Data Science Toolbox Project 3 - Topic Modelling with Password Data

Katarzyna Staniewicz, Sydney Vertigan, Junfan Huang

January 2, 2022

Introduction

In this task, our aim was to analyse one of the most important parts of the cyber-security world – plaintext passwords. A password usually contains some users' information such as their names, birthday, etc. So a rational thought is to mine these plaintext passwords and determine what the most common password looks like (for instance, is it a combination of the user's the name with his/her birthddate). In order to achieve this, our simulation is divided into 3 parts: topic exploration, password prediction, password security (password strength).

You can see how we planned the split in the mind map below.

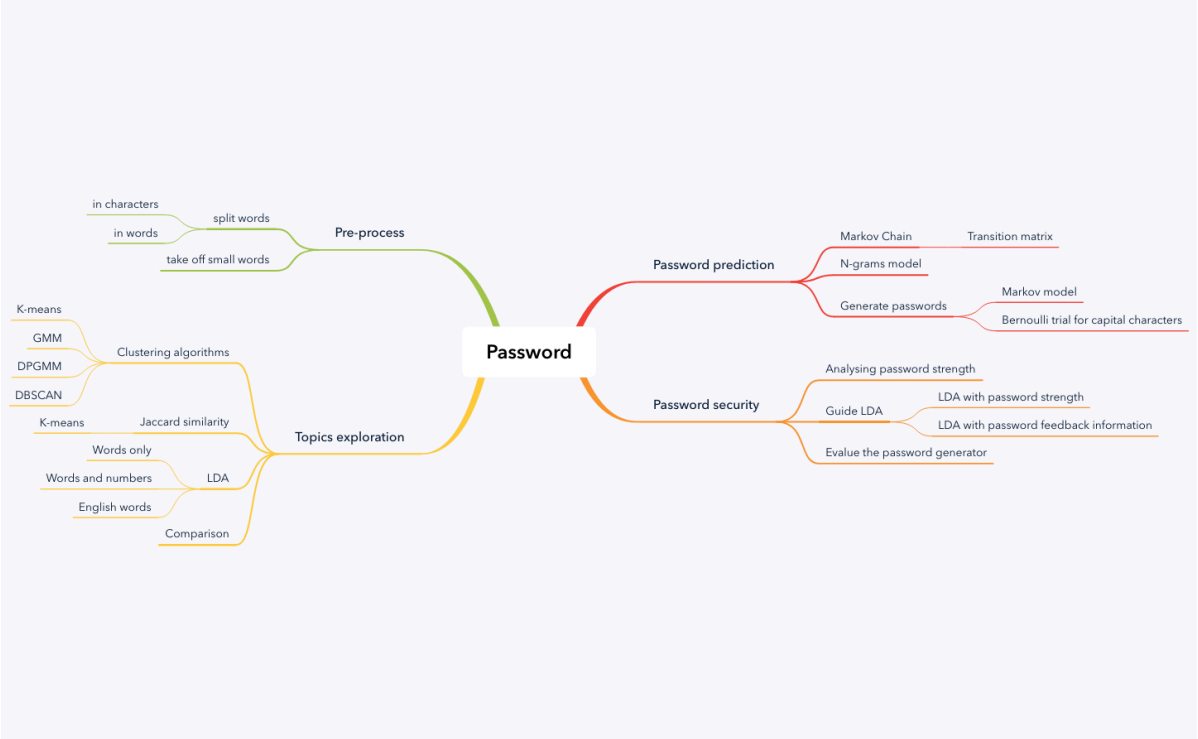

Figure 1: Plan by Mind Map

In a previously submitted work, we all felt as though we had only addressed what had been suggested in the project brief. This time, however, we not only look at the topics in our password data, but also look at how we can combine these topics with an analysis of password strength and maybe even predict passwords.

In the first part, we start with a baseline model of K-means to get a vague topic classification after obtaining the TF-IDF matrix. Then, we determine weak-points of the model. Next, we try different classification algorithms and compare them with K-means. From what we obtained, we are able to increase our performance. Then, we use another non-trivial algorithm, Latent Dirichlet Allocation (LDA), and provide a visualisation of the results. Additionally, we explore how to improve the performance of the LDA algorithm to get better results.

In the password prediction section, we build a Markov Chain for the password and predict the most likely words or characters following one another, given the observed plaintext. We also use a model to get a better result (more like a password).

Finally, we examine the password strength by using **zxcvbn** and trying to guide the LDA model to obtain reasonable topics. Moreover, we are able to evaluate our password generators by analysing the password strength.

The data we used is from a publicly accessible database that was created due to a password leak and therefore, the plaintext was brute forced in order to generate the passwords. This database can be found on the website https://github.com/duyetdev/bruteforce-database and contains 38,650 usernames and passwords from a film website https://sktorrent.eu.

