Fabio Pruneri

Harvard University

Keystroke logging in second language writing

RWTH Aachen University
Anglistische Sprachwissenschaft
Elma Kerz & Marcus Ströbel

UROP – Undergraduate Research Opportunities Program

Aachen
July 12, 2019

Table of Contents:

page:

1. Introduction 4

2. Project Description 4
   - 2.1 Description of available data
   - 2.2 Objectives of the analysis

3. Pre-processing, cleaning and summarizing data 5
   - 3.1 Transforming JSON into .csv
   - 3.2 Cleaning copy-pasted data
   - 3.3 Computing summary statistics of our dataset
     - 3.3.1 Keystroke speed and frequency
     - 3.3.2 Sentence length and speed
   - 3.4 Running CoCoGen to obtain complexity scores

4. Data analysis and interpretation 7
   - 4.1 Gathering metrics from keystroke data
     - 4.1.1 Defining Fluency and Latency
   - 4.2 Relating linguistic complexity with keystroke metrics
     - 4.2.1 Number of clauses
     - 4.2.2 Linguistic sophistication and Diversity
     - 4.2.3 Information theoretic (Kolmogorov complexity)

5. Project outcome and evaluation 12

1. Introduction

Extracting information from written text is a challenging task. In the past, most research was based on the final output, losing all the information produced during the writing process itself [1]. Only limited information was available on the development of the text over the course of a writing session, and most research was heavily speculative in nature [2]. Electronic writing allows to overcome difficulties associated with collection of this data through the use of appropriate software. One kind of data that is now more widely available is a *keystroke log* associated with the text that records the exact timestamps of every key press that happened in the editor [3] [4]. This information can be used to explore the cognitive processes that lie behind writing, and can be valuable to future linguists, cognitive psychologists, etc [5].

In our project, we extract data from the raw keystroke log and convert it into a useful format. We compute metrics that summarize characteristic of the writing process by sentence. Finally, to assess the quality of the final product, we compute linguistical metrics using a tool developed by the Linguistics department, *CoCoGen* [6]. By exploring the relationships between keystroke and linguistical metrics, we obtain novel insights about the writing process and the cognitive process that lie behind it.

2. Project description

For this project, we built tools to analyze keystroke logs for large corpora. We then explore the texts produced during English classes at RWTH Aachen. Our aim is to find relationships between writing patterns as found in the keystroke logs and the resulting text, building on results of [4] and [7]. We believe our results are representative of writing patterns of other populations as well, and in the future the same process can be run on more data to explore similarities and differences.

2.1 Description of available data

Our database consists of 7,012 files and keystroke logs, produced over four semesters by over 600 students at RWTH Aachen. The subjects were required to write between a weekly essay over the course of the semester, summarizing the content learned during the week. The essays were graded on a Pass/Fail basis, and 10 essays were required for successful completion of the course.

Students were instructed to use *Etherpad* [8], an open-source text editor, while composing their essays. This software automatically collected keystroke data that were later anonymized for research. The log was saved in compressed JSON format, recording information about every key press in a coded format (see example below).


```
{"text":"\n","attribs":"|1+1"}}},{"changeset":"Z:1>1*0+1$L","meta":
{"author":"a.SOsOoMqkqdc2tepz","timestamp":1524218645504}},{"changeset":"Z:
2>2=1*0+2$ea","meta":{"author":"a.SOsOoMqkqdc2tepz","timestamp":1524218646014}},
{"changeset":"Z:4>4=3*0+4$rnin","meta":{"author":"a.SOsOoMqkqdc2tepz","timestamp":
1524218646529}},{"changeset":"Z:8>2=7*0+2$g ","meta":
{"author":"a.SOsOoMqkqdc2tepz","timestamp":1524218647043}},
{"changeset":"Z:a>1=9*0+1$J","meta":{"author":"a.SOsOoMqkqdc2tepz","timestamp":
1524218647545}},{"changeset":"Z:b>3=a*0+3$our","meta":
```


Sample of a compressed JSON file.

