Learning Password Modification Patterns with Recurrent Neural Networks

Alex Nosenko, Yuan Cheng(✉), and Haiquan Chen

California State University, Sacramento, CA 95819, USA
{anosenko,yuan.cheng,haiquan.chen}@csus.edu

Abstract. The majority of online services continue their reliance on text-based passwords as the primary means of user authentication. With a growing number of these services and the limited creativity and memory to come up with new memorable passwords, users tend to reuse their passwords across multiple platforms. These factors, combined with the increasing amount of leaked passwords, make passwords vulnerable to cross-site guessing attacks. Over the years, several popular methods have been proposed to predict subsequently used passwords, such as dictionary attacks, rule-based approaches, neural networks, and combinations of the above. In this paper, we work with a dataset of 28.8 million users and their 61.5 million passwords, where there is at least one pair of passwords available for each user. We exploit the correlation between the similarity and predictability of these subsequent passwords. We build on the idea of a rule-based approach but delegate rule derivation, classification, and prediction to a Recurrent Neural Network (RNN). We limit the number of guessing attempts to ten yet get an astonishingly high prediction accuracy of up to 83% in under five attempts in several categories, which is twice as much as any other known models or algorithms. It makes our model an effective solution for real-time password guessing against online services without getting spotted or locked out. To the best of our knowledge, this study is the first attempt of its kind using RNN.

Keywords: Authentication · Passwords · Recurrent neural networks

1 Introduction

Passwords remain the first and sometimes the only line of defense for most online services. Having a strong, unique password is extremely important to keep users' data safe. The government agencies, especially those storing users' personally identifiable information (PII), medical and legal records, follow the password guidance of the National Institute of Standards and Technology (NIST). Different online services have independent definitions of secure passwords. They also enforce different password composition, expiration, and reuse policies. This puts a lot of responsibility on users to create and maintain a large number of passwords. To cope with this burden, a user can either use a password manager, create and remember a unique password for each account, or create a very

R. Krishnan et al. (Eds.): SKM 2021, CCIS 1549, pp. 110–129, 2022.
https://doi.org/10.1007/978-3-030-97532-6_7

strong but memorable password that follows all the guidelines and use it across all platforms. According to a survey by a cybersecurity company NordPass, 50% of the respondents in the UK find it extremely difficult to remember unique passwords for multiple accounts [21]. The problem worsens when a user is required to change passwords due to password expiration or known security breaches. Based on numerous studies, the majority of people reuse their passwords by modifying them slightly every time a new password is required [5,9,20,28]. Since most of these studies were conducted in academic institutions and involved participants with higher education levels and better security awareness, the situation for the rest of the Internet community is probably even worse.

The habit of password reuse is detrimental to account security due to the increasing threat of *cross-site password guessing attacks*. In this form of attack, an attacker leverages previously leaked password datasets to guess passwords potentially used by the same user at different sites [4]. With an abundance of password leaks and data breaches, there is a large pool of publicly available passwords for a swarm of users. Attackers have various tools at their disposal, such as dictionary-based attacks, rule-based attacks, and machine learning models for effective and automated guessing. Suppose each online service allows up to three attempts to enter a password before locking down the account and consider that each user has registered on at least five popular online services. An attacker thus has at least 15 attempts to guess a password before being spotted or flagged. The traditional brute-force attack and dictionary attack will not be effective in this setting due to the rate limit. However, rule-based and neural network-based predictions can still yield a high probability of successful guesses with rate-limiting enforced [11,14,16,17].

In this paper, we leverage the rule-based approach and automate the guessing process using a neural network model to derive modification patterns, complete the classification, and generate a password guess. We resort to neural networks, which outperform traditional classifiers like Naïve Bayes and k-nearest neighbors (KNN) used in prior research [28] to solve the classification problem. We use a character-based bidirectional long short-term memory (BiLSTM) model to generate passwords for each modification category. We then build an experimental model that can make predictions without knowing the modification patterns. We use the character-based LSTM encoder-decoder model as it is commonly used when one sequence of characters (e.g., the original password) needs to be transformed into another sequence of characters (e.g., a subsequent password). This model delivers outstanding prediction results for a significant amount of password pairs.

The main contributions of our study can be summarized as follows:

- We created a neural network-based classifier for password modification category prediction.
- We built an LSTM based model for password generation for each category.
- We designed an LSTM based model that predicts a subsequent password based on the original password with up to 83% accuracy in under five attempts.

- We quantified the vulnerability of password reuse based on Levenshtein distance, Jaro-Winkler distance, and modification patterns.
- We made recommendations for online services to enhance their password security.

The remainder of this paper is organized as follows. We discuss the related work in Sect. 2. We describe the original dataset, the pre-processing steps, and the resulting datasets in Sect. 3. Section 4 elaborates the prediction process and the model architecture. Section 5 presents our results and compares them with the results from the existing models. Section 6 talks about the security recommendations and key takeaways of this research. We conclude this study and identify some future directions for this line of research in Sect. 7.

2 Related Work

The problem of password guessing is not new. And over time, there has been an abundance of methods proposed to solve it. Among the most prominent guessing approaches are dictionary-based attacks, rule-based attacks, and neural network-based attacks. While some methods train and test their prediction models on the same dataset, other methods extract rules from one dataset and try to guess passwords from another dataset. The first type of method is referred to as the *single-site password guessing attack*, where attackers crack passwords from a single password leak. The second type is known as the *cross-site password guessing attack*, which exploits leaked passwords from multiple online services. Our research falls into the second type, where we aim to guess users' subsequent passwords from their known passwords from the same or a different site.

Next, we will provide an overview of different password guessing methods, including the most recent advance of using neural networks for this purpose.

2.1 Dictionary Attacks

Dictionary attacks depend upon the assumption that a password is either a word that belongs to a pre-compiled word list or dictionary [9], a valid, complete word (used in vocabulary attacks) or a valid passphrase (used by Markov model-based methods). The first two attacks may need many attempts to make a correct guess, require constant maintenance of dictionaries, and cost a tremendous amount of time and resources. Markov model-based methods, on the other hand, enable efficient password and passphrase cracking by only generating and testing linguistically likely passwords [19] or linguistically correct phrases [24]. These methods are commonly used for single-site password guessing attacks but can also instigate cross-site attacks with slight modifications. Unfortunately, most of the passwords in our dataset do not fall under the category of valid dictionary words or linguistically correct phrases. Thus, these methods are of limited use in solving our problem.

