On Predicting the Final Character of Passwords

Richard Shay

December 8, 2010

Abstract

As a means of authentication and authorization, passwords are ubiquitous. Passwords are frequently the sole guardian of critical systems and data. This paper seeks to gain a better understanding of the internal structure of passwords; this paper uses techniques from machine learning to predict the final character of passwords when given the prior characters and some outside information. This paper is a final project for a class on machine learning, and follows the methodology of that discipline. The results, moreover, may be of interest to security researchers.

1 Introduction

Logging into a bank account. Signing up for Netflix. Checking your email. What do these activities have in common? They all require dealing with a password.

Passwords are now a commonplace component of daily life. They provide security by ensuring that only authorized individuals access a given resource. But the security afforded by a password is no stronger than the password itself. Passwords can be cracked – that is, an attacker can use different means to determine the password of a given user.

Previous papers have discussed whole-password cracking. This work uses machine learning to predict last character of a password, given information about the preceding characters and other contextual data.

Understanding whether this prediction is possible holds implications for security researchers. Many password-cracking techniques utilize assumptions about the composition of characters within a password. This research also gives a better understanding of an attack known as *shoulder surfing*, in which an attacker watches the victim enter a password. This paper provides an understanding of how well the attacker could reconstruct the final character, having observed the prior characters being entered.

The rest of this paper is organized as follows. Background and related work are covered in Section 2. Section 3 outlines the collection of the dataset used in this study, which has been collected for prior research. The methodology for machine learning analysis is outlined in Section 4. There follows a detailed discussion of my analysis, culminating in the final results being presented in Section 9. The paper concludes with a discussion of the findings in Section 10.

2 Related Work

The data I use, described in Section 3, was collected as part of an ongoing research project by CyLab at Carnegie Mellon University concerning passwords and password-composition policies. The project began with a survey conducted on students at Carnegie Mellon University, in which we asked participants about their passwords and sentiment regarding them [8]. Subsequently, the project began to ask participants to create passwords under various conditions online using Amazon Mechanical Turk. While we have no publications yet using the dataset used in this paper, we are working on two papers. We are exploring the amount of *entropy* [6], or "guessability," contained in bodies of passwords created under our conditions. And we are studying how quickly an attacker could "crack" these passwords, using various techniques.

The work presented here relates to password cracking; it involves using part of a password to determine another part. Many cracking efforts, however, involve guessing a whole password. The idea, in many of these papers, is to explore how an attacker might crack a victim's password with no prior knowledge of the password itself.

Previous work in using text characters to determine the probability of subsequent characters was performed by Claude Shannon in 1951 [7]. Shannon estimated the entropy of the English language as a whole. He generated probabilities for *n-grams*, sequences of characters in text; and he measured how well the first $n-1$ characters could predict the *nth* character.

Prior work has used password cracking as a measure of the strength of password corpora. In 1995, Bishop et al. [1] used automated tools to crack passwords submitted by system administrators, and found that the first password was cracked within two minutes. Proctor et al. [5] performed a laboratory experiment in which they cracked passwords gathered in a laboratory setting under differing conditions.

A well-known password-cracking tool is John the Ripper [2]. This freely available program allows its user to perform cracking attempts on a given set of passwords, either in plaintext form or hashed. A malicious attacker could use this tool to decipher a stolen file of encrypted passwords.

Narayanan et al. [4] present a password guess-generation mechanism based on a Markov model. Given a training set of passwords, this algorithm learns the probabilities of starting characters in passwords. Then, the algorithm learns the probabilities for digrams found in passwords in the training set. Based on starting characters and digrams, the algorithm can assign a probability to a given password, thereby enabling guess attacks using the most likely passwords as guesses first. Subsequent research has shown that this technique can be more effective than John the Ripper [3, 9].

Weir et al. [10] present another algorithm for password cracking, generating a list of guess passwords based on estimated probability. This scheme involves training on a set or sets of passwords. The algorithm learns structural probabilities within the training passwords; for example, a structure of two uppercase letters followed by six lowercase letters might be the most probable structure in a given training set. Character string frequencies are also learned from the training data; for example, for three consecutive lowercase letters, the most common string might be “xyz.” The algorithm then uses this data to generate guess passwords; these guesses can be considered attempts to crack an unknown password. A subsequent study has found this technique especially effective in password cracking in a short period of time [11].

