Exploring Password Complexity: A Comparative Analysis of Supervised and Unsupervised Approaches for Password Strength Classification

Brianna Schuh

Advised by Priya Panda, PhD

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Statistics and Data Science at Yale College

Yale University

New Haven, Connecticut

May 4th, 2023

Abstract

Within the first four months of 2023 [1], 17 major companies reported data breaches. Although many companies store password encryptions instead of raw passwords, these breaches still pose a threat to account holders. Attackers can access personal information if they are able to crack the password. Companies use algorithms to determine password strength and derive their password policy accordingly. In 2012, Dropbox released their zxcvbn, their own algorithm for classifying passwords [2]. In this project, we examined password complexity as defined by zxcvbn through the use of supervised and unsupervised methods. 700,000 passwords were randomly sampled from the 2015 000webhost data breach. Their strengths were classified by the zxcvbn algorithm, and we extract several features to capture the underlying pattern of the data, including n-grams, Shannon entropy, Levenshtein distance, and character repetition weight sum. For the supervised methods, we used a Multilayer Perceptron (MLP) and a Convolution Neural Network (CNN). For the unsupervised methods, we used k-means clustering at different clustering levels. By comparing the performance of these models, we aimed to uncover patterns in password strength and identify the most influential features in predicting password complexity. This analysis offers valuable insights into password complexity and its relationship with the zxcvbn algorithm, contributing to the development of more robust password policies and enhanced user security.

Acknowledgments

Graduating from Yale is the culmination of all of the support my family has provided me. I would like to thank my family for all they have sacrificed so that I can pursue my interest in statistics. I would also like to thank my first-year suitemates for all of the support they have given me throughout my time at Yale, and especially during my last semester while working on this. Lastly, I would like to thank Professor Priya Panda for her guidance.

Contents

1 Executive Summary

In this project, we analyzed the complexity of passwords through the use of supervised and unsupervised machine learning techniques. We were interested in discovering the underlying patterns and sequences in password strength so that we can identify the most influential features in predicting password complexity. The analysis was based on a random sample of 700,000 passwords obtained from the 2015 000webhost data breach. Password strengths were classified using the zxcvbn algorithm, and several features were extracted, such as n-grams, Shannon entropy, Levenshtein distance, and character repetition weight sum.

The supervised models employed in this project were a Multilayer Perceptron (MLP) and a Convolution Neural Network (CNN). K-means clustering was used as an unsupervised method. Supervised techniques achieved high performance in password strength classification task - both models achieved an accuracy of 70%. Both methods were analyzed to determine which features contributed its classification task, and it revealed that Levenshtein distance and Shannon entropy were common influential features. The k-means clustering method provided insights into the underlying structure of the password feature space. The Silhouette scores and Davies-Bouldin scores highlighted the trade-offs between cluster quality and correspondence with true password strength classes.

In conclusion, supervised techniques like MLPs and CNNs showed potential to succeed in password strength classification tasks. However, the insights gained from unsupervised K-means clustering should also be taken into account to improve classification models and enhance the understanding of password strength. This analysis offers insights into password complexity and its relationship with the zxcvbn algorithm, and can be further applied to other deep learning techniques and password strength algorithms.

2 Introduction

Data breaches have become increasingly common in recent years, with numerous high-profile cases affecting millions of users worldwide. In the first four months of 2023 alone, 17 major companies reported data breaches. While many organizations store encrypted passwords rather than raw passwords, these breaches still pose a significant threat to account holders. Attackers can access personal information if they manage to crack the encrypted passwords. Consequently, companies use algorithms to determine password strength and establish password policies that enhance user security. In 2012, Dropbox introduced the zxcvbn algorithm for classifying passwords based on their complexity. This project focuses on examining password complexity, as defined by the zxcvbn algorithm, using supervised and unsupervised machine learning techniques.

The data used in this study consisted of a random sample of 700,000 passwords from the 2015 000webhost data breach. Password strengths were classified by the zxcvbn algorithm, and several features were extracted to capture the underlying patterns in the data. These features included n-grams, Shannon entropy, Levenshtein distance, character repetition weight sum, most common character set count, character frequency ratio, and password length to unique ratio.

The remainder of the paper is organized as follows: Section 3 will present an overview of the dataset. Section 4 will introduce the methodologies used for this research. It will discuss what exactly a password is, the zxcvbn algorithm, and the features extracted from our dataset. Section 5 will discuss the architecture of our supervised models, MLP and CNN. Section 6 will discuss the algorithm behind k-means clustering.

