An Investigation of Machine Learning for Password Evaluation

by

Margaret Nicole Todd

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2016 by the
Graduate Supervisory Committee:

Guoliang Xue, Chair
Gail-Joon Ahn
Dijiang Huang

ARIZONA STATE UNIVERSITY

December 2016

ABSTRACT

Passwords are ubiquitous and are poised to stay that way due to their relative usability, security and deployability when compared with alternative authentication schemes. Unfortunately, humans struggle with some of the assumptions or requirements that are necessary for truly strong passwords. As administrators try to push users towards password complexity and diversity, users still end up using predictable mangling patterns on old passwords and reusing the same passwords across services; users even inadvertently converge on the same patterns to a surprising degree, making an attacker's job easier. This work explores using machine learning techniques to pick out strong passwords from weak ones, from a dataset of 10 million passwords, based on how structurally similar they were to the rest of the set.

ACKNOWLEDGMENTS

I would like to thank my family, friends, professors, mentors, classmates and colleagues who have helped me grow tremendously over the past few years. I am particularly indebted to and thankful for those who fit into more than one of the above categories, magnifying the impact of their support throughout this journey.

I am especially grateful for the support and encouragement from my advisor, Dr. Guoliang (Larry) Xue, and the other members of my wonderful committee, Drs. Gail-Joon Ahn and Dijiang Huang.

This work was completed under the generous support of Arizona State University's Scholarship for Service program, part of the National Science Foundation's CyberCorps initiative.

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A 16-character password with uppercase characters, lowercase characters, numbers, and punctuation marks should be "secure" in that it would take about 1 trillion years to crack using current computing technology and paradigms [1]. Unfortunately for this model, humans can remember roughly 7 pieces of information at once [2]. This means that there is a fundamental disconnect between the required entropy for truly secure passwords and the entropy the human brain can keep track of. Naive password strength checkers, meaning those looking only for diversity in character sets and length, may not be able to discern the usable but low-entropy password of "PassWord1234567!" and the unusable but truly computationally secure "amal-Xd8m*na7rYb" [3].

This is all the more troubling because when forced into using long computer generated passwords with real randomness, the average user is likely to write it down and leave it lying around for an attacker with physical access to use, which would obviously undermine the security of this type of password. An alternate coping strategy is to come up with one password perceived to be very strong and reuse it for every new account; this makes their high-value password even more valuable to an attacker and only needs to be compromised once to be compromised everywhere [4]. The passwords humans come up with and remember when given these length and character set diversity requirements end up conforming to a shockingly small set of patterns [5], which means that attackers have to simply add these patterns to their existing dictionaries to continue to easily crack a majority of users' passwords [6] [7] [8] [9]. A naive password strength meter, working from a checklist of requirements, may encourage users to create passwords that are harder for them to remember yet only a multiplicative increase in time for attackers to crack, instead of the exponential increase in attacker difficulty that would be desired in exchange for this additional cognitive load on the user. A password strength meter that could recognize and learn these patterns that emerge could be a valuable tool to combat this problem.

Background and Motivation

Passwords have some clear positives in their favor that imply that they will continue to annoy users indefinitely. For example, passwords are the best way humans currently have implemented for shared secrets, which is a concept that underpins both cryptography and authentication. In fact, of authentication schemes either proposed or implemented in at least small-scale, passwords are something of an unbeatable triple threat: they are simultaneously secure, usable and deployable. No other proposed mechanism can meet, let alone beat, them in all three categories simultaneously. The often overlooked aspect when proposing alternative authentication schemes is deployability: passwords are essentially free to implement and support, while biometrics, hardware tokens or graphical patterns require that each user have specialized equipment in order to authenticate to the service. Few services can get away with this kind of expensive change [10]. Passwords have an edge in that they should be user memorable because, in general, they are user-created. It is intuitive that the easiest string for someone to remember is a string that was composed by that person.

Unfortunately, passwords also have their challenges, many of them owing to the limits of human cognition. As alluded to above, there is a constant struggle between usability and security. Sometimes a proposed so-called improvement can even reduce one or both instead [11].

One 16-character random password already unnaturally strains the human brain, but there is evidence that humans can be trained to recall such strings. An experiment used spaced repetition to successfully teach strings made up of either six words or twelve characters to a set of users. The twelve characters had meaning for the researchers but likely appeared to be random to the subjects, so this technique might be able to be extended to sixteen random characters [12]. If everyone only needed a single password, this might be tenable; however, the actual demands on users far, far exceeds that. One 2007 study found that participants accessed an average of 25 accounts in a six-month window, and logged into eight of those per day [13]. Another study found that their average user had 27 accounts, accessing 11 per week; these participants reported having “between 2 and 15 unique passwords, with a median of 5 passwords” [4]. By necessity, assuming these two studies hold up when being generalized to the

rest of the population, this also means that the average user must be reusing passwords across services [14]. Your author took a few minutes to get a rough count of the demands on her memory and cognition by services requiring log-ins, and stopped counting after identifying over 50 unique services that require a password, with access frequencies of those services varying between multiple times each day to less than once per year. Humans might be able to be taught a rarely created but often used 16-character password, but only the most exceptional would be able to both recall and differentiate dozens of such passwords.

An interesting problem with passwords is that of poor internationalization. The Internet is implemented in languages and protocols that grew overwhelmingly out of the English language. This legacy carries on such that many password fields, even on non-English sites, are not set up to accept non-Latin characters, such as those that may be used by Mandarin or Cantonese speakers or speakers of languages using the Cyrillic or the Hebrew alphabets. Even alphabets that include accented Latin characters may not be fully supported, hampering many other native speakers of such languages [3] [15]. In these cases, it appears that patterns using numbers, such as dates and, in Chinese, rough transliterations become overwhelmingly popular. Very simple English phrases also become overrepresented in these sets, above even their overuse in English [16].

Users converge in not only the coping strategies discussed above, but in common password use and even password creation patterns and mangling rules [6] [17] [18] [19]. In some password dumps, the most common password in the set is used by nearly 5% of the accounts [16]. Over a quarter of users in one study in a condition where they needed to include a symbol simply added an exclamation point to the end of the password they had just attempted to submit [20].

Without explicit discussion on the point, users end up with the same mangling patterns in password creation and even semantically similar starting word sets, passphrase grammar, and even character choices. Generally, lowercase characters are most common, followed by digits; users rarely use uppercase or symbol characters unless required [20]. Users, as a population, have surprisingly stable character case placement preferences: an uppercase character is much

likelier to be the first character than any other character, while numbers and symbols are overrepresented as the last one or two characters [17]. Combinations of these preferences give rise to convergence on patterns [19]. Within character sets, there is convergence in individual character choices: for example, a digit is most likely to be a ‘1’ and some symbols are overused, like ‘.’, ‘_’ and ‘!’ [17]. Semantically, root words found in passwords conform to a small set of themes, such as love, animals, names, places, sexual terms and profanity [7] [20]. Naturally, this also means that the same root words end up being used in multiple passwords, which is a source of password collision. Passphrases, being longer than passwords, may seem stronger, but evidence shows that word frequencies in both passphrases and long passwords are different than they would be in English but are consistent across these populations [7] [21].

Coping strategies involving direct or modified reuse make this convergence problem worse and also effectively weaken a user’s password for every new service it is used on. Reuse can threaten even the strongest of passwords, particularly when considering attacks other than online or offline guessing attacks, such as keylogging and phishing. In this case, no matter how well-protected a great password is on many services, compromise once on any service could compromise every service [14]. On a related note, accounts utilizing weak passwords can only be protected by service administrators so much; even when they’re properly salted and hashed, they may be cracked offline from the hash or simply guessed in an online guessing attack [19]. An administrator’s best defense against this at least partly involves keeping those obvious but popular passwords from being used by any account on their system.

This convergence makes an attacker’s job easier. An intelligent attacker will start with the most common passwords used on any site that doesn’t blacklist them, which can be found in lists all over the Internet [22]. When investigating new breaches, the same passwords are found over and over again, used by many users at the same service. In addition to those identical passwords, predictable patterns that the users have converged upon can be seen in the data. These patterns will definitely be leveraged by savvy attackers trying to better tailor their attacks for passwords that they have high confidence will appear in any given password database; they

should be similarly leveraged by researchers and administrators to help users away from these passwords through blacklisting and intelligent strength meters [6] [19].

There are two tragedies of the commons at work in the password ecosystem as it stands today. This idea refers to a situation in which there is a shared resource and incentives of those using it such that they gain from overuse, until the resource is depleted for all users. The traditional tragedy of the commons involves a common cattle grazing ground, with the restriction that, while farmers can have their cows graze for free, they can each only have one. From each farmer's perspective, if he can sneak an extra cow in, his profits will double, while only negatively affecting every other farmer by what seems like a surely negligible amount – what's one more cow in a herd? Unfortunately for the savvy farmer trying to get ahead, every other farmer has the same set of incentives, so many others add a second cow, and maybe even a third, until the common land becomes overgrazed and is not able to support any cows, leaving both the farmers who were following the community rules and the rule-breakers with no cows and no profits. The shared resources in the password ecosystem that are being tapped too aggressively are user memory and user time.

User memory is simultaneously treated as limited yet inexhaustible. Few account registration or password reset pages require passwords containing dozens of characters, because that seems obviously absurd to require users to remember. Of course, even sites that do not really need users to strongly authenticate their identities may derive some small benefit from it, like better user tracking, and what is just one more password for the average user to remember when it can do so much better for the targeted advertising on the service? Bonneau and Preibusch segment services into high security services and low security services. High security services have reason to have users strongly authenticate and have strong security, in general, as a central requirement to their business. Examples of high security services could be online banking services, email and e-commerce sites with significant repeat business and therefore reason to not only process but also to store sensitive personally identifiable information of users such as legal names, addresses and credit card numbers. Low security sites, on the other hand, could be gimmicky single-use apps, news sites, and blogs that have the public sign in to comment or otherwise interact with the

content. There is little reason as far as the user is concerned for low security sites to require passwords of any kind, and they may not protect user data, including passwords, as stringently as a high security site would. And, as the low security site may think, what is the harm in just one more password for each user? Meanwhile, from even security-conscious users' perspectives, fatigue from trying to create and remember unique passwords for every service they interact with, even incidentally, eventually pushes them into only slight mangling or even simply direct reuse across services. From the attacker's perspective, low security sites may be poorly defended compared to high security services, but may contain passwords that are the keys to high security accounts. In this scenario, every actor is acting completely rationally, but users lose. The authors recommend a somewhat counterintuitive but very interesting partial solution to this equilibrium: low security services should require weak passwords. They will still have most of the benefits of having users authenticate for tracking purposes with perhaps a slight loss in accuracy due to compromised accounts, but users will not be able to reuse passwords across sites with different security postures and requirements, so there should be a net harm reduction in the ecosystem as a whole [23] [24].

It has been argued that user time is similarly not accounted for properly either. In an analysis delightfully titled "So Long and No Thanks for the Externalities," the writer shows that users are acting completely rationally when they choose weak passwords or reuse passwords across sites. The common wisdom is sometimes that user education is required to make users understand why strong passwords are needed, and why the definition of "strong password" is so hard to pin down; alternately, researchers simply give up and say that users act irrationally and against their best interests. In the work referenced, Cormac shows that because the monetary cost of breaches is still fairly low and as a rule, financial institutions with deep pockets have been set up to absorb almost all of this cost, the economy as a whole is harmed more by administrator-enforced password difficulty or complexity increases requiring small additional amounts of user time when logging in than the poor passwords and resulting breaches [25].

Some grand problem statements can be derived from this broad, systems-view literature review. For example, how can the problems of the password ecosystem be ameliorated? If that is

too expansive a scope, it can be narrowed towards fixing the problems inherent in a single service or system's passwords. There is evidence that strength meters can affect this positively: in the presence of a good strength meter, users choose to create longer and more complex passwords, even though they are not being required to [11] [26]. Narrowing the scope even further, how can a password strength meter algorithm be designed to learn and thus discourage common patterns, based on what it has seen in blacklisting and in practice? This question strongly implies an online machine learning algorithm, provided with good training data to start but also the ability to incorporate data as it is gathered. Two such algorithms are Random Forest and K-Means Clustering, but there is still a gap between this identification of a possible solution and its successful implementation. Random Forest requires labeled training data and K-Means Clustering requires the number of clusters to be defined ahead of time. What would these labels be, and how are they derived? How many clusters might there be, and what does identifying the cluster a candidate password belongs to say about the strength or weakness of that string as a password?

Problem Statement

This thesis investigates the application of Kernel Density Estimation (KDE) to explore the clustering of passwords in the space whose dimensions are defined by structural features extracted from each password, which are transformed and down-selected using Principal Component Analysis (PCA). The questions at the outset were those of precision, and also how this data pipeline would scale up if it were to be used as a part of a deployed strength meter. Evaluating accuracy, as opposed to precision, would require some ground truth understanding or definition of what makes a strong password; this remains something of an open question to fully define concretely, which is why precision is the chosen metric for some notion of correctness for this work.

Related Work

This project is in line with but set apart from current research on passwords. Other researchers are using Markov models to enhance password cracking tools and estimate the

number of guesses before a proposed password would be cracked [27] [28] [29] [30]. This project investigates using a broader set of features with the goal of creating a model that can see if a machine learning model can pick out inputs that "look like" previous passwords it has seen as well as recognize outliers, which should be stronger passwords that would not be easily broken from an attacker with knowledge of how passwords tend to be composed. Work has been done fairly exhaustively investigating algorithms behind strength meters and revealed that they need more deliberate design decisions and standardization across them [3]. This work is intended to be a move in the proposed direction. A peer pressure meter, showing users how strong their password was in comparison to previous users, was shown to influence users to create better passwords similarly to a traditional, absolute-strength meter, but without the potential risk communication problems that an absolute meter may cause [26]. This project is effectively a step towards a more comprehensive peer pressure meter, and could ultimately adopt similar messaging. A strength meter using a Support Vector Machine (SVM) has been developed and implemented into a framework [31] [32]. An SVM is a supervised learning technique that requires labelled data; it is not completely clear where they derived their labels from, and so it seems as though they are using the SVM to approximate another strength meter. This work investigates a different, unsupervised learning algorithm.

CHAPTER 2

APPROACH

This thesis investigates leveraging machine learning to, in the future, create a flexible password strength meter algorithm that can more accurately reflect the strength of a proposed password by taking not only length and diversity of character sets but also patterns it has seen into account. A similar password structure implies a similar pattern used in creation. Structure refers to the coarse case composition of the password. A pattern is the human strategy in creating the password. Similar passwords should be nearer to each other in the password space defined by the features used in this work. To illustrate these terms, take the passwords “Password1;” and “Dictionary5!”. They conform to the structure of an uppercase character, followed by a string of lowercase characters, then appended with a number and a symbol. They were both clearly created by taking the same pattern, which is taking a common word, capitalizing the first character, adding a single digit and a symbol to the end. An approach that matched password case compositions too finely could miss the identical strategy used in their generation because they aren’t the same length.

Statically blacklisting common words and patterns is one of the most effective measures an administrator can take to push users into making more secure passwords [33]. This is encouraging, but there is the potential of a blacklisting-and-reconverging arms race between administrators and users; the inadvertent convergence on the relatively small number of patterns humans choose without coordination when creating passwords under similar conditions suggests that such convergence may occur again if barred from the current handful of common patterns [18]. Additionally, a static blacklist is most effective against the developers who created it; their blind spots may let other weak, common passwords or substrings through.

This thesis aims to start to create a set of tools that can dynamically nudge users away from the common patterns to encourage a broader diversity of patterns, which would then also make it less and less feasible for attackers to crack passwords by predicting common patterns. If this reconvergence occurs again, a constantly-training, or online, machine learning model could continually encourage users to move onto new patterns. Encouraging users to fill

underrepresented sectors of the potential password-space would make it less tractable for attackers to discover and incorporate every pattern into their dictionaries. Regarding the usability aspect of this strategy, as users would be creating their own passwords still, instead of relying on purely random computer-generated passwords, so the hope would be that any given user would be able to come up with something that has up to 7 pieces of entropy for them yet looks to others as if it has greater entropy.

