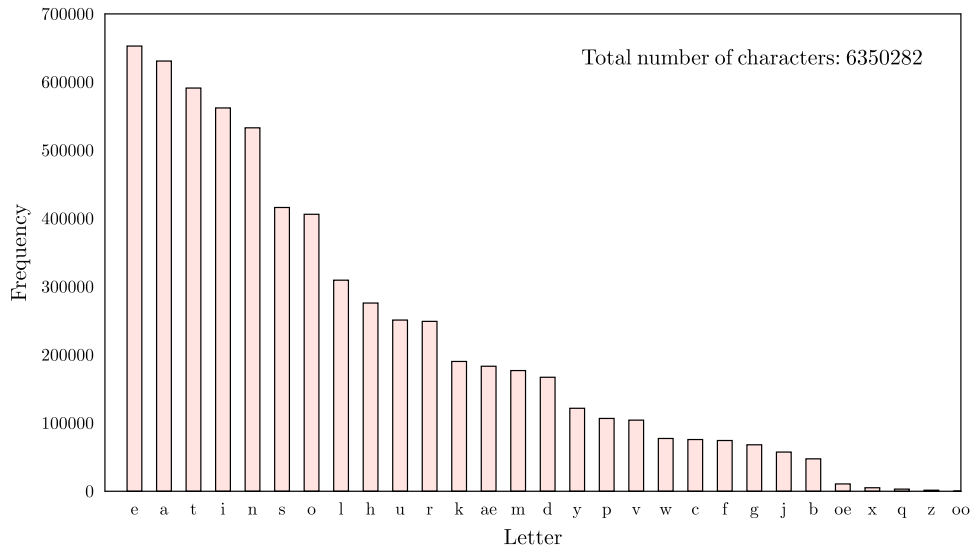

Figure 1: Frequencies of individual characters. As expected, the vocals *e*, *a* and *i* share the top positions with consonants *t*, *n* and *s*.

More complete results will be presented later once we have acquired a larger dataset for testing.

2.2 Small characters raw data analysis

In the preliminary analysis of text patterns, we examined both small and capital letters and some special characters using a test data from the Gutenberg e-book library. It can be inferred from the results that *small letters dominate*. Therefore, the first task was to develop a small characters analysis. In short, it does the following:

- reads a text input and finds all sequential patterns of two, three or four characters.
- creates a subfolder (name specified by user) in `C:\TouchMiner\` and writes all analysis files in that folder.

TouchMiner creates files for *absolute frequencies* and *normalized values*. More specifically, using 2-dimensional (2-character) filenames as an example:

- `2Ddistribution.dat` holds the text pattern frequency distribution data from the analysis of two sequential characters. We take a closer look at this in subsection 2.3.
- `2Dindex.txt`, `2Dlet.txt` and `2Dzval.dat` contain frequency (count) data listing indices, letters and frequency counts only, respectively. An example data from `2Dindex.txt` looks like the following:

```
17 24 7759
17 25 25
17 26 0
17 27 2692
```

```
17 28 338
18  0 42155
18  1 446
18  2 2476
18  3 246
18  4 56437
```

The first number is the index of the first letter, the second number corresponds to the second letter in sequence and the third value is the frequency of patterns, i.e. how many times a certain pattern was found. This becomes perhaps clearer when we compare with the same example data from the file `2Dlet.txt` which is shown below.

```
r  y 7759
r  z 25
r  å 0
r  ä 2692
r  ö 338
s  a 42155
s  b 446
s  c 2476
s  d 246
s  e 56437
```

So, in this example, we found 7759 patterns of 'ry', zero patterns of 'rx', 42155 times 'sa', and so on. The indices are coded so that zero corresponds to letter `a` and 28 to `ö`. As a concluding observation, the same data from file `2Dzval.dat` is presented below.

```
7759
25
0
2692
338
42155
446
2476
246
56437
```

We used the `.dat` extension for `zval` since the data is employed as an input for GLE (Graphics Layout Engine), which will be discussed shortly in section 3.

- The files `2Dnorm.txt`, `2Dnlet.txt` and `nzval.dat` are similar but contain *normalized* values, i.e. the sum of all nzvalues in each dimension is equal to 1. Example data from `2Dnorm.txt` is presented next.

```
17 24 0.001505
17 25 0.000005
17 26 0.000000
17 27 0.000522
```

```
17 28 0.000066
18  0 0.008175
18  1 0.000086
18  2 0.000480
18  3 0.000048
18  4 0.010945
```

Note that raw analysis data takes about 63 MB of disk space. A sample plot from preliminary analysis is presented in figure 2.

