Classification using data compression

Lab Work 2

Bruno Nunes | 80614 | 7%
Catarina Marques | 81382 | 1%
David Raposo | 93395 | 2%
Gonçalo Machado | 98359 | 90%

Introduction

In the field of Algorithmic Information Theory, a pertinent challenge is discerning whether a particular text has been rewritten by ChatGPT or not. This assignment addresses this problem by developing a program that classifies a given text as either rewritten or not rewritten by ChatGPT, based on its statistical features.

The concept of representing original data using a smaller set of features aligns closely with lossy data compression principles. Therefore, this project explores an information-theoretic approach to classification, leveraging data compression techniques to directly assess the similarity between files, thus bypassing the need for separate feature extraction stages.

To accurately classify a given text, the developed program will utilize finite-context models to represent statistical information and compute the similarity between target text and reference texts representing both rewritten and not rewritten categories.

Methodology and Algorithm

This assignment we were tasked with creating a single program that would take text written by humans, text written by ChatGPT and a target text. The program would then create two finite-context models, one using the human text as basis and the other using the AI text as basis. Finally, it would compress the target text using each of the models and classify the text as belonging to the class that better compressed the target text.

To increase performance, speed up the retrieval of the results and to better develop and test the overall functionality, we decided to split the main program into two programs: the **fcm** (from finite-context model) and the **was_chatted**. The objective project is to create finite-context models and save them to a file. The **was_chatted** is built so that instead of needing to read the texts and create the models, it simply reads from files the models already created and compresses the target text.

A finite-context model, also called a discrete time Markov chain, is used in lossless data compression to represent data dependencies. It works by collecting counters that represent the number of times that each symbol occurs after each context. The context of order **k** is a window composed of **k** symbols. In its essence, it is simply a table that in the first column has the window/context, and the others have the number of times a symbol appears after the context. To note that a symbol is not only letters, but also numbers, punctuation and also white spaces.

This table is then used to predict the probability of a symbol appearing after the current context. This is done using the formula: $P(e|c) \approx N(e|c)/ s \in \Sigma\, N(s|c)$, which is the frequency that the symbol *e* appears after the context *c* divided by the sum of the frequency of every symbol *s* after the context *c*.

However, since the frequency of a symbol that exists in the universe but was not seen after a context is 0, it will lead to a probability of 0, which when compressing means the number of bits needed to represent the symbol are infinite, a value **alpha** is used, with the formula becoming $P(e|c) \approx N(e|c) * alpha \,/\, s \in \Sigma\, N(s|c) * alpha.$

FCM (Finite-Context Model)

This program has the task of taking a file with text, creating a finite-context model and saving it to a file.

The developed program takes the following arguments:

- **f** - The name of the file containing the texts
- **k** - The order of the finite-context model
- **t** - The type of the model (H for human and A for AI)

The program starts by reading and extracting the values of the values, using the defaults when the flags are not provided with the exception of the **f** flag. It then initializes the unordered map where the contexts of order **k**, as well as every symbol that appears after a context and the frequency with which it appears will be saved.

After it starts reading the file character by character in a loop performing the following actions:

- Check if the length of the context is the same as the order of the model, and if so, it will increment the frequency of the character appearing after the context.
- Update the context to have the last **k** characters

Finally, it saves the unordered map to a file with a name composed of the order of the model, the type of the model and the name of the file that was used to create the model. An example is ModelType_H_K_4_human_text.txt, where the type is H for human, the order is 4 and the filename is human_text.

The text inside this file obeys the following rules:

- The first line is the type of model (H or A).
- The second line is the order of the model.
- The third line is the number of contexts that exist in the model.
- For each context:
    - A line with the context.
    - A line with the number of symbols and frequencies that appear after the context.
    - For each symbol:
        - A line with the symbol.
        - A line with the frequency.