3 Data Collection

The data set used for this project is part of an ongoing research effort by CyLab at Carnegie Mellon University. This project involves gathering passwords under explicit sets of password-composition requirements, and asking participants to fill out surveys about password usage habits and sentiment. This section provides an overview of the password-collection effort. Amazon Mechanical Turk is used to recruit and compensate participants. Custom code, written in Ruby on Rails, walks the participant through the study. SurveyGizmo collects survey data from participants.

The study is divided into two parts. Participants see a notice about the study on Amazon’s Mechanical Turk service. From there, they are directed to its website. They are given a scenario for password-creation and asked to create a password adhering to a set of requirements, determined by their condition. If the password does not meet the requirements, they are asked to create a new password until a satisfactory password is made. They answer a brief survey, and are asked to recall their password. Two days later, participants are sent an email inviting them to return for the second part of the study. Participants are asked to recall their passwords again, and then are given a second survey, which concerns how they stored their password and the process of password recall.

Each participant is assigned to one of eight conditions and creates exactly one password. No participant may take the study more than once. Over one thousand passwords have been collected per condition.

Participants in Condition basic8survey are told “To link your survey responses, we will use a password that you create below; therefore it is important that you remember your password.” In the other seven conditions, participants are asked to imagine that their email service provider has become compromised, and that they must create a new password under their specified set of requirements. They are asked to behave as if this were their real password.

Condition basic8 provides the following instructions: “Password must have at least 8 characters.”

Condition basic16 provides the following instructions: “Password must have at least 16 characters.”

Condition dictionary8 uses a dictionary check: the non-letter characters are removed from the submitted password and the password is rejected if the remaining letters are in a dictionary, ignoring case.

Condition comprehensive8 required an uppercase letter, a lowercase letter, a digit, a symbol, and a dictionary check.

Conditions blacklistBasic,blacklistHard, and blacklistFiveBillion reject any password in a blacklist. The sizes of these blacklists are 234936, 40 million, and 5 billion words; and the check is made without regard to case.

4 Procedure Outline

The objective of this paper is to train a classifier that can predict the final character of a password, given the preceding characters and meta-data. This section outlines that process. Data preparation and feature selection are outlined in Section 5; this details the process of migrating data from an ongoing security research project for this paper, and the process of selecting and creating features. This section also discusses the division of data into *development*, *train*, *test*, and *holdout* sets. In Section 6, exploratory data analysis is performed on the *development* data set. Section 7 presents a baseline performance, using SMO with default settings trained on the *training* dataset and run on the *test* dataset. Parametric optimization is performed in Section 8; this includes a comparison of baseline and optimized performance. Finally, the optimized model is trained on both the *training* and *test* datasets, and used to classify instances in the *holdout* dataset. The results of this are presented in Section 9.

5 Data Preparation

I divided the data into four sets: the *development* set, used for data exploration; the *train* set; the *test* set; and the *holdout* set, to be used after optimization.

I used one thousand participants from each of the eight password-composition conditions, meaning 8000 instances total were used. For each condition, 200 were randomly assigned to *development*, *test*, and *holdout*; and 400 to *train*. Thus *development*, *test*, and *holdout* each contain 1600 instances, and *train* contains 3200.

The next step was to prepare the data for use by WEKA. The data includes a participant's password, how long that participant took to complete each step of the study, how many attempts the participant required to create and recall a password, and whether the participant returned for the second part of the study. It included survey data such as gender and age, usability data such as how difficult the participant found creating a password, and a large amount of data about the participant's password storage and usage habits.

Preparing this data for WEKA required a combination of domain expertise and trial-and-error with the *development* dataset. Much of the data was removed, including any open-ended free-response survey answers. Other data was removed when I decided it would be very unlikely to be helpful for this project. This process involved thinking about the data, and examining the *development* dataset.