The code for this project can be found at https://github.com/xihajun/data-science-toolbox-kate-syd-jun:

- ./report/project.ipynb contains the code for running our simulations and producing the figures in this report.
- ./documentation/ contains rough versions of these and provides evidence of our work.
- Project HTML version

We have chosen to split the equity as follows:

- Junfan 1/3
- Sydney 1/3
- Kate 1/3

Data Pre-processing

Data analysis

In order to made our algorithms run smoothly, it was important to to clean our password data. Passwords often consist of different characters, mixed with numbers and capital letters. In order to analyse these passwords, we split them into separate words, numbers and special characters (like *,?,/, or any punctuation). Afterwards, we can process the data in the three following ways:

- Splitting character by character;
- Splitting them into strings, numbers and special characters;
- Splitting them into separate words, numbers and special characters.

As will be discussed in the next few sections, the first case represents every character as a 'word' for the LDA algorithm and Markov Chain model. However, it is clear that this method will lose too much useful information, such as the meaning of plaintext, common combinations, and so on.

In order to retain as much information as possible, it seems reasonable to implement the second splitting method. Here, we focus on the length of every split and extract information about the ways in which characters can be combined in a password (*e.g. some passwords that combine numbers and strings belong to topic 1, while passwords only combining numbers belong to topic 2*). In this way, we are able to learn the password generating pattern and get the distribution of the password combination pattern. This helps to better understand the password structure.

For the last splitting method, our idea was to use clustering algorithms or methods in order to generate topics. We have focused on this idea throughout our topic modelling. Individual words obviously contain more information than the combination of word strings and also make topics more readable. However, as we found, you can use some relationships between the words in each passwords this way.

From above, we believe the best approach was to split the data into 'words'. Our definition of a word is as follows:

- Any real word (or what the function that we used considered to be a word)
- Any string of numbers e.g 2018
- Any string of special characters e.g. !"*£'?

In order to extract words from the passwords we first split the password strings. Our strategy was to split the strings whenever the "type" of the next character was different from the previous (the three types being: letters, numbers, special characters). Once we had split the password this way, we then used the function **segment** from the library **wordsegment**. This function split our strings of letters into separate words. Because we were unsure as to how accurate this function was, we tested it with many English words - for which the function performed very accurately. However, from our research, our collection of passwords seems to include many words in other languages and we are not sure how accurate the function performs in these cases.

Once we started to look at the results we were getting with our models, we decided to try a few different ways to further clean the data. These were all attempts to get more consistent topics. We removed the short words e.g. 'a', as we felt they would not make a difference to our overall topics because they have no real meaning. We also took out the numbers because in some cases they seemed to be confusing the topics. One last thing that we tried was to extract all the English words and let our models run on this set. The rationale for this approach was that English is a language that we all speak, so we could see whether our models were performing well and whether the words were clustered together correctly. This reduction was necessary for us to judge the performance of the algorithms because the whole data set contains many languages. The model would cluster similar words together but if they were in a language none of us knew, we would not be able to confirm the validity of the combination.

To summarise, we further cleaned the data and tested each of the 'cleaner' sets to see if our topic modelling performed better. We did this by:

- Removing small words, with only 1 or 2 characters
- Removing numbers
- Extracting words from only one language. We chose the English language.

Topic exploration

TF-IDF Matrix

Our initial idea for exploring password data was to get the importance of a word in a document. A well known method to reflect how important a word is to a document in a collection or corpus is to use TF-IDF matrix. This matrix is the product of these two statistics (*Term Frequency times In- verse Document Frequency*). [5] It is normally computed as follows:
Suppose we have a collection of N documents. Define f_{ij} to be the frequency (number of occurrences) of term (word) i in document j.
Then, define the term frequency TF_{ij} to be:

$$TF_{ij} = \frac{f_{ij}}{max_k f_{kj}}$$

The most frequent term in document j gets a TF of 1, and other terms get fractions as their term frequency for this document.

On the other hand, the IDF is defined as follows:
Suppose term i appears in n_i of the N documents in the collection.
Then,

$$IDF_i = log(\frac{N}{n_i})$$

So

$$tf - idf = TF_{ij} \times IDF_i$$

The terms with the highest TF-IDF score are often the terms that best characterize the topic of the document.

Using the TF-IDF matrix, we can run a slew of clustering algorithms to better understand the hidden structure within the passwords and also identify vague groups/topics.

K-Means

We used K-means clustering because we thought it could help us perform the classification automatically, after which we would be able to see the hidden structure in the passwords.

However, the size of the K-means clusters are unbalanced. There is a huge class dominating nearly 95% of the whole dataset. That is to say, the K-means algorithm could not appropriately divide the data into reasonable clusters. However, we still obtained some justifiable results, which are shown here as the K-means classes.