2.2 Objectives of the analysis

Extracting data from the logs is not a straightforward task. We need to use several tools and programming languages to convert information between formats and design tools for cleaning and standardizing the output. An additional obstacle is the size of our dataset: certain algorithms would take several days to complete, and we need to find alternatives with a more efficient runtime.

Our first goal is to obtain measures of the cognitive effort to use in later analysis. Furthermore, we would like to distinguish between different aspects of complexity of a given text and correlate them with different metrics from the keystroke data. This is already common practice in the eye-tracking literature, where specific events are related with specific components of language understanding.

Everything will be written with reusability and extensibility in mind. Software should be properly documented, and data will be stored in the cleanest and most readily-usable format. This will allow others to more easily reproduce and build on the results of our research.

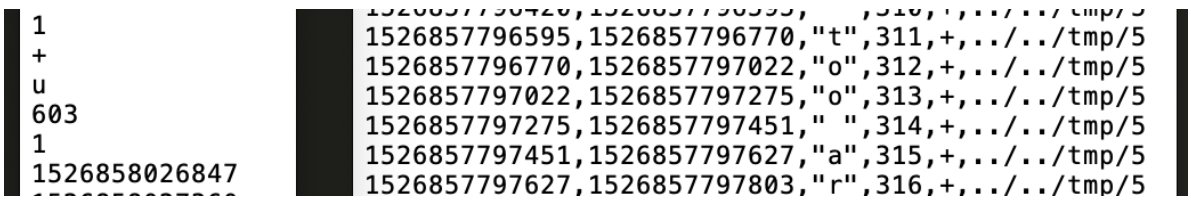
3. Pre-processing, cleaning and summarizing data

The first obstacle was transforming the ugly JSON format into the accessible .csv. This allows us to powerful Python libraries like Pandas and Matplotlib for data cleaning and visualization. We then run the CoCoGen tool to obtain different complexity scores for the final text. All source code in in the /src/ directory.

3.1 Transforming JSON into .csv

First, we build a node.js script that parses the JSON, throws away unnecessary information, and outputs the changesets in separate lines in a new file. This file is then run through a Java engine that uses the *Changeset* library provided by *Etherpad* to unpack the changeset.

One of the inconsistencies we had to face is that the Changeset library records the position at which the character was inserted in the current version of the text, which usually differs from the final position in the file. We wrote a Python script that fixed this issue by keeping track on the current state of the file from which the position of every character at the end could be derived. Unfortunately, being quadratic, this would have taken several days to run. We re-wrote a quicker C++ implementation, and in just under 10 hours our script was done. All of the results were accumulated in the keystroke.csv file for later use.



Start and end results of our node.js / C++ scripts.

3.2 Cleaning copy-pasted data

One of the challenges we faced was removing contaminated data. Some of the students did not follow instruction for proper use of the Etherpad application, and instead of typing the text directly into the editor, used a series of copy-and-pasting from other sources. This corrupted their keystroke log, and the Changeset library offered no convenient way to solve this issue, resulting in sentences written inconceivably quickly.

Our solution consisted of several filters and conditions that sentences had to satisfy in order not to be marked as copy-pasted and removed. These included being written with uneven spacings, or occasional deletions, along with other subtler criteria. We took this opportunity to also remove excessively short (< 5 words) sentences and corrected some mistakes by the Stanford-CoreNLP tokenizer esp. regarding bullet points, fragmented lists and paragraph breaks. This reduces the size of our dataset by around 40% but guarantees that all remaining data is not corrupted in any way.