2.2 Rule-Based Attacks

Rule-based attacks rely on password creation and reuse patterns extracted from previously leaked datasets or user surveys and are widely used for cross-site attacks. Researchers conducted statistical analyses of leaked password datasets and discovered that most users stick to simple and easily memorable patterns [4,28–30,33]. Based on the patterns, researchers were able to build algorithms and prediction trees that indicate the most probable modification categories and the most likely transformations. These data-driven algorithms aim to minimize the number of guesses and maximize prediction accuracy. Among the several rule-based mechanisms is Probabilistic Context-Free Grammar (PCFG). This method analyzes leaked datasets and existing wordlists to create grammars for generating word-mangling rules [30]. The next generation of rule-based guessing mechanisms is based on PCFG but leverages previously used passwords as an additional input to help predict subsequent passwords. This targeted prediction algorithm is known as TarGuess-II [29]. Zhang et al. developed a generic algorithmic framework for searching out possible transformations that convert a user's previous passwords to future ones [33]. Their optimal search strategy successfully cracked an average of 13% of the accounts in the experiment within five online guesses and 18% within ten attempts. Wang et al. introduced the next iteration of rule-based predictors by breaking a process into two steps [28]. The first step uses a Naïve Bayes classifier to guess a modification category, and the second step applies the rule-based mechanism to guess the actual password. This approach shows significant improvements in prediction, but the accuracy within ten attempts is still below 30%.

2.3 Neural Network-Based Guessing

This relatively new neural network-based approach was surfaced in 2016 with a premise that Recurrent Neural Networks (RNNs) can predict the next symbol in a character string if provided with enough training data [17]. It presented promising results in single-site and cross-site attacks and was used in combination with rule-based attacks [16] and Markov model [14]. It helped overcome several previous limitations, such as the utilization of fixed-length context, by using long short-term memory (LSTM) network. LSTM is a subset of RNNs and can store features discovered over a longer period of time [22]. Generative Adversarial Networks (GANs), a subset of neural networks, were used as a specific training approach for neural network models to eliminate the need for learning modification patterns on a single-site attack [11]. Despite showing good prediction rates, most of these studies did not limit the number of guessing attempts and ran models until they exhausted every possibility. This assumption makes these models impractical in online password guessing in the real world.

Our research is based on the understanding of password modification patterns. We specifically focus on the subsequent password guessing problem, that is, guessing future passwords from preceding ones. We propose an RNN-based approach, which focuses on lowering the number of attempts used for cross-site password guessing. We generate two models, one with rules provided and the other one that derives the rules itself, and explore the prediction accuracy of the two models in under ten attempts.

3 Dataset

The dataset was provided by Wang et al. [28], which consists of 61,552,446 individual passwords that belong to 28,836,775 users from 107 online services over eight years. The dataset has been sanitized and anonymized to protect personally identifiable information. Every user in the dataset has at least two passwords; thus, we can form at least one pair of subsequently used passwords. Although some users have more than two passwords, we only choose two passwords (i.e., one pair) for each user for simplicity.

We then pre-process the dataset to eliminate pairs of identical passwords and those that appear more than once. The resulting dataset contains 17,133,333 unique pairs. This process helps us identify a set of the most common passwords for further analysis. Among the most popular passwords are “123456,” “password,” “qwerty,” “111111,” “123123,” “dragon,” “monkey,” “shadow,” and “love.”

When looking at the length of the passwords, we discover that 99% of the passwords are 5–17 characters long, which seems to be consistent with the common password requirements enforced by online services as well as the general human memory capacity [25]. The passwords that are longer than 17 characters (hard to memorize) or shorter than 5 (not acceptable by most online services), as well as the passwords that contain non-ASCII characters, are considered outliers and are thus removed from further consideration.

We analyze the distribution of passwords based on two metrics, Levenshtein distance and Jaro-Winkler distance.

The Levenshtein distance is the number of edits (e.g., substitution, insertion, or deletion) needed to transform one string into another. For example, transforming “rain” to “shine” requires three steps, consisting of two substitutions and one insertion: “rain” $\rightarrow$ “sain” $\rightarrow$ “shin” $\rightarrow$ “shine.” These operations could have been done in other orders, but at least three steps are needed [10]. In our dataset, the Levenshtein distance between passwords ranges from 0 to 17, as shown in Fig. 1. And the majority of password pairs have a Levenshtein distance in the range from 1 to 11. The Levenshtein distance can help set up password reuse rules that are easy for users to understand (e.g., make sure the subsequent password is different from the original by three characters). However, it suffers from a major limitation. For example, the Levenshtein distance between the words “password” and “password12345678” is 8, which is relatively high, although both words exhibit an easy-to-guess pattern.

We then resort to the Jaro-Winkler distance for a more meaningful measure. The Jaro-Winkler distance considers the substitution of two close characters less important than the substitution of two characters that are far from each other [7]. It computes the string similarity by returning a value that lies in the

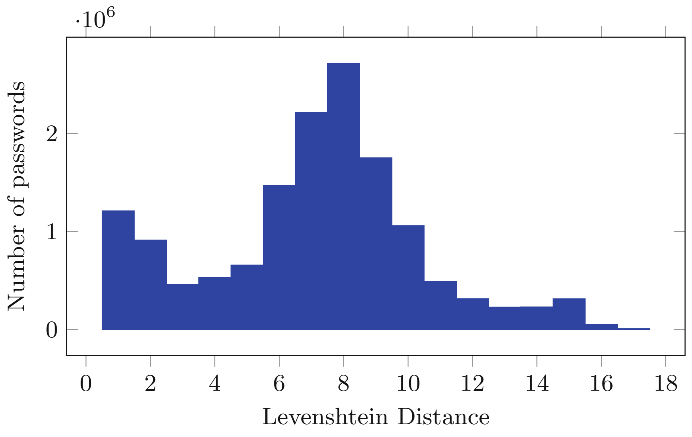

Fig. 1. Password distribution by Levenshtein distance

interval of [0.0, 1.0]. For instance, the Jaro-Winkler distance between the words "password" and "password12345678" is 91.7%. Figure 2 shows the distribution of password pairs in the dataset based on the Jaro-Winkler distance. The majority of the password pairs fall in between 0.4 and 0.99.