Table 1: K-means clusters with passwords

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
V76szXAoQ9	martinrosko	Spartak666	pipinka123	ferbman
mixvibes123	martincak	666fericko666	matulko123	manss123
PancharT1983	martin24	ekb666	matulko123	tescoman
bulldog	martin0842681246	casucci666	nana123	wjcman
nereknu9558285913	26martin1986	skull666	Acer*123	blindman

At first we inserted all the data in the algorithms, but all the numbers tend to be randomly allocated across the groups. We got better results when all the numbers were excluded. One can treat all numbers as one group/topic. Investigating the patterns of the numbers would be a separate task.

We could see when working without the numbers, the main feature of the K-means algorithm is that it clusters similar words together. The first cluster is the biggest one and it tends to group many random words. These words do not seem to have much in common and one can often see a cluster consisting of the words starting with the same letter. For instance, the first 20 words of the second cluster in our code are 'natal, kap, kamo, ink, zzz, jaro, jav, jason, jaso, jasmin, jas, jaroslav, jar mil, jarek, jard, jcj, jar, janus, janosik, janko'. Most of the words in this cluster start with 'ja'.

When we performed this algorithm on only the English words it became more clear as to how the algorithm was clustering. Many of the clusters were very similar, however you could see some things you would expect to be clustered together. Here is the top words in one of the 5 clusters: ['ice'], ['wow'], ['hell'], ['jack'], ['internet'], ['ink'], ['hop'], ['hip'], ['him'], ['her'], ['has'], ['fer'], ['had'], ['god'], ['genius'], ['fun'], ['fox'], ['forever'], ['for'].
From this we can see patterns we would expect - like 'him'and 'her' being clustered together. However, the overall topics still do not seem to make sense.

In light of this, we started to try some different algorithms that we thought could create better topics.

Comparison with other Clustering Algorithms

Here are three algorithms used for comparison.

- **Gaussian Mixture Model**

We wanted to compare K-Means with other clustering algorithms. We decided to use GMM as it is a lot more flexible, especially in terms of cluster covariance. This is because K-Means is actually a special case of GMM, where each cluster' s covariance along all dimensions approaches 0. [6] With GMM, each cluster can have unconstrained covariance structure, which is different from K-Means where each point gets assigned to the cluster closest to it.

- **Dirichlet Process with GMM**

  The benefit of a Dirichlet Process with Gaussian mixture model is that it obtains the number of clusters automatically. However, the speed of the algorithm depends on the size of data.

- **DBSCAN**

  DBSCAN stands for density-based spatial clustering of applications with noise. [2] We thought this might therefore be something that would work well as some passwords have a lot of noise in the form of random characters.

The results of these different algorithms are shown as follows.

Table 2: Other algorithms

plaintext	K-means	GMM	DPGMM	DBSCAN
V76szXAoQ9	0	1	4	0
milanek22	0	6	6	1
PancharT1983	0	1	4	0
pipinka123	3	6	6	2
bulldog	0	1	4	0

In general, different models learned different similarity of our passwords, but there was still a dominant class no matter which method we used.

Jaccard Distances with K-Means

Our original K-Means model was not clustering how we thought it would, or indeed, wanted it to. We thought this was because of how we defined the distances between two passwords. Therefore, we thought we could use a different type of distance measure that would make more sense to cluster similar passwords together. The Jaccard Index is used as a way of identifying the similarity between two sets. [4]. Simply put, it is the intersection divided by the union. The formula for the Jaccard Coefficient is:

$$J(A,B) = \frac{|\ A \cap B\ |}{|\ A \cup B\ |} \tag{1}$$

We used these coefficients to form our 'distance matrix'. From this distance matrix we used K Means in the same way as above.

It took far too long to run on all of the data - so, to see if it clustered our passwords better, we ran it on a section of our data.

From this section, we did a comparison with our K-means results. As shown, the Jaccard method clustered the more similar words together irrespective of whether they appear at the beginning of the passwords or at the end of the passwords.
The table seems to show that the Jaccard distances are able to obtain a better results (put them into one class) for different order combination. (Table on next page).

Table 3: Jaccard distances

Original K-means clusters	K-means clusters with Jaccard distances
3	nana123
3	loko123
3	jasov123
7	1234a5
3	Venda123

LDA

We used different methods to group the set of words. Ideally, all the elements of the group share common features. The Latent Dirichlet Allocation finds a suitable distribution to generate passwords which may make our topics more interpretable. We wanted to use this feature for the password analysis, as well as the password prediction. LDA is able to generalise to the model it uses to separate our words into topics. [3, 1]

In our data, every password is taken as a distinct document in LDA. Passwords consist of words coming from different topics. For simplicity, we assumed there are k topics (in practice we set $k = 5$). This is clearly not a correct assumption since there are more topics. We did not consider finding the exact number of topics essential to our project. The main point was to design a program that would group words with the same topic.