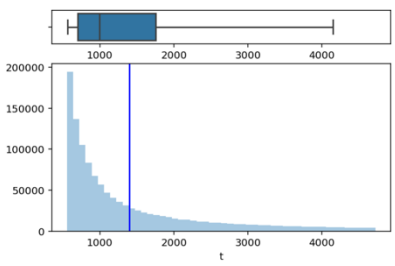
3.3 Computing summary statistics of our dataset

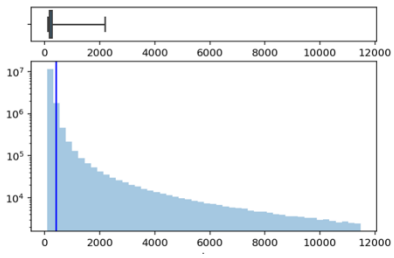
These preprocessing steps allowed us to import all our data in Python through the Pandas library. It now consists of 14,966,923 keystrokes, forming 1,831,361 words and 95,354 sentences over 3,466 text files. Each file was roughly 530 words in length and was written at a pace around ~20 words per minute. We encourage the reader to follow our analysis by importing the files in the /pads/ folder in their preferred software.

3.3.1 Keystroke speed and frequency

Typical values for time between keypresses (from now on referred as a *pause*) are 150 ~ 2000 ms. You can see pause distribution below (log scale on the right).

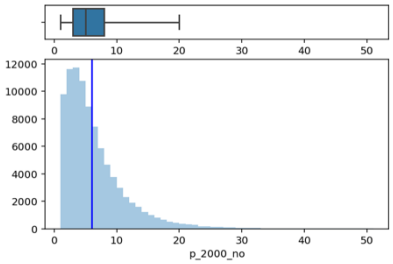
char	freq	speed	deleted
	14.41 %	424.3	16.493 %
e	9.63 %	255.6	14.730 %
t	6.67 %	386.2	14.232 %
a	6.01 %	361.3	14.818 %
i	5.27 %	352.1	14.840 %
s	5.11 %	342.3	15.492 %
n	5.08 %	265.7	14.107 %
o	4.81 %	351.0	14.772 %
r	4.60 %	282.1	15.000 %
h	3.31 %	294.0	15.650 %
l	2.82 %	315.2	13.924 %
c	2.60 %	405.7	14.152 %

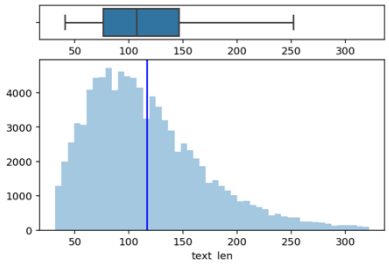


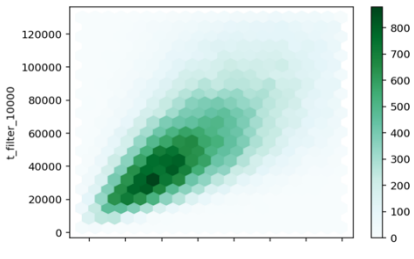


3.3.2 Sentence length and speed

The average sentence is ~120 characters long. Each contains around ~7 pauses longer than 2s. There is an almost linear relationship between time and length.







3.4 Running CoCoGen to obtain complexity scores

Our last step in the data collection process is obtaining linguistical metrics about the final text output the students wrote. We felt the *sentence* was the smallest unit we could analyze without excessive variance, and from now on we assume all metrics are computed per sentence, unless otherwise stated. To split our files, we used the Java implementation of the Stanford-CoreNLP library, that automatically tokenizes, lemmatizes and splits the text according to several criteria. We accumulated the outputs into a new sentences.csv file.

We requested access to CoCoGen, a tool developed by the Linguistics department here at RWTH Aachen for generating complexity scores for a text based on a sliding window technique [6] [9]. We ran it on all our samples to obtain more than 30 different complexity scores for each sentence. This took approximately 4 hours, and provided us with a wide array of Syntactical, Morphological, Lexical, and Computational Complexity scores ready for analysis.

`Syntactic.ClausesPerSentence_value`	No. of clauses per sentence	Typically, 1 ~ 5
`Lexical.Sophistication.BNC_value`	Average frequency of words in the British National Corpus	Typically, 0.2 ~ 0.8
`Lexical.Diversity.TTR_value`	Type-Token ratio	Typically, 0.6 ~ 1
`KolmogorovDeflate_value`	Information theoretical metric: how compressible is the text	Typically, 0.8 ~ 1.6

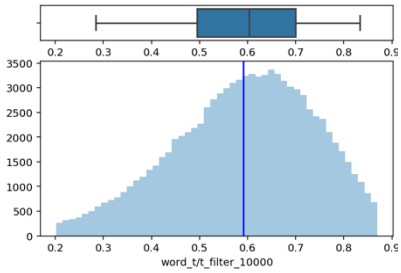
Some of the linguistical measures available to us from CoCoGen along with typical values.