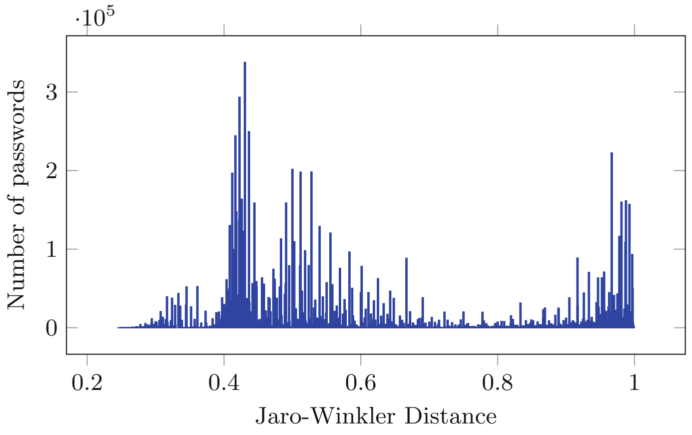

Fig. 2. Password distribution by Jaro-Winkler distance

The passwords that exhibit the highest similarity will have the smallest Levenshtein distance and the highest Jaro-Winkler distance and will be the best candidates for performing cross-site guessing attacks. We use the NLTK Python library for the Levenshtein distance and the StrsimPy library for the Jaro-Winkler distance to implement both metrics.

In prior research, several most common modification patterns were identified, including Substring, Common Substring, Capitalized, Leet, and Sequential Keys [28]. We label each password pair based on these five patterns to create a labeled dataset. The pairs that do not fit into any of these rules are dropped. The labeled dataset contains 3,006,871 unique password pairs.

Figure 3 demonstrates the number of password pairs for the original dataset ("All passwords"), the set where an original password is not the same as its subsequent password ("Unique pairs"), the set of unique pairs where each password is between 5 to 17 characters long and contains only valid ASCII characters that will be used for most of this research ("Working set"), and the labeled set ("Known rules").

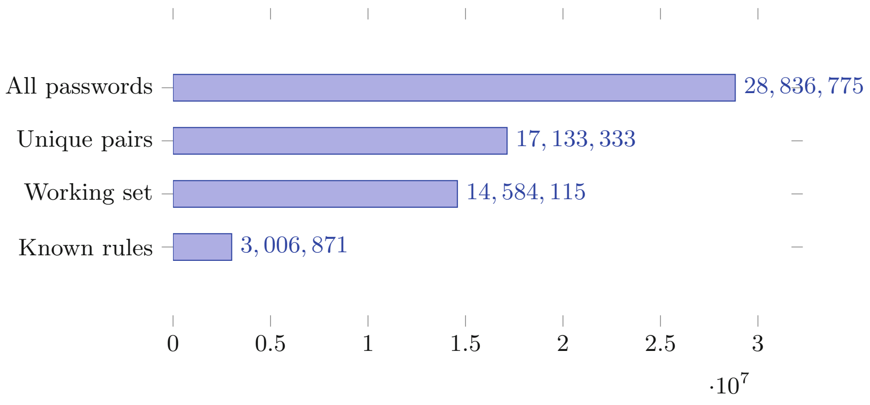

Fig. 3. The number of password pairs in each dataset

4 Password Prediction Process

So far, we have reviewed the data pre-processing steps necessary for password prediction. Next, we want to build a pipeline to take the resulting dataset, identify and tag modification patterns, classify passwords into appropriate buckets, and generate predictions for each original password. We also seek to take a step further by skipping tagging and classification and go straight to password prediction. We will refer to this process as *direct password prediction*. This approach will be especially beneficial when users combine multiple modification patterns or no rules can be identified. To the best of our knowledge, this approach has not been used for launching a single-user cross-site password guessing attack yet.

The prediction pipeline consists of four steps. During the first step, common modification patterns are defined, and each password pair is analyzed and labeled with a corresponding category. We use a neural network model to assign each original password into a single modification category during the second step.

This process is known as the single-label prediction problem. During the third step, we build a second model to learn about possible modifications within each category. With 90% accuracy on the test data, the model can understand and generate all possible modifications for each category. Both models are then combined, and the resulting pipeline is assembled in the last step. Our approach can take just one original password as an input, classify it into a modification category, and generate password guesses. We will now review each step in further detail.

4.1 Tagging

Before tagging password pairs, let us first introduce the common password modification patterns we borrowed from [28]. Leet refers to any transformation of alphanumeric characters to visually similar symbols and vice versa. The Substring category includes password pairs where one password is a substring of the other. Adding symbols to the head or tail of a string is the most common modification of this category. Capitalization is where one or more symbols are in uppercase. The Common Substring category contains password pairs that share common letter combinations. The Sequential Keys category consists of passwords that contain alphabetically ordered letters (e.g., “abcd”), sequential numbers (e.g., “1234”), and adjacent keys on the keyboard (e.g., “qwert,” “asdfg,” etc.). We define a function to identify which pattern each password pair fits in and tag each pair with a corresponding modification pattern category. After the tagging is completed, we drop the passwords that do not match any rules or contain non-ASCII symbols. The resulting dataset has 3,006,871 pairs of passwords. Figure 4 shows the distribution of each password modification category in the tagged dataset. The most common patterns found in the dataset are Substring and Common Substring. Together they cover 2,674,521 password pairs, which are around 89% of the dataset. These password pairs provide sufficient training data for our proposed model. The other three categories represent about 10% of the dataset. Except for the Sequential Keys category, we still collect enough training data from the categories of Capitalization and Leet.