Collecting significant human interaction with the machine learning model is beyond the scope of this proposed thesis, and has been set aside as future research. Instead, the actual password data used will be a disclosure of 10 million passwords released by security researcher Mark Burnett in February 2015 [34] [35]. This is to balance data validity and the ethical concerns of the author. Obviously, passwords look more like passwords than any other possible category of strings, so research on convergence on the patterns underlying password creation can only really be performed on actual passwords. Generally, this type of research makes use of password dumps from various data breaches, such as Rockyou or Myspace [3] [8] [9] [17] [28] [30] [36]. This is common practice, with the ethical justification being that no additional harm is coming to the victims through white-hat research on the now-public list; attackers are leveraging these resources, so it is seen as leveling the playing field to do ethical password research using these same resources [30]. The author generally agrees with this justification for analyzing real password leaks, but the author's personal preference was to never have the identity of the service, the usernames or emails and passwords combined, and such single-service leaks necessarily contain all of this information; in contrast, Burnett's 10 million passwords include only usernames, scrubbed of any email associations, and the corresponding passwords, with no direct links to the service they were obtained from. Most password research on real passwords necessarily involves leveraging stolen data. This research can be a net positive, making future passwords and therefore systems safer, but this was not an aspect of this research this author wanted to take lightly and without the approval and supervision of a separate ethics body. Burnett's passwords are not individually linked to how he obtained them, and he reports that some of them were willingly shared. There may not be any meaningful distinction between

knowingly using a database of all stolen passwords from a single source as opposed to using a database of passwords, many of which were likely stolen from various sources, but this distinction helped the author sleep a bit easier at night. Unfortunately, the intended positive effect of using this corpus that is more anonymized and separated from the source than is standard practice comes with some downsides. Any analysis of this database could not be generalized to all passwords in general, because nothing is known about the password creation requirements or conditions; in fact, because the methodology for gathering and choosing these passwords is not particularly rigorously documented, the broadest generalizations that could be made would be about passwords obtained and released by Mark Burnett. (For character counts and some other simple metrics about this dataset, see Appendix A.) For the purposes of this work, this limitation is not a significant problem, but it does strongly imply that a near-term future work step would be to ethically obtain real passwords under known and controlled password creation contexts.

The data is processed by extracting features, particularly focusing on the structure, from each password, scaling those features using a Min-Max Scaler, using principal component analysis (PCA) to reduce dimensionality, and using kernel density estimation (KDE) to see how close a tested password is in feature-space to others, with the expectation that a password nearer to other passwords should be structurally similar to them, and therefore predictable to an attacker concerned with mining patterns known to be used to create passwords. This project uses Scikit-Learn as its library of machine learning algorithms [37].

This thesis sidesteps the quagmire of active research that is defining metrics for absolute password strength. This is a very difficult problem, because passwords are generally only strong against a given attacker with a defined approach to cracking passwords [11] [17] [20] [30] [38] [30] [18]. What may be a strong password against one adversary with one guess generation approach might be relatively weak against an adversary with a different guess generation strategy. For a simple example of this, an adversary trying to break short passwords might be utterly thwarted by a longer passphrase, but an adversary expecting passphrases and optimizing for them may find that same passphrase to be a trivial challenge. Except in the case where a password would only ever be generated by brute-force guessing, or cases where a password can

be found in an easily enumerable closed set, there is no agreed-upon absolute strength metric for passwords.

The features extracted from each password include the number of characters of each character class and some composed features looking at composition, the class of each character at each position, the ASCII number of each character at each position, the number of cases in each bigram and trigram, the number of case changes in each bigram and trigram, the distance between each character in each bigram and trigram, and the characteristics of the last character, bigram and trigram specifically. The five character classes are lowercase letters, uppercase letters, digits, symbols and whitespace characters; when four classes are referred to as the complete set of classes, whitespace is either included in the symbol class or is excluded from allowable characters. ASCII is a very common character encoding scheme. A bigram is two adjacent characters, and a trigram is three adjacent characters.

The features that are extracted are intended to capture only a limited definition of the structure of the passwords, which does not take into account semantics or string matching or repetition. This was to explore this approach's feasibility, with the idea that should it prove useful, it could be incorporated within a more robust feature set or pipeline that could cover these blind spots.

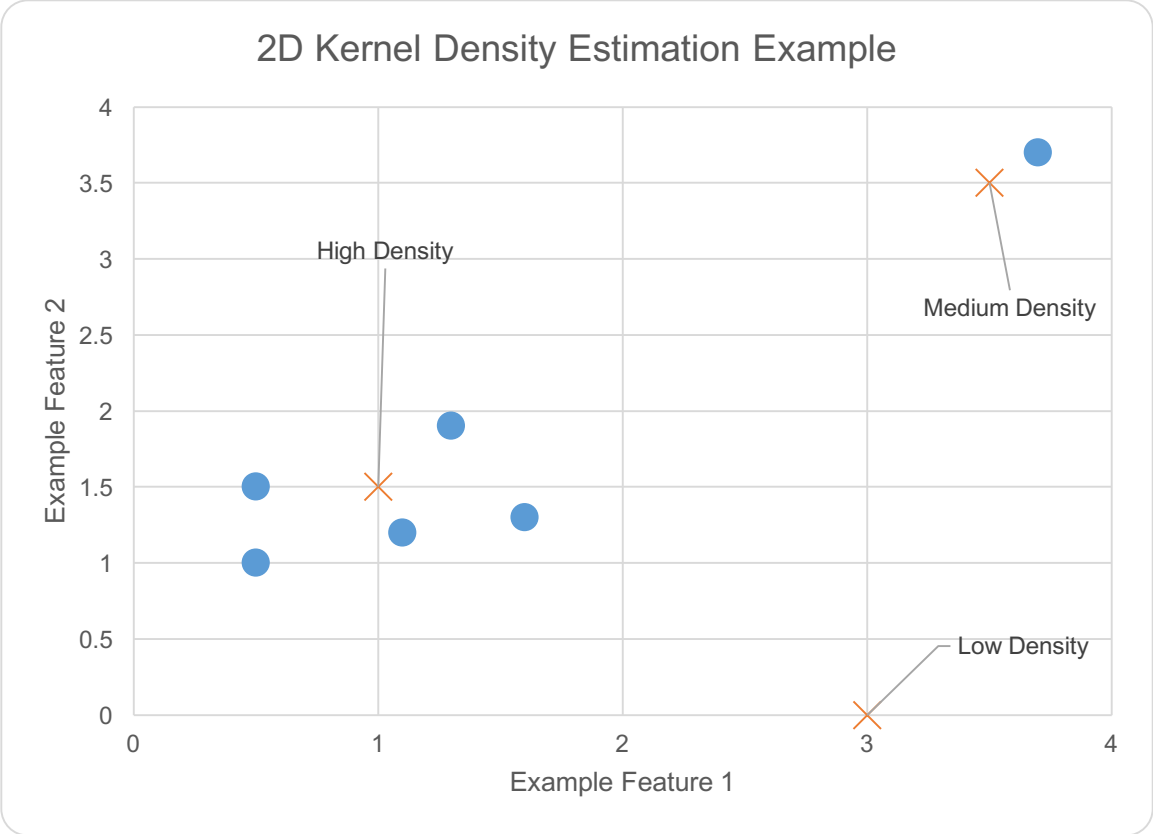

Figure 1: A Simple Two-Dimensional Toy Example to Illustrate KDE

KDE smooths the density distribution of the training set so that there are continuous densities at all points in space, even if there isn't training data that directly overlaps the space being tested. Training data fills the space and thus defines the density of the space, like the blue dots are doing in Figure 1. The orange Xes show the possible density measurements, qualitatively, that test data could get at that spot. KDE outputs a unitless number, which was used to compare the relative densities of different passwords in the testing set so that the testing set could be ordered.

Tests are performed against folds of the database that the model had not yet encountered. The different folds or cuts of the database are non-overlapping, so any repetition is true repetition of that password within the list as opposed to selecting the same instance of a password twice. The results of these tests are an ordered list of the testing set; this is further

simplified in this work by taking the top 10% of each test set as the best passwords of that set and the bottom 10% as the worst passwords.

It may seem strange to be using training and testing sets with an unsupervised learning method, but this was done very deliberately. This work is assuming there is no available ground truth on which passwords within a given group are strong or weak on their own, but that there are groups of pre-existing passwords and proposed new passwords. In this model, there is a sort of ground truth in the density of the feature-space of the existing list of passwords against which the proposed passwords are tested, without necessarily then being added. Given the imagined application, borrowing the existence of training and testing sets from more overtly supervised methods was deemed to be a useful abuse of the differences between data preparation for supervised and unsupervised learning.

CHAPTER 3

IMPLEMENTATION

This work began on a laptop with 16GB RAM and a 2.8GHz Intel Core i7 CPU. The machine learning pipeline used required significant amounts of memory, so to speed up the evaluation process, the work was moved to a server with 168GB RAM and 32 2.1GHz Intel Core i7 CPUs. Comparisons of performance are only ever done within tests performed on the same machine.

The following features are extracted from each password in the subset that is being worked on: ASCII number of each character, the bigram and trigram 'distance,' the number of character set changes in each bigram and trigram, the above bigram and trigram characteristics specifically for the last characters of the password, the length of the password, the total number of each character set in the password (including whitespace), the percentage of the password composed of each character set, and some logic to capture the combination of character sets used within each password.

ASCII table is a semi-arbitrary ordering, but so would any other ordered list that might be created for this. The ASCII table has symbols split apart, with some appearing before digits, others between digits and uppercase, more between uppercase and lowercase, and a few after the lowercase set. This could be improved upon. Using the ASCII table also gives the artifact that certain character sets are "closer" to each other than others, which is odd but would recur in any other ordering that could be created. The ASCII table does not contain all printable characters, so a move towards at least one encoding with broader character sets, such as UTF-8, would be a more complete version of this.

There is no string matching in this model. It is not explicitly set up to catch common words or repetitions of substrings. These are standard practice in password strength assessment, and could certainly be added as an augmentation of this approach.

Additionally, leet, l33t or 1337 transformations are not specifically accounted for. As they are used, they should become common in the training set, so should be in some sense learned automatically. Unfortunately, this would mean that there is a corpus of passwords in the training

set that were known a priori to be less complex than they might appear, so this approach to leet transformations would not be ideal in deployment of this algorithm. This is actually true of all known-predictable or known-weak passwords: they need to be included in the training set to be learned by the algorithm, or else their distinctive features, such as character equivalence mapping, would need to be explicitly included in the model. Including leet transformations may be a nontrivial matter, particularly given that some leet transforms are from one character to multiple characters, and so transforming back to the alphabetic representation from a given leet-enhanced password could be complex. Another work created a leet dictionary by transforming non-leet passwords in their other dictionaries; this is the more straightforward direction than checking an alphanumeric password for meaningful leet transformations, which would also require string matching to a blacklist or use of semantic tests [3].

Catching keyboard patterns would require including an explicit mapping of the keyboard in feature extraction. This would mean being able to record character distances as a function of their distance on a keyboard, which would be a useful measure for recognizing simple keyboard patterns [9] [36]. There may be a need to include multiple keyboard layouts, such as QWERTY, DVORAK, and international keyboard variations; common mobile keyboard layouts would also be interesting and possibly fruitful to capture in the feature set.

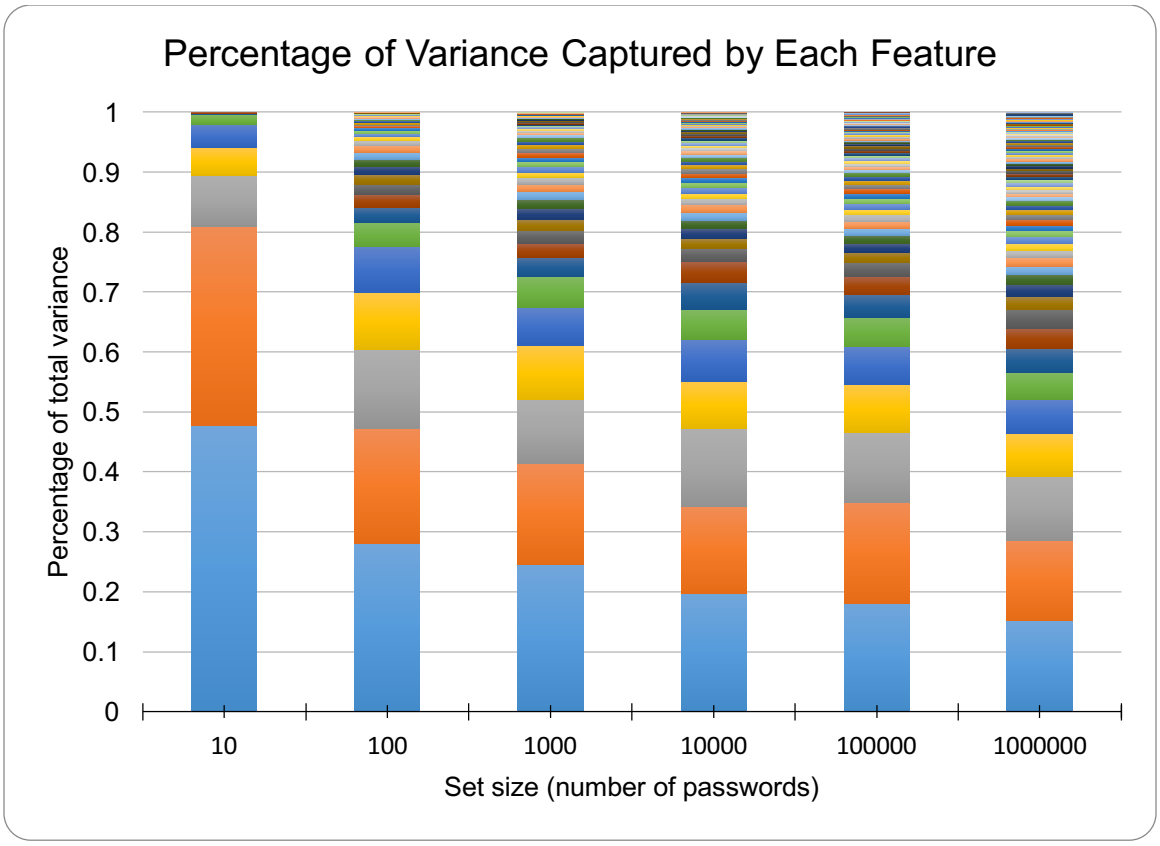

Figure 2: Percentage of Variance Captured by Each Feature After Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is applied to the original feature set to get a new feature set, ordered by the percentage of variance that each new independent feature can explain of the dataset. Figure 2 shows the results of PCA being applied to different batch sizes of passwords, where each colored bar corresponds to the amount of variance explained by a given constructed feature. In the 10 password batch, it makes a certain amount of sense that only 3 variables can differentiate the handful of passwords, explaining almost 90% of the variance among them. This trend continues as the batch sizes increase, with only five factors accounting for 50% of the variance in the 1 million password batch.

natalya.vildanovanatalya.vildanova	ei4qh753951nat8520123846jkl320	gkfytnfptvkzxfcnmyjvth2
chelseajapanesefightingmonkeys	elena.sergeeva.1962elena.sergeeva.1962	deadmanmarcelofabiano1
ssabitalouloulostack_paypa247	szhezhzhchvlapszhezhzhchvlap	w5Z9d.19377$sY3.17378
9619836847katyushka_ryzhova	739086277341921481351iza48920	jess4tsayescorpio_dota
rhbcnzdyfvbcnb1234567890poi	uyieryitydfgsdfdgjfgsgfsgfjsg	1q2a3w4s5e6d352052040159118
3MC3TCH6564464IYH64T6LIU3	angelfibirdkristibailey_75	p9*J@X#N8q4s78gWGM^W
230106VTwHuk260190SE880i	6kQz0dGX2yRMAwnrO4GJw63aQ	iloveyouaddpost.sulman
michellenewyorkislanders1	tiggermarcosantonionascimento	221.177.2007-03-1503
michelleoazariyt_roj0188f	jenniferisidor.eva_1989ja	markes_garsiamarkes_garsia
baseballmistermannumber1	killerhjivenko_alev1984ya	igordaniel1410199886831454
p1q2z4prmtuz34503xz5p2	pointuxer1212pointuxer1212	??????
qqqqqqqqqqqqqqqqqqqqqqqqq	letmeinspring_branch_baby	23903313bf593f573d8
familiengersusan_16guanzon	18001043shkola43gawnopozornoe	abc123TheEternalSpectator
hhhhhhhhhhhhhhhhhhhhhhhhhh	********	williammacc-anton99_1990
georgeabrahammartinez2008	tBPb2z7uCxlUJ6PaMTMs5	peppermarquito_pm030490
galichka1992matchenko1991	gadhavidhaval07081987**	queen_of_the_north_wind
******	michellerafa_benites_83	s0i9r8i7n6o5v4i3c2h1
passworsohapi2geder_100708	iloveyoujavier_becerra86	0998389003-INAgrujovic
internetthomasrosselli8	footballjarylkasymov_90	111111middlefinger_666

Figure 3: Examples of Some Passwords Found in Best Sets

142536	2111986	300367	110356
12101975	12041975	102166	224466
14121989	220263	23101965	1101974
17101986	124356	2111975	2121983
200101	4101978	145347	25121986
456456	112311	31121987	125565
12101996	346789	200366	223100
13011977	125224	200275	10201974
2011976	12011963	354565	131523
2323	22031978	20011984	102121

Figure 4: Examples of Some Passwords Found in Worst Sets

Some of the initial results are shown in Figures 3 and 4. The Worst sets are populated almost exclusively by digit-only strings. Given the way that the original features were extracted and the relative prevalence of this type of password, this was not unexpected. Of all the character sets, digit-only has the smallest password space, with only 10 characters, and the intercharacter

distance between them will be similarly small. Other research has shown that these types of passwords frequently correlate to dates in MMDDYY, MMDDYYYY, DDMMYY or DDMMYYYY format, with certain ranges overrepresented, the implication being that they are birthdates and other popular milestone events in users' lives, like anniversaries and holidays. The range of years is constrained from the mid-1900s to the current date, which narrows an attacker's search space even more. Many of these date strings should cluster from having many of bigrams in common, being composed of concatenations of "01" - "12", "01" - "31", "19" or "20", and "00" - "99" (with a concentration from "69" to "12", and now likely up to "16") [7]. Other common numerical passwords are just patterns, like "142536", "456456", "124356", and "112311" in the example group.