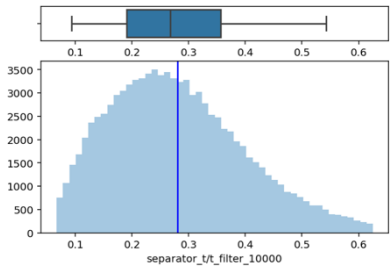
4. Data analysis and interpretation

We defined a set of metrics from the keystroke set and compared those to select CoCoGen linguistic complexity scores.

4.1 Gathering metrics from keystroke data

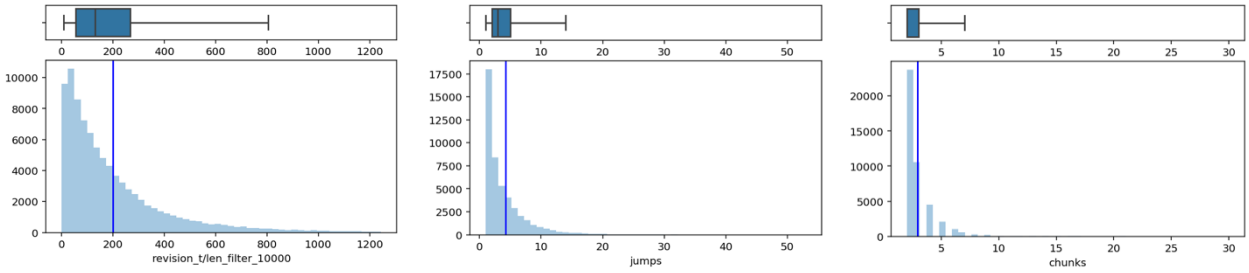
Apart from the obvious 'time spent' and 'no. pause above x' class of metrics, we chose to differentiate between time spent on words themselves vs. time spent on *separators* between words. These indicate what percentage of the total time[1] (and thus what proportion of the cognitive effort) was devoted to words themselves vs. structuring the sentence as a whole.





[1] Whenever a metric specifies _filter_10000 or variants, pause above 10s are reduced to 10s. This is to reduce the effect of outliers in sensitive metrics. For instance, the user might be taking a break or using other software.

We gather information on what percentage of time is spent revising the text, or rewriting text that was already written. Additionally, we keep track of how many times the user moves to a different part of the sentence, perhaps changing a word that was written earlier (*jumps* from now on), and how many times the user moves to a different sentence altogether before going back to the current one (*chunks*).