4.2 Classification

Conventional classifiers have been used to solve the classification problem as they are easy to use and do not require heavy data pre-processing. However, neural networks have proven to deliver better prediction rates, especially on larger datasets [23].

We build a 4-layer LSTM classifier using the Keras Python library, as shown in Fig. 5. The Input Layer takes a single sequence of characters with the same length as the longest password in the dataset, which is 17 characters long. The One-Hot Encoder processes every password as a sequence of characters and transforms those sequences into a one-hot numeric array. It assigns the value of 1 if the character is present in a given word or 0 if otherwise. The encoding is passed to the LSTM units. We use a character-level Bidirectional LSTM (BiLSTM) Layer,

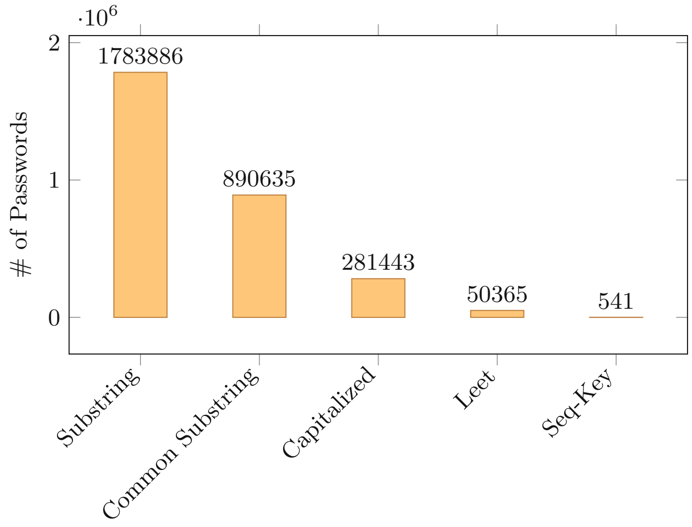

Fig. 4. Category distribution of the tagged dataset

which is an extension of traditional LSTMs and can improve the model performance on sequence classification problems. The BiLSTM Layer runs inputs in two directions, one from the past to the future and the other from the future to the past. Unlike unidirectional LSTM, BiLSTM uses two hidden states and can preserve information from both past and future at any point in time. Because of these qualities, BiLSTM can better understand the context around each character in the sequence [31]. The output of the BiLSTM cells is fed to a dense Activation Layer. The Activation Layer contains an activation function, which defines how the weighted sum of the input is transformed into an output. To ensure that a model learns features and does not converge prematurely, we use the Adam optimizer with a small learning rate [12]. This optimizer is used to update network weights during each training iteration.

As a result of this step, we build a neural network classifier to predict password modification patterns.

4.3 Password Generation

In this step, we train a 7-layer character-level BiLSTM model to generate passwords within each modification category. The model shown in Fig. 6 has two input streams. The left stream includes Input Layer #1, the One-Hot encoder, and the BiLSTM layer. These three layers act similarly to what is described in Sect. 4.2. The right input stream includes Input Layer #2 and a Repeat Vector Layer, which are both used to add a list of modification patterns to the model as each prediction is generated within a single category. The Concatenation Layer combines the outputs of both streams and feeds the combined outputs into the

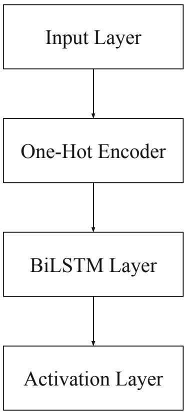

Fig. 5. Classification model architecture

Activation Layer. The Activation Layer acts the same as described in Sect. 4.2. The resulting model can generate password guesses for each modification category with high accuracy.

To generate password guesses for all categories, we put the classification model and the password generation model together into one prediction pipeline, namely Pipeline Prediction Mechanism (PPM). As an original password enters the pipeline, the first model predicts the modification category. Then, the predicted category, along with the original password, is fed into the second model, which generates password candidates and chooses the top 10 candidates with the highest probability. Once the prediction is completed, we verify if a password is guessed correctly.

4.4 Direct Password Prediction

To the best of our knowledge, RNN has never been used to solve cross-site password guessing for the same user before. This problem, however, is similar to the problem of machine translation. In both problems, the source may vary in length and character dictionary. The model architecture consists of at least two LSTM layers, encoder and decoder. The encoder takes the input sequence and summarizes the information into a context vector or hidden states of LSTM [3]. The outputs are not important and thus are dropped, but the hidden states are saved. This context vector encapsulates the information for all input elements and will be used by the decoder to generate predictions. The decoder is also an LSTM layer that takes the encoder output as an initial state and produces an output sequence. We use the Softmax layer from the Keras library as an Activation Layer in our model. The Softmax function is based on normalized

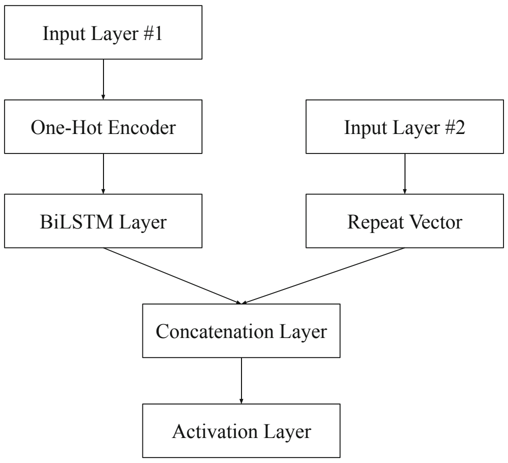

Fig. 6. Password generation model

exponential function and is used as the last layer on a neural network to normalize the output to a probability distribution over predicted output classes [13]. In our model, it will determine the output modified password.

Since a model can generate multiple predictions with different degrees of confidence, we need an algorithm to choose the top 10 most probable outputs. We use the Beam search algorithm, one of the most widely used for sequence-to-sequence machine translation problems [32], to help us identify the most probable predictions. Direct password prediction proves to be the most promising approach as it eliminates the need for constant rule derivation and distribution analysis and eases dataset pre-processing. The resulting model, namely Direct Prediction Mechanism (DPM), is rule-independent and delivers high prediction rates.