The order between the Best and Worst sets was explored as a check on the expected functionality of the data pipeline, but was not further interacted with. Above digit-only strings were lowercase-only strings, which are more common than digit-only but less densely clumped together because of varying distance between adjacent characters, less rigid structure and fewer repeated characters: '1' is present in nearly every password in Figure 4, and "19" and "20" are present with a high frequency; there is a nearly identical frequency of 'e' and 'a' to '1' at around 6% of the total characters each, with every other character appearing less than 4.5% of the time but no other bigrams can compare. Uppercase characters anywhere in the password ranked it higher, and the presence of symbols, even the most common symbols, guaranteed a place in about the top quartile of the passwords given that symbols of any kind make up only about 1.9% of the total characters.

The passwords in the Best sets, examples of which are shown in Figure 3, are clearly set apart by their uncommon lengths, but there is evidence that this is not the only feature involved in ranking these passwords as being the most idiosyncratic. These passwords are nearly all mixed-case, as would be expected, but the mixing of cases is a little more varied than the kind of patterns that have been discussed in the Background section. The most interesting and initially perplexing results are the single case passwords, which include 29 'q's, 28 'h's, 6 and 8 '*'s, and 6'?'s. Note that the group shown in Figure 3 may not be, and likely is not, representative of the

passwords in a single Best set, nor the population of all of the Best sets; these seeming outliers in particular were selected for the purpose of further analysis and discussion. Although this work does not use a defined universal metric for “strong” or “weak” passwords, it is probably safe to assume that these five highlighted passwords are not great passwords against many smart adversaries. They are very predictable, and because passwords should approximate “randomness” as much as possible so as to be only able to be brute forced by even the savviest attacker, these passwords fail that heuristic for what a password should look like, much more than some of the others included in this figure. Inspecting the structure of these passwords, it becomes clearer why they are placed in less dense areas of the feature-space. The lowercase-only strings are long, like the other passwords in the group; the symbol-only strings are short, but contain not just one or two symbols like some of the other Best passwords, at a maximum. These traits of length and more symbols than expected (and symbol-only) are fairly rare in the corpus being examined, as is the single character being repeated for the length of the password, making all intercharacter distances zero: this is another uncommon trait. These five may not be good passwords in practice, but they are structurally dissimilar to the rest of the passwords, so it makes sense that they are less dense areas of the feature-space; this simply shows that the feature-set chosen is differentiating but not sufficient to feed into a strength meter algorithm.

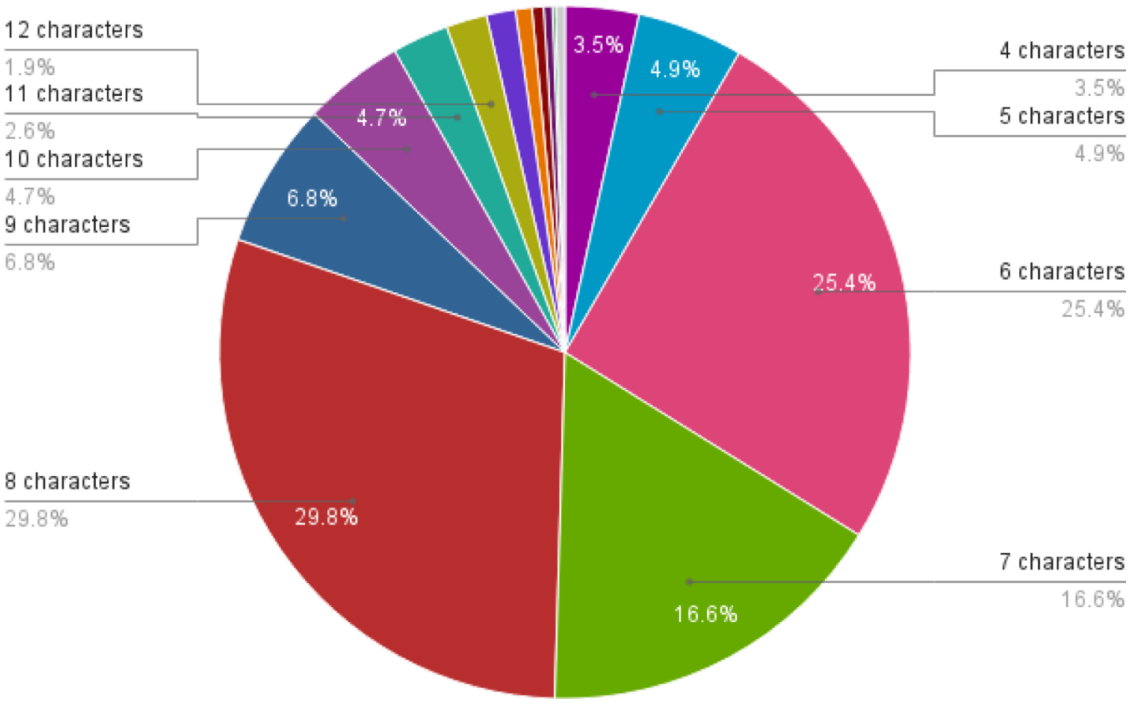

Figure 5: Breakdown of the Composition of the 10 Million Password List by Length

Because length appeared to be something of a dominating factor in the Best group shown in Figure 3, the decision was made to move to a single-length group for further testing to better explore the other factors in the feature set. To control for length while keeping, ideally, the most variety in the training and testing sets, the largest single-length group was identified and used. As shown in Figure 5, the largest group of single-length passwords was the eight-character group at 2.98 million passwords.

knFTeKmT	9FLrLCtU	dax68rue	hjad9Tq2	w7VkPr8T	pV7U8BzS	z6iwqpr8	UvEhEtOK
aGB17SC5	WarranT0	vfNmn6du	CANYON96	QP33OWIO	FeB44r6N	8YKtzChX	5590947!
NJr5Zh5a	7LaJwrSV	0q9a6567	DKstN2xT	C5hqkMq0	RR9852m4	TAKiTdfQ	NEVIe728
40po7gyf	OnexuDUT	CrwhJz7T	y2tD3h3x	26g2lma3	andr_ish	3PJVa7M8	MfniCIh1
1984.cta	FFWpower	4Z8zTIeR	7yV7thiv	53M6n5Lt	jjjB03b3	aG5PQI2E	J5SCNY85
JR87A6ex	czN2mZDQ	yPAEGNZN	RhetPvYk	CruFETkT	ch5riss$	WBV2jJGx	7EQRK6JV
2dPy8rgv	3LHE7n8N	hejywa7j	X7DCCA6c	te92dahs	2F6Jvpde	EAgDy5EG	NOEmlrDg
BpondiQQ	55GtPY5h	v25v79zf	Jordan_1	EFXyuN4H	KvacEP3i	Bugab01!	q2j8nKez
544mPSmF	M7th7z2r	AasqAtV2	9jaFYGhK	toe3mn14	7xF4t2hN	Aa2626Va	F9pFD3Qb
xCU8BpOo	mANg3yyz	ia2silwy	RR9Sq6pC	2B93HD87	5My8p6Vo	FmVeb4sd	#1marine

Figure 6: Examples of Some Passwords Found in the Eight-Character-Only Best Sets

24051992	22061961	8082002	33015072	68021322	12031971	17081951	14081973
22121965	96745831	98766789	1101988	22838430	29635238	19121973	32358100
11111122	16488726	25011953	95387389	24700031	12051983	10101996	10091983
19041996	13793346	4081976	31081983	56701452	49613758	19890713	25032003
14237464	12121996	23021999	61221101	12102004	12011978	18011952	14260712
21011987	19940125	18121998	40312580	27071966	13058822	19750820	83021111
19802310	26101982	13051996	3051972	31071940	2041971	8082003	15111979
16925422	22101960	21111972	2091939	10250780	18061976	24021993	7092004
14091995	26071953	42424242	24081966	28102006	19051988	20072006	11081963
36651116	6021955	15328897	19461946	10111775	14789321	16101946	31323132

Figure 7: Examples of Some Passwords Found in Eight-Character-Only Worst Sets

Some examples from the Best and Worst sets in the eight character-only tests are shown in Figures 6 and 7, respectively. There is little change in the Worst sets, as the passwords included appear again to be overwhelmingly dates and patterns. An effect of limiting the length was that some of the Worst sets started including lowercase-only strings that were particularly overrepresented in the training and testing sets, such as “superman”, “password”, “baseball”, “poohbear”, “jennifer” and “corvette”. The Best sets look as expected, separating out the passwords that contain two or more character sets, with distinctive compositions. Some of the passwords shown have some semantic meaning that makes their structure more predictable, like “#1marine” and “Jordan_1”, but that is not unexpected, because none of the features were chosen to segment, detect or bias against meaningful strings. The addition of such features or separate subroutines to the data pipeline would aid in the goal towards a future comprehensive strength meter. Note that any examples shown in Figures 3 and 4 that are eight characters

should still appear in the Best and Worst sets when testing and training is limited to eight character passwords only, even if they are not explicitly shown again in the mostly illustrative example group shown here. From non-systematic manual inspection, the segmentation of the testing sets appears to be working as envisioned, even after controlling for length.

CHAPTER 4

RESULTS

Analysis

Although this work sidesteps the definition of absolute strength of a non-random password, a comparison against a well-respected strength meter algorithm, zxcvbn by Dropbox, was performed against all Best and Worst sets [39]. The approach taken by zxcvbn is to segment passwords into different atoms and determine the time it would take an optimal attacker to break each one. This gives a sort of average worst-case bound, in that the hypothetical attacker already knows the best approach and dictionary size to break a given password but has to check about half of that optimal dictionary. Some weaknesses of zxcvbn are that it falls prey to the limitations of any other hardcoded dictionary or blacklist, and will miss words or mangling that are not explicitly included in it, and that it cannot handle non-ASCII characters, such as accented alphabetic characters or many symbols. It is intended to be able to be deployed client-side, and so is intentionally light-weight and, by necessity, not fully comprehensive. The average crack time, as given by zxcvbn, for all Best set passwords is 3.1 million seconds, or about a year; the average crack time for all Worst set passwords is around 150 seconds, or 2.5 minutes. The optimal attacker strategy is shown in zxcvbn results, and many of the Best passwords have no better cracking strategy than brute force, while many of the Worst passwords would be cracked essentially instantly with the use of pre-enumerated dictionaries such as dates or common English words. No significant deviations were found from these numbers based on the number of dimensions used in generating the sets.

Because the output from the pipeline is two sets, there needed to be some way to quantitatively compare them across different conditions, particularly because this thesis does not tackle absolute strength metrics of passwords. Small changes to the training and testing conditions were found to change the order of passwords in the Best and Worst sets, but this was deemed too fine and fiddly a metric, particularly given the imagined application of this data pipeline, a strength meter. That strength meter would not report that a password is the #3

strongest or weakest password it has ever encountered, but would instead report a strength category that password falls into. The overlap of the Best or Worst sets, given some changed variable, is more applicable to measure than some investigation of small, local rank reversals. The Jaccard index or coefficient, which is the cardinality of the intersection of two sets divided by the cardinality of the union of those sets, is the metric that is used in this section. A Jaccard coefficient of 0 would mean that there are no shared members in the sets, because their intersection is empty; a Jaccard coefficient of 1 would mean that all members in the sets are shared, because their intersection is equal to their union. The Jaccard coefficient underreports the amount of overlap, because the input is in the form of lists, which may have repeated members; although the training and testing sets are thus misnamed, the Best and Worst sets are truly treated as sets for the purpose of this evaluation. This overlap arises extremely rarely in Best sets, but is very common in the Worst sets: “12345678” could be in one training “set” six times, but would only be reported once in the Worst set.

The 2.98 million eight character passwords were divided into 10 non-overlapping cuts, and these cuts were each used as a training set against each other cut as the testing set, with the number of features, or dimensions, varying from 8 to 64, in increments of 8. This range of dimensions was chosen because for the eight-character group, 95% of the variance was captured by the first 16 features, 98% was captured by the first 24 features, and 99% variance was captured by the first 30 features. The goal was to include at least one test below this range and a few beyond this range. The Best set and Worst set were the top 10% and bottom 10% of the testing set; because of repetition in the testing “set” list, the Worst set always had fewer members than its corresponding list form would have had. In the 10 million passwords, there are fewer than 5.2 million unique passwords; in the 2.98 million eight character passwords, there are a slightly lower proportion of unique passwords, with fewer than 1.54 million unique passwords in that subgrouping.

Three variables were tested to better understand this data pipeline, with only one ever changing at one time. The three variables investigated were training data, testing data, and the number of features, which are also interchangeably referred to as dimensions because they

define the space under investigation. Set agreement within a given test is measured by Jaccard coefficient and is reported as the minimum, average and maximum values encountered in 5760 comparisons for testing and training data and 5040 comparisons for the number of dimensions.

To examine the importance of specific testing data, the training set and number of dimensions was held constant while the testing data was varied. If two unrelated test sets had overlap in their Best sets, this would imply that this technique is not working, which would be unexpected. It would mean that the same password happens to be repeated in the unrelated and non-overlapping testing sets, but that the training data showed it to be fairly idiosyncratic compared to the rest of the corpus. If this happened with any significant frequency, that would either mean there is a problem with this approach or else that the amount of training data was insufficient to adequately cover the password corpus in feature-space. On the other hand, overlap in the Worst sets would imply that the diversity of passwords in the dataset, or at least in the eight-character subgroup of the dataset, is awful. Frankly, this is expected to the point that if there were no overlap, it would again raise the concern that this approach is not working. Passwords were repeated in input lists with enough regularity that Worst sets were always shorter than 10% of the input testing list -- the number of times a password appeared in the list was recorded and showed that some passwords, like “password” showed up thousands of times in each and every cut of the data. As such, they should always have some strings that match, and these shockingly frequent strings should always fall to the bottom of the ordered list, into or very near the Worst set. Quite a bit of overlap in Worst sets is expected, if the data pipeline is ordering the testing set as intended.