I also transformed the password itself to be more useful to WEKA. In the database, passwords are stored as strings. But I instead created eight attributes which contain the first four and last four characters of the password. This is possible because each password has at least eight characters. The last of these characters is the "class" of the instance for WEKA. This allowed WEKA to view these password characters as individual nominal attributes rather than viewing the password as a whole. I also created attributes for the character type (uppercase letter, lowercase letter, digit, or symbol) of each of these characters, except the last character.

The full set of attributes is presented in Table 1.

Table 1: Instance attributes

Name	**Type**	**Explanation**
	Study Behavioral Data	
condition	nominal	Participant condition
start_at_hour	numeric	0-23, the hour participant started
start_at_day	numeric	0-6, the week day participant started
did_finish	nominal	T/F, If participant completed whole study
num_failed_policy	numeric	# Failures creating a password
prrec_count	numeric	# tries required to recall password
	Password Data	
pwd_len	numeric	Password length
f1, f2, f3, f4	nominal	The first four characters in the password
l1, l2, l3, l4	nominal	Last four characters in password
tf1, tf2, tf3, tf4	nominal	Types of the first four characters
tl1, tl2, tl3	nominal	Types of 4th, 3rd, 2nd last characters

6 Data Exploration

I performed exploratory data analysis using the *development* dataset. This included manual examination of the instances, and seeking to understand how their final character could be predicted. In addition, feature selection was an iterative process, closely tied to exploratory data analysis.

To gain an initial understanding of what sort of performance I might expect, I ran several machine learning algorithms in WEKA to predict the last character of instances, using ten-fold cross-validation on the *development* dataset. This provided a sanity-check on my final feature selection, and helped confirm my choice of using *SMO* as my machine learning algorithm. I had hypothesized, before performing any tests, that SMO would provide the best results, based on observing its performance in classification tasks throughout the semester. The results of using different algorithms on *development* are in Table 2. The best results were obtained using SMO and feature selection on SMO.[1]

Feature selection with both SMO and J48 selected 15 attributes. In both cases, the top attributes were the seven known characters and character types, plus participant condition. Although feature selection on SMO did perform slightly better than SMO itself, the number of correctly classified instances was not significantly greater (χ^2 test, $\chi^2 = 0.0078$, df = 1, p-value = 0.9298). Thus in order to take advantage of having more data,

[1]Feature selection involved using *AttributeSelectedClassifier* with *ChiSquaredAttributeEval* and *Ranker* selecting the top 15 attributes.

Table 2: Results of using ten-fold cross-validation on *development*. All settings are default unless otherwise indicated. Feature selection involved using *AttributeSelectedClassifier* with *ChiSquaredAttributeEval* and *Ranker* selecting the top 15 attributes.

Algorithm	%Correct	Kappa
Naive Bayes	19.125	0.159
J48	19.9375	0.167
SMO	20.0625	0.169
Feat.Sel. w SMO	20.25	0.171
Feat.Sel. w J48	10.625	0.033

I chose to proceed using SMO rather than feature selection on SMO.

The next step in my exploratory data analysis was looking at the distribution of final characters in *development*. Among those 1600 passwords were 68 different final characters. The most common final character is "1," the final character in 7.6875% of passwords. The most common final characters are shown in Table 3. The whole distribution is visualized in Figure 1.

Table 3: The most common final characters in *development*

%Frequency	Character
7-8%	1
6-7%	3
4.5-6%	0, 9, 7
4.0-4.4%	5, 8, 6
3.5-3.9%	4, 2, n, !, s
3.0-3.4%	e, a, r, y

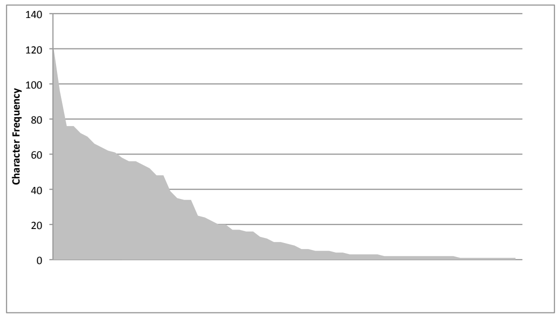

Figure 1: Distribution of final characters in the *development* dataset. The $y-axis$ indicates frequency among the 1600 total passwords.