Section 7 will explain the actual experiment: how the models were trained/fitted, the evaluation metrics, and the performance based off these metrics. Section 8 will discuss the ethics and limitations behind this task. Lastly, Section 9 will conclude the paper with recommendations, ideas for future work, and improvements to enhance password policies and user security.

3 Dataset

In October 2015, a data breach at 000webhost, a company that provided website hosting services, exposed over 13 million usernames, email addresses, and plaintext passwords. 000webhost did not hash passwords before storing them, hence why attackers were able to access the raw passwords.

700,000 passwords from the breach were randomly sampled and split into a training set, testing set, and a validation set. All models were trained on the same training set, tested on the same testing set, and validated on the same validation set.

There were 91 unique characters in the dataset. The largest password length is 193 characters, and the shortest password length is one character. The dataset is positively skewed since the mean password length is greater than the median password length; the average password length is 9.68 characters and the median password length is 9 characters. Figure 1 visualizes the skew of the data.

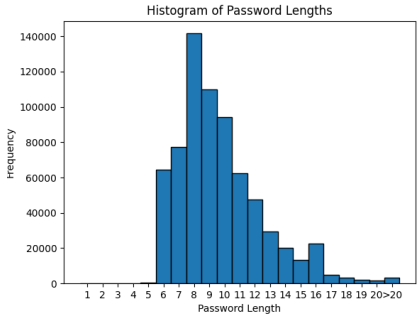


Figure 1: Histogram of Password Lengths

The passwords were assigned a strength classification by the zxcvbn algorithm, which is further explained in section 4.2. There are five classes total: very weak, weak, medium, strong, and very strong. When we applied this algorithm to our dataset, less than 2,500 passwords were classified as very weak. Since this number is so small compared to size of the dataset, the very weak passwords and the weak passwords were merged into one class, weak. Thus, our new scale now goes from 1, weak, to 4, very strong. As shown in Figure 2, the distributions of the classes are unequal. In order to avoid biasing any of our models to a class with larger samples, we performed undersampling, a technique that randomly removes observations from the larger classes, before training any of our models. Undersampling seemed appropriate given that our dataset is already so large, so it is unlikely that any important information would have been lost.

Feature engineering was performed on each observation in the dataset, and these features were used to train both our superivsed and unsupervised models. Our chosen features are described in greater detail in section 4.3.

4 Methodology

In this section, we will discuss the methodologies employed in our analysis. We will start off by defining a password and its elements to establish a common understanding. We will then discuss the details of the zxcvbn algorithm. Lastly, we will go through the all of the features that were extracted in the dataset that were used in our subsequent analysis.

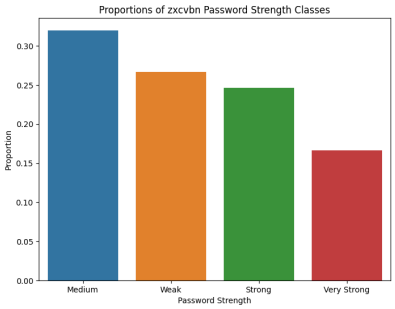


Figure 2: Proportions of zxcvbn password strength classes.

4.1 Passwords

Let us begin by formally defining a password and its elements. Let $\mathcal{U}$, $\mathcal{L}$, $\mathcal{D}$, and $\mathcal{S}$ denote the sets of upper case, lower case, digits, and special characters, respectively:

$$\mathcal{U} = \{A, B, \ldots, Z\}, \tag{1}$$

$$\mathcal{L} = \{a, b, \ldots, z\}, \tag{2}$$

$$\mathcal{D} = \{0, 1, \ldots, 9\}, \tag{3}$$

$$\mathcal{S} = \left\{ \begin{matrix} !,\ \#,\ \$,\ \%,\ \&,\ ',\ (,\ ),\ *,\ +, \\ -,\ .,\ /,\ :,\ ;,\ <,\ =,\ >,\ ?,\ @, \\ [,\ ],\ \hat{},\ \_,\ `,\ \{,\ |,\ \},\ \tilde{},\ ¡ \end{matrix} \right\}. \tag{4}$$

Then, let $\mathcal{A}$ denote the complete alphabet, which is the union of these sets:

$$\mathcal{A} = \mathcal{U} \cup \mathcal{L} \cup \mathcal{D} \cup \mathcal{S}. \tag{5}$$

A password of length n is a string of text $P = p_1 p_2 \ldots p_n$, where each character p_i is an element of the alphabet $\mathcal{A}$:

$$P \in \mathcal{A}^n, \tag{6}$$

where $\mathcal{A}^n$ represents the set of all strings of length n made up of the characters in $\mathcal{A}$.