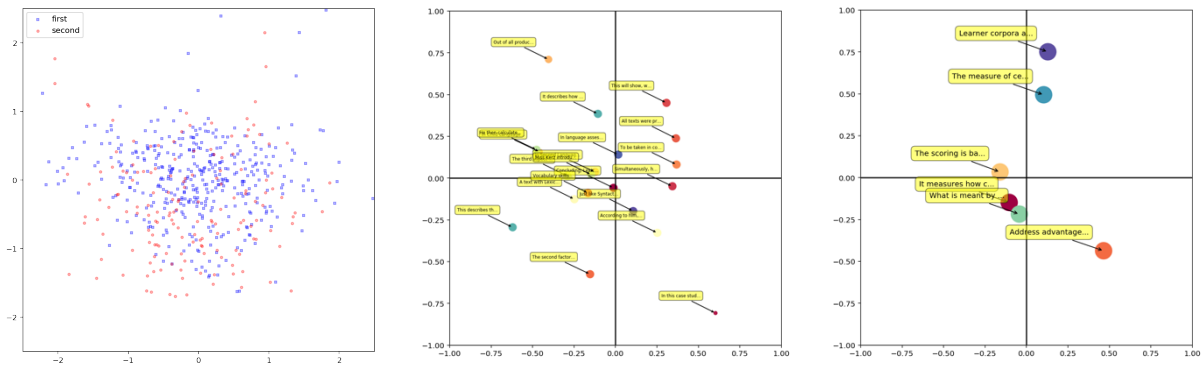
4.1.1 Defining Fluency and Latency

'Time spent' and 'pauses longer than x' metrics are not adequate to capture the full distribution of pause length. A sentence written at a constant pace, one with a long break at the beginning, and one with multiple stops are not easily distinguishable. We define two new metrics, Fluency and Latency, to capture this range of possibilities.

First, take the natural logarithm of the time associated with each keystroke to suppress outliers. We group them by user and normalize them to suppress differences in typing speed between students. We call these values `adjusted_log_t`.

We define Fluency of a sentence to be the average of its `adjusted_log_ts`. This is similar to the median: few long pauses have little effect compared to the frequent, short ones. We define Latency to be the sum of (1/2^n) `adjusted_log_ts`, sorted by length. This gives massive weight to longer breaks, and disregards shorter ones.

The strength of these metrics is that they are both independent of the length of the sentence, and the average typing speed of the user who wrote it. A text at a constant, brisk pace with a couple long breaks will have higher fluency that one with no long breaks but slower average pace. A sentence with a single long break and a brisk pace overall will have longer latency that a sentence with no long breaks but slower average pace. We can plot them on the x-y plane as below:



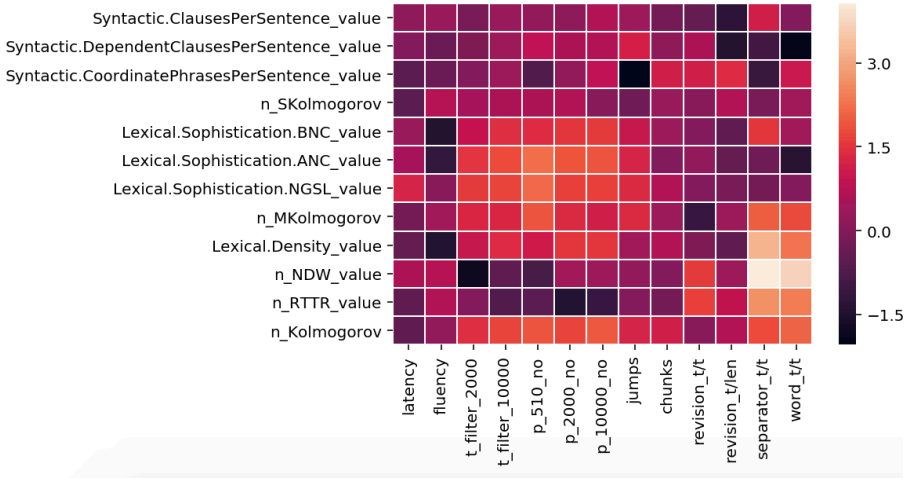
Left, x-y = fluency-latency plots for sentences written by two different users. Center, for 20 selected sentences. Right, for six sentences that were written multiple times (circles contain >50% of the occurrences). Similar sentences tend to have similar positions on the fluency-latency plane.

4.2 Relating linguistic complexity with keystroke metrics

We are now ready to find relationships between keystroke metrics and the linguistic complexity of the sentences produced. Since most of our metrics are heavily dependent on the length of the text, and the typing/linguistic ability of the writer, we choose to adopt a unique strategy for standardization.

First, fix two different criteria that separate sentences with different linguistic complexity values. For instance, we might choose to separate sentences with only one clause from sentences with multiple clauses. In the case where only a numerical metric is specified, we choose to compare the top 20% with the bottom 20% of the sentences with respect to that metric. Then, we try to find pair of sentences of the same length and written by the same user that belong to the two different categories. This allows to construct two unbiased pools for comparing keystroke metrics.