7 Baseline Performance

I performed a baseline analysis using default WEKA settings. The models discussed in this section were trained on the *train* dataset, and used to classify the instances of the *test* dataset.

To gain a better understanding of the meaning behind the Kappa statistic, I began using the *ZeroR* algorithm, which assigns each instance to the majority class. ZeroR correctly classified 160 of the 1600 instances, for a 10% success rate. The Kappa statistic was, of course, zero.

I then used *OneR*, which used the second-to-last character to predict the final character. This lead to 20% of instances being classified correctly, for a Kappa statistic of 0.1619.

I next used SMO with the default setting $c = 1.0$. The result was 322 instances, or 20.125% of the total, being classified correctly. The Kappa statistic was 0.1689.

Next, I performed an error analysis on the

SMO classification. The massive number of classes made this challenging; the *test* data set contains 71 different class values, leading to a confusion matrix of over 5000 values.

The confusion matrix showed "1"s and "2"s scattered throughout with no discernible pattern, suggesting that there are many passwords whose final character follows no pattern at all. The more interesting observation I made is that many of the errors predicted a character of the same character type (uppercase letter, lowercase letter, digit, or symbol) as the actual character. This indicates that the algorithm may be able to determine at least the type of the final character in many cases when it cannot determine the character itself.

Given the apparent success of SMO in correctly predicting the character type of the final character in a password, I decided to explore this concept a bit further. I created a version of the *train* dataset with the type of the final character replacing the character itself. I used SMO with default settings to perform a ten-fold cross validation to predict that final character type. Of the 3200 instances, 2582 were correctly classified (80.688%). This resulted in a Kappa statistic of 0.6735. The greater success of predicting character class is not a huge surprise; there are many more possible characters than character types.

This error analysis, along with the inclusion of character types among the top-ranked attributes in my exploratory data analysis using feature selection, confirms my inclusion of character types among attributes. It appears that character types are an important structural component of passwords, which is consistent with the password-cracking model of Weir et al. [10], which I discuss in Section 2. This error analysis also suggests ideas for future work, which I will discuss in Section 10.

8 Optimization

My next step was to optimize SMO – to find the parametric value that would result in the best performance for the algorithm on my dataset. The parameter that I optimized was C, the complexity parameter. This value being higher may lead to SMO using more data from instances, but being too high may result in over-fitting.

I used *CVParameterSelection* in WEKA, with *train* for training and *test* for testing. I began by using five folds with SMO, and configuring parameter selection to use C values between 1 and 6, in four steps. The result was that $C = 1$ was the best value. The estimated the Kappa statistic for using a model thus tuned on unseen data was 0.1708.

Since the best value I tried was also the smallest value I tried, I decided to try smaller values of C, this time ranging from 0.25 to 1.0 with four steps. The most successful parametric setting was $C = 0.25$, leading to a Kappa statistic of 0.1957 for estimated performance on unseen data.

Since I once again had optimal performance with the lowest value in my range, I tried yet another, lower range in another optimization: with C values ranging from 0.05 to 0.25. This resulted in an optimal C parameter of 0.15, predicting a Kappa statistic of 0.1893 for performance on unseen data.

I then performed a comparison of baseline SMO performance to optimized SMO performance; this meant comparing the performance of SMO with C values of 1 and

0.15. With $C = 1$, 322 instances of the *test* dataset were correctly classified, and 1278 were incorrectly classified. With $C = 0.15$, 362 instances were classified correctly and 1238 were not. This difference in not statistically significant (χ^2 test, $\chi^2 = 2.8282$, df=1, $p - value = 0.09262$). The experimenter in WEKA indicated that the Kappa statistics were also not significantly different.

9 Final Result

To generate a final result, *train* and *test* were combined into a single training set. The testing set was *holdout*, which had been otherwise unused. Using SMO with $C = 0.15$, 421 of 1600 instances were correctly classified (26.3125%), for a Kappa statistic of 0.2282. Using 1R, 370 instances were correctly classified (23.125%), for a Kappa statistic of 0.1934. The difference between the numbers of correctly classified instances is statistically significant (χ^2 test, $\chi^2 = 4.1983$, df=1, $p - value = 0.04046$).