The LDA algorithm takes the lists of words as inputs and calculates relevant topics in the corpus, along with respective weights of words. This way, we can see how much each word contributes to the topic. Since there are too many words per topic and the words do not tend to reoccur, the weight of most words per topic is usually about 0.001.

Understanding LDA is not trivial. Below we explain what the mathematics of LDA in our case stands for:

- $\beta_{1:K}$ are the topics where each β_k is a distribution over the vocabulary for topic k. Since the words rarely reoccur it is close to a distribution.
- θ_d are the topic proportions for password d.
  Since the password contains at maximum only a couple of words.
- $\theta_{d,k}$ is the topic proportion for topic k in password d

  Therefore the $\sum_d \theta_{d,k}$ gives us the proportion of the topic k over all the passwords.
- z_d are the topic assignments for password d
- $z_{d,n}$ is the topic assignment for word n in password d
- w_d are the observed words for password d

$$p(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D}) = \Pi_{i=1}^{K} p(\beta_i) \Pi_{d=1}^{D} p(\theta_d) \Pi_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\beta_{1:K}, z_{d,n})$$

Once we got the LDA model, our goal was to obtain the parameters $\beta_1, ..., \beta_k$ and $\theta_1, ..., \theta_d$. Our understanding is based on the gibbs sampling, where the LDA updates the word in topics in every iteration through the probability. After convergence, we are able to obtain the topic-word matrix which is what we want to learn.

Here is what we learn from LDA algorithm by using **pyLDAvis**.(click for more details)

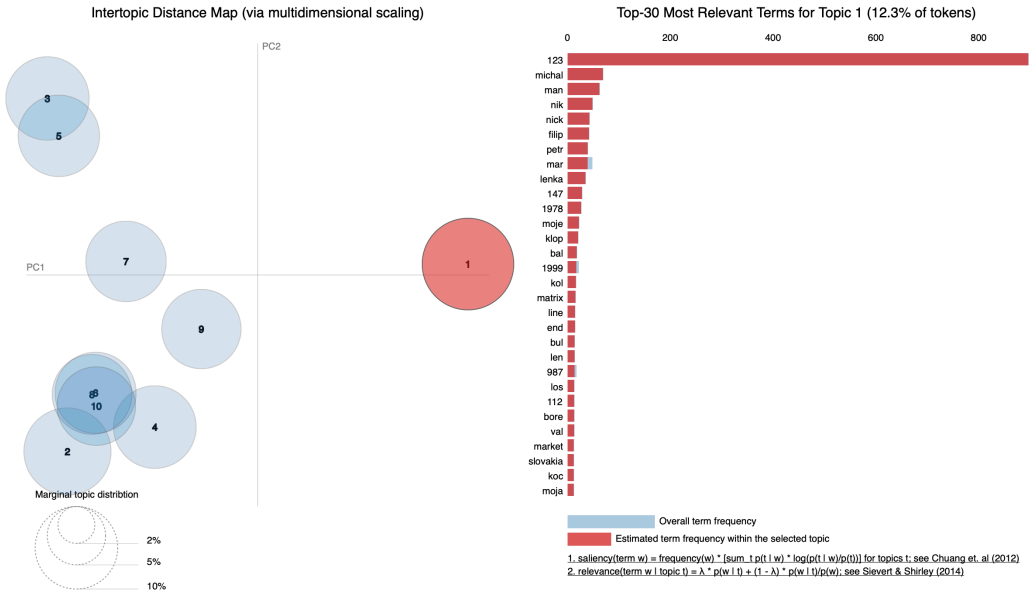

Figure 2: LDA results with numbers

To make more sense of the dataset, we tried to get rid of the number part and explored the pure string words. The picture below shows the LDA results without numbers.