4.5 Prediction on a Reversed Dataset

To increase the number of password pairs, Wang et al. [28] switched source and target passwords and attempted a prediction of original passwords based on the subsequent ones. We decide to replicate the same experiment providing a model with modified passwords as a source and asking it to predict the original passwords. This adds additional 3.1M password pairs to our experiment. No changes to the model are necessary to run this experiment, which exhibits another advantage of using neural networks to solve the password guessing problem. We realize that predicting the original password can be easier for the DPM model since most users tend to use simpler passwords to begin with and add complexity (e.g., Leet, Capitalization, String, etc.) as they change them. Additionally, we try to reverse the order of characters in the original password before passing it to the model for prediction. This approach was used by Sutskever et al. [26], where the authors reversed the word order in the source sentence before passing it to an RNN model, which ultimately yields better prediction results.

5 Experimental Results

5.1 Hardware Requirements

Most of this project was executed on Google Colab Pro, which is a cloud-based Jupyter notebook environment. The hosted runtime environment uses Tesla P100-PCIE-16 GB GPU, Intel(R) Xeon(R) CPU @ 2.20 GHz processor, 25 GB of RAM, and 109 GB of disk space. The most hardware demanding parts of the experiment were dataset pre-processing, model training, and direct password prediction. Direct password prediction was the most computationally expensive. Based on Google Colab measurements, it took 13 GB of RAM to train a model on 400k records and 20 GB for 500k records. Even though it might sound like a significant amount of resources, it is not unreachable for a sophisticated attacker. Better hardware resources will yield better performance results since we can train the model on a larger dataset and make faster predictions with higher accuracy.

5.2 Results

We first compared the results of an LSTM classifier and a Naïve Bayes classifier. Table 1 shows that the LSTM based model delivers better results than the Naïve Bayes classifier, especially in underrepresented categories, such as Leet and Common Substring.

Table 1. LSTM model vs. Naïve Bayes classifier

		Precision	Recall	F1-score
LSTM	Capitalized	0.65	0.58	0.61
	Common substring	0.58	0.31	0.4
	Leet	0.49	0.23	0.31
	Seq-Key	0	0	0
	Substring	0.73	0.91	0.81
Naïve Bayes	Capitalized	0.6	0.39	0.48
	Common substring	0.38	0.05	0.08
	Leet	0.14	0.01	0.01
	Seq-Key	0	0	0
	Substring	0.66	0.9	0.79

LSTM outperforms traditional classifiers because it captures and preserves the sequential order (i.e., the order in which the characters appear in a password), while the classical models do not. For example, "abac" is different than "aabc" for LSTM, while it is the same for a classical model as it only considers the frequency of the featured characters 'a,' 'b,' 'c,' which are the same for both strings.

We then evaluated the performance of our password generation model. Since our approach is novel, we do not have an exact baseline to make a one-to-one comparison with the existing work. The closest studies are the ones that estimate how many guesses are needed to predict a password correctly [4,28,29]. All of these papers published their prediction rates within the first ten attempts. Therefore, we use ten attempts as part of the experiment parameters. We compared the results of our Pipeline Prediction Mechanism (PPM) and Direct Prediction Mechanism (DPM) with three existing algorithms from [28] (referred to as "Domino" hereafter), [29] (referred to as "TarGuess-II" hereafter), and [4] (referred to as "Tangled" hereafter).

On average, both of our models exhibited a 5% improvement of overall prediction rates comparing to "Domino" and three times more accurate predictions than TarGuess-II and "Tangled" as shown in Fig. 7. Although the improvement does not seem drastic, if we can predict 5% more passwords out of 6 million, that is around 300,000 more compromised accounts.

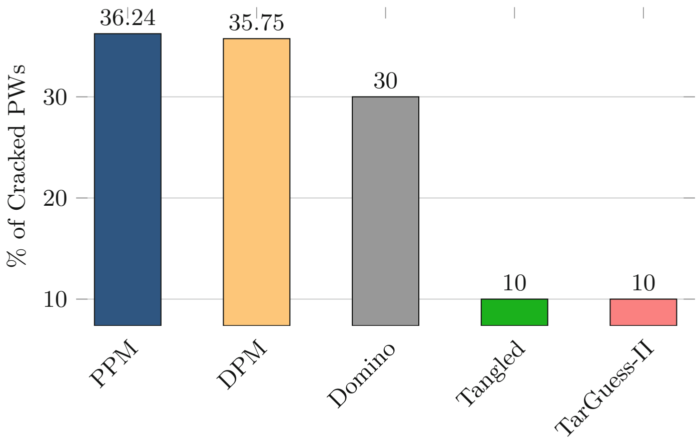

Fig. 7. Model comparison at ten attempts

A larger difference in prediction rates between "Domino," PPM, and DPM can be observed in the first four attempts, as shown in Fig. 8, where our model is 100% more effective. This is especially surprising considering that DPM was only provided with the original password and no other prior information. It can also be observed that the DPM model makes most of the predictions during the first few attempts, and then its confidence decreases as well as the number of predicted passwords (see Fig. 9). On the other hand, traditional approaches exhaust all possible combinations and thus predict more as they try more. This reveals a fundamental difference between our method and those traditional ones.