10 Discussion

This paper has explored using machine learning tools and techniques to predict the final characters of passwords. This has included data preparation, feature selection, baseline performance observation, error analysis, and optimization. Over one-quarter of the instances in the *holdout* dataset were successfully predicted by optimized SMO.

As shown in Section 2, much of the prior research work in password-cracking has dealt with guessing passwords without any prior knowledge of their characters. This research, on the other hand, has used machine learning to determine the final characters of passwords given their prior characters and other meta-data.

The value of this work is found in better understanding how an attacker might reconstruct a whole password, having observed part of it. This might occur, for example, during a shoulder-surfing attack. Moreover, this paper provides deeper insight into the internal structure of passwords.

Optimization of the C parameter in SMO indicated the optimal value being rather small: 0.15 rather than the default 1.0. The smaller this value, the less SMO will overfit the training data. The low optimized C value likely indicates that the internal structure of passwords can easily lead to overfitting the training data. This is likely due to many arbitrary patterns emerging within the training dataset that are not helpful in classifying the testing dataset.

Another interesting finding is the relative strength of *OneR*, the machine learning algorithm in which a single rule is used to classify instances. While its final performance was worse than that of the more complex SMO algorithm, OneR successfully classified over one-fifth of *holdout* using only the second-to-last character. This is an interesting finding because the concept of generating character probabilities using prior characters is found in related work, as discussed in Section 2. The Markov algorithm developed by Narayanan et al. [4] uses the prior character to predict probabilities for subsequent characters. And Shannon [7] used prior characters to determine subsequent character probabilities in his estimation of entropy in the English language.

The error analysis in Section 7 indicates that SMO is better at classifying passwords by final character type than by the final character itself. Future work may involve an analysis of using SMO to predict the final character types of passwords. Moreover, future work on classifying the final character in passwords may include more attributes pertaining to character types; this may include, for example, digrams of character types.

A limitation of this paper is that the algorithms used throughout this work are given only one guess to determine the final character of a password. Whereas, an actual attacker who witnessed all but the final character of a password being entered would likely have multiple chances to crack that password before being locked out of a system.

Finally, this paper is a class project. Future work may be to examine the results of this paper and consider whether and how they may be translated into a paper for publication. This may include a breakdown of instance prediction by password-composition policy, or the use of machine learning to predict multiple characters in a password at once. I would be interested in discussing this further.

References

[1] Matt Bishop and Daniel V. Klein. Improving system security via proactive password checking. *Computers & Security*, 14(3):233–249, 1995.

[2] Solar Designer. John the Ripper. `http://www.openwall.com/john/`, 1996-2010.

[3] Simon Marechal. Advances in password cracking. *Journal in Computer Virology*, 4(1):73–81, 2008.

[4] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 364–372, New York, NY, USA, 2005. ACM.

[5] Robert W. Proctor, Mei-Ching Lien, Kim-Phuong L. Vu, E. Eugene Schultz, and Gavriel Salvendy. Improving computer security for authentication of users: Influence of proactive password restrictions. *Behavior Res. Methods, Instruments, & Computers*, 34(2):163–169, 2002.

[6] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Comp. Comm. Rev.*, 5(1), 1949.

[7] C. E. Shannon. Prediction and entropy of printed english. *Bell Systems Tech. J.*, 30:50–64, 1951.

[8] R. Shay, S. Komanduri, P.G. Kelley, P.G. Leon, M.L. Mazurek, L. Bauer, N. Christin, and L.F. Cranor. Encountering stronger password requirements: user attitudes and behaviors. In *Proc. SOUPS'10*, 2010.

[9] C. M. Weir. *Using Probabilistic Techniques To Aid In Password Cracking Attacks*. PhD thesis, Florida State University, 2010.

[10] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek. Password cracking

using probabilistic context-free grammars. In *2009 30th IEEE Symposium on Security and Privacy*, pages 391–405. IEEE, 2009.

[11] Yinqian Zhang, Fabian Monrose, and Michael K. Reiter. The security of modern password expiration: An algorithmic framework and empirical analysis. In *Proc. ACM CCS'10*, 2010.