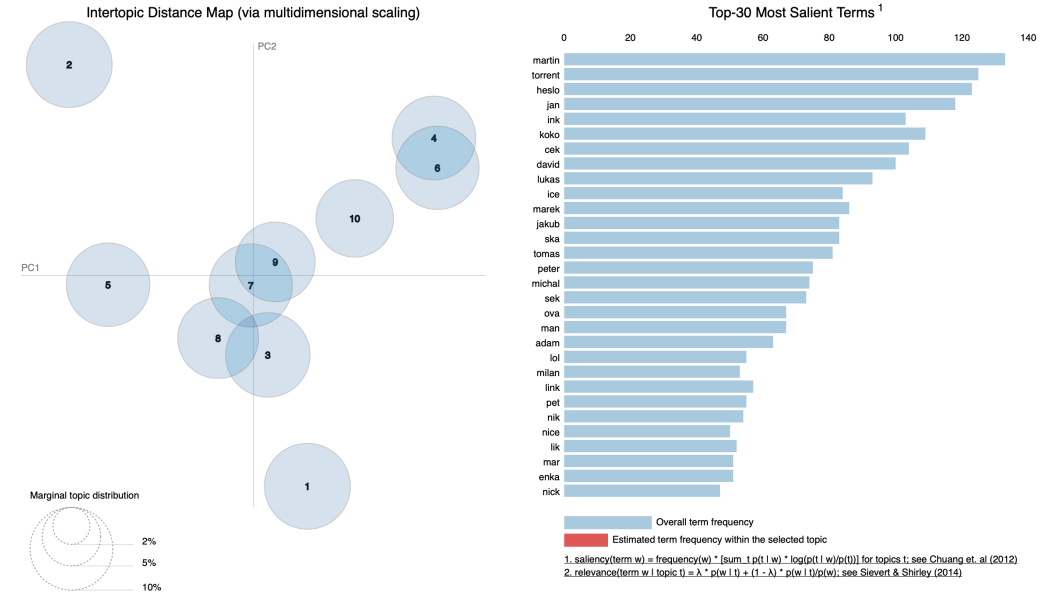

Figure 3: LDA results without numbers

(The interactive versions of both of the visualisations can be found on the Online Jupyter Notebook.) It is interesting that we can see the website name: torrent in the top 2 terms which provides us another way to learn the sources of the dataset.

We included the **WordClouds** for each of the 5 topics we created through LDA to get a better visualisation effect. Here, we used the data that had simply been split into 'words' and then had the short words and numbers removed since the numbers seemed to be randomly allocated all across the topics. To create the word clouds we used the library **wordcloud** and the function **WordCloud**.

Figure 4: LDA Topics' Word Clouds for words without numbers and short words

From the picture, we can see that every topic contains some words that we would consider similar. However, there are many words we do not understand. Perhaps they are not words in any language. Moreover, there are many words that have not much to do with each other. This is partly due to our unrealistic assumption of 5 topics. Topic number 3 contains many names. One of the most unexpected result was that a large number of passwords included words related to death. For instance in the 5th topic you see 'hell', 'death', 'sink', 'monster'. Many topics also contain words to do with the website name, e.g. 'torrent' or 'torr'.

The problem with LDA is that it only processes words in one language (English is probably the default since the majority of the words are English). In order to create multilingual topics, our model would need be much more complex, but we would not gain much as we do not fluently speak most other languages. (https://hal.archives-ouvertes.fr/tel-01706347/document) For this reason, we decided to check the LDA results when running only on the English words. The results are shown below.

Figure 5: LDA Topics' Word Clouds for English words, without numbers and short words

Here, we can make greater sense of some topics. For example, 'hip' and 'hop' tend to be clustered together. However, there are still seemingly unrelated words clustered in the same topic. Again, we think this is because we should have increased our number of topics. We attempted this but the run time seemed

to be incredibly long. The number of topics is not known, so we would have to run the algorithm for nearly as many times as there are number of words. Moreover, in every case we would have to analyse the results and this would not be efficient.

Password Prediction

In order to have a deep understanding of the password structure, we tried a number of methods to predict the password by some given sequences. The core idea is to calculate the probability of the next passwords, given the sequences we have which has a strong relationship with a Markov model.

Markov Chain

The models we have used above do not take into account the order in which the words appear. Therefore, word co-occurrence information is lost. We thought that a good way to model some of the relationships between the characters in a password would be to use a Markov chain, from one state to another. For our Markov chain, we look at how likely it is that a certain character comes after a previous character. For example, the Q_{ab} position of our transition matrix is the probability that a b would come after an a in any password.

To work out the transition rates, we used some mathematics we had learnt last term. This is: Assuming we have at least one transition from state i (which we have, as every letter is followed by another):

$$\widehat{Q}_{ij} = \frac{number\ of\ transitions\ from\ i\ to\ j}{number\ of\ transitions\ from\ i} \tag{2}$$

[7]

In order to do this we created a list of all possible characters and named this **alphabet**. We then had to introduce a stop character. In order not to infect our data with '.' and make the transition rates involving this character incorrect, we used the Chinese character for a full stop '停' as a stop status (to tell the function when to stop). We then used the formula above to calculate the transition matrix. From this we can see the relationships between characters and how likely one is to come after the last character.

We thought this would be an interesting way to try and 'crack a code'. For example, when guessing a password for this website you could use the transition matrix to do so. If we are looking at value i, we can first guess the next character to be the value j such that position j has the highest value for all of row i in Q.

We also created a password generator based on this Markov Chain. This means, following our intuition, these are weak passwords (if you had the transition matrix and you were using these probabilities in order to guess passwords).