Since a password is just a sequence of characters, we can refer to specific positions in a password. Let P[i] refer to the ith character, p_i, of a given password, P. Moving forward, we will use this notation from this section to describe the features extracted from passwords.

4.2 Password Strength Classification

To classify passwords, we used zxcvbn, a password strength classifcation tool created by Dropbox in 2012. The zxcvbn algorithm considers several factors such as dictionary words, patterns, sequences, and entropy. Each password is assigned a strength score between 0 and 4, with 0 being very weak and 4 being very strong. As mentioned in section 3, the very weak class and the weak class were merged into one class, weak. Thus, our new scale now goes from 1, weak, to 4, very strong. The strength classifications are determined by the following factors:

1. **Dictionary Words:** The algorithm uses a large dictionary of common words, such as passwords, names, and English words. It checks if the password contains any of these words or their combinations and adjusts the strength score accordingly.

2. **Spatial Patterns:** The algorithm looks for common keyboard patterns like "qwerty" and "asdfgh" and reduces the estimated strength based on the presence of such patterns.

3. **Sequences:** The algorithm identifies simple ascending or descending sequences (i.e., "abcdef", "123456") and adjusts the strength score accordingly.

4. **Repeated Patterns:** The algorithm recognizes repeated characters, substrings, or patterns and reduces the strength score based on their presence.

5. **Entropy:** The algorithm calculates the entropy for each matched pattern independently and incorporates the minimum entropy into the strength score.

6. **Password Structure:** The algorithm searches for the simplest (lowest entropy) non-overlapping sequence of patterns and adjusts the strength score based on this structure.

Let us a define a function, zxcvbn, which takes a password, P, as input and outputs a score of either 0, 1, 2, 3, or 4, which is based on the factors defined above.

The password strength classification can then be derived from the zxcvbn score as follows:

$$\text{classification}(P) = \begin{cases} \text{Weak} & \text{if zxcvbn}(P) = 0 \text{ or zxcvbn}(P) = 1 \\ \text{Medium} & \text{if zxcvbn}(P) = 2 \\ \text{Strong} & \text{if zxcvbn}(P) = 3 \\ \text{Very Strong} & \text{if zxcvbn}(P) = 4 \end{cases} \tag{7}$$

4.3 Feature Engineering

Feature engineering describes the process of extracting information that can be used to train models from raw data. Essentially, features encode structural information of the password. Ten features will be considered: Shannon entropy, bigram frequency, trigrams frequency, four-grams frequency, Levenshtein distance, and character repetition weight sum, most common character type count, character frequency ratio, and password length to unique ratio. The features are described in-depth below.

4.3.1 Shannon Entropy

Shannon entropy measures the unpredictability of a string. For a given password, P, it measures of the randomness of it, considering the frequency of each character and the set of all possible characters. A higher entropy indicates a greater randomness.

Consider a password P of length N, made up of characters from set $\mathcal{A}$. Let q_i be the probability of character i appearing in the password P. This probability can be calculated as:

$$q_i = \frac{\text{frequency of character } i \text{ in } P}{N} \tag{8}$$

The Shannon entropy $H(P)$ of the password P can be calculated using the following formula:

$$H(P) = -\sum_{i \in P} q_i \log_2 q_i \tag{9}$$

4.3.2 N-gram Frequency

We use bigrams, trigrams, and four-grams as features in our dataset, and we will define them more generally as n-grams. An n-gram is a contiguous sequence of n items from a given string. Let P denote a password of length N. An n-gram of P is a contiguous sequence of n characters from P, where n is a positive integer less than or equal to N. An n-gram of P starting at position i can be defined as follows:

$$P[i : i + n] = (P[i], P[i + 1], \ldots, P[i + n - 1]) \tag{10}$$

where $P[i]$ is the first character of the n-gram, $P[i+1]$ is the second character, and so on, up to $P[i+n-1]$, which represents last character of the n-gram.