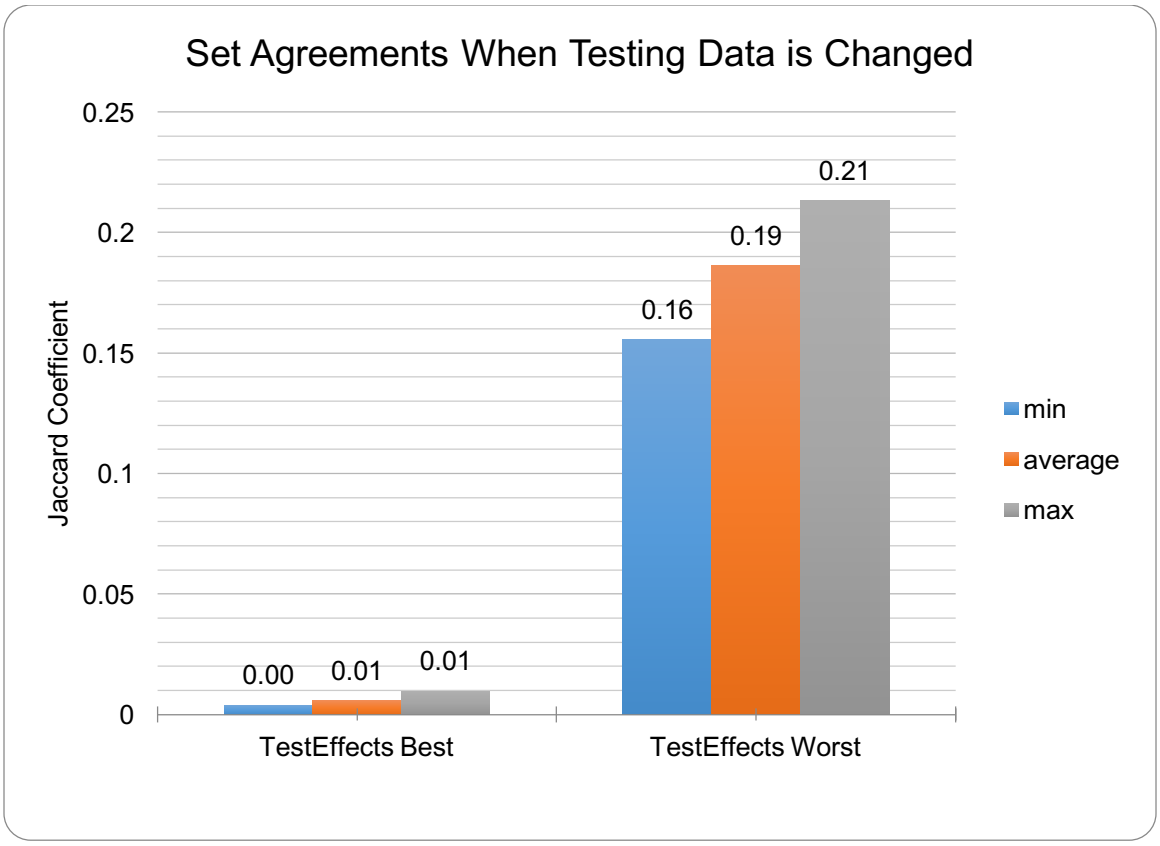

Figure 8: Set Agreements When Testing Data is Changed

The results from varying testing data is shown in Figure 8. There is a very small amount of overlap in the Best sets of unrelated testing cuts with an average Jaccard coefficient of 0.01, which is potentially troubling. Again, this means that there are passwords that are being repeated in both test sets but are being placed in a low-density area, which should mean that they are structurally idiosyncratic or rare. This is an obvious problem, the extent of which should be investigated further through careful manual inspection, which may lead to the creation of additional features that more fully characterize the convergence that is leading to these passwords being created repeatedly yet being reported by this technique as having rare structures. Mixed case, semantically meaningful passwords, like those leveraging leet transformations, or mixed case keyboard patterns may be some of the culprits. As expected, there is quite a bit of overlap in the Worst sets, which again, is actually the lower bound on the

amount of overlap between the corresponding lists of testing data. Overall, these results seem to show that the pipeline is mostly working but that the feature set is incomplete, which was already known or that the amount of training data is insufficient, which will be explored a bit more in the next test.

To examine the importance of specific training data, the testing data and the number of dimensions is held constant while the specific training data is varied. More overlap in the test set's Best and Worst sets implies less sensitivity to the exact training data. No generalization can be made about the quality of the training data, because training data is effectively being randomly selected from the same dataset, which means that each training set should be roughly equal quality by any metric. Poor overlap may mean that a larger training set is needed to better fill out the feature-space, that truly higher quality (by some metric) training data is needed, or that there is a fundamental problem with the approach that is being taken. The effect of the size of the testing dataset was never explicitly evaluated.

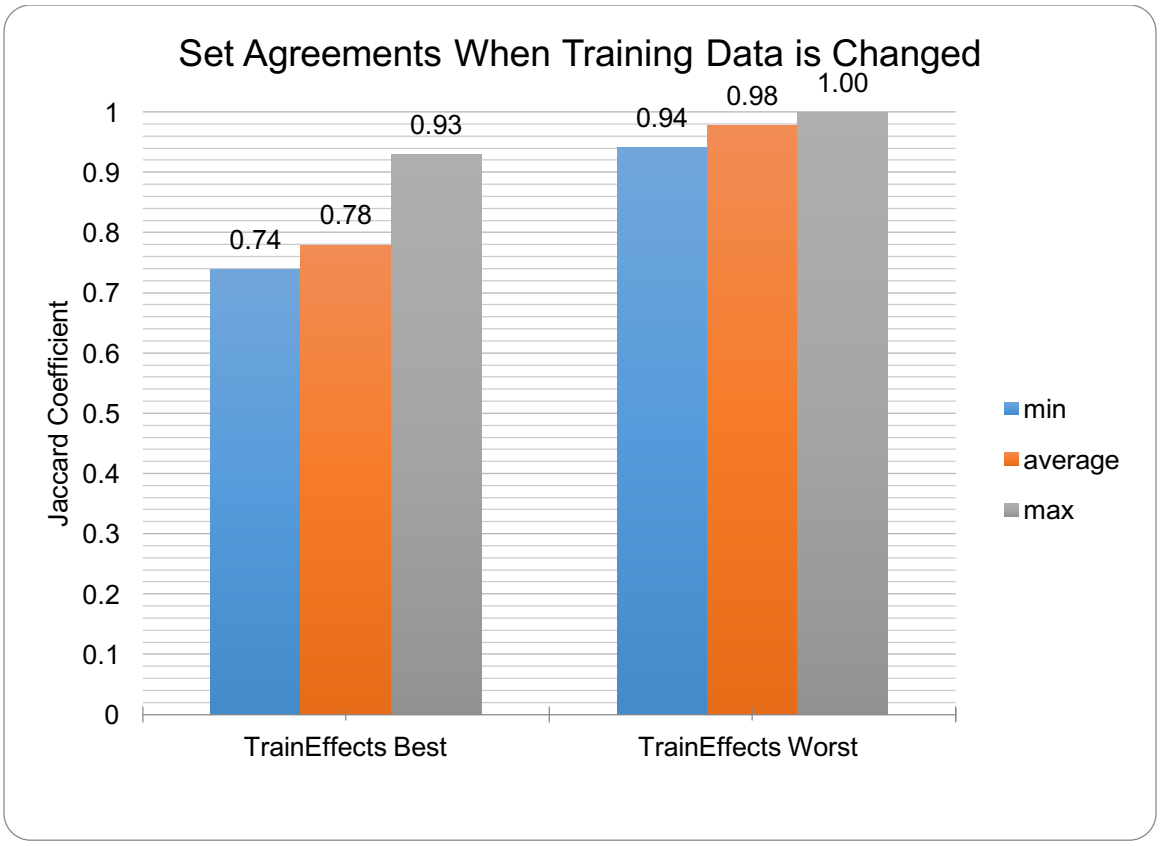

Figure 9: Set Agreements When Training Data is Changed

The results from varying the training data is shown in Figure 9. Changing the training data can change the Best set by around 22% on average. The Worst set is actually quite resilient to the change. This is especially interesting to see given that, with the strength meter application, one would conceivably be more concerned with weeding out weak passwords as opposed to positively and accurately identifying the strongest passwords. An additional test varying the size of the training set would be interesting, but was not performed in this analysis.

To examine the importance of the number of features used, the training data and the testing data is held constant while the number of features is varied. Achieving more overlap with fewer dimensions means that there is little or no need to spend the time and space required to use more dimensions. It would also mean that the structural differences between passwords can be captured in very few discriminating features, at least as chosen for this work. The hope is for some dimension reduction from the 195 features that are extracted from each eight-character

password, if only for time and speed savings, but also to have convergence happen at a relatively high number of features, because that would show that the features selected actually matter in looking for uncommon password structures and patterns.

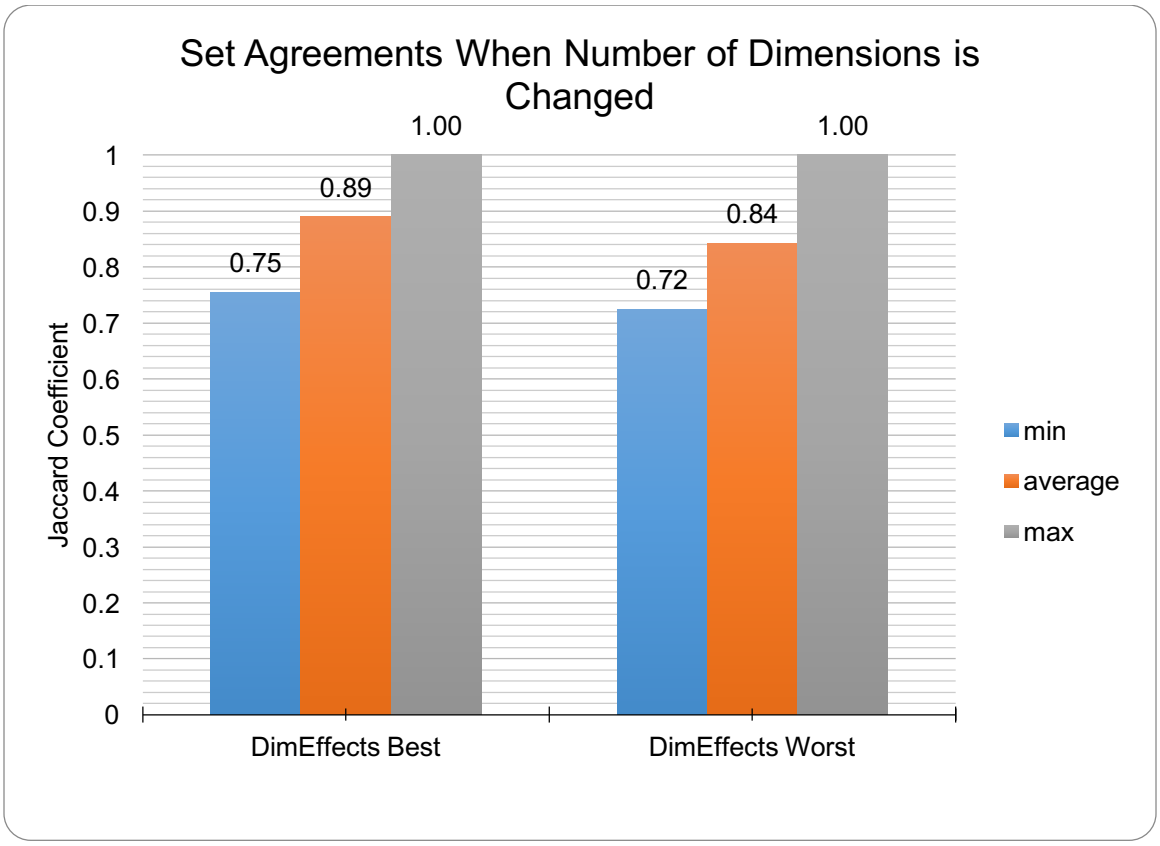

Figure 10: Set Agreements When Number of Dimensions is Changed

The results from varying the number of dimensions is shown in Figure 10. This is a coarse look at the effects, which can be broken down to look at the changes that happen with each feature set size. Both the Best and Worst sets are pretty sensitive, and similarly sensitive, to the number of features.

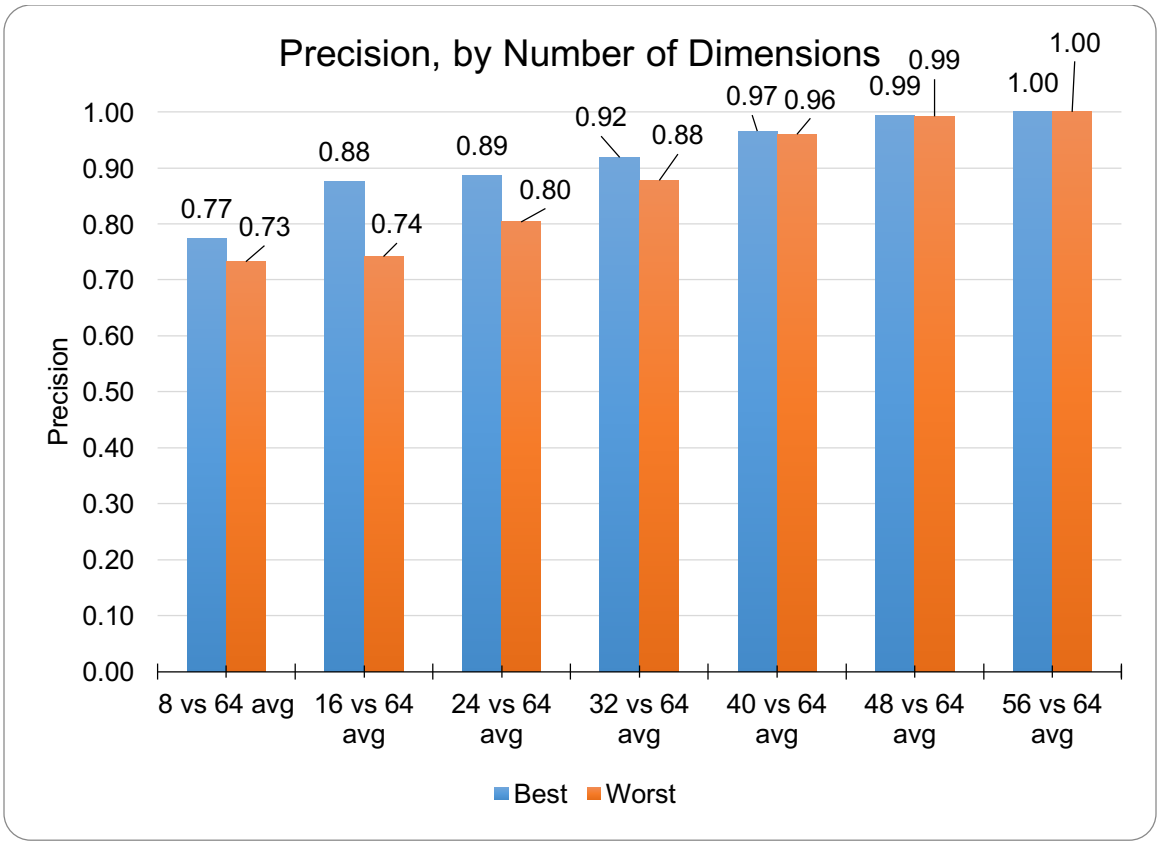

Figure 11: Precision, by Number of Dimensions

A closer look at the effects of changing the number of dimensions can be seen in Figure 11. Some metric of accuracy would require that the “right” answer for the Best and Worst set compositions are known; the best that can be managed in the absence of this “right” answer is instead precision. For this comparison, the sets obtained with 64 features is the goal, and the other sets are compared against them. Of course, as more features are used, higher precision is achieved on both the best and the worst passwords, with identical results achieved with 56 features as 64 features. The additional eight features take more time and space to include but do not make any difference in the final result; based on the lack of apparent perturbations these last eight features seem to make, it may even be extrapolated that any changes in the Best and Worst set after 64 features would be small in general while adding quite a bit of time and space requirements.