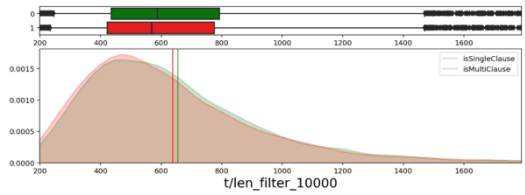
We develop an automated routine for generating these pools and comparing metrics. It generally runs in ~10 seconds and produces two pools of 3000~10000 sentences depending on the criteria utilized. This allows future researchers to easily replicate and build on our conclusions.



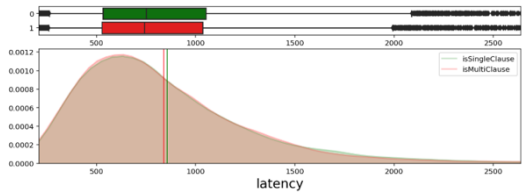
A summary of the correlation between selected keystroke and linguistic metrics. A brighter color indicates that a given keystroke metric is a better predictor of a given linguistic complexity score.

4.2.1 Number of clauses

We compare sentences composed of a single clause with sentences made of multiple clauses. We find that for a given user and length, single-clause sentences take longer to write ($p = 2 \times 10^{-9}$). This might be counterintuitive at first glance, but it is often more challenging to write a long, continuous sentence rather than breaking it up in multiple chunks.

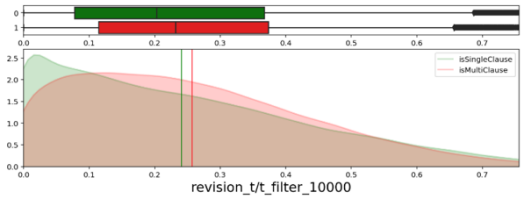



```
Sample size:  13941    Mean diff:  16.3281
stds:  0.0496    p-value:  2.32270E-09
```

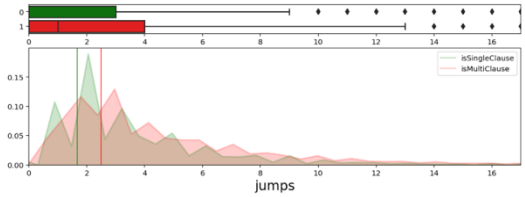



```
Sample size:  13941    Mean diff:  16.8461
stds:  0.0358    p-value:  0.000012
```

If we remove the constraint on sentence length, multi-clause sentences have higher latency (but still higher fluency). This presumably indicates a larger block of time devoted to thinking about sentence structure, along with less time spent overall on single words. As far as secondary measures go, multi-clause sentences have a higher proportion of time spent revising and on separators. This, along with the difference in number of jumps and chunks, very strongly suggest that when revising after the sentence is completed for the first time writers are turning single-clause sentences into multi-clause ones. From our data, it is unclear whether writers happen to go through the same process in the middle of writing a sentence as well.




```
Sample size:  40296    Mean diff:  0.0163
stds:  0.0876    p-value:  1.48570E-69
```

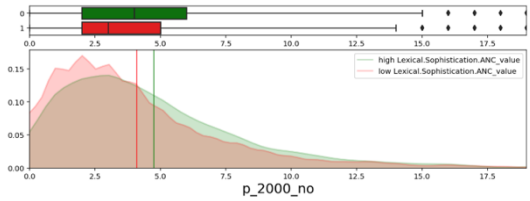



```
Sample size:  40296    Mean diff:  0.8306
stds:  0.2549    p-value:  0.00000E+00
```

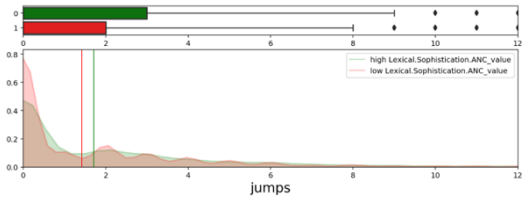
As a final note, there is no appreciable difference in typing speed between subordinate and coordinate clauses. However, subordinate clauses require a higher proportion of revision and separator time, have more internal jumps, and a higher number of pauses above 5 seconds. Intuitively, it's easier to write a correlative 'and' without stopping than a more complex subordinate preposition.