Another idea that we had was to create 'strong passwords' with the least likely character to come next. Therefore for value i the next value would be j such that the j^{th} position in row i of Q has the smallest value (even if that is 0). However, this did not work the way in which we thought it would. For most rows the character j that gave the smallest transition rate would be /. This meant you would tend to get a cycle of / and then the character that had the smallest rate from there.

If we had more time we would try to find another way in which we could use the Markov Chain to generate strong passwords.

After we generated the Markov chain for characters, we did the same as before for the 'words' in each password. We found the transition matrix in the same way.

The time complexity of calculating the transition matrix was different for each version of the Markov Chain.

Bigrams Model

The idea of Bigrams model is similar to that of a Markov Chain, but only takes two adjacent elements into account. It considers the relationship between the last two statuses and next status, which is more realistic. Consequently, we can do a better prediction and get sequences more like passwords.

Now, we can calculate the probability of the new state given the observed passwords. We predict the next character by setting it to be the most likely one.

Password generation

Other than the password prediction, we can perform some more exciting tasks. Through our transition matrix, we are able generate passwords. The core idea is to have an initial random word, and through this word, generate the remaining sequences. As we already take the stop state into account, the algorithm stops automatically. Here are some generated passwords:

Table 4: Different password generators

Markov Chain(character)	Markov Chain(words)	Bigrams(character)	Bigrams(words)
jqgukbojkoAy	sexmail.0810	domo19870	buksatko12345hope
nmuvv123	stenlivierka	1xdra34772	anothercentricky
zfnpcmL1	ivtuvzlxot	skyhob15	1996palopalo132
stqkhbylerka	devondavidko228	52036ya	17101835
hi1wohv88887	alexandr123aaa166	hodikot681	popteamdjpecc

Table 4 shows the performance of different models. In particular, from left to right, with increasing model complexity, the plaintexts are more likely to be set as passwords. Furthermore, we still have some other ways to generate passwords. For example, as in the LDA generative model, once we learn the distributions of words in topics and topics in documents, we are able to generate passwords through these distributions. By first selecting the topics and then through this topic-words-'dice' we can generate words.

Password Security

Password strength

Password strength is a measure of the effectiveness of a password against guessing or brute-force attacks. A strong password usually contains a mixture of strings, numbers and capital characters. In contrast, a weak password usually uses more common, 'easy to remember' phrases like 123, abc and so on. In general, strong passwords usually consist of the weak passwords strings - just combined in a less intuitive way. In this section, we are going to explore the password strength for our dataset. By using our strength results, we can guide LDA to obtain a better results.

Topic Modelling Using Strength

Some of our topics did not make sense to us. We were unsure why the models had clustered certain words together. As an attempt to create topics that clustered in a way we would understand, we added in the strength of each password. This also allowed us to see which 'words' were in strong passwords and which words would be more likely to appear in weak ones. To start, we used **zxcvbn**, a good password analysing tool, in order to find how strong each password was. We then added this word (*strongest, strong, medium, weak, weakest*) onto our split up password. Then we could run LDA on our new documents again and see what would cluster together. We used data visualisation to make it easier to view the clusters. Below is

a screenshot of our visualisation and on our Jupyter report you can find the interactive version. (Online interactive version)