An extract of the start of the model in file ModelType_A_K_4_ai_train_010.txt is presented in Figure 1. In order to better explain this extract, we added text that can be identified by the starting characters **##**.

```
A                           ## Type of the model.
4                           ## Order of the model (k).
191187                      ## Number of contexts in the model
p yo                        ## Context
4                           ## Number of symbols that appear after the context
u                           ## Symbol that appears after the context
1658                        ## Frequency that the symbol appears after the context
c                           ## Another symbol
1
b
1
w
1
TXp                         ## Another context
1
t
1
ram                         ## Another context
44
a
357
…
```

Figure 1: Example of model file generated.

Was_chatted

This program has the task of compressing a text and classifying it as being written by humans or by AI.

The developed program takes the following arguments:

- **h** - The name of the file with the human-based model.
- **c** - The name of the file with the ai-based model.
- **t** - The name of the file with the text to be classified.
- **a** - The alpha.
- **k** - The order of the model.

This program starts by reading each model, the human-based model and the ai-based model, to an unordered map. In this process, the program checks if the files provided are of the correct type and have the correct order.

After, it reads the target file character by character in a loop performing the following actions:

- If the character is a newline or a tab, the character is skipped.
- If the context is the same size as the order of the models, do the following for both models:
    - Get the frequency of the character after the context.
    - Get the sum of the frequencies of every symbol that appears after the context.
    - Calculate the probability.
    - Calculate the number of bits needed to represent the probability.
    - Add to the total number of bits.

- Update the context to have the last **k** characters

Finally, the number of bits needed by both models to compress the text are compared, and whichever model needs the less bits to compress the text is the type that the text is classified as.

By “compressing” the text with both models at the same time we achieve smaller times of execution.

Datasets and Metrics

The assignment requires two types of text: written by humans or rewritten by ChatGPT. In order to obtain sufficient results to draw conclusions, we need a large number of texts from both types. For this reason, we used the dataset “AI_Human.csv” available on Kaggle.

The dataset is a 1.11GB CSV file with 487235 texts that contains 2 columns:

- **text**: Contains the text.
- **generated**: Indicates whether the text was generated by AI (1.0) or by humans (0.0).

Due to both time limitations and process capability limitations, smaller datasets were retrieved. These datasets are split into three categories:

- **train**: Used to create the models. Two kinds of datasets were created, one with human texts and another with ai texts. Each kind has 5 datasets, each with a different number of texts. The naming of the datasets is ##_train_???.txt were ## is the kind of dataset (human/ai) and ??? is the percentage of texts the dataset contains in relation to the number of texts of the same kind in the original dataset (002/004/006/008/010), which range from 2% to 10%.
- **test**: Used to obtain results. In order to properly evaluate and compare the results, 500 human texts and 500 ai texts were retrieved from the original dataset. These are not included in any train dataset. The naming of the files is ##_!!!!.txt where ## is the kind of file (human/ai) and !!!! is the index of the text in the original dataset.
- **smaller_test**: These are text files created from a small subset of files (10 human written files and 10 ai written) from the **test** category but with 50% or 25% of the text, creating 40 files. These files will be used to compare accuracy when the length of the target texts varies. The naming of the files is &&_==.txt where && is the percentage of text in relation to the original file (50/25) and == is the name of the original test file.

As for metrics to evaluate the results, we chose to focus only on the accuracy of the program regarding correctly classifying a text. In the **Results and Analysis** section, we will talk about 3 kinds of accuracy:

- **Human** : The accuracy of the program in classifying human written texts.
- **ChatGPT** : The accuracy of the program in classifying AI written texts.
- **Total**: The accuracy of the program in classifying texts. This will be the main accuracy compared between results.

Procedures

In order to evaluate the developed program, we performed a series of tests.

The range of input parameters was the following:

- **k** - 2,4,6,8.
- **alpha** - 1,10,100.

Before obtaining the results, we first needed to use the **fcm** program to create the files with the models with the different **k** values for all datasets of the **train** category. This resulted in the creation of 20 models for each kind of text, for a total of 40 models.

After, we used **was_chatted** and ran all 1000 test files using all combinations of the input parameters and models, which resulted in 60 different combinations and 60000 program executions.

Finally, we used the **was_chatted** and ran the **smaller_test** files with all combinations of the input parameters and models, which again is 60 different combinations and 2400 program executions.