The set of all n-grams of P can be denoted as follows:

$$\mathrm{NG}(P,n) = \{(P[i], P[i+1], \ldots, P[i+n-1]) : 0 \leq i \leq N-n\} \tag{11}$$

In other words, $\mathrm{NG}(P,n)$ is the set of all n-grams of P, where each n-gram is represented as a tuple of n characters from P.

For a given password P and a given positive integer n, let n-gram frequency of P be the the number of occurrences of each n-gram in P. Let $f(w)$ be the number of occurrence of n-gram w in P. Then the n-gram frequency of P can be defined as a vector $\mathbf{F}_n(P)$ with length of $|\mathrm{NG}(P,n)|$, where each element of the vector maps to a unique n-gram in P:

$$\mathbf{F}_n(P) = [f(w) : w \in \mathrm{NG}(P,n)] \tag{12}$$

Thus, $f(w)$ is defined so that it takes a password P and a positive integer n as input, and outputs a vector $\mathbf{F}_n(P)$ of length $|\mathrm{NG}(P,n)|$ that represents the frequency of each unique n-gram in P. The vector is constructed by iterating over all n-grams in P and counting the number of occurrences of each unique n-gram.

The total number of n-grams in P is the cardinality of $\mathrm{NG}(P,n)$, which is represented as:

$$|\mathrm{NG}(P,n)| = N - n + 1 \tag{13}$$

Let the set of all n-grams $\mathrm{NG}(P,n)$ be converted to a set, which eliminates duplicates. The size of this set is the number of unique n-grams in P of length n:

$$|\mathcal{U}(\mathrm{NG}(P,n))| = \mathrm{len}(\mathrm{set}(\mathrm{NG}(P,n))) \tag{14}$$

Thus, the n-gram frequency of P can be defined as:

$$\mathrm{freq}_n(P) = \frac{|\mathcal{U}(\mathrm{NG}(P,n))|}{N-n+1} \tag{15}$$

This value ranges between 0 and 1, with higher values indicating that P contains more unique n-grams of length n relative to its length.

4.3.3 Levenshtein Distance

Levenshtein distance is a string metric that measures the similarity between two strings, which is based on the smallest number of single-character edits such that one word becomes another word. Informally, smaller distances indicate that two words are similar to each other, so only a few single-character edits would need to be done so that one word is changed to the other. Similarly, larger distances indicates that two words differ, so a lot of edits would be required so that one becomes the other.

The Levenshtein distance between two strings x, y, where x is of length $|x|$ and y is of length $|y|$, written as $\mathrm{lev}(x,\ y)$, can be defined as:

$$\mathrm{lev}(x,y) = \begin{cases} |x| & \text{if } |y| = 0 \\ |y| & \text{if } |x| = 0 \\ \mathrm{lev}(\mathrm{tail}(x),\mathrm{tail}(y)) & \text{if } x[0] = y[0] \\ 1 + \min \begin{cases} \mathrm{lev}(\mathrm{tail}(x),y)) \\ \mathrm{lev}(x,\mathrm{tail}(y)) \\ \mathrm{lev}(\mathrm{tail}(x),\mathrm{tail}(y)) \end{cases} & \text{otherwise} \end{cases} \tag{16}$$

where tail is a function that takes in some string a as its input and outputs all but the first character.

The Levenshtein distance is used to measure the similarity between a password, P, and a dataset of 200 commonly used passwords. The smallest distance between P and each of the 200 common passwords is used as the feature. Let P_c represent a set of 200 common passwords. Thus, the Levenshtein distance of P is:

$$\mathrm{lev}(P) := \min_{c \in P_c} \mathrm{lev}(P,c) \tag{17}$$

4.3.4 Character Repetition Weight Sum

Character repetition weight sum is a measure of consecutive repetitions of a character in a given password, P, which is weighted by how often a specific character is repeated. It is computed as follows:

Let P represent the password string of length n. Let $C(i)$ be a characteristic function that indicates whether the i-th and $(i-1)$-th characters of the password are the same:

$$C(i) = \begin{cases} 1 & \text{if } P[i] = P[i-1] \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

Let $R(i)$ be a function that outputs the number of consecutive repeating characters starting at position i in the password:

$$R(i) = \begin{cases} 1 + R(i-1) & \text{if } C(i) = 1 \\ 1 & \text{otherwise} \end{cases} \tag{19}$$

The character repetition weight sum is then calculated as the sum of the products of $R(i)$ and $R(i)-1$, considering only the last character of each sequence of consecutive repeating characters:

$$\text{weight sum} = \sum_{\substack{i=1 \\ C(i)\neq 1 \wedge C(i-1)=1 \\ \text{or} \\ i=n \wedge C(i)=1}}^{n} (R(i) \cdot (R(i)-1)) \tag{20}$$

The summation iterates over the positions i in the password where either a repeating sequence ends ($C(i) \neq 1 \wedge C(i-1) = 1$) or it is the last character of the password and part of a repeating sequence ($i = n \wedge C(i) = 1$).