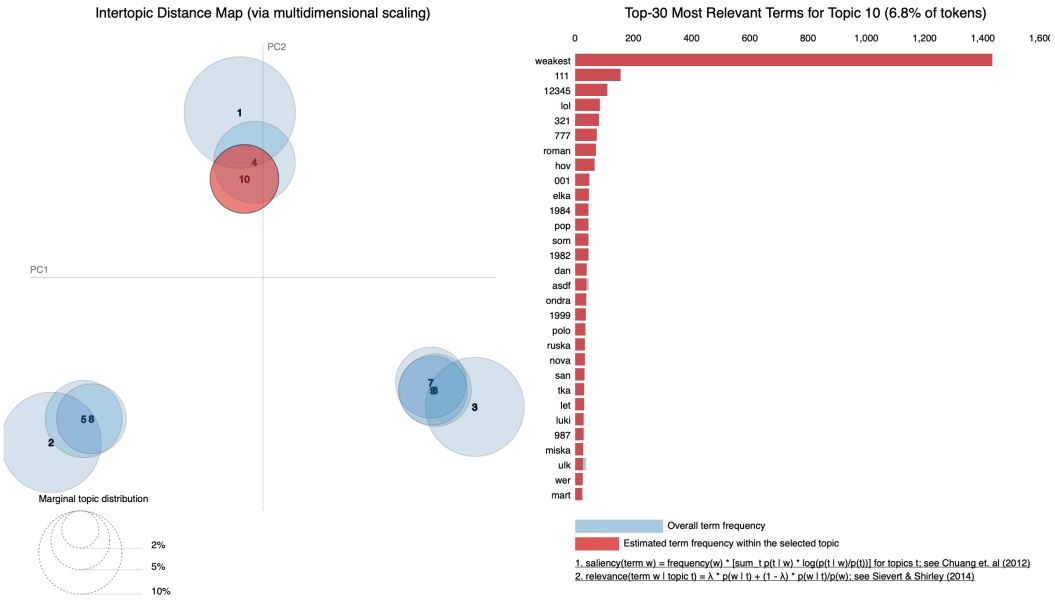

Figure 6: Guide LDA with password strength weakest labels

As seen here, in the weakest part, the topics contain a lot of easy combinations like 111, 12345 which makes more sense compared to the previous LDA results. In general, it is obvious that the graph is divided into 3 parts: weak, medium and strong parts respectively, although there is a weakest part near to the strong part (this could be due to some strong passwords being combinations of weak passwords). It shows that the text labels work. In that case, it is worth to add more features. **zxcvbn** provides a password feedback function, which does an analysis of the password and generates feedback about the context of the password. For example, feedback can consist of: containing too many repeat words, too short and so on. After doing this, we obtain LDA results as follows. (Online interactive version)