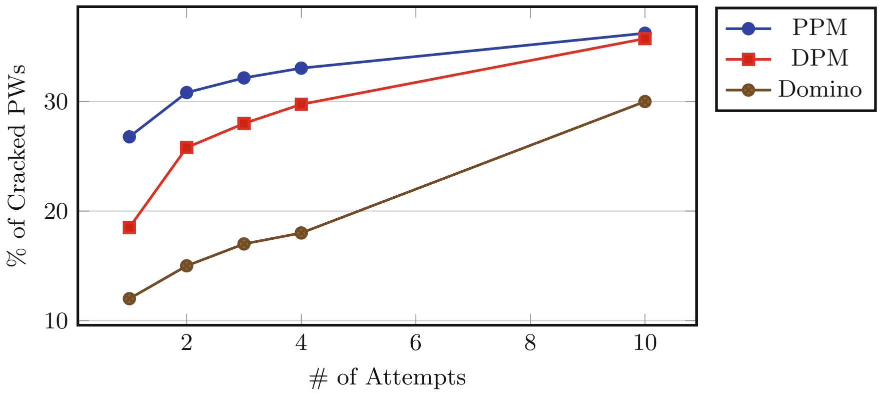

Fig. 8. Model comparison under ten attempts

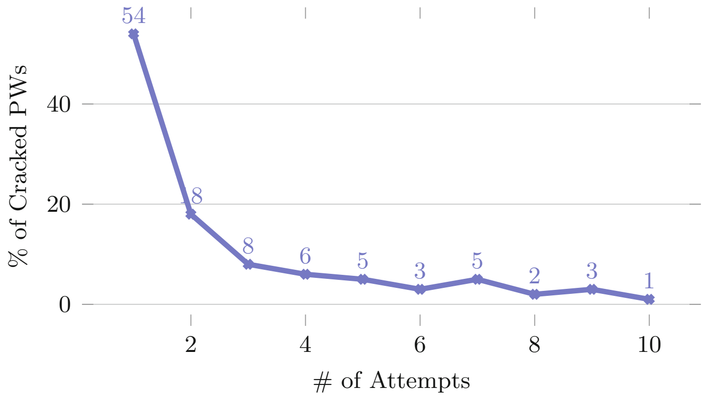

Fig. 9. DPM prediction rate for each attempt

Next, we zoomed in on the results of the DPM predictions to quantify for the first time the correlation between password predictability and Levenshtein distance for subsequent passwords. The prediction rates, as shown in Fig. 10, are broken down by the Levenshtein distance on the x-axis (x-coordinates correspond to the distance values 1, 2, 3, 4, 5, and 6, respectfully). The y-axis refers to the percentage of guessed passwords. We found that it is 6–8 times harder to predict a subsequent password when the Levenshtein distance is 4 compared to the same task when the distance is 1. Changing only one character in subsequent passwords does almost nothing to improve the overall password strength since the prediction rate is as high as 83%. In other words, almost 8 out of 10 subsequent passwords can be predicted regardless of modification patterns if the two passwords only differ in one character.

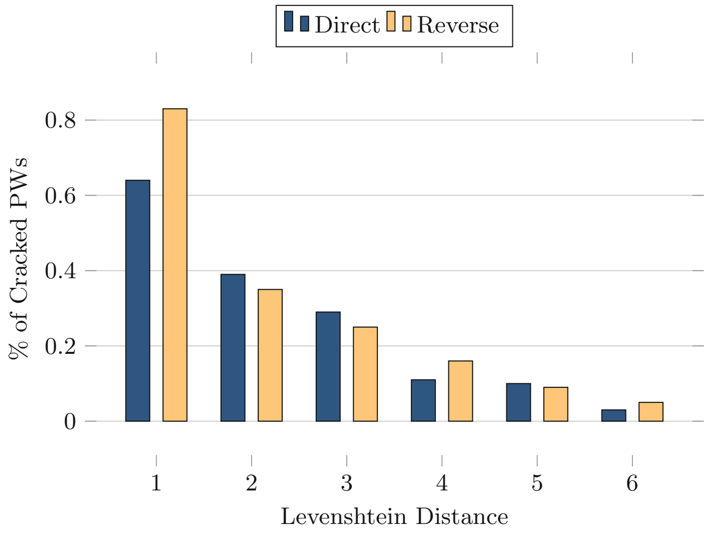

Fig. 10. Prediction rate vs. Levenshtein distance

We also examined the association between password predictability and Jaro-Winkler distance by running a DPM model on each distance interval, as shown in Fig. 11. Passwords with less than 0.7 Jaro-Winkler distance result in a prediction rate of below 1% under five guesses, comparing to passwords with the same metric of above 0.9, which have a prediction rate of around 50%.

We then investigated the prediction results to see how specific modification patterns contribute to the overall password predictability as most of the predicted passwords exhibit some sort of pattern. The unrelated password pairs that do not have any syntactic or semantic similarity proved to be the hardest to predict. Figure 12 shows that Capitalization along with having a subsequent password being a substring or containing a substring of the original are the easiest categories to break and are at least 20%–50% less secure. Considering how easy it is for a user to remember a password with a few added characters or some capitalization changes, it is as easy for the model to guess it. Since Substring is the most represented category in the original dataset, it is not surprising to see that DPM is well trained in this category and is thus able to make a successful guess around 50% of the time, as shown in Fig. 12. However, the interesting fact is that Capitalization represents only around 9% of the dataset, yet a model successfully predicts around 64.5% of those passwords. It is also worth noticing

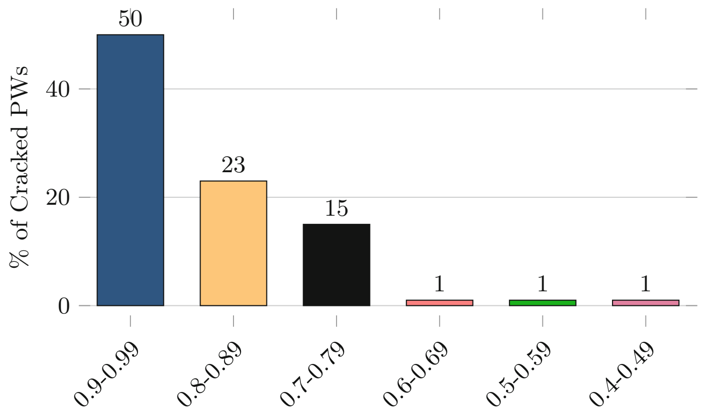

Fig. 11. Prediction rate vs. Jaro-Winkler distance

that DPM is able to predict around 7.4% or 25,000 of passwords that do not follow any known modification patterns.

Overall, both models demonstrated great prediction rates, beating the close competitors 2:1 in scenarios where only a small number of guessing attempts is allowed. Our models adopt a more fine-grained prediction approach and are capable of predicting 83% of passwords in certain common modification categories. With more data available, the performance of our models improves. But our models can also work well on a smaller training set (e.g., 50k size).