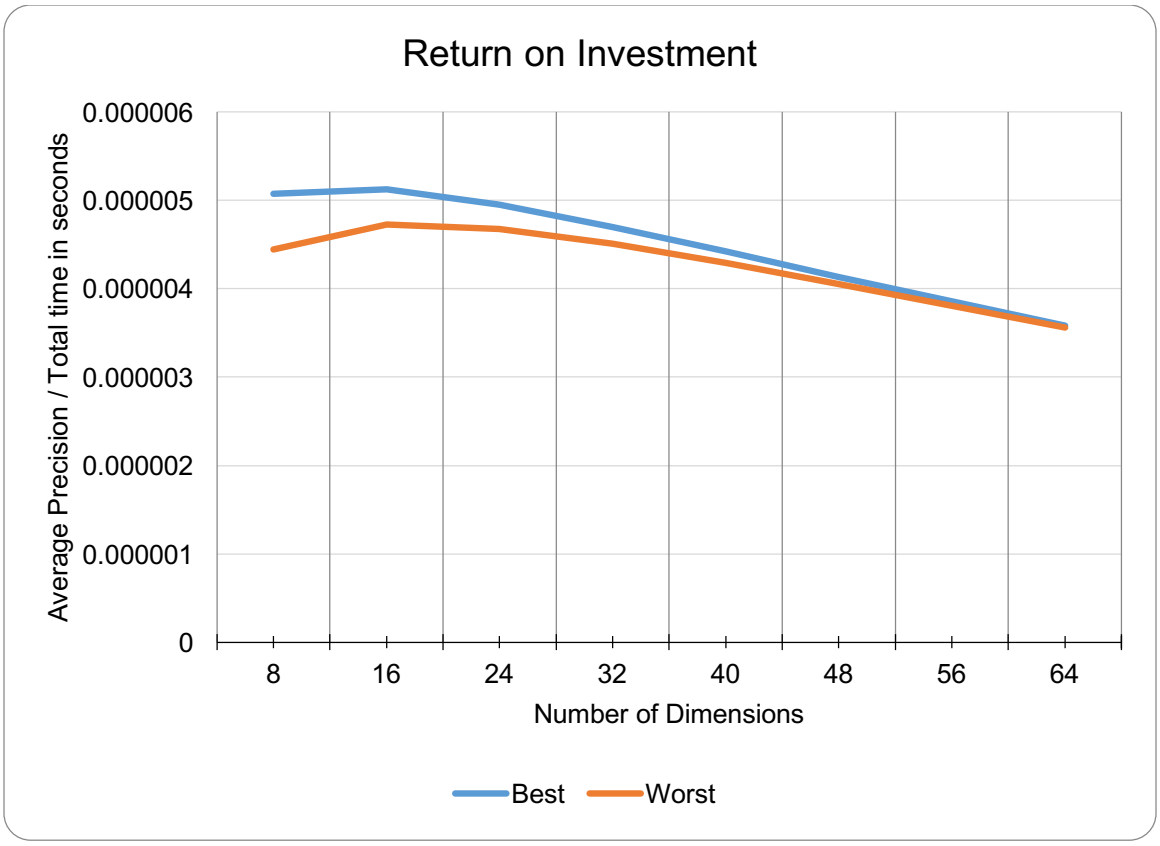

Figure 12: Return on Investment of Different Numbers of Features

After some point, there should be diminishing marginal returns on adding more of these features to the pipeline. (Other, additional features not explored here might certainly make more of a difference if they were added, but this analysis is only on this feature set.) This is explored in Figure 12, which shows the return on investment for each feature set. The return on investment is comparing the total time it took to achieve some level of precision with respect to the Best and Worst sets that would be obtained with 64 features. The inflection point, which is where the marginal ROI starts to decrease, is around 16 features for both the Best and Worst sets. This seems early, because only about 80% of the sets' members are the same as would be obtained with 64 features. An assumption going into the ROI analysis was that the inflection point would be closer to 40 features, which had previously intuitively appeared to be a good balance of precision and time during testing. Clearly though, the increase in time does not offset the additional precision beyond 16 features as compared to the ROI achieved by moving from 8 features to 16

features. The actual inflection point could be more precisely found by varying the feature set size from 9 to 23 with increments of one, because this analysis with increments of 8 may be too coarse to achieve the best marginal ROI for this dataset and these features.

Turning to the scaling of this pipeline, it was quite time- and memory-intensive. The development and testing began on a machine with 16GB of RAM, but had to move to one with 168GB of RAM because some of the tests simply required more memory than the original machine had. Use of virtual memory on the laptop had slowed the performance considerably above training and testing dataset sizes above 10,000 passwords. Time scaling was investigated in more depth, and these results are below.

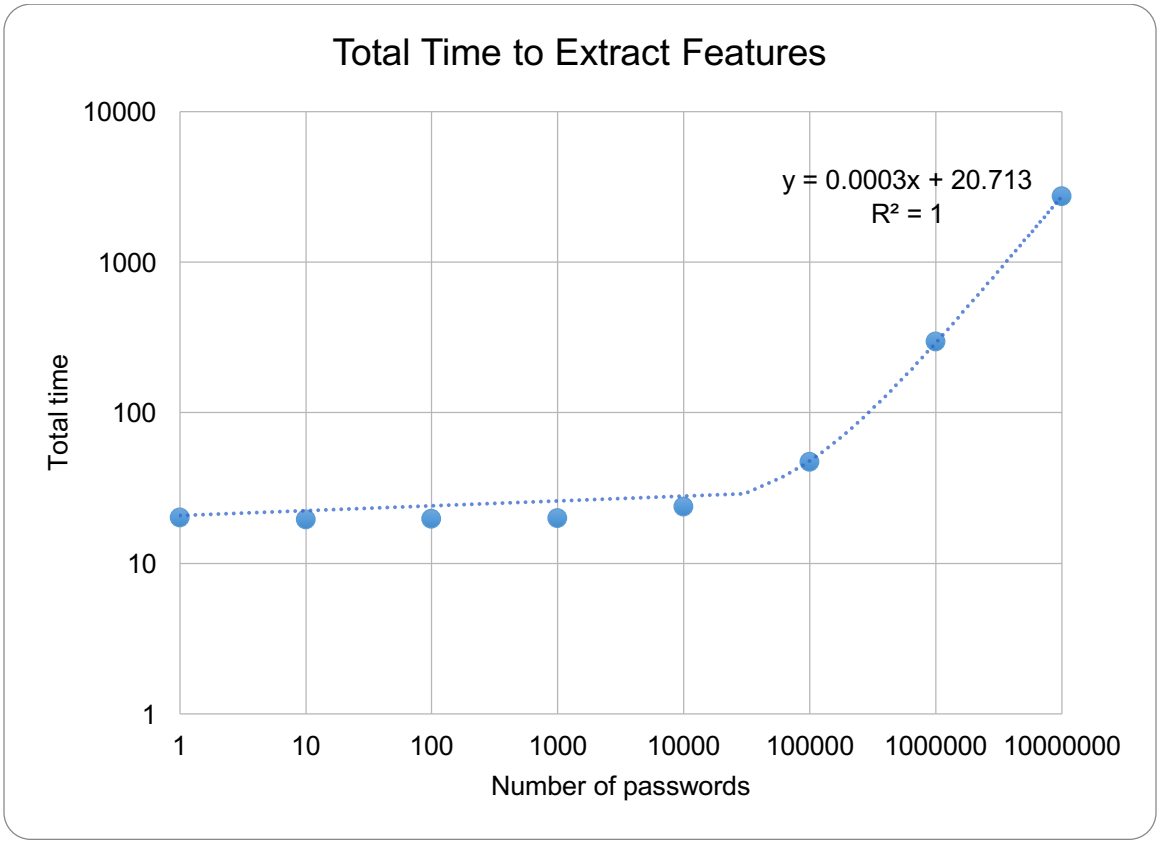

Figure 13: Total Time to Extract Features

Feature extraction, as implemented, involves opening a file to read a section of the 10 million passwords and then writing the features of the passwords of that section to another file, in CSV format. These I/O operations are the dominating time sink involved in feature extraction until about 10,000 password batches, where it becomes clearer that it takes about 300 microseconds (mcs) to extract features from each password. This is shown in Figure 13.

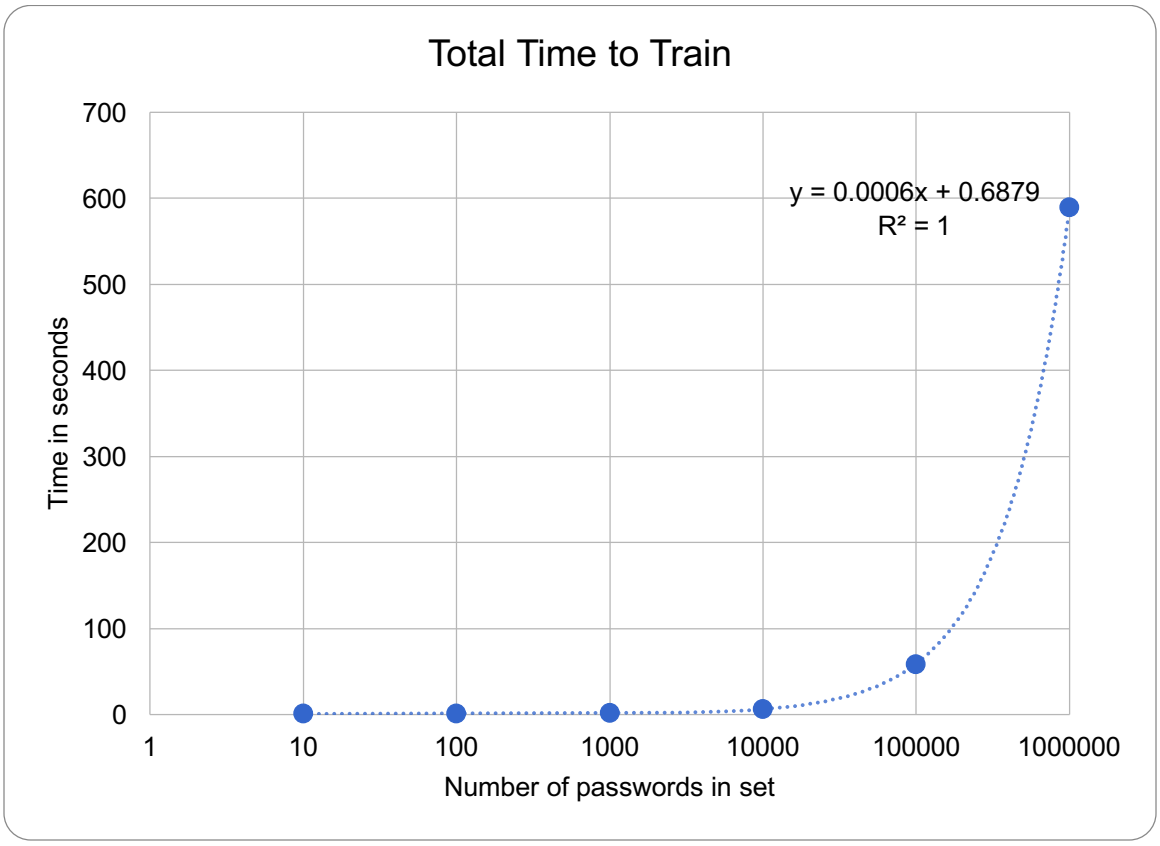

Figure 14: Total Time to Train

Using Principal Component Analysis and training the KDE algorithm on the training dataset also scaled linearly with the number of passwords being trained, taking around 600 mcs per password. This can be seen in Figure 14.

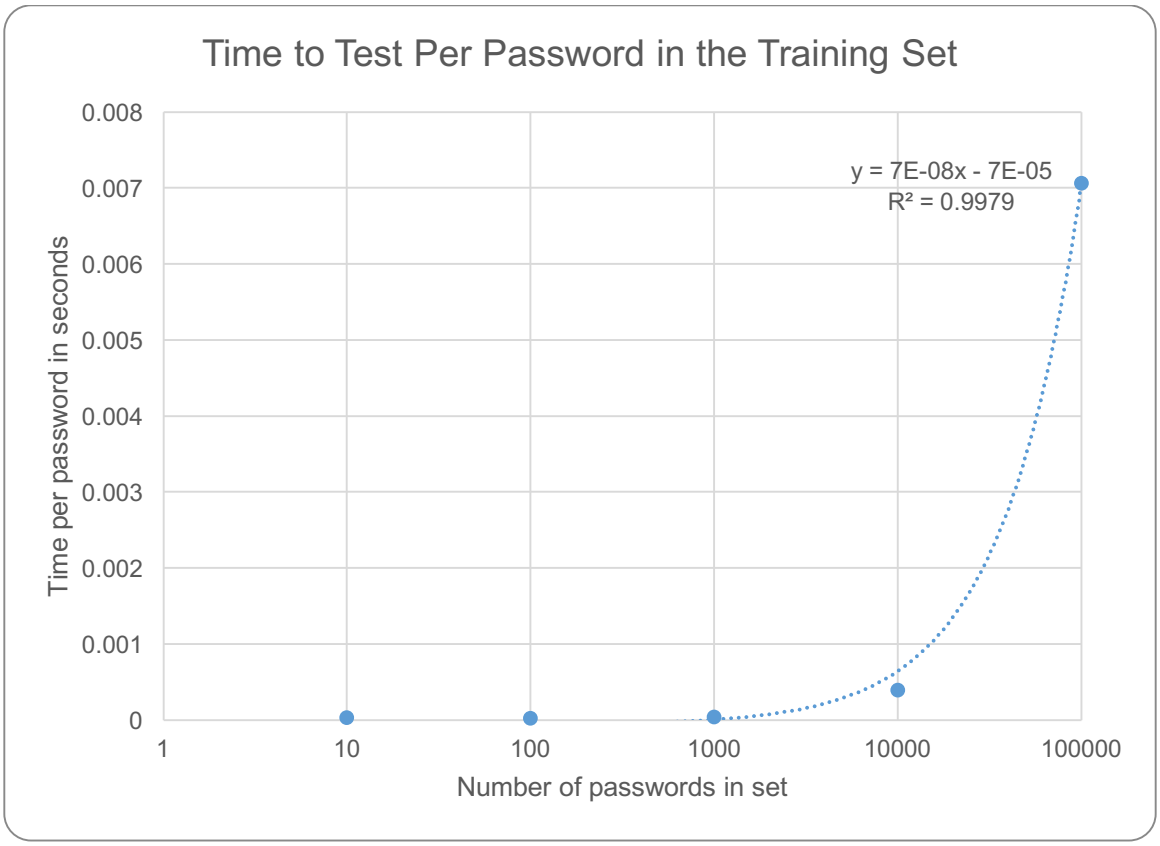

Figure 15: Time to Test Per Password in the Training Set

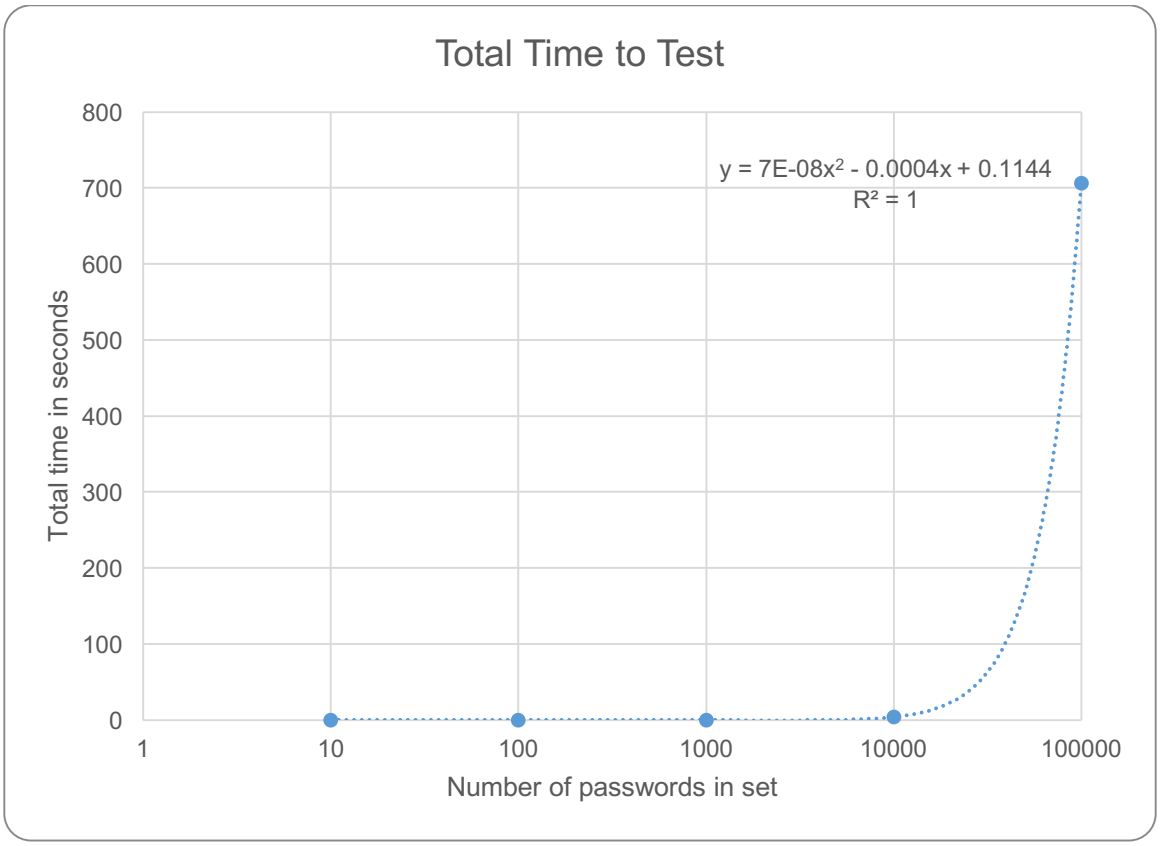

Figure 16: Total Time to Test

So far, total time had scaled linearly with the number of passwords, which was fairly ideal. In a divergence from this pattern, time to test per password increased linearly with the training batch size, as shown in Figure 15. This means, as shown in Figure 16, that the total time to test increases with the square of the number of passwords in the training set. These tests had been done originally with equally-sized training and testing sets, but further testing not shown here revealed that the variable driving this increase was indeed the number of passwords in the training set, not the number of passwords in the testing set.

Figure 17: Total Time to Train and Test Large Batches, by Number of Dimensions

Previous scaling comparisons involved runs with the batch sizes of the datasets changing but the number of features held constant. Total time with constant batch sizes but varying dimensions was recorded separately, and this increases linearly with the number of dimensions, shown in Figure 17. Runs with training and testing sets of just less than 300,000 passwords added about 2300 seconds per eight additional features. This data is messier because of variable load on the shared machine, but the trend is clear.