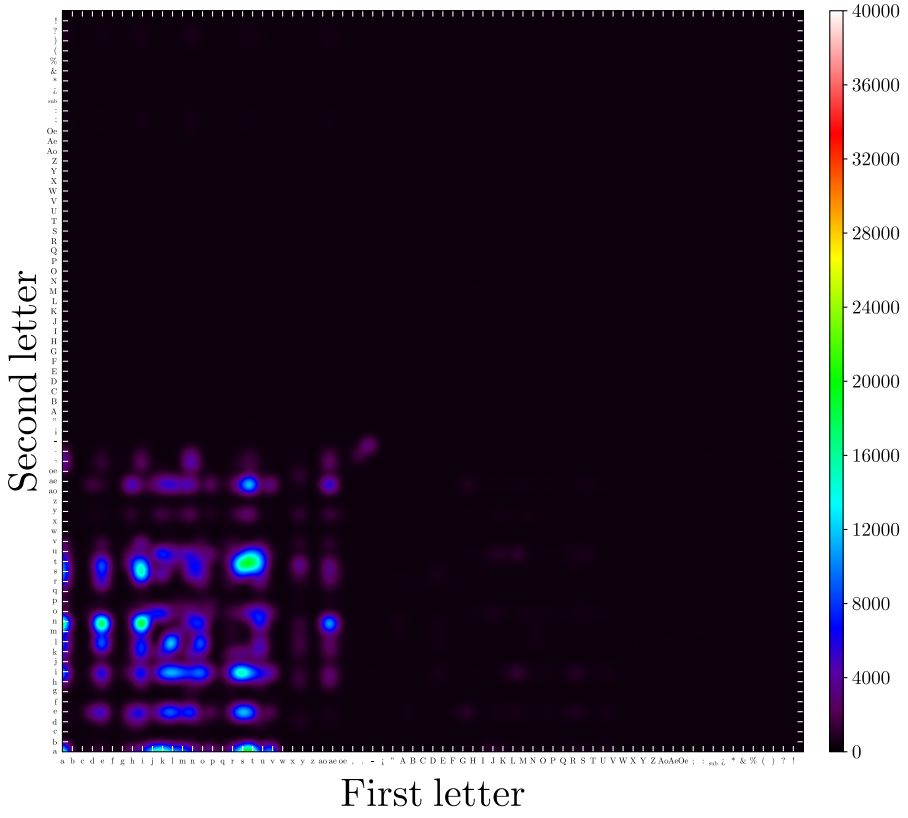

Figure 2: An example of two-character pattern frequencies from preliminary analysis, which also included capital letters and some special characters. The figure is not very accurate since the z values are averaged. We are currently working on a better, discrete graph.

2.3 Text pattern frequency distributions

TouchMiner divides the observed frequencies into 116 classes of constant width w, which is the largest value of a single pattern observed divided by 116 (and rounded down). In future versions, adaptability may be added so that user can specify the class width. Figure 3 shows an example graph for two-character patterns.

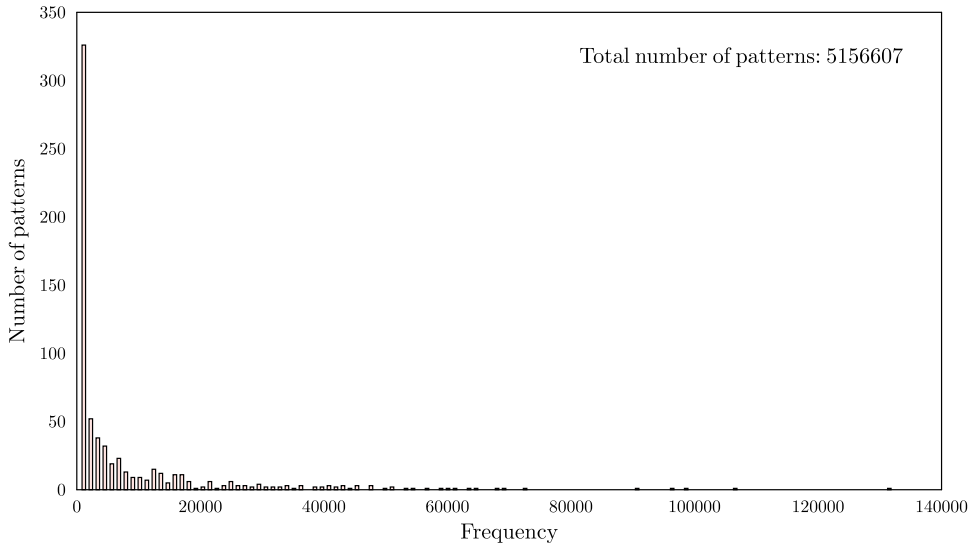

Figure 3: Frequency distribution of two-character text patterns. We instantly see that the majority of patterns have a relatively low frequency (the x-axis values are upper limits). The observed distribution is far from normal. In a way, however, we have identified a major irrelevant component in data which can easily be removed by *filtering* (see section 4.1).