Results and Analysis

This section will have the results regarding the project and their analysis.

Input Parameter Variation

As shown by graphs presented in Figure 2, a lower **alpha** value results in more stable outcomes across different **k** values. This stability diminishes as **alpha** increases, showing that the performance is sensitive to this parameter.

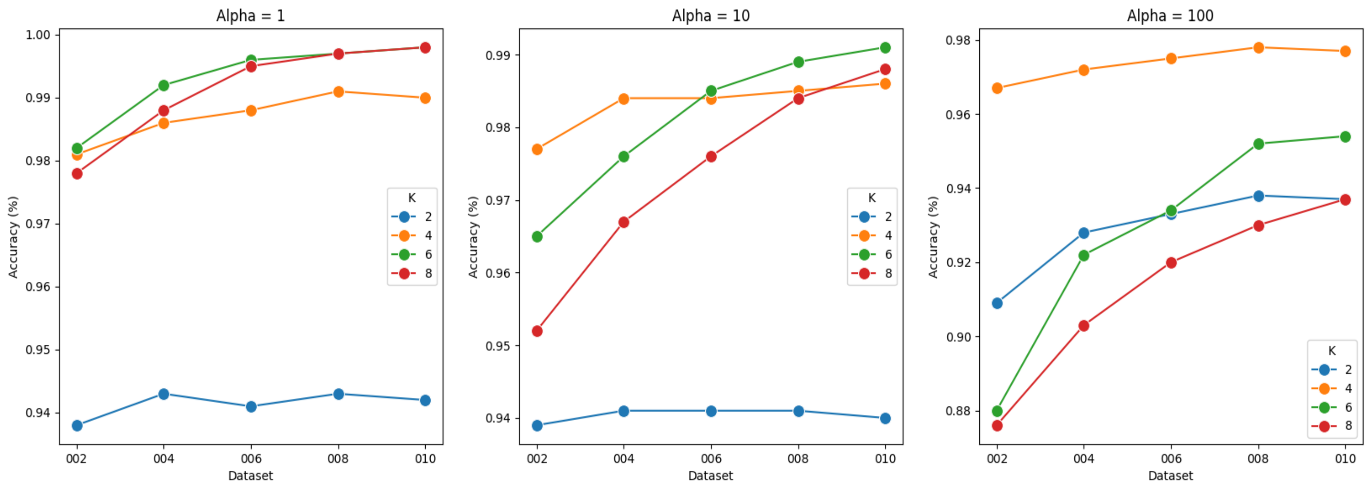
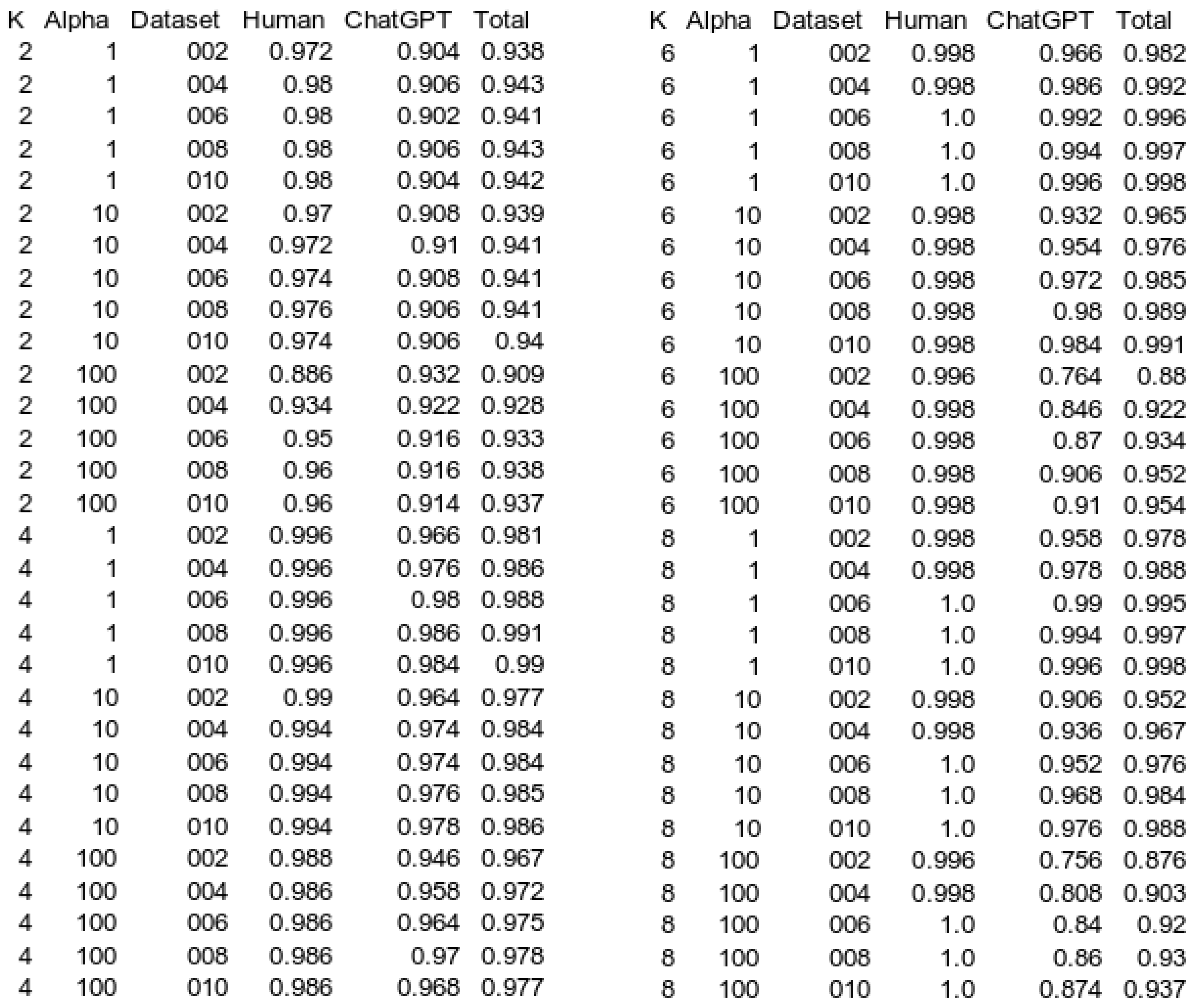