6 Discussion

Predicting slightly modified subsequent passwords is getting easier. With more data breaches occurring each year, there is an abundance of leaked passwords, including subsequent passwords for the same users. This allows us to extract more fine-grained modification patterns and train more robust prediction models. Our experiment showed that having the original password alone is enough to predict a subsequent password with a high probability in just a few attempts. With most online services allowing up to ten attempts [1], an attacker can use the proposed mechanisms and models to generate subsequent passwords and try to compromise an account without raising the alarm. With the availability of more powerful computational resources and companies delaying the public release of data breach details, an attacker can re-train the models to account for newly acquired data and enhance the chance of success.

The length and complexity requirements recommended by the NIST guideline [2] are outdated in practice today. Having a long password that consists of various symbols might add extra security only when it comes to single-site dictionary attacks and eavesdropping [8]. Making slight modifications based on an original password provides little to no additional security in cross-site attacks. It may even give users a false sense of security as these changes seem to align with

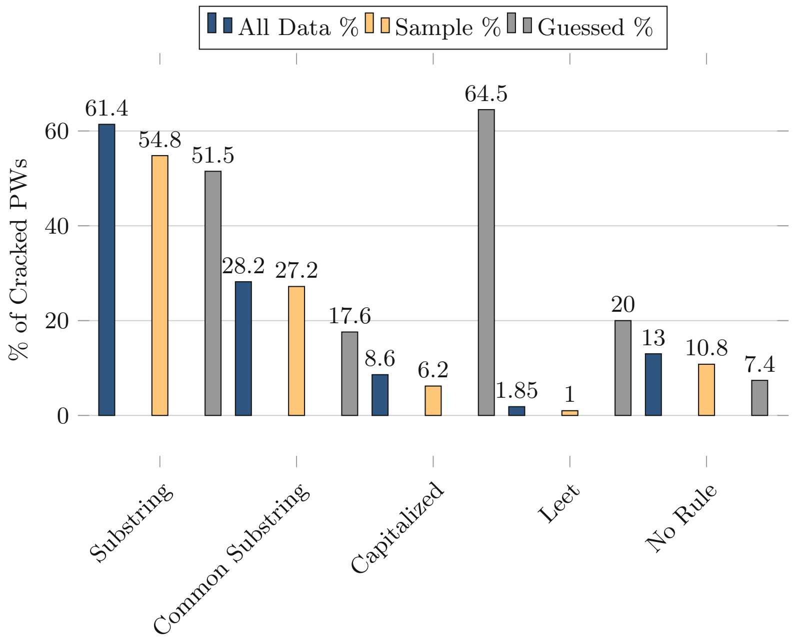

Fig. 12. Prediction within each category

the characteristics of "strong passwords" defined by the guideline. Our studies showed that we need to consider the similarity between an original password and its successors as a new requirement in a future edition of user authentication guidelines. The less similarity there is between subsequent passwords, the more secure user data is [18]. Even with sufficient training data, neural network-based guessing algorithms require much more attempts to crack passwords that are distant in terms of similarity metrics. For example, we observed that passwords that are four edits away are at least three times harder to break than those with just one modification away. Having completely unrelated passwords can mitigate cross-site attacks in the presence of a password breach.

Updating user authentication guidelines also helps standardize web frameworks and development tools include libraries that support similarity-based proactive password checking. A proactive password checker can prevent users from choosing an easy-to-guess subsequent password, especially when it is similar to the used ones. This process, however, should not have a detrimental effect on user experience and may include a system to suggest a secure password if a user struggles to come up with one.

The use of passwords alone should be re-considered by service providers in favor of two-factor authentication, biometrics, and other alternative means. Users' creativity, memory capacity, or the level of education in computer security should not be the decisive factors of data security. Most survey participants involved in the password guessing studies represent a younger and higher educated cohort compared to the general population [4,27]. The safe guess would be that a general population would have even more insecure password habits. It is hard to change users' habits as we consistently see users reusing the same passwords or modifying them slightly. As users entrust their data and private information to more and more online services, these services should step up and carry more responsibilities to ensure that users' negligence or lack of education does not jeopardize the safety of their data.

Password managers are considered by many as an appealing substitute for alleviating password fatigue. However, password managers usually require a local or cloud-based password storage, and the access to such storage relies on a master password. An insecure implementation or a lost master password adds a single point of failure to authentication. Prior studies showed that most popular password managers in use suffered from various kinds of vulnerabilities [6,15]. The widespread adoption of password managers will not overshadow the efforts on making passwords stronger. In fact, the advances in both directions complement each other.

7 Conclusion and Future Work

In this research, we investigated the problem of subsequent password prediction. We built a password prediction pipeline to automate password categorization and password generation using Recurrent Neural Networks. The prediction results were superior to the ones delivered by traditional classification and guessing algorithms. The performance boost was especially significant when we limit the number of guessing attempts to five. We combined the understanding of rule-based prediction algorithms and the power of LSTM neural networks to solve the problem of cross-site prediction for passwords created by the same user. It is a relatively new approach and perhaps one of the very first attempts to use Recurrent Neural Networks for this specific task. We were able to quantify the correlation between the similarity, modification patterns, and predictability of subsequent passwords. In addition, we demonstrated the ease of prediction and high accuracy of the most common modification strategies, such as adding head or tail symbols to the original password or capitalization. We showcased that such a prediction process could be facilitated by affordable hardware or online computing resources, such as Google Colab, due to the low complexity and shallow nature of the RNN model used. The efficiency of prediction allows it to run on platforms where five or fewer attempts are allowed before an account gets locked. We also discussed the concrete steps the online services should take to improve the security of the authentication process.

In the future, we would like to apply the model to a single-site attack scenario to see how well it can perform compared to the other models. We would also like to investigate password pairs that are not syntactically but semantically similar (e.g., synonyms, associated words). The prediction of these passwords was out of the scope of this research. We may also improve the password prediction by training the model on a synthetic balanced dataset that contains various under-represented modification patterns found in this research (e.g., common names).