4.2.2 Lexical sophistication and diversity

We now explore the differences between sentences with high and low sophistication scores, as computed by CoCoGen over the American National Corpus. The is no appreciable differences in fluency or latency or typing speed between the two groups. However, sentences with more sophisticated lexicon have a much greater number of pauses >2s. This implies the writers often needs to stop before writing a relatively infrequent word. The lower revision percentage suggests students avoid replacing uncommon words after investing the effort to write them in the first place. The higher number of jumps and chunks suggests that people often do go back to replace easier words with sophisticated ones.




```
Sample size:  4235    Mean diff:  0.6567
stds:  0.1718   p-value:  2.55498E-29
```



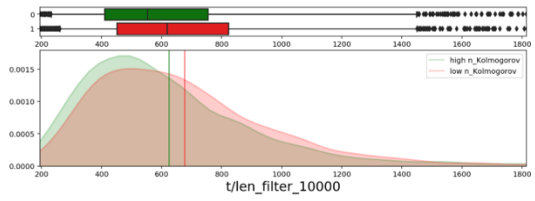

```
Sample size:  4235    Mean diff:  0.2796
stds:  0.1072   p-value:  1.49795E-12
```

When evaluated on the British National Corpus instead, discrepancies between sentence groups are lower. This suggests that the population sample is more familiar with uncommon words from the BNC rather than the ANC. For the analysis of other sophistication and density metrics, please refer to section 6 of our Jupyter Notebook in the /src/py/ folder.

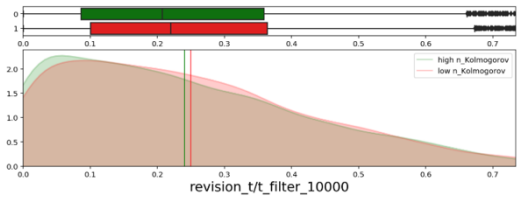
4.2.3 Information theoretic (Kolmogorov complexity)

All measures of linguistic complexity analyzed so far rely on categories and notions invented by humans. Recently, new measures have been borrowed from information theory to measure the density of information contained in a given sentence. One of these is the Kolmogorov Deflate, which roughly measure by what factor the given text can be compressed through standard algorithms that retain the full information. The theory is that more compressible texts contain less information and are simpler in nature. From now on, we will use the implementation provided by CoCoGen and compare the top and bottom 20%, after normalizing by writer and length as usual.

On top of being an excellent predictor of traditional complexity scores such as sophistication, diversity, more clauses and verbs and longer words, Kolmogorov Deflate is also correlated with most of our keystroke-derived measures.




```
Sample size:  3711    Mean diff:  52.1265
stds:  0.1618   p-value:  3.24403E-23
```

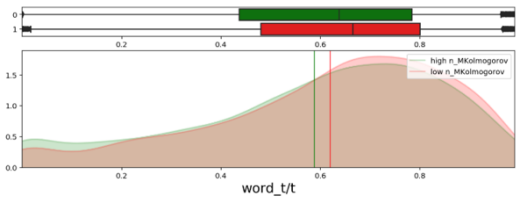



```
Sample size:  3711    Mean diff:  0.0090
stds:  0.0486   p-value:  0.001549
```

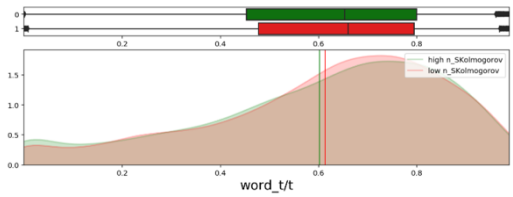
At first glance, it might be difficult to pinpoint exactly what factors are responsible for high Kolmogorov Deflate, precisely because its value is automatically generated by a computer. However, certain tricks allow us to differentiate, at least in part, between different components of information theoretical complexity. By randomly removing a fraction of the characters in the text, we are able to isolate a measure of of Morphological complexity, while randomly removing words evaluates Syntactic complexity.