Higher values of CRWS indicate that there are more consecutive values of repetition in a given password, P, which suggests that the password is weaker since there are more patterns in it.

4.3.5 Most Common Character Set Count

The most common character set count feature, C_{max}, calculates the maximum frequency of character sets in the alphabet. Let P be a password, and let A be the set of subsets of the alphabet, as defined in section 4.1. The most common character set count feature is calculated as:

$$C_{max}(P) = \max_{S \in A} \sum_{i=1}^{N} \mathbb{I}(P[i] \in S) \tag{21}$$

Higher values indicate that one character set is much more prominent than the others, which signals that the password is less complicated.

4.3.6 Character Frequency Ratio

The character frequency ratio feature, R_{freq}, computes the ratio between the maximum and minimum character frequency in the password. Let P be a password of length N, and let C be the set of unique characters in P. The character frequency ratio feature is calculated as:

$$R_{freq}(P) = \frac{\max_{c \in C} \sum_{i=1}^{N} \mathbb{I}(P[i] = c)}{\min_{c \in C} \sum_{i=1}^{N} \mathbb{I}(P[i] = c)} \tag{22}$$

If the password contains only one unique character, the ratio is set to 1. Higher values indicate a less even distribution of characters, which suggests that the password is more complicated.

4.3.7 Password Length to Unique Ratio

The password length to unique ratio feature, R_{unique}, calculates the ratio between the password length and the number of unique characters in the password. Let P be a password of length N, and let C be the set of unique characters in P. The password length to unique ratio feature is calculated as:

$$R_{unique}(P) = \frac{N}{|C|} \tag{23}$$

Higher values indicate that the password consists of fewer unique characters relative to its length, suggesting that it may be less complicated.

5 Supervised Learning Models

Both models were implemented using the Keras library. The models takes Shannon entropy, bigram frequencies, trigram frequencies, four-gram frequencies, Leveshtein distance, character repetition weight sum, consecutive character type count, most common character type count, character frequency ratio, and the password length to unique ratio as inputs and outputs one of the four strength classes discussed in section 4.2.

5.1 Multilayer Perceptron (MLP)

The MLP model is a feed-forward, fully connected neural network consisting of multiple layers of nodes with non-linear activation functions, dropout layers, and L1 regularization, as described in detail below.

5.1.1 Model Architecture

Let $x \in \mathbb{R}^n$ denote the input feature vector, where n is the number of input features. The model consists of several layers of transformations, including fully connected (dense) layers, dropout layers, and non-linear activation functions.

For a fully connected layer i with k_i nodes, let $W_i \in \mathbb{R}^{k_{i-1} \times k_i}$ denote the weight matrix and $b_i \in \mathbb{R}^{k_i}$ denote the bias vector. The output of a fully connected layer can be computed as follows:

$$h_i = W_i \cdot h_{i-1} + b_i \tag{24}$$

where h_{i-1} is the output of the previous layer, and h_i is the output of the current layer.

Non-linear activation functions are applied element-wise to the output of the fully connected layers:

$$h_i = f_i(h_i) \tag{25}$$

where $f_i(\cdot)$ is the activation function of layer i (e.g., ReLU, softmax).

Dropout layers are applied after certain fully connected layers during training. In a dropout layer i, each element of the output from the previous layer h_{i-1} is set to zero with probability p_i:

$$h_i = d_i(h_{i-1}) \tag{26}$$

where $d_i(\cdot)$ represents the dropout operation with dropout rate p_i.

For the 7th layer, L1 regularization is applied with a regularization weight λ. The L1 regularization term is added to the loss function and computed as follows:

$$L1 = \lambda \sum_{j=1}^{k_6} \sum_{i=1}^{k_7} |W_7(i,j)| \tag{27}$$

The final output of the model, h_{12}, is a probability distribution over the four password strength categories, produced by applying the softmax activation function to the final dense layer.

The complete architecture and the hyperparameters of the MLP model is described in greater detail in Appendix A.1.