Overall, this data pipeline is $O(N^2 * M)$, where N = the number of passwords in the training set and M = the number of dimensions used. Unfortunately, this means that it does not scale as nicely as would be needed for the algorithm underpinning a strength meter. Because N is the number of passwords previously seen, the hypothetical strength meter would slow unacceptably over time as it accumulated new data that it would have to be periodically retrained on. The unsuitability of KDE as the algorithm underpinning the strength meter was already

known, because it is not an online algorithm, but this is further confirmation that KDE would be a step in the development process towards the strength meter algorithm as opposed to a prototype of the algorithm itself.

Discussion

Part of the reason this data pipeline was time-consuming was that it was implemented on traditional computing architecture using CPUs instead of leveraging the speed with which graphical processing units (GPUs) can process machine learning algorithms. A next step would be to move this investigation to one or more GPUs to speed up the entire process. This step would not improve the poor scaling of this particular pipeline, and KDE in particular, because that is an algorithmic constraint, but it would improve the multi-day run times of some of the tests that were run.

Other near-term future work on this data pipeline is to add additional features or algorithms to cover the gaps already noted, such as string matching for both blacklisting and catching repetition, keyboard patterns, leet transformations, and other such password creation or mangling patterns.

The development of labels or identified clusters is another clear next step, so the use of Random Forest or K-Means Clustering could be transitioned to. This is partly a usability issue, a risk communication issue and a technical question. There could be the concern that giving users feedback that a given password is “strong” or “excellent” might give them too much confidence in it, and ignores broader problems about reuse essentially weakening passwords [3]. Of course, the strength meter in one system cannot necessarily capture the problems of reuse across multiple systems or even exhaustively test for every conceivable attack pattern to properly deem a given password as excellent; therefore, an approach that is aimed at culling clearly weak passwords but not necessarily confirming that a tested password is strong may be sufficient and be closer to the truth. Studies have suggested that users focus more on not being weak rather than pushing to get a strong password [11]. If a system had the fidelity to report strength in very small increments, such as 0 to 100 (which still presents its own risk communication problems), it

would be interesting to see where users stop and decide a password is good enough to proceed; this kind of experiment could feed back into the design choices for the strength meter.

An interesting route to explore could be a strength algorithm that approximates guess numbers as used in studies such as “Guess again (and again and again)” [30]. Computing guess numbers requires high performance computing, terabytes of storage and days of time. Regression learning might be able to give a good enough approximation of given adversaries’ guess numbers that it could become a lightweight replacement or estimator. This could be very useful both in reporting scores to users in the password creation process as well as in evaluation of passwords obtained from experiments or in evaluation of other password strength algorithms.

This dataset, which, again, may not be representative of passwords in general, seems to support the idea that the best users can do is about seven discrete chunks of information together. This was shown in the best passwords chosen from the 10 million password dataset when not controlling for length: some of the passwords in the least dense part of the password space as measured by the features chosen were those that were long but with that length padded by variations of “password,” dates, and other particularly common chunks. This presents an interesting approach to cracking longer passwords that might appear to be stronger in some way than shorter passwords, but might actually be comparable: if an attacker assumes 7 unknowns, where those unknowns could be words, letters, dates, leet substitutions, and other such “chunks” that have been discussed, and combines them together, these passwords that are long but still lack in complexity could be assessed to be much weaker than they initially seem. The scaling and efficacy of this password cracking approach is, of course, beyond the scope of this work but remains identified future work.

The placement of the pieces of the future strength meter is also something of a struggle from a security perspective. It was not directly designed, implemented or tested in the course of this thesis, but some evaluation of the idea itself was attempted. There are, unfortunately, readily conceivable security concerns with the envisioned strength meter. There would be many potential avenues for information leakage to attackers, both in transit and at rest. The strength meter could leak information to malicious users in the course of normal use, which may give insights to

attackers about the corpus of passwords it has seen; stealing the algorithm's training data could result in a plaintext password dump. These issues have been raised and are in consideration, but no solutions are in a state to be reported here. A study facing some similar challenges did show that one could thoughtfully add dummy information without diluting the usefulness of the algorithm, which could be a partial solution to the security issues raised above [28].

In addition to the security challenges this meter would present, there are performance and networking issues to work out as well. Running the meter's computation as a server side meter obscures the details of the algorithm and the data it is using from observers. Per Kerckhoff's Principle, this is far from sufficient for security, but it remains an appropriate and acceptable defensive implementation decision. This placement would also increase usability, given that resource-intensive computation can take place on an architecture specifically designed for it as opposed to on whatever device the user is accessing the service from; particularly given the growth of the Internet of Things and ubiquity of mobile devices, the fewer assumptions about the capabilities of any given user's available resources made, the better.

On the other hand, purely server side computation may require quite a bit of superfluous communication between client and server. A hybrid approach can reduce the load on the service's server as well as constrain the amount of traffic required in the password creation and checking process. A simple set of requirements can be easily checked on the client's side, such as length of the proposed password, number of character sets, and whether or not the password matches a short blacklist of extremely common passwords or password patterns, as is done by many other strength meters as the sum total of their tests. Any password that does not pass these baseline compliance checks does not need to be sent to the server for evaluation, because the answer of its inappropriateness will already be known. Only after a candidate password succeeds against these minimum requirements would it be transmitted to the server and assessed, reducing overall network traffic and load on the server as compared to the purely server side computation approach.

Going beyond this particular approach and corresponding envisioned strength meter, another question arose during this work. The following question was posed to the author: given

this feature set, what would different password generators' results look like with this data pipeline? Would they be clumped together because of a similarly unpredictable structure, or would they be uniformly spread out because none of them look alike? Would the results of multiple generators reveal strategies, patterns and preferences embedded in those pseudorandom generators? Of particular personal interest to the author, when a population of users is presented with a set of possible passwords from a generator, which can be regenerated indefinitely, with the instruction to pick one of the passwords that has been generated to use as their high-value password, are there any shared biases or preferences in selection that weaken the population of chosen passwords as compared to the population of passwords as generated? In an extension of this machine-generated and user-chosen approach, what if users are provided with a randomly generated password but are able to make a limited number of edits to it? This could possibly enhance usability in that it would be able to be made more memorable for that user, but if users make only a small set of conforming edits, it could weaken the corpus of such passwords as a whole. Investigations of these questions are planned alongside further development and evolution of this data pipeline for password evaluation; symbiotic co-validation of both sets of hypotheses and approaches could be worked towards.

There was previous mention of ethically gaining a corpus of real passwords under controlled conditions, requirements and password creation and use contexts. This will be done by leveraging Mechanical Turk, as is the current state of the art for such studies; the ecological validity of these password corpuses are not identical to those of real services' password corpuses, but it is acceptably close and ethical. A much broader and more diverse base of users can be reached than the pool that would be involved in lab studies, with more valid passwords than are obtained in lab studies, and paying thousands of MTurk workers on the order of a dollar each for password creation, password recall and user sentiment studies is a frankly stunning return on investment for a human subject study [11] [18] [20] [30] [33] [38].

Returning to the complex set of problems within the password ecosystem, initial partial solutions were conceived of, partly as future work and partly to better understand where this work might fit into such a solution. An open source toolset for password management and

administration for low and high security services has been envisioned, with appropriate protections and practices for each built in, to make consistency and good security easy for service operators of both kinds of services while keeping passwords from being able to be reused across the categories of sites. The future strength meter that was imagined as the proximate application for this data pipeline or future versions of this pipeline could provide user feedback when creating a password for a high security site or when creating a master password for a password manager. It may also be able to validate pseudorandom password generators and check the choice of and alterations made to passwords pulled from those generators.

CHAPTER 5

CONCLUSION

Broadly, this work was born of interest in how users might be encouraged towards idiosyncratic password creation patterns. Exploring this led to investigating one set of machine learning techniques as possible components to a future password strength meter. The features selected, which focus on the structure of each password, appear to be promising attributes for clustering structurally similar passwords and letting structurally unusual passwords stand out, but is not a covering or sufficient set of features for distinguishing good passwords from bad, which in itself is an unsolved question. Kernel density estimation also showed interesting results and will continue to be used in the aim of developing features or understanding meaningful clusters for a future comprehensive and intelligent strength meter algorithm, but lacks multiple traits required of that algorithm, such as scaling appropriate to the application and the ability to continually learn from new data.

REFERENCES

[1] M. Wales, "How Secure Is My Password?," Small Hadron Collider, [Online]. Available: https://howsecureismypassword.net. [Accessed July 2016].

[2] G. A. Miller, "The magical number seven, plus or minus two: some limits on our capacity for processing information," *Psychological review,* vol. 63, no. 2, p. 81, 1956.

[3] X. D. C. D. Carnavalet, *A Large-Scale Evaluation of High-Impact Password Strength Meters,* Concordia University, 2014.

[4] E. Stobert, "The agony of passwords: can we learn from user coping strategies?," in *CHI'14 Extended Abstracts on Human Factors in Computing Systems*.

[5] J. Fox, "The Big Password Mistake That Hackers Are Hoping You'll Make," 16 October 2014. [Online]. Available: http://www.huffingtonpost.com/jeff-fox/the-big-password-mistake_b_5995208.html. [Accessed September 2016].

[6] E. I. Tatlı, "Cracking More Password Hashes With Patterns," *IEEE Transactions on Information Forensics and Security,* vol. 10, no. 8, pp. 1656-1665, 2015.

[7] R. Veras Guimaraes, *An investigation of semantic patterns in passwords,* University of Ontario Institute of Technology, 2013.

[8] V. Rafael, C. Collins and J. Thorpe, "On Semantic Patterns of Passwords and their Security Impact," in *NDSS*, 2014.

[9] H.-C. Chou, H.-C. Lee, H.-J. Yu, F.-P. Lai, K.-H. Huang and C.-W. Hsueh, "Password cracking based on learned patterns from disclosed passwords," *International Journal of Innovative Computing, Information and Control,* vol. 9, no. 2, pp. 821-839.

[10] J. Bonneau, C. Herley, P. C. Van Oorschot and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *2012 IEEE Symposium on Security and Privacy*, 2012.

[11] B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. L. Mazurek, T. Passaro, R. Shay, T. Vidas, L. Bauer and N. Christin, "How does your password measure up? the effect of strength meters on password creation," in *21st USENIX Security Symposium (USENIX Security 12)*, 2012.

[12] J. Bonneau and S. Schechter, "Towards reliable storage of 56-bit secrets in human memory," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[13] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proceedings of the 16th international conference on World Wide Web*, 2007.

[14] A. Das, J. Bonneau, M. Caesar, N. Borisov and X. Wang, "The Tangled Web of Password Reuse," in *2014 Network and Distributed System Security Symposium*, 2014.

[15] J. Bonneau and R. Xu, "Of contraseñas, סיסמאות, and 密码: Character encoding issues for web passwords," in *Web 2.0 Security & Privacy (W2SP'12)*, 2012.

[16] Z. Li, W. Han and W. Xu, "A large-scale empirical analysis of chinese web passwords," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[17] M. Weir, S. Aggarwal, M. Collins and H. Stern, "Testing metrics for password creation policies by attacking large sets of revealed passwords," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010.

[18] S. Komanduri, R. Shay, L. F. Cranor, C. Herley and S. Schechter, "Telepathwords: Preventing weak passwords by reading users' minds," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[19] R. Redman, "Supercharged Jack the Ripper Techniques (presentation only)," in *Spring 2011 OWASP*, Austin, 2011.

[20] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin and L. F. Cranor, "Can Long Passwords Be Secure And Usable?," in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, 2014.

[21] J. Bonneau and E. Shutova, "Linguistic properties of multi-word passphrases," in *International Conference on Financial Cryptography and Data Security*.

[22] M. Burnett, "10,000 Top Passwords," 20 June 2011. [Online]. Available: https://xato.net/passwords/more-top-worst- passwords/#more-269.

[23] J. Bonneau and S. Preibusch, "The Password Thicket: Technical and Market Failures in Human Authentication on the Web," in *9th Annual Workshop on the Economics of Information Security*, 2010.

[24] S. Preibusch and J. Bonneau, "The password game: negative externalities from weak password practices.," in *International Conference on Decision and Game Theory for Security*, 2010.

[25] C. Herley, "So long, and no thanks for the externalities: the rational rejection of security advice by users.," in *Proceedings of the 2009 workshop on New security paradigms workshop*, 2009.

[26] S. Egelman, A. Sotirakopoulos, I. Muslukhov, K. Beznosov and C. Herley, "Does my password go up to eleven?: the impact of password meters on password selection," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2013.

[27] J. Ma, W. Yang, M. Luo and N. Li, "A study of probabilistic password models," in *2014 IEEE Symposium on Security and Privacy*, 2014.

[28] C. Castelluccia, M. Dürmuth and D. Perito, "Adaptive Password-Strength Meters from Markov Models," in *NDSS*, 2012.

[29] 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*, 2009.

[30] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor and J. Lopez, "Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms," in *2012 IEEE Symposium on Security and Privacy*, 2012.

[31] K. S. Jamuna, S. Karpagavalli and M. S. Vijaya, "A novel approach for password strength analysis through support vector machine," *International Journal on Recent Trends in Engineering,* vol. 2, no. 1, pp. 79-82, 2009.

[32] G. Suganya, S. Karpgavalli and V. Christina, "Proactive password strength analyzer using filters and machine learning techniques," *International Journal of Computer Applications,,* vol. 7, no. 14, pp. 1-5, 2010.

[33] R. Shay, L. Bauer, N. Christin, L. F. Cranor, A. Forget, S. Komanduri, M. L. Mazurek, W. Melicher, S. Segreti and B. Ur, "A Spoonful of Sugar?: The Impact of Guidance and Feedback on Password-Creation Behavior," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 2015.

[34] J. Biggs, "A Security Researcher Just Dumped 10 Million Real Passwords," 10 February 2015. [Online]. Available: https://techcrunch.com/2015/02/10/a-security-researcher-just-dumped-10-million-real-passwords/. [Accessed September 2016].

[35] M. Burnett, "Today I Am Releasing Ten Million Passwords," 9 February 2015. [Online]. Available: https://xato.net/passwords/ten-million-passwords/. [Accessed November 2015].

[36] H. C. Chou, H. C. Lee, C. W. Hsueh and F. P. Lai, "Password cracking based on special keyboard patterns," *nternational Journal of Innovative Computing, Information and Control,* vol. 8, no. 1, pp. 387-402, 2012.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[38] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay and B. Ur, "Measuring password guessability for an entire university," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013.

[39] D. L. Wheeler, "zxcvbn: Low-budget password strength estimation," in *Proc. USENIX Security*, 2016.

APPENDIX A

CHARACTER COUNTS IN DATASET BY PASSWORD CASE COMPOSITION

The following section is a character count of every password from the 10 million password dataset. Passwords are additionally broken down by case composition, where spaces are separated out from the rest of the symbol set. Any non-ASCII characters are put into the password set; this includes accented and superscript alphabetic characters. Additional metrics are the number of passwords falling into each composition set as well as the number of characters in those passwords and their average length. A grayed-out cell shows a character that does not fit into the class composition of that column; a 0 shows a character that fits into the class composition of that column, but was not seen in the dataset. Across the columns, the relative sizes of the number of passwords, the number of characters in those passwords, and the average length of those passwords is denoted by a blue bar partially filling the cells, relative to the maximum of each of those metrics: the number of passwords and characters in “All” and the average length of “Digit, Space + Symbol”. The green bars down each column show the relative proportion of each character in that column, again with the maximum count represented as a full bar and every other bar as being the relative proportion. The legend, corresponding metrics for “All”, and the column names and aggregated metrics are printed on every page for ease of reference. The spreadsheet is converted to pages such that the pages for each column (case composition classes) are together, but the pages for each row (characters) may be separated. For example, the character counts for “Lower” can be found on pages 50 to 53; the character counts for the letter ‘a’ can be found on pages 50, 54, 58, 62, 66 and 70.