Comparing their values with keystroke metrics available to us, we found this trick very successful at isolating different components of language complexity. Syntactic complexity is a better predictor of fluency, typing speed and revision frequency, indicating repeated emphasis on limited portions of text (likely, words). Morphological

complexity, on the other hand, is a better predictor of latency, jumps and separator time, indicating more time spent on structural components of the sentence.




```
Sample size:  3980    Mean diff:  0.0314
stds:  0.1259   p-value:  1.00574E-15
```




```
Sample size:  4057    Mean diff:  0.0116
stds:  0.0468   p-value:  0.001441
```

Morphological Deflate is a much better predictor of the percentage of time spent on words.

5. Project outcome and evaluation

Our team successfully preprocessed and cleaned the available data and into a convenient format for statistical analysis. We developed several keystroke metrics, and provided a convenient framework for defining new metrics on the dataset in the future. We explored and found intuitive and statistically significant relationships between out metrics and complexity measures. Our project successfully showcases the predictive power of data available from keystroke logs, along with the validity of measures unique to CoCoGen such as the Kolmogorov Deflate.

This experience allowed me to explore previously unknown NLP tools such as the Stanford-CoreNLP and taught how to build a quick and reliable pipeline for data processing and cleaning. I had a chance to refine my knowledge of statistical and graphical tools in Python, picking up on some new features of the most popular data science libraries along the way. The most interesting challenge was extracting information out of keystroke logs, a format of data previously unknown to me. Collaboration between us and the supervisors was extremely helpful for thinking about pathways and implementation strategies.

Due to time constrains, we were not able to investigate many potentially interesting facets of our data. Possible avenues for future research include clustering users based on their writing patterns, linking distribution of pauses and bursts with syntactic features, and predictive analysis of CoCoGen/keystroke contours over entire files.

We encourage the interested reader to explore our Notebooks and READMEs for further analysis, suggestions and implementation helpers. Please ask Marcus for access to our repository on RWTH GitLab.

References

[1] Matsuhashi, A. (1982). Explorations in real-time production of written discourse. In M. Nystrand (Ed.), What writers know: The language, process, and structure of written discourse (pp. 269-290). New York, NY: Academic Press.

[2] Matsuhashi, A. (1982). Explorations in real-time production of written discourse. In M. Nystrand (Ed.), What writers know: The language, process, and structure of written discourse (pp. 269-290). New York, NY: Academic Press.

[3] Wengelin, A. (2006). Examining pauses in writing: Theories, methods and empirical data. In K. P. H. Sullivan & E. Lindgren (Eds.), Computer key-stroke logging and writing: Methods and applications (Vol. 18, pp. 107-130). Oxford, UK: Elsevier.

[4] Leijten, Mariëlle, and Luuk Van Waes (2013). “Keystroke Logging in Writing Research: Using Inputlog to Analyze and Visualize Writing Processes.” Written Communication, vol. 30, no. 3, pp. 358–392.

[5] Berninger, V. W. (Ed.). (2012). Past, present, and future contributions of cognitive writing research to cognitive psychology. New York, NY: Psychology Press.

[6] Marcus Stroebel, Elma Kerz, Daniel Wiechmann, and Stella Neumann (2016). Cocogen-complexity contour generator: Automatic assessment of linguistic complexity using a sliding-window technique. In Proceedings of the Workshop on Computational Linguistics for Linguistic Complexity (CL4LC), pages 23–31.

[7] Alvès, R. A., Castro, S. L., de Sousa, L., & Strömqvist, S. (2007). Influence of typing skill on pause-execution cycles in written composition. In M. Torrance, D. Galbraith & L. Van Waes (Eds.), Recents developpements in writing-process research (Vol. 20, pp. 55-65). Dordrecht, Netherlands: Kluwer.

[8] https://etherpad.org/

[9] Ströbel, Marcus (2014). Tracking complexity of l2 academic texts: A sliding-window approach. Master’s thesis. RWTH Aachen University.