5.2 Convolution Neural Network (CNN)

The 1D CNN model is a deep learning model designed to capture local patterns and spatial relationships in the input data through the use of convolutional and pooling layers. The model is composed of multiple layers including convolutional layers with filters and non-linear activation functions, max pooling layers, fully connected layers, and a softmax activation function for the output, as described in detail below.

5.2.1 Model Architecture

Let $x \in \mathbb{R}^{n \times 1}$ denote the input feature matrix, where n is the number of input features. The model consists of several layers of transformations, including 1D convolutional layers, max pooling layers, and fully connected (dense) layers.

For a 1D convolutional layer i with k_i filters, kernel size s_i, and stride t_i, let $W_i \in \mathbb{R}^{s_i \times k_{i-1} \times k_i}$ denote the filter weights and $b_i \in \mathbb{R}^{k_i}$ denote the bias vector. The output of a 1D convolutional layer can be computed as follows:

$$h_i[j] = f_i\left(\sum_{m=0}^{s_i-1} W_i[m, j] \cdot h_{i-1}[t_i \cdot j + m] + b_i[j]\right) \tag{28}$$

where h_{i-1} is the output of the previous layer, h_i is the output of the current layer, and $f_i(\cdot)$ is the activation function of layer i (e.g., ReLU).

Max pooling layers are applied after certain convolutional layers. In a max pooling layer i with pool size p_i, the output is computed as follows:

$$h_i[j] = \max_{m=0}^{p_i-1} h_{i-1}[p_i \cdot j + m] \tag{29}$$

Following the convolutional and max pooling layers, the feature maps are flattened into a 1D vector. The fully connected (dense) layers are then added. For a fully connected layer i with k_i nodes, let $W_i \in \mathbb{R}^{k_{i-1} \times k_i}$ denote the weight matrix and $b_i \in \mathbb{R}^{k_i}$ denote the bias vector. The output of a fully connected layer can be computed as follows:

$$h_i = W_i \cdot h_{i-1} + b_i \tag{30}$$

Non-linear activation functions are applied element-wise to the output of the fully connected layers:

$$h_i = f_i(h_i) \tag{31}$$

where $f_i(\cdot)$ is the activation function of layer i (e.g., ReLU for intermediate dense layers and softmax for the final dense layer).

The complete architecture and the hyperparameters of the CNN model is described in greater detail in Appendix A.2.

6 Unsupervised Learning Models

We were interested in using unsupervised learning methods to gain a deeper understanding of the underlying features in the data. By incorporating these methods, we hoped to identify which features are better at discriminating between classes.

6.1 K-Means Clustering

K-means is an unsupervised clustering algorithm that aims to partition the input data into k clusters, where each observation belongs to the cluster with the nearest mean. We chose k-means as our unsupervised learning method to explore the underlying structure of the password dataset. By partitioning the data into k clusters, we aimed to uncover patterns or relationships that could help us understand password strength evaluations.

6.1.1 Algorithm

The algorithm iteratively updates the cluster centroids and assigns each data point to the cluster with the nearest centroid. The steps are as follows:

1. Initialize the centroids randomly by selecting k data points.
2. Assign each data point to the closest centroid.
3. Update the centroids by calculating the mean of all data points assigned to each centroid.
4. Repeat steps 2 and 3 until the centroids' positions do not change significantly or a predefined number of iterations are reached.

The distance metric used for determining the closest centroid is the Euclidean distance. Given two points x_i and x_j in the feature space, the Euclidean distance is computed as follows:

$$d(x_i, x_j) = \sqrt{\sum_{p=1}^{n} (x_{ip} - x_{jp})^2} \tag{32}$$

where n is the number of input features.

6.1.2 Optimal Number of Clusters

To determine the optimal number of clusters (k) for the k-means algorithm, we employed the elbow method. This method involves running the K-means algorithm with varying values of k, ranging from 1 to 10 in this case. For each value of k, we computed the sum of squared errors (SSE) between each data point and its corresponding cluster centroid. The SSE measures the compactness of the clusters, with lower values indicating better clustering.

The SSE values were plotted against the number of clusters, as illustrated in Figure 3. In this plot, the value of k was chosen by an "elbow" point, which represents the value of k beyond which the SSE does not decrease significantly with the addition of more clusters. In other words, the elbow point indicates the optimal number of clusters where the within-cluster variance is minimized without unnecessarily increasing the number of clusters. In Figure 3, there appears to be two elbow points, $k = 2$ and $k = 4$. Thus, we the analysis will consider both cases, $k = 2$ and $k = 4$.