	All	Lower	Upper	Digit	Space	Symbol
# passwords	*9,997,985*	3,824,547	109,258	2,035,160	0	607
# characters	*75,916,675*	27,228,063	768,877	14,221,036	0	3,414
Avg length	*7.59*	7.12	7.04	6.99	N/A	5.62
'a'	*4,765,881*	2,781,120				
'b'	*1,394,757*	794,688				
'c'	*1,629,304*	950,821				
'd'	*1,705,096*	972,885				
'e'	*4,117,197*	2,545,795				
'f'	*1,019,573*	553,422				
'g'	*1,246,221*	717,978				
'h'	*1,445,050*	843,951				
'i'	*2,901,135*	1,781,152				
'j'	*677,976*	327,909				
'k'	*1,341,596*	712,147				
'l'	*2,288,102*	1,394,123				
'm'	*1,732,512*	972,506				
'n'	*2,825,016*	1,697,131				
'o'	*3,032,936*	1,874,353				
'p'	*1,150,607*	666,764				
'q'	*320,816*	103,334				
'r'	*3,093,749*	1,872,128				
's'	*2,757,228*	1,666,565				
't'	*2,330,857*	1,417,111				
'u'	*1,394,794*	796,310				
'v'	*769,297*	375,330				
'w'	*720,790*	365,917				
'x'	*445,369*	184,415				
'y'	*1,178,415*	632,684				
'z'	*529,710*	227,524				
'A'	*223,322*		66,764			
'B'	*134,408*		24,482			
'C'	*133,753*		28,731			
'D'	*133,401*		28,540			
'E'	*174,236*		61,219			
'F'	*109,205*		17,548			
'G'	*113,031*		22,383			

	All	Lower	Upper	Digit	Space	Symbol
# passwords	*9,997,985*	3,824,547	109,258	2,035,160	0	607
# characters	*75,916,675*	27,228,063	768,877	14,221,036	0	3,414
Avg length	*7.59*	7.12	7.04	6.99	N/A	5.62
'H'	*110,162*		24,362			
'I'	*115,795*		45,373			
'J'	*93,063*		14,189			
'K'	*107,284*		21,048			
'L'	*135,003*		38,414			
'M'	*142,667*		29,030			
'N'	*134,861*		42,998			
'O'	*113,121*		48,311			
'P'	*120,634*		21,714			
'Q'	*66,359*		9,136			
'R'	*151,967*		47,622			
'S'	*168,694*		42,401			
'T'	*137,600*		37,107			
'U'	*102,116*		23,693			
'V'	*87,765*		14,781			
'W'	*86,311*		14,657			
'X'	*74,301*		11,872			
'Y'	*97,692*		20,497			
'Z'	*77,862*		12,005			
'0'	*3,310,606*			2,003,501		
'1'	*4,751,205*			2,551,049		
'2'	*3,111,201*			1,595,168		
'3'	*2,168,754*			1,087,241		
'4'	*1,845,614*			974,031		
'5'	*1,980,406*			1,084,531		
'6'	*1,890,052*			1,043,140		
'7'	*1,931,518*			1,053,708		
'8'	*2,036,416*			1,174,103		
'9'	*2,790,004*			1,654,564		
<Space>	*61*				0	
'_'	*38,379*					86
'-'	*20,394*					37
';'	*625*					5

	All	Lower	Upper	Digit	Space	Symbol
# passwords	*9,997,985*	3,824,547	109,258	2,035,160	0	607
# characters	*75,916,675*	27,228,063	768,877	14,221,036	0	3,414
Avg length	*7.59*	7.12	7.04	6.99	N/A	5.62
':'	*9*					0
'!'	*10,353*					131
'?'	*4,527*					1,124
'.'	*37,959*					58
'"'	*744*					0
'»'	*19*					0
'('	*93*					2
')'	*11*					0
'['	*167*					0
']'	*133*					6
'{'	*14*					1
'}'	*20*					1
'@'	*7,719*					119
'*'	*9,939*					1,606
'/'	*67*					5
'\'	*382*					0
'&'	*1,011*					24
'#'	*2,781*					45
'%'	*1,042*					67
'`'	*65*					1
'^'	*379*					18
'+'	*271*					1
'<'	*2*					0
'='	*810*					3
'>'	*3*					0
'¬'	*3*					0
'|'	*75*					0
'¦'	*3*					0
'~'	*228*					39
'$'	*3,950*					35
'¼'	*5*					0
'é'	*2*					0
'ª'	*20*					0

	All	Lower	Upper	Digit	Space	Symbol
# passwords	*9,997,985*	3,824,547	109,258	2,035,160	0	607
# characters	*75,916,675*	27,228,063	768,877	14,221,036	0	3,414
Avg length	*7.59*	7.12	7.04	6.99	N/A	5.62
^B	*2*					0
^C	*31*					0
^G	*1*					0
^O	*2*					0
^W	*1*					0

	All	Lower + Upper	Lower + Digit	Lower + Space	Lower + Symbol	Upper + Digit
# passwords	*9,997,985*	251,578	2,985,686	24	40,103	110,227
# characters	*75,916,675*	1,991,431	24,686,517	260	447,902	877,075
Avg length	*7.59*	7.92	8.27	10.83	11.17	7.96
'a'	*4,765,881*	132,204	1,567,790	38	44,737	
'b'	*1,394,757*	34,915	479,009	3	9,892	
'c'	*1,629,304*	44,409	528,340	10	13,042	
'd'	*1,705,096*	46,512	575,194	9	13,124	
'e'	*4,117,197*	131,847	1,191,796	19	35,136	
'f'	*1,019,573*	30,087	358,364	3	8,913	
'g'	*1,246,221*	37,537	402,022	5	9,514	
'h'	*1,445,050*	44,852	454,831	6	12,996	
'i'	*2,901,135*	94,173	845,031	9	26,209	
'j'	*677,976*	19,800	276,425	5	4,794	
'k'	*1,341,596*	37,061	490,730	1	13,211	
'l'	*2,288,102*	64,141	698,983	11	20,747	
'm'	*1,732,512*	40,850	614,037	9	15,266	
'n'	*2,825,016*	87,214	862,105	15	25,494	
'o'	*3,032,936*	91,742	882,493	26	25,186	
'p'	*1,150,607*	28,546	383,644	7	8,539	
'q'	*320,816*	14,019	164,615	1	753	
'r'	*3,093,749*	92,850	937,256	12	26,906	
's'	*2,757,228*	67,519	863,423	14	24,721	
't'	*2,330,857*	68,108	704,053	7	18,982	
'u'	*1,394,794*	50,854	440,740	8	11,705	
'v'	*769,297*	25,035	299,047	2	7,979	
'w'	*720,790*	23,452	267,410	2	4,191	
'x'	*445,369*	19,396	185,741	1	2,761	
'y'	*1,178,415*	43,992	403,792	4	10,827	
'z'	*529,710*	19,761	224,606	3	4,499	
'A'	*223,322*	36,258				43,152
'B'	*134,408*	31,571				15,464
'C'	*133,753*	29,003				15,765
'D'	*133,401*	26,799				18,009
'E'	*174,236*	29,171				26,666
'F'	*109,205*	25,025				11,761
'G'	*113,031*	24,187				12,988

	All	Lower + Upper	Lower + Digit	Lower + Space	Lower + Symbol	Upper + Digit
# passwords	*9,997,985*	251,578	2,985,686	24	40,103	110,227
# characters	*75,916,675*	1,991,431	24,686,517	260	447,902	877,075
Avg length	*7.59*	7.92	8.27	10.83	11.17	7.96
'H'	*110,162*	22,335				12,888
'I'	*115,795*	20,345				18,798
'J'	*93,063*	19,938				9,575
'K'	*107,284*	20,313				14,685
'L'	*135,003*	22,512				20,136
'M'	*142,667*	29,096				18,020
'N'	*134,861*	19,938				21,418
'O'	*113,121*	17,774				19,670
'P'	*120,634*	25,298				13,013
'Q'	*66,359*	13,132				6,257
'R'	*151,967*	23,799				24,059
'S'	*168,694*	32,777				22,678
'T'	*137,600*	25,451				19,030
'U'	*102,116*	21,483				11,484
'V'	*87,765*	17,297				11,219
'W'	*86,311*	17,381				9,118
'X'	*74,301*	14,200				8,209
'Y'	*97,692*	20,547				11,173
'Z'	*77,862*	14,925				8,642
'0'	*3,310,606*		1,117,836			51,097
'1'	*4,751,205*		1,857,447			67,133
'2'	*3,111,201*		1,271,970			56,028
'3'	*2,168,754*		896,238			41,978
'4'	*1,845,614*		708,914			36,324
'5'	*1,980,406*		716,389			39,946
'6'	*1,890,052*		677,206			37,372
'7'	*1,931,518*		707,671			38,446
'8'	*2,036,416*		691,184			37,607
'9'	*2,790,004*		940,185			47,267
<Space>	*61*			30		
'_'	*38,379*				15,564	
'-'	*20,394*				7,646	
';'	*625*				366	

	All	Lower + Upper	Lower + Digit	Lower + Space	Lower + Symbol	Upper + Digit
# passwords	*9,997,985*	251,578	2,985,686	24	40,103	110,227
# characters	*75,916,675*	1,991,431	24,686,517	260	447,902	877,075
Avg length	*7.59*	7.92	8.27	10.83	11.17	7.96
':'	*9*				4	
'!'	*10,353*				2,857	
'?'	*4,527*				547	
'.'	*37,959*				13,416	
'"'	*744*				649	
'»'	*19*				19	
'('	*93*				25	
')'	*11*				1	
'['	*167*				84	
']'	*133*				75	
'{'	*14*				8	
'}'	*20*				8	
'@'	*7,719*				1,637	
'*'	*9,939*				2,582	
'/'	*67*				3	
'\'	*382*				254	
'&'	*1,011*				333	
'#'	*2,781*				434	
'%'	*1,042*				205	
'`'	*65*				27	
'^'	*379*				69	
'+'	*271*				80	
'<'	*2*				2	
'='	*810*				121	
'>'	*3*				3	
'¬'	*3*				0	
'|'	*75*				24	
'¦'	*3*				0	
'~'	*228*				72	
'$'	*3,950*				640	
'¼'	*5*				3	
'é'	*2*				0	
'ª'	*20*				18	

	All	Lower + Upper	Lower + Digit	Lower + Space	Lower + Symbol	Upper + Digit
# passwords	*9,997,985*	251,578	2,985,686	24	40,103	110,227
# characters	*75,916,675*	1,991,431	24,686,517	260	447,902	877,075
Avg length	*7.59*	7.92	8.27	10.83	11.17	7.96
^B	*2*				1	
^C	*31*				0	
^G	*1*				0	
^O	*2*				1	
^W	*1*				0	

	All	Upper + Space	Upper + Symbol	Digit + Space	Digit + Symbol	Space + Symbol
# passwords	*9,997,985*	2	834	0	8,430	0
# characters	*75,916,675*	20	7,384	0	74,433	0
Avg length	*7.59*	10.00	8.85	N/A	8.83	N/A
'a'	*4,765,881*					
'b'	*1,394,757*					
'c'	*1,629,304*					
'd'	*1,705,096*					
'e'	*4,117,197*					
'f'	*1,019,573*					
'g'	*1,246,221*					
'h'	*1,445,050*					
'i'	*2,901,135*					
'j'	*677,976*					
'k'	*1,341,596*					
'l'	*2,288,102*					
'm'	*1,732,512*					
'n'	*2,825,016*					
'o'	*3,032,936*					
'p'	*1,150,607*					
'q'	*320,816*					
'r'	*3,093,749*					
's'	*2,757,228*					
't'	*2,330,857*					
'u'	*1,394,794*					
'v'	*769,297*					
'w'	*720,790*					
'x'	*445,369*					
'y'	*1,178,415*					
'z'	*529,710*					
'A'	*223,322*	3	569			
'B'	*134,408*	0	154			
'C'	*133,753*	1	204			
'D'	*133,401*	1	240			
'E'	*174,236*	1	486			
'F'	*109,205*	2	183			
'G'	*113,031*	1	132			

	All	Upper + Space	Upper + Symbol	Digit + Space	Digit + Symbol	Space + Symbol
# passwords	*9,997,985*	2	834	0	8,430	0
# characters	*75,916,675*	20	7,384	0	74,433	0
Avg length	*7.59*	10.00	8.85	N/A	8.83	N/A
'H'	*110,162*	1	146			
'I'	*115,795*	1	362			
'J'	*93,063*	0	76			
'K'	*107,284*	0	209			
'L'	*135,003*	0	248			
'M'	*142,667*	0	226			
'N'	*134,861*	0	427			
'O'	*113,121*	2	340			
'P'	*120,634*	1	148			
'Q'	*66,359*	0	32			
'R'	*151,967*	0	393			
'S'	*168,694*	1	311			
'T'	*137,600*	1	248			
'U'	*102,116*	1	196			
'V'	*87,765*	0	112			
'W'	*86,311*	0	90			
'X'	*74,301*	0	58			
'Y'	*97,692*	1	160			
'Z'	*77,862*	0	57			
'0'	*3,310,606*			0	10,605	
'1'	*4,751,205*			0	9,526	
'2'	*3,111,201*			0	6,686	
'3'	*2,168,754*			0	4,867	
'4'	*1,845,614*			0	4,195	
'5'	*1,980,406*			0	4,415	
'6'	*1,890,052*			0	3,941	
'7'	*1,931,518*			0	4,047	
'8'	*2,036,416*			0	4,378	
'9'	*2,790,004*			0	6,640	
<Space>	*61*	2		0		0
'_'	*38,379*		312		1,696	0
'-'	*20,394*		83		2,658	0
';'	*625*		7		24	0

	All	Upper + Space	Upper + Symbol	Digit + Space	Digit + Symbol	Space + Symbol
# passwords	*9,997,985*	2	834	0	8,430	0
# characters	*75,916,675*	20	7,384	0	74,433	0
Avg length	*7.59*	10.00	8.85	N/A	8.83	N/A
':'	*9*		0		0	0
'!'	*10,353*		151		676	0
'?'	*4,527*		295		1,117	0
'.'	*37,959*		294		5,347	0
'"'	*744*		21		3	0
'»'	*19*		0		0	0
'('	*93*		2		1	0
')'	*11*		0		0	0
'['	*167*		3		3	0
']'	*133*		2		1	0
'{'	*14*		1		0	0
'}'	*20*		0		0	0
'@'	*7,719*		130		896	0
'*'	*9,939*		86		2,053	0
'/'	*67*		1		27	0
'\'	*382*		0		11	0
'&'	*1,011*		29		40	0
'#'	*2,781*		49		117	0
'%'	*1,042*		20		36	0
'`'	*65*		0		3	0
'^'	*379*		21		16	0
'+'	*271*		5		27	0
'<'	*2*		0		0	0
'='	*810*		3		210	0
'>'	*3*		0		0	0
'¬'	*3*		0		0	0
'|'	*75*		0		0	0
'¦'	*3*		0		0	0
'~'	*228*		3		7	0
'$'	*3,950*		55		164	0
'¼'	*5*		2		0	0
'é'	*2*		2		0	0
'ª'	*20*		0		0	0

	All	Upper + Space	Upper + Symbol	Digit + Space	Digit + Symbol	Space + Symbol
# passwords	*9,997,985*	2	834	0	8,430	0
# characters	*75,916,675*	20	7,384	0	74,433	0
Avg length	*7.59*	10.00	8.85	N/A	8.83	N/A
^B	*2*		0		0	0
^C	*31*		0		0	0
^G	*1*		0		0	0
^O	*2*		0		0	0
^W	*1*		0		0	0

	All	Lower, Upper + Digit	Lower, Upper + Space	Lower, Upper + Symbol	Lower, Digit + Space	Lower, Digit + Symbol
# passwords	*9,997,985*	570,968	2	7,137	2	36,722
# characters	*75,916,675*	4,922,333	24	83,160	19	422,933
Avg length	*7.59*	8.62	12.00	11.65	9.50	11.52
'a'	*4,765,881*	197,888	2	4,395	0	30,298
'b'	*1,394,757*	65,827	0	1,266	2	7,253
'c'	*1,629,304*	80,086	1	1,729	0	8,497
'd'	*1,705,096*	84,175	0	1,629	0	8,951
'e'	*4,117,197*	181,632	1	4,140	2	20,469
'f'	*1,019,573*	60,081	0	1,075	0	6,197
'g'	*1,246,221*	69,544	0	1,446	0	6,200
'h'	*1,445,050*	76,674	1	1,590	2	7,936
'i'	*2,901,135*	131,736	2	2,829	1	16,065
'j'	*677,976*	43,035	0	968	0	3,811
'k'	*1,341,596*	75,462	0	1,585	0	9,390
'l'	*2,288,102*	91,927	0	2,146	0	12,876
'm'	*1,732,512*	75,670	1	1,614	0	10,205
'n'	*2,825,016*	130,119	1	2,767	1	15,892
'o'	*3,032,936*	136,768	2	3,014	1	15,264
'p'	*1,150,607*	54,384	0	1,242	0	5,668
'q'	*320,816*	35,317	0	685	0	1,091
'r'	*3,093,749*	139,154	2	3,173	0	17,432
's'	*2,757,228*	113,291	0	2,632	2	15,354
't'	*2,330,857*	105,120	2	2,289	1	11,784
'u'	*1,394,794*	82,861	1	1,803	0	7,405
'v'	*769,297*	54,177	0	1,132	0	5,248
'w'	*720,790*	54,184	1	995	0	3,152
'x'	*445,369*	48,548	0	803	0	2,310
'y'	*1,178,415*	75,932	0	1,626	0	6,961
'z'	*529,710*	47,489	0	851	0	3,768
'A'	*223,322*	71,037	1	1,749		
'B'	*134,408*	59,890	1	1,042		
'C'	*133,753*	57,542	1	861		
'D'	*133,401*	56,930	0	1,061		
'E'	*174,236*	53,206	0	1,169		
'F'	*109,205*	52,485	0	815		
'G'	*113,031*	51,147	0	895		