Figure 2: Graphical representations of results varying alpha, k and percentage of dataset sampling.

Also, lower **k** values generally show poorer performance. However, this trend reverses as **alpha** increases, indicating a complex interaction between these parameters. Specifically, with **alpha** set to 1, the best results are observed at **k=8** and **k=6**, while **k=2** shows the poorest outcomes. Conversely, at alpha 100, performance for **k=8** and **k=6** decreases, while it improves for **k=2**.

The proportion of training data significantly impacts **accuracy**. Larger datasets consistently enhance performance, emphasizing the importance of robust training sets.

The attached tables (Figure 3) further clarify these outcomes, distinctly highlighting the best and worst performance scenarios. The superior results are predominantly associated with higher **k** values, lower **alpha** levels, and richer datasets. For instance, configurations (6,1,010) and (8,1,010) achieved nearly perfect **accuracy** scores of 0.998. In contrast, the least favorable results occur with higher **alpha** values and poorer datasets, as seen with (6,100,002) and (8,100,002), where **accuracy** dropped to around 0.880 and 0.876, respectively.



K	Alpha	Dataset	Human	ChatGPT	Total
2	1	002	0.972	0.904	0.938
2	1	004	0.98	0.906	0.943
2	1	006	0.98	0.902	0.941
2	1	008	0.98	0.906	0.943
2	1	010	0.98	0.904	0.942
2	10	002	0.97	0.908	0.939
2	10	004	0.972	0.91	0.941
2	10	006	0.974	0.908	0.941
2	10	008	0.976	0.906	0.941
2	10	010	0.974	0.906	0.94
2	100	002	0.886	0.932	0.909
2	100	004	0.934	0.922	0.928
2	100	006	0.95	0.916	0.933
2	100	008	0.96	0.916	0.938
2	100	010	0.96	0.914	0.937
4	1	002	0.996	0.966	0.981
4	1	004	0.996	0.976	0.986
4	1	006	0.996	0.98	0.988
4	1	008	0.996	0.986	0.991
4	1	010	0.996	0.984	0.99
4	10	002	0.99	0.964	0.977
4	10	004	0.994	0.974	0.984
4	10	006	0.994	0.974	0.984
4	10	008	0.994	0.976	0.985
4	10	010	0.994	0.978	0.986
4	100	002	0.988	0.946	0.967
4	100	004	0.986	0.958	0.972
4	100	006	0.986	0.964	0.975
4	100	008	0.986	0.97	0.978
4	100	010	0.986	0.968	0.977

K	Alpha	Dataset	Human	ChatGPT	Total
6	1	002	0.998	0.966	0.982
6	1	004	0.998	0.986	0.992
6	1	006	1.0	0.992	0.996
6	1	008	1.0	0.994	0.997
6	1	010	1.0	0.996	0.998
6	10	002	0.998	0.932	0.965
6	10	004	0.998	0.954	0.976
6	10	006	0.998	0.972	0.985
6	10	008	0.998	0.98	0.989
6	10	010	0.998	0.984	0.991
6	100	002	0.996	0.764	0.88
6	100	004	0.998	0.846	0.922
6	100	006	0.998	0.87	0.934
6	100	008	0.998	0.906	0.952
6	100	010	0.998	0.91	0.954
8	1	002	0.998	0.958	0.978
8	1	004	0.998	0.978	0.988
8	1	006	1.0	0.99	0.995
8	1	008	1.0	0.994	0.997
8	1	010	1.0	0.996	0.998
8	10	002	0.998	0.906	0.952
8	10	004	0.998	0.936	0.967
8	10	006	1.0	0.952	0.976
8	10	008	1.0	0.968	0.984
8	10	010	1.0	0.976	0.988
8	100	002	0.996	0.756	0.876
8	100	004	0.998	0.808	0.903
8	100	006	1.0	0.84	0.92
8	100	008	1.0	0.86	0.93
8	100	010	1.0	0.874	0.937

Figure 3: Tables of configurations tested and results.

Specific to text origin detection, the optimal parameters for identifying human-generated text align with the observations mentioned above. The best settings for human text detection are consistently with lower **alpha** values and richer datasets. For AI-generated text, while following the same pattern, better outcomes are associated with larger training datasets.

There is a notable disparity in accuracy between detecting human and AI-generated texts. All tested configurations did not fall below 88.6% **accuracy** for human texts, while for ChatGPT texts it went as low as 75.6%. It also verified the existence of some configurations that allowed the correct identification of all text samples of human origin tested (Human accuracy = 1.0).

This difference may relate to the size of the datasets, with human text datasets being larger compared to those for AI-generated texts.

The remaining graphs showing the variation of the dataset and **alpha** for a given **k** and the variation of **k** and **alpha** for the same dataset were also generated, available in the annexes section (Figure 4 and Figure 5). Some of the conclusions presented can be better visualized in these graphs.

Target Text Length Variation

Initially, when thinking about how the models and the compression works, we thought that when using the same input parameters and using two target texts where one of them was a portion of the other, the accuracy for the smaller target texts would be lower.

After performing the tests with the **smaller_test** texts and comparing them with the results from the original **test** texts, we found out that the accuracy remained more or less equal (this comparison can be seen in a csv file named *small_results_accuracy.csv*), although when comparing the difference in bits needed for compression between the results for the original texts and smaller texts, we saw that the smaller the target text, the lower the difference was.

As such, we believe that the accuracy when using smaller target texts remained the same due to 2 reasons: not enough smaller target texts which can lead to results that do not correctly represent the reality and not using even smaller target texts.

Conclusion

The developed programs and the applied strategies helped us converge with the assumption that data compression allows to directly classify a target text, between human and AI-generated origins, by addressing similarity with a dataset made of human text samples and a dataset made of AI-generated text samples.