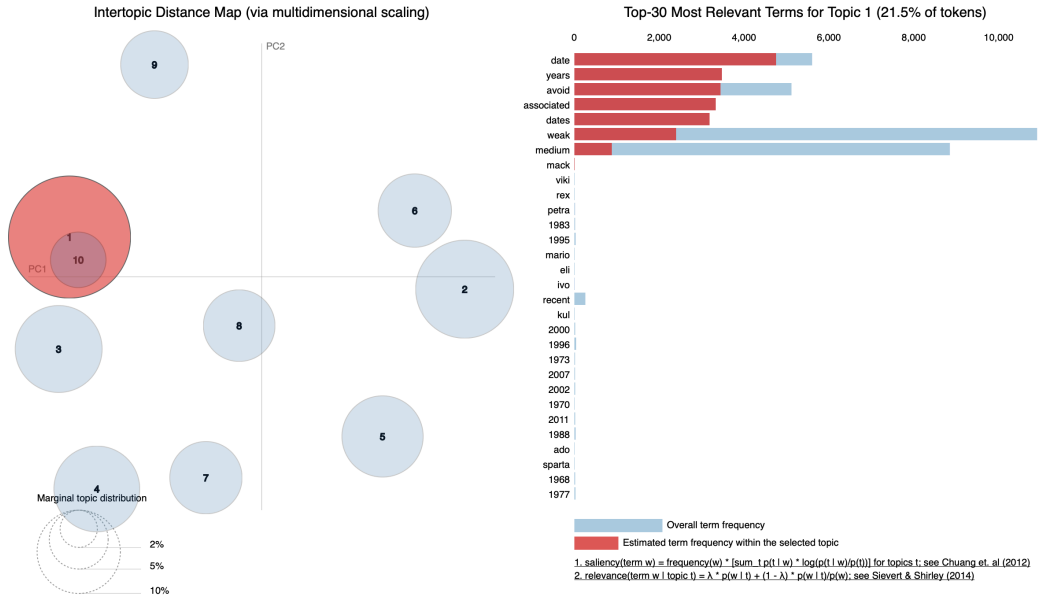

Figure 7: Guide LDA with password analysis - date

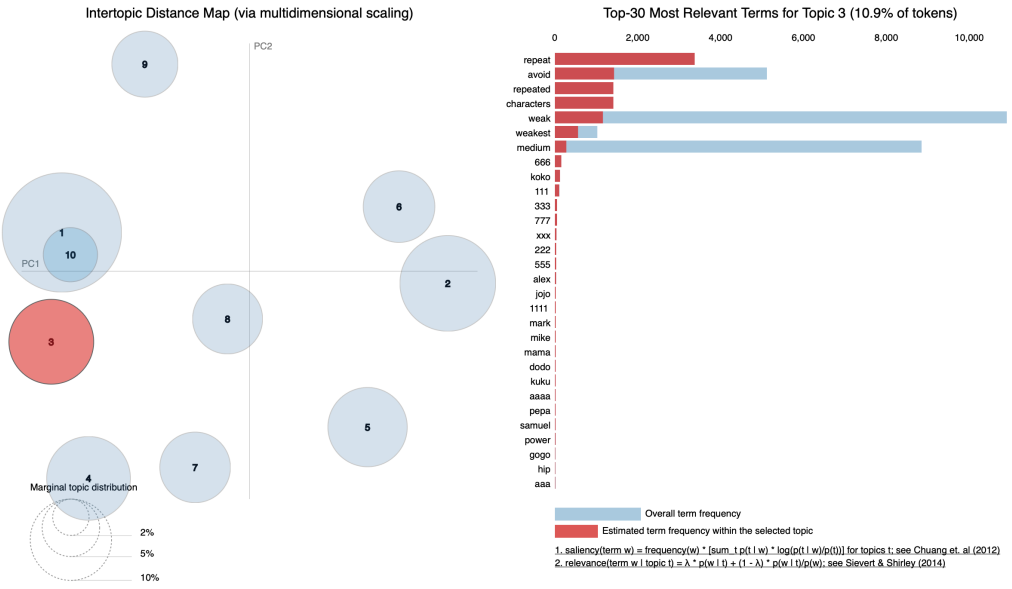

Figure 8: Guide LDA with password analysis - repeat

What is interesting about these clusters is that a lot of topics replicate what we would expect. For example, repeated values are in the same topic as 'weak' (e.g 3333 or ****). Another thing that comes up a lot in the same topic as 'weak' are values in order (e.g. 12345 or abc). Words that appear in the same topic as 'strong' tend to be more random words that are less likely to be guessed or 'words' which are not real words at all and just a random string of letters (e.g. dtkbsl).

Creating Strong Passwords

We can now evaluate our password generators. In general, this model got a satisfactory result in password

Table 5: Passwords evaluation

Bigrams model	password strength score
smile29hfmnd	4
456as3862quorthon	4
budek4l	2
dopice1983	3
hilkaurl56	3

strength, as well as making passwords easier to remember. However, it could be improved: one way to do this would be to combine capital characters by using a Bernoulli trial model. The rough idea is to get the probability of capital characters successfully appearing.

Another idea we had to generate strong passwords was to create a Markov Chain only on the passwords that are strong. Therefore this will give us the relationships between characters and words in strong passwords. Consequently, we can use the transition matrix in order to generate strong passwords.

Discussion and Conclusion

Pre-processing allowed us to run our algorithms with ease in most cases. In the future, we could think more about problems we may face at the start of the project. In particular, we could have removed

numbers and small words at an earlier date thereby allowing the project to run even more smoothly.

In our data set, most passwords contain some words, which is definitely not secure considering that the words that people use reoccur, even in such a small data set. We recommend never using any names, repetition, dates or words related to death in a password. Swearwords seem to appear more frequently than one would expect, so maybe these words should also be avoided when creating a secure password.

In the table below we briefly compare different methods of the topics exploration.

Algorithm	**Advantages**	**Disadvantages**
K-Means	runs fast, easy to understand	clustering sensitive to the position of the word in the password
Gaussian Mixture Model	flexible in terms of covariance	fails if dimensionality of problem is too high
Dirichlet Process with GMM	obtains no. of clusters automatically	can take a long time to run
DBSCAN	works well with noise	not entirely deterministic
Jaccard Indexes with K-Means	groups similar words together well	slow
LDA	groups word according to a topic	we do not know the number of topics

As seen above, different models are optimal for different reasons. For example, the Jaccard Indexes with K-Means may be the best in terms of accuracy. However, the more standard version of K-Means is much quicker.

Using the Markov Chain algorithm, we can predict what the most likely subsequent letter in the password is. Moreover, if we know a password contains a certain word, our algorithm suggests the most likely passwords. This could be a great way for a company to see if their passwords are secure or not. They could run our predictor and type words from their own passwords to see if the predictor guessed correctly - letting them know whether they should change a password or not.

Probably the most useful result of out project is the exploration of the strength of the password using LDA. The program shows the security level of the passwords and what characteristics make a password the most secure. The results agree with our intuition and our ideas from the start of the project.

Another observation is that passwords contain limited information in the limited length.

In the cybersecurity industry, passwords are a very important area of research. If, from our model, it is shown that a password is easily generated then it would be easy for the system to be compromised. Therefore, it can be important to use encryption techniques to ensure that the password is less predictable.

Future work:

- For the password generation part, we can also consider generating capital characters by building a Bernoulli trial model
- Try a different dataset
- Use the relative usernames (password and username data doesn't correspond for some reason) to see if the passwords are similar

References

[1] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[2] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[3] Thushan Ganegedara. Intuitive guide to latent dirichlet allocation. https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-latent-dirichlet-allocation-437c81220158 Accessed: 2019-02.

[4] Daniel Lawson. Key algorithms. Accessed: 2019-02.

[5] Zhongduo Lin. *Indoor location-based recommender system*. PhD thesis, University of Toronto, 2013.

[6] Siraj Raval. Gaussian mixture models - the math of intelligence. Accessed: 2019-02, 2017.

[7] Patrick Reuben-Delanchy. Introduction to mathematical cybersecurity notes - markov chains. Accessed: 2019-02.