	All	Lower, Upper + Digit	Lower, Upper + Space	Lower, Upper + Symbol	Lower, Digit + Space	Lower, Digit + Symbol
# passwords	*9,997,985*	570,968	2	7,137	2	36,722
# characters	*75,916,675*	4,922,333	24	83,160	19	422,933
Avg length	*7.59*	8.62	12.00	11.65	9.50	11.52
'H'	*110,162*	48,363	0	849		
'I'	*115,795*	28,853	0	946		
'J'	*93,063*	47,154	0	815		
'K'	*107,284*	48,943	0	815		
'L'	*135,003*	51,067	0	1,091		
'M'	*142,667*	63,128	1	1,284		
'N'	*134,861*	47,410	0	862		
'O'	*113,121*	25,203	0	755		
'P'	*120,634*	57,933	0	967		
'Q'	*66,359*	35,977	0	762		
'R'	*151,967*	53,455	1	966		
'S'	*168,694*	67,219	0	1,208		
'T'	*137,600*	53,322	0	927		
'U'	*102,116*	42,620	0	998		
'V'	*87,765*	42,409	0	801		
'W'	*86,311*	43,313	0	730		
'X'	*74,301*	38,307	0	653		
'Y'	*97,692*	42,745	0	1,045		
'Z'	*77,862*	40,386	0	721		
'0'	*3,310,606*	105,363			0	16,518
'1'	*4,751,205*	234,083			0	23,774
'2'	*3,111,201*	159,877			1	15,438
'3'	*2,168,754*	123,778			1	10,264
'4'	*1,845,614*	111,027			1	7,600
'5'	*1,980,406*	123,704			1	7,984
'6'	*1,890,052*	117,701			1	7,363
'7'	*1,931,518*	115,488			0	8,492
'8'	*2,036,416*	116,799			0	8,740
'9'	*2,790,004*	123,388			0	13,571
<Space>	*61*		2		2	
'_'	*38,379*			1,913		14,737
'-'	*20,394*			685		6,599
';'	*625*			8		169

	All	Lower, Upper + Digit	Lower, Upper + Space	Lower, Upper + Symbol	Lower, Digit + Space	Lower, Digit + Symbol
# passwords	*9,997,985*	570,968	2	7,137	2	36,722
# characters	*75,916,675*	4,922,333	24	83,160	19	422,933
Avg length	*7.59*	8.62	12.00	11.65	9.50	11.52
':'	*9*			1		3
'!'	*10,353*			620		2,458
'?'	*4,527*			276		385
'.'	*37,959*			3,621		11,678
'"'	*744*			22		32
'»'	*19*			0		0
'('	*93*			9		41
')'	*11*			2		8
'['	*167*			19		39
']'	*133*			16		29
'{'	*14*			1		1
'}'	*20*			1		7
'@'	*7,719*			415		2,838
'*'	*9,939*			359		1,945
'/'	*67*			1		30
'\'	*382*			6		99
'&'	*1,011*			73		259
'#'	*2,781*			232		637
'%'	*1,042*			92		164
'`'	*65*			9		18
'^'	*379*			37		97
'+'	*271*			14		96
'<'	*2*			0		0
'='	*810*			66		299
'>'	*3*			0		0
'¬'	*3*			0		1
'|'	*75*			12		30
'¦'	*3*			0		1
'~'	*228*			16		53
'$'	*3,950*			421		931
'¼'	*5*			0		0
'é'	*2*			0		0
'ª'	*20*			0		0

	All	Lower, Upper + Digit	Lower, Upper + Space	Lower, Upper + Symbol	Lower, Digit + Space	Lower, Digit + Symbol
# passwords	*9,997,985*	570,968	2	7,137	2	36,722
# characters	*75,916,675*	4,922,333	24	83,160	19	422,933
Avg length	*7.59*	8.62	12.00	11.65	9.50	11.52
^B	*2*			1		0
^C	*31*			0		27
^G	*1*			1		0
^O	*2*			0		0
^W	*1*			0		1

	All	Lower, Space + Symbol	Upper, Digit + Space	Upper, Digit + Symbol	Upper, Space + Symbol	Digit, Space + Symbol
# passwords	*9,997,985*	0	1	1,568	0	24
# characters	*75,916,675*	0	10	15,441	0	553
Avg length	*7.59*	N/A	10.00	9.85	N/A	23.04
'a'	*4,765,881*	0				
'b'	*1,394,757*	0				
'c'	*1,629,304*	0				
'd'	*1,705,096*	0				
'e'	*4,117,197*	0				
'f'	*1,019,573*	0				
'g'	*1,246,221*	0				
'h'	*1,445,050*	0				
'i'	*2,901,135*	0				
'j'	*677,976*	0				
'k'	*1,341,596*	0				
'l'	*2,288,102*	0				
'm'	*1,732,512*	0				
'n'	*2,825,016*	0				
'o'	*3,032,936*	0				
'p'	*1,150,607*	0				
'q'	*320,816*	0				
'r'	*3,093,749*	0				
's'	*2,757,228*	0				
't'	*2,330,857*	0				
'u'	*1,394,794*	0				
'v'	*769,297*	0				
'w'	*720,790*	0				
'x'	*445,369*	0				
'y'	*1,178,415*	0				
'z'	*529,710*	0				
'A'	*223,322*		0	670	0	
'B'	*134,408*		0	232	0	
'C'	*133,753*		0	225	0	
'D'	*133,401*		0	271	0	
'E'	*174,236*		0	402	0	
'F'	*109,205*		0	171	0	
'G'	*113,031*		0	175	0	

	All	Lower, Space + Symbol	Upper, Digit + Space	Upper, Digit + Symbol	Upper, Space + Symbol	Digit, Space + Symbol
# passwords	*9,997,985*	0	1	1,568	0	24
# characters	*75,916,675*	0	10	15,441	0	553
Avg length	*7.59*	N/A	10.00	9.85	N/A	23.04
'H'	*110,162*		1	159	0	
'I'	*115,795*		0	278	0	
'J'	*93,063*		0	109	0	
'K'	*107,284*		0	221	0	
'L'	*135,003*		0	217	0	
'M'	*142,667*		0	299	0	
'N'	*134,861*		0	505	0	
'O'	*113,121*		0	259	0	
'P'	*120,634*		2	202	0	
'Q'	*66,359*		0	50	0	
'R'	*151,967*		0	320	0	
'S'	*168,694*		2	365	0	
'T'	*137,600*		0	231	0	
'U'	*102,116*		0	129	0	
'V'	*87,765*		0	150	0	
'W'	*86,311*		0	99	0	
'X'	*74,301*		0	103	0	
'Y'	*97,692*		1	125	0	
'Z'	*77,862*		0	84	0	
'0'	*3,310,606*		0	951		121
'1'	*4,751,205*		1	1,170		55
'2'	*3,111,201*		0	884		59
'3'	*2,168,754*		0	536		27
'4'	*1,845,614*		0	484		27
'5'	*1,980,406*		0	476		28
'6'	*1,890,052*		0	462		18
'7'	*1,931,518*		1	568		34
'8'	*2,036,416*		1	532		31
'9'	*2,790,004*		0	726		22
<Space>	*61*	0	1		0	24
'_'	*38,379*	0		502	0	0
'-'	*20,394*	0		272	0	48
';'	*625*	0		6	0	0

	All	Lower, Space + Symbol	Upper, Digit + Space	Upper, Digit + Symbol	Upper, Space + Symbol	Digit, Space + Symbol
# passwords	*9,997,985*	0	1	1,568	0	24
# characters	*75,916,675*	0	10	15,441	0	553
Avg length	*7.59*	N/A	10.00	9.85	N/A	23.04
':'	*9*	0		0	0	0
'!'	*10,353*	0		127	0	0
'?'	*4,527*	0		603	0	0
'.'	*37,959*	0		483	0	59
'"'	*744*	0		5	0	0
'»'	*19*	0		0	0	0
'('	*93*	0		1	0	0
')'	*11*	0		0	0	0
'['	*167*	0		0	0	0
']'	*133*	0		2	0	0
'{'	*14*	0		0	0	0
'}'	*20*	0		0	0	0
'@'	*7,719*	0		139	0	0
'*'	*9,939*	0		135	0	0
'/'	*67*	0		0	0	0
'\'	*382*	0		1	0	0
'&'	*1,011*	0		26	0	0
'#'	*2,781*	0		46	0	0
'%'	*1,042*	0		92	0	0
'`'	*65*	0		3	0	0
'^'	*379*	0		8	0	0
'+'	*271*	0		19	0	0
'<'	*2*	0		33	0	0
'='	*810*	0		0	0	0
'>'	*3*	0		0	0	0
'¬'	*3*	0		2	0	0
'|'	*75*	0		2	0	0
'¦'	*3*	0		2	0	0
'~'	*228*	0		5	0	0
'$'	*3,950*	0		84	0	0
'¼'	*5*	0		0	0	0
'é'	*2*	0		0	0	0
'ª'	*20*	0		2	0	0

	All	Lower, Space + Symbol	Upper, Digit + Space	Upper, Digit + Symbol	Upper, Space + Symbol	Digit, Space + Symbol
# passwords	*9,997,985*	0	1	1,568	0	24
# characters	*75,916,675*	0	10	15,441	0	553
Avg length	*7.59*	N/A	10.00	9.85	N/A	23.04
^B	*2*	0		0	0	0
^C	*31*	0		1	0	0
^G	*1*	0		0	0	0
^O	*2*	0		0	0	0
^W	*1*	0		0	0	0

	All	Lower, Upper, Digit + Space	Lower, Upper, Digit + Symbol	Lower, Digit, Space + Symbol	Upper, Digit, Space + Symbol	Lower, Upper, Digit, Space + Symbol
# passwords	*9,997,985*	0	15,105	0	0	0
# characters	*75,916,675*	0	165,790	0	0	0
Avg length	*7.59*	N/A	10.98	N/A	N/A	N/A
'a'	*4,765,881*	0	7,409	0		0
'b'	*1,394,757*	0	1,902	0		0
'c'	*1,629,304*	0	2,369	0		0
'd'	*1,705,096*	0	2,617	0		0
'e'	*4,117,197*	0	6,360	0		0
'f'	*1,019,573*	0	1,431	0		0
'g'	*1,246,221*	0	1,975	0		0
'h'	*1,445,050*	0	2,211	0		0
'i'	*2,901,135*	0	3,928	0		0
'j'	*677,976*	0	1,229	0		0
'k'	*1,341,596*	0	2,009	0		0
'l'	*2,288,102*	0	3,148	0		0
'm'	*1,732,512*	0	2,354	0		0
'n'	*2,825,016*	0	4,277	0		0
'o'	*3,032,936*	0	4,087	0		0
'p'	*1,150,607*	0	1,813	0		0
'q'	*320,816*	0	1,001	0		0
'r'	*3,093,749*	0	4,836	0		0
's'	*2,757,228*	0	3,707	0		0
't'	*2,330,857*	0	3,400	0		0
'u'	*1,394,794*	0	3,107	0		0
'v'	*769,297*	0	1,347	0		0
'w'	*720,790*	0	1,486	0		0
'x'	*445,369*	0	1,394	0		0
'y'	*1,178,415*	0	2,597	0		0
'z'	*529,710*	0	1,209	0		0
'A'	*223,322*	0	3,119		0	0
'B'	*134,408*	0	1,572		0	0
'C'	*133,753*	0	1,420		0	0
'D'	*133,401*	0	1,550		0	0
'E'	*174,236*	0	1,916		0	0
'F'	*109,205*	0	1,215		0	0
'G'	*113,031*	0	1,123		0	0

	All	Lower, Upper, Digit + Space	Lower, Upper, Digit + Symbol	Lower, Digit, Space + Symbol	Upper, Digit, Space + Symbol	Lower, Upper, Digit, Space + Symbol
# passwords	*9,997,985*	0	15,105	0	0	0
# characters	*75,916,675*	0	165,790	0	0	0
Avg length	*7.59*	N/A	10.98	N/A	N/A	N/A
'H'	*110,162*	0	1,058		0	0
'I'	*115,795*	0	839		0	0
'J'	*93,063*	0	1,207		0	0
'K'	*107,284*	0	1,050		0	0
'L'	*135,003*	0	1,318		0	0
'M'	*142,667*	0	1,583		0	0
'N'	*134,861*	0	1,303		0	0
'O'	*113,121*	0	807		0	0
'P'	*120,634*	0	1,356		0	0
'Q'	*66,359*	0	1,013		0	0
'R'	*151,967*	0	1,352		0	0
'S'	*168,694*	0	1,732		0	0
'T'	*137,600*	0	1,283		0	0
'U'	*102,116*	0	1,512		0	0
'V'	*87,765*	0	996		0	0
'W'	*86,311*	0	923		0	0
'X'	*74,301*	0	899		0	0
'Y'	*97,692*	0	1,398		0	0
'Z'	*77,862*	0	1,042		0	0
'0'	*3,310,606*	0	4,614	0	0	0
'1'	*4,751,205*	0	6,967	0	0	0
'2'	*3,111,201*	0	5,090	0	0	0
'3'	*2,168,754*	0	3,824	0	0	0
'4'	*1,845,614*	0	3,011	0	0	0
'5'	*1,980,406*	0	2,932	0	0	0
'6'	*1,890,052*	0	2,848	0	0	0
'7'	*1,931,518*	0	3,063	0	0	0
'8'	*2,036,416*	0	3,041	0	0	0
'9'	*2,790,004*	0	3,641	0	0	0
<Space>	*61*	0		0	0	0
'_'	*38,379*		3,569	0	0	0
'-'	*20,394*		2,366	0	0	0
';'	*625*		40	0	0	0

	All	Lower, Upper, Digit + Space	Lower, Upper, Digit + Symbol	Lower, Digit, Space + Symbol	Upper, Digit, Space + Symbol	Lower, Upper, Digit, Space + Symbol
# passwords	*9,997,985*	0	15,105	0	0	0
# characters	*75,916,675*	0	165,790	0	0	0
Avg length	*7.59*	N/A	10.98	N/A	N/A	N/A
':'	*9*		1	0	0	0
'!'	*10,353*		3,333	0	0	0
'?'	*4,527*		180	0	0	0
'.'	*37,959*		3,003	0	0	0
'"'	*744*		12	0	0	0
'»'	*19*		0	0	0	0
'('	*93*		12	0	0	0
')'	*11*		0	0	0	0
'['	*167*		13	0	0	0
']'	*133*		8	0	0	0
'{'	*14*		2	0	0	0
'}'	*20*		3	0	0	0
'@'	*7,719*		1,545	0	0	0
'*'	*9,939*		1,173	0	0	0
'/'	*67*		5	0	0	0
'\'	*382*		6	0	0	0
'&'	*1,011*		227	0	0	0
'#'	*2,781*		1,221	0	0	0
'%'	*1,042*		366	0	0	0
'`'	*65*		4	0	0	0
'^'	*379*		113	0	0	0
'+'	*271*		29	0	0	0
'<'	*2*		0	0	0	0
'='	*810*		75	0	0	0
'>'	*3*		0	0	0	0
'¬'	*3*		0	0	0	0
'|'	*75*		7	0	0	0
'¦'	*3*		0	0	0	0
'~'	*228*		33	0	0	0
'$'	*3,950*		1,620	0	0	0
'¼'	*5*		0	0	0	0
'é'	*2*		0	0	0	0
'ª'	*20*		0	0	0	0

	All	Lower, Upper, Digit + Space	Lower, Upper, Digit + Symbol	Lower, Digit, Space + Symbol	Upper, Digit, Space + Symbol	Lower, Upper, Digit, Space + Symbol
# passwords	*9,997,985*	0	15,105	0	0	0
# characters	*75,916,675*	0	165,790	0	0	0
Avg length	*7.59*	N/A	10.98	N/A	N/A	N/A
^B	*2*		0	0	0	0
^C	*31*		3	0	0	0
^G	*1*		0	0	0	0
^O	*2*		1	0	0	0
^W	*1*		0	0	0	0