Lastly, we would like to build a neural network model to classify the passwords that involve more than one type of modification pattern (e.g., Capitalization and Leet, Substring and Leet, etc.).

References

1. Brostoff, S., Sasse, M.A.: "Ten strikes and you're out": increasing the number of login attempts can improve password usability. In: CHI 2003 Workshop on Human-Computer Interaction and Security Systems (2003)
2. Burr, W., Dodson, D., Perlner, R., Gupta, S., Nabbus, E.: NIST SP800-63-2: Electronic Authentication Guideline. Tech. rep, National Institute of Standards and Technology, Reston, VA (2013)
3. Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: encoder-decoder approaches. arXiv preprint arXiv:1409.1259 (2014)
4. Das, A., Bonneau, J., Caesar, M., Borisov, N., Wang, X.: The tangled web of password reuse. In: NDSS. vol. 14, pp. 23–26 (2014)
5. Florencio, D., Herley, C.: A large-scale study of web password habits. In: 16th International Conference on World Wide Web, pp. 657–666 (2007)
6. Gasti, P., Rasmussen, K.B.: On the security of password manager database formats. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 770–787. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33167-1_44
7. Gomaa, W.H., Fahmy, A.A., et al.: A survey of text similarity approaches. Int. J. Comput. Appl. **68**(13), 13–18 (2013)
8. Grassi, P.A., Garcia, M.E., Fenton, J.L.: DRAFT NIST SP800-63-3 digital identity guidelines. Tech. rep, National Institute of Standards and Technology, Los Altos (2017)
9. Haque, S.T., Wright, M., Scielzo, S.: A study of user password strategy for multiple accounts. In: Third ACM Conference on Data and Application Security and Privacy, pp. 173–176 (2013)
10. Hardeniya, N.: NLTK Essentials. Packt Publishing Ltd., Birmingham (2015)
11. Hitaj, B., Gasti, P., Ateniese, G., Perez-Cruz, F.: PassGAN: a deep learning approach for password guessing. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 2019. LNCS, vol. 11464, pp. 217–237. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21568-2_11
12. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint hyperimagehttp://arxiv.org/abs/1412.6980arXiv:1412.6980 (2014)
13. Kouretas, I., Paliouras, V.: Simplified hardware implementation of the softmax activation function. In: 2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST), pp. 1–4. IEEE (2019)
14. Li, H., Chen, M., Yan, S., Jia, C., Li, Z.: Password guessing via neural language modeling. In: Chen, X., Huang, X., Zhang, J. (eds.) ML4CS 2019. LNCS, vol. 11806, pp. 78–93. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30619-9_7
15. Li, Z., He, W., Akhawe, D., Song, D.: The emperor's new password manager: Security analysis of web-based password managers. In: 23rd USENIX Security Symposium. pp. 465–479 (2014)

16. Liu, Y., et al.: GENPass: a general deep learning model for password guessing with PCFG rules and adversarial generation. In: 2018 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2018)
17. Melicher, W., et al.: Fast, lean, and accurate: modeling password guessability using neural networks. In: 25th USENIX Security Symposium, pp. 175–191 (2016)
18. Murray, H., Malone, D.: Exploring the impact of password dataset distribution on guessing. In: 2018 16th Annual Conference on Privacy, Security and Trust (PST), pp. 1–8. IEEE (2018)
19. Narayanan, A., Shmatikov, V.: Fast dictionary attacks on passwords using time-space tradeoff. In: 12th ACM Conference on Computer and Communications Security, pp. 364–372 (2005)
20. Notoatmodjo, G., Thomborson, C.: Passwords and perceptions. In: Seventh Australasian Conference on Information Security, vol. 98, pp. 71–78. Citeseer (2009)
21. Rawlings, R.: Password habits in the US and the UK: This is what we found (2020). https://nordpass.com/blog/password-habits-statistics/
22. Schmidhuber, J., Hochreiter, S.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
23. Schumacher, M., Roßner, R., Vach, W.: Neural networks and logistic regression: Part i. Comput. Stat. Data Anal. **21**(6), 661–682 (1996)
24. Sparell, P., Simovits, M.: Linguistic cracking of passphrases using Markov chains. IACR Cryptol. ePrint Arch. **2016**, 246 (2016)
25. Stobert, E., Biddle, R.: The password life cycle: user behaviour in managing passwords. In: 10th Symposium on Usable Privacy and Security (SOUPS), pp. 243–255 (2014)
26. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. arXiv preprint arXiv:1409.3215 (2014)
27. von Zezschwitz, E., De Luca, A., Hussmann, H.: Survival of the shortest: a retrospective analysis of influencing factors on password composition. In: Kotzé, P., Marsden, G., Lindgaard, G., Wesson, J., Winckler, M. (eds.) INTERACT 2013. LNCS, vol. 8119, pp. 460–467. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40477-1_28
28. Wang, C., Jan, S.T., Hu, H., Bossart, D., Wang, G.: The next domino to fall: empirical analysis of user passwords across online services. In: Eighth ACM Conference on Data and Application Security and Privacy, pp. 196–203 (2018)
29. Wang, D., Zhang, Z., Wang, P., Yan, J., Huang, X.: Targeted online password guessing: an underestimated threat. In: 2016 ACM Conference on Computer and Communications Security, pp. 1242–1254 (2016)
30. Weir, M., Aggarwal, S., De Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: 2009 30th IEEE Symposium on Security and Privacy, pp. 391–405. IEEE (2009)
31. Xu, G., Meng, Y., Qiu, X., Yu, Z., Wu, X.: Sentiment analysis of comment texts based on BiLSTM. IEEE Access **7**, 51522–51532 (2019)
32. Yoo, J.Y., Morris, J.X., Lifland, E., Qi, Y.: Searching for a search method: benchmarking search algorithms for generating NLP adversarial examples. arXiv preprint arXiv:2009.06368 (2020)
33. Zhang, Y., Monrose, F., Reiter, M.K.: The security of modern password expiration: an algorithmic framework and empirical analysis. In: 17th ACM Conference on Computer and Communications Security, pp. 176–186 (2010)