In our particular experience, the classification given by the developed program, supported by finite-context models generated, are concentrated in intervals of 88.6% to 100% for Human generated texts, and 75.6% to 99.6% for AI-generated text samples, on tests performed under 4 to 10 seconds, in accordance to the input data.

Moreover, it is made evident with the experiment that the richer the dataset used to generate the finite-context models, the better the accuracy achieved. This in alliance with greater order models (greater value of **k** factor) and a lower smoothing factor (lower value of **alpha** factor) is conducent to optimal results. In these conditions our program "**was_chatted**" can provide a global maximum accuracy of 99%.

Additionally, we can also state, even though with some reservation, that the accuracy of classification is target test sample size independent. However, we leave the caveat of the limitations of both the maximum possible size of the generated texts, limited by the possible use of ChatGPT, and the lack of more data to support the statement.

Future Work

For future work, we would firstly test with larger datasets, datasets with different kinds of text and a wider range of input parameters to reach a more certain conclusion about the accuracy of our programs. We would also test with smaller target texts in order to better understand the impact of target text length in the program accuracy.

In order to facilitate testing, we would change the code so that instead of a single value of alpha we could input an array of values, which would greatly speed up the **was_chatted** program without much increase in memory use.

We would also improve the overall use of memory of our program by researching if there is a better way to store and access the model instead of loading them to memory before performing the “compression”. This could come at the cost of some speed, but would allow for more parallel processes in the same machine.

We would also investigate if using threads in either/both **fcm** and **was_chatted** would increase the overall performance.

We would also add the option to create and use more than 2 models and customize what types of models we could create and use, allowing for a more customizable use of the programs.