7 Experiment

In this section, we will discuss the techniques used to evaluate the performances of our supervised and unsupervised models. First, we will discuss how we trained our models. Next, we will discuss the metrics used to evaluate the performances. Then, we will review the performances of our models using the metrics we defined. Lastly, we will go over the different insights we can deduce from both models.

7.1 Training Run

We will begin by explaining our training processes for our supervised models and our unsupervised models.

7.1.1 Supervised Models

The training process of the MLP model used a batch size of 32 observations and ran for 50 epochs. The entire training process took approximately 10 minutes to complete. No adjustments to the model's hyperparameters were made during the training process. As show in Figure 4a, both the training loss and the validation loss followed an overall decreasing trend. Although there were some points where the training loss was higher than the validation loss, which could signal that the model was underfit during the instances, we can conclude that model was able to generalize the features well.

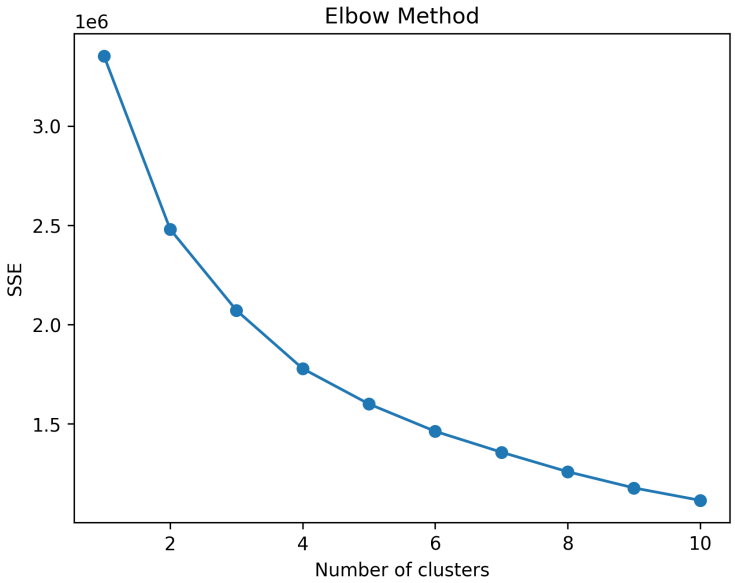
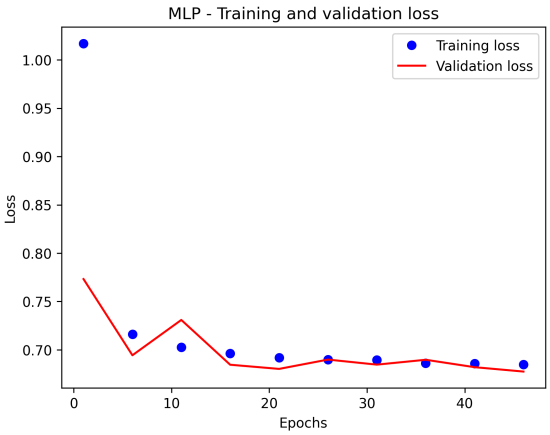
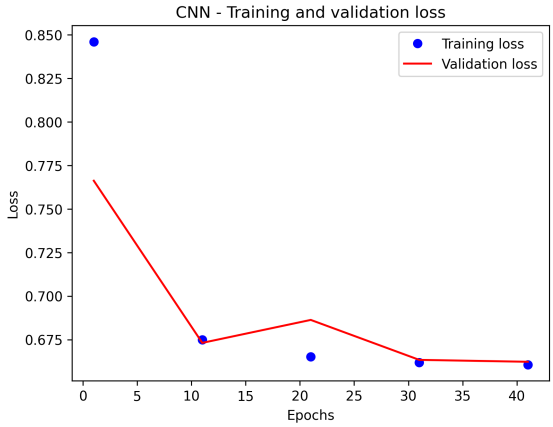


Figure 3: Scree plot for k-means.

The training process of the CNN model used a batch of size 32 and ran for 50 epochs. The entire training process took approximately 15 minutes to complete. Like the MLP training process, there were no adjustments to the CNN's hyperparameters were made during the training process. As show in Figure 4b, both the training loss and the validation loss decreased throughout the process. Thus, the model has adapted well to the training data.