3 Interaction with other programs

As for graphical representations of data generated by TouchMiner, we recommend the GLE, short for *Graphics Layout Engine*. It is a free, high-quality plotting tool equipped with advanced programming features. The plots and bar graphs presented in this manual are all drawn with GLE.

Since the *purpose of TouchMiner is to provide data mining results*, it will not include any features of a touch type training software. There are many promising Open Source typing projects for which the mining results could be applied, such as Klavaro and Ktouch.

4 Future directions

TouchMiner is still at its (infant) alpha stage, so there's a lot of development ahead. There have been many suggestions for additional features, but for the time being we only focus on the core

functions. In this section, we outline some features we plan to add in near future.

If you have read this manual till this point, we believe you might be interested in Touch Mining. *If you feel like you could do basic development and testing for the project*, contact the corresponding author *Riku Järvinen* via email for more information. The CC license can be waived if necessary.

4.1 Filters

As we saw in section 2.3, the distributions of patterns are not normal. In fact, they have a "high" tail which contains text patterns rarely encountered. It would be preferable to *filter the irrelevant patterns* so that e.g. 99.5 % of total frequency count is still accounted for. Because the use of Gaussian statistics is not justified, we cannot utilize measures like 'standard deviation' or 'mean value': they have no practical value. Instead, we do something else:

> We store the number of patterns P in each dimension and calculate numbers $k = P/200$. Then we set values $j = k/N$, where N is the number of possible patterns in a dimension. Next we remove all patterns that have a frequency lower than this limiting value, resulting in a maximum total contribution of k of removed patterns.

As an example, we consider a situation from our preliminary testing. The number of observed patterns for two consecutive letters was 5156607. In two dimensions, we have $N = 29^2 = 841$ theoretically possible patterns. The maximum number of counts to be removed is $k = 5156607/200 > 25783$. Dividing this by N gives $j = 25783/841 > 30$ and therefore we should remove all patterns that have a frequency of 30 counts or less. As a comparison, note that the largest frequency found (combination '*in*') was 131547.

The advantage of using a filter before ordering the results is a considerable reduction of computational strain. Nevertheless, the end results contain most of the relevant knowledge. One filter will be programmed for the next release of TouchMiner, and it will be for 99.5 %.

4.2 Keyboard creation

It is a well-known fact that the qwerty keyboard is outdated. The design principles used in its planning (hammers that keep jamming themselves etc.) are no longer valid. Alas, we need something better, from the viewpoint of usability and ergonomics.

TouchMiner was originally meant to be a *keyboard constructor*. The idea is to use combined results from the analysis of 2-, 3- and 4-character patterns to include characters into the keyboard, one by one, with a greedy algorithm, in each step determining the best possible outcome (highest probability in a given set of characters). The mathematical idea is almost finished, and will be presented here in near future. The algorithm, however, will take some time to program, because the application of abstract 4-dimensional set algebra to programming is nontrivial.

4.3 Words analysis

In conjunction with the previous section, we are also interested in including a contribution from words analysis in the keyboard creation process. **The main assumption of this research was**

that a word is a well-defined "unit of writing". Short text patterns are constructors of these units, so if one masters the patterns, one should pretty easily be able to conquer whole words. The grounding idea comes from Finnish elementary schools, where children are taught to read and write by practicing short patterns and syllables, then whole words, and finally sentences and larger pieces of text.

This feature will not be implemented for some time because the mathematical basis is not yet finished.

4.4 Special characters analysis

TouchMiner can be modified to analyze patterns in any set of characters. Information obtained this way helps to develop tools for special character training, which is essential in many professional areas, e.g. programming. Our plan is to include a special character analysis (at least for raw data) targeted at programmers some time during Spring 2012.

5 Changelog

Most significant changes between released project versions will be presented here.

Version 0.03 alpha

- First alpha version released. It still misses many of the core features, such as filtering nonessential data and keyboard creation. It has frequency distributions, small character raw data analysis and individual character frequency analysis.
- The source code is still a mess, mainly because the structure fo the program as a whole is not yet fully determined.