Annexes

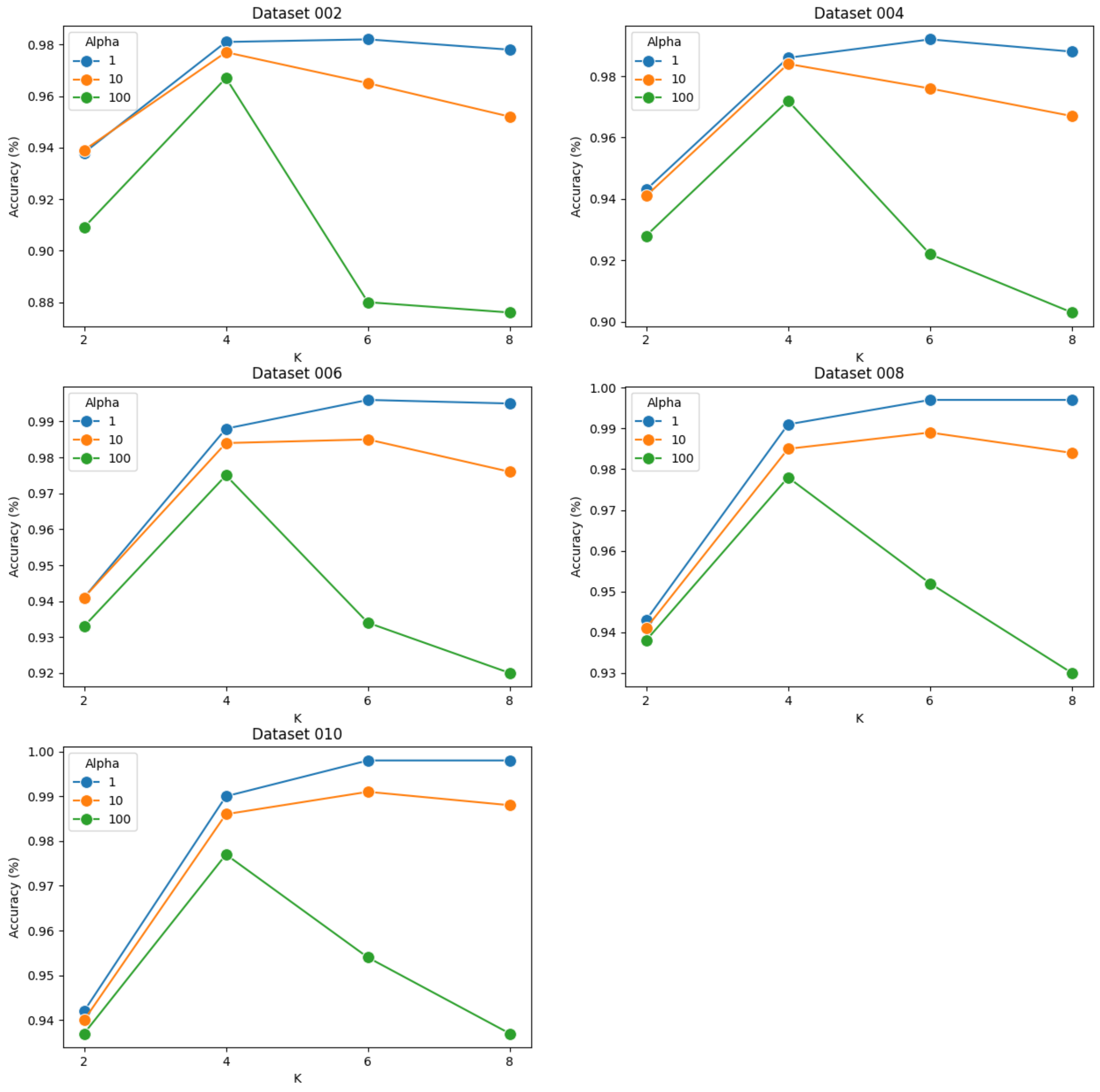


Figure 4: Graphs with the variation of **k** and **alpha** for the same dataset.

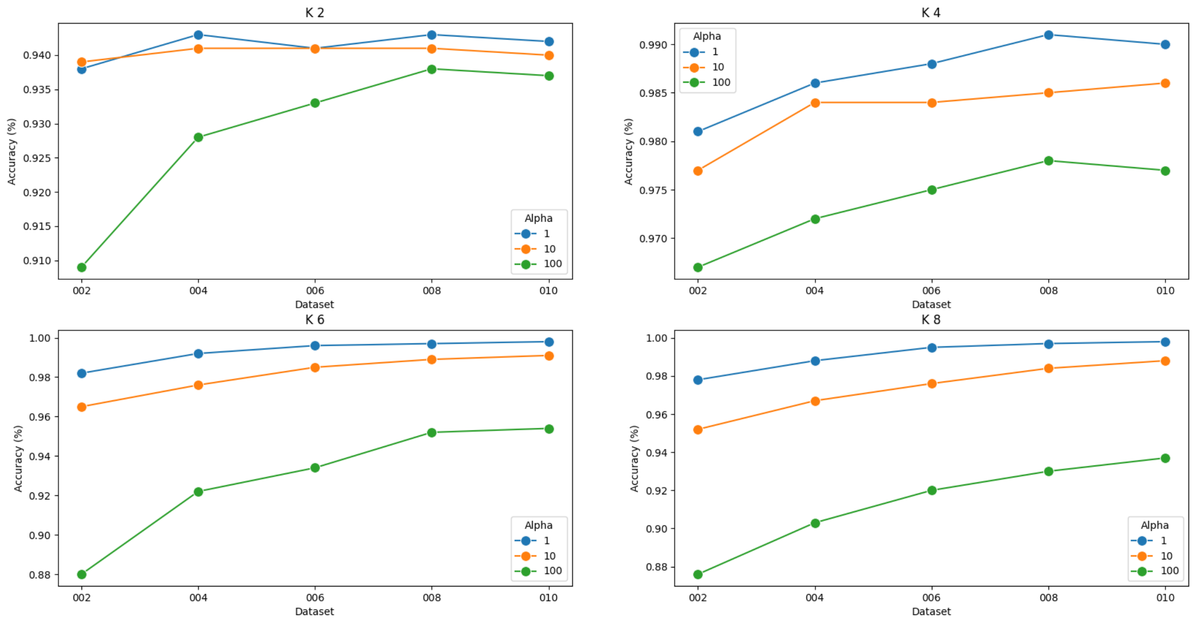


Figure 5: Graphs with the variation of the dataset and **alpha** for a given **k**.