(a) MLP training and validation loss.



(b) CNN training and validation loss.

Figure 4: Training and validation loss for MLP and CNN models.

The final validation accuracy for the MLP is 70.68%, and the final validation accuracy for the CNN is 70.05%. Both models performed well in the training run overall, and the MLP only performed slightly better than the CNN.

7.1.2 Unsupervised Models

K-means is sensitive to outliers, so the features were normalized before the algorithm was executed to ensure that none of the features dominated the clusters. The hyperparameters are described in detail in

Appendix A.3.1 .

7.2 Evaluation Metrics

Next, we will introduce the metrics we used to evaluate the performance of the models.

7.2.1 Supervised Models

- **Test accuracy:** The ratio of correct predictions to the total number of predictions. A higher value indicates that a model performs better.
- **Precision:** The ratio of true positives to the sum of true positives and false positives. A higher value indicates a higher accuracy for positive predictions.
- **Recall:** The ratio of true positives to the sum of true positives and false negatives. A higher value indicates that there are few false negatives.
- **F1-score:** The harmonic mean of precision and recall. A higher F-1 score indicates the model is good quality classifier.
- **AUC/ROC curve:** The area under the receiver operating characteristic (ROC) curve, which plots the true positive rate (TPR) against the false positive rate (FPR). It measures the model's ability to discriminate between different classes. The value represents the model's ability to correctly classify the instances across all classes, with an AUC of 1 being a perfect classifier and an AUC of 0.5 indicating no better performance than random chance.

7.2.2 Unsupervised Models

- **Silhouette score:** Measures how similar an object is to its own cluster compared to other clusters. Ranges from -1 to 1, where a higher value indicates better clustering.
- **Davies-Bouldin score:** Measures the average similarity between each cluster and its most similar one. A lower DB score indicates better separation between clusters.
- **Adjusted Rand score:** Measures the similarity between true labels and predicted labels, adjusted for chance. Ranges from -1 to 1, with 1 indicating perfect agreement between true and predicted labels.
- **Normalized Mutual Information (NMI) score:** Measures the mutual information between true labels and predicted labels, normalized by the geometric mean of their individual entropies. Ranges from 0 to 1, with 1 indicating perfect agreement between true and predicted labels.
- **Adjusted Mutual Information (AMI) score:** Similar to NMI but adjusted for chance. Ranges from 0 to 1, with 1 indicating perfect agreement between true and predicted labels.

7.3 Performance of Supervised Models

As shown in Table 1 for the MLP model and Table 2 for the CNN model, both models achieved an overall accuracy of 0.70, demonstrating that they can correctly classify the password strength approximately 70% of the time. The weighted averages for precision, recall, and f1-score were also similar for both models.

The performance of the model varies by the category. For the MLP, the weak class had a precision of 0.71 and a recall of 0.64, indicating that when the model predicted weak, it was correct 71% of the time, and it correctly identified 64% of the true weak instances. On the other hand, the very strong class had a higher precision of 0.89 but a lower recall of 0.52, suggesting that the MLP model was more conservative in predicting very strong and missed a significant portion of the true strong instances. In other words, when the model predicted an observation as very strong, it was correct 89% of the time, but it only identified 52% of the samples in the very strong class. The strong class had a precision of 78% and a recall of 85%. The high precision and low recall for the very strong class and the lower precision from the strong class suggests that the model struggled to distinguish the very strong class from the strong class. Since the recall of the

strong class was 85%, this suggests that the model was able to correctly identify most of the observations in the strong class. It is possible that the model falsely labeled those in the very strong class as just strong, which would explain the lower precision and higher recall in the strong class.

	Precision	Recall	F1-score	Support
Weak	0.71	0.64	0.67	23211
Medium	0.57	0.81	0.67	23365
Strong	0.78	0.85	0.81	23155
Very strong	0.89	0.52	0.65	23509
Accuracy			0.70	93240
Macro avg	0.74	0.70	0.70	93240
Weighted avg	0.74	0.70	0.70	93240

Table 1: MLP model metrics.

We can see that the CNN model experienced a similar trend. The very strong class had a high precision of 83%, but had a lower recall of 56%, which reveals that the model did not notice all of the true very strong classes. The strong class had a precision of 76%, and an even higher recall of 88%. So, the model was able to identify most of the strong classes, but the precision suggests that it also incorrectly classified a substantial number of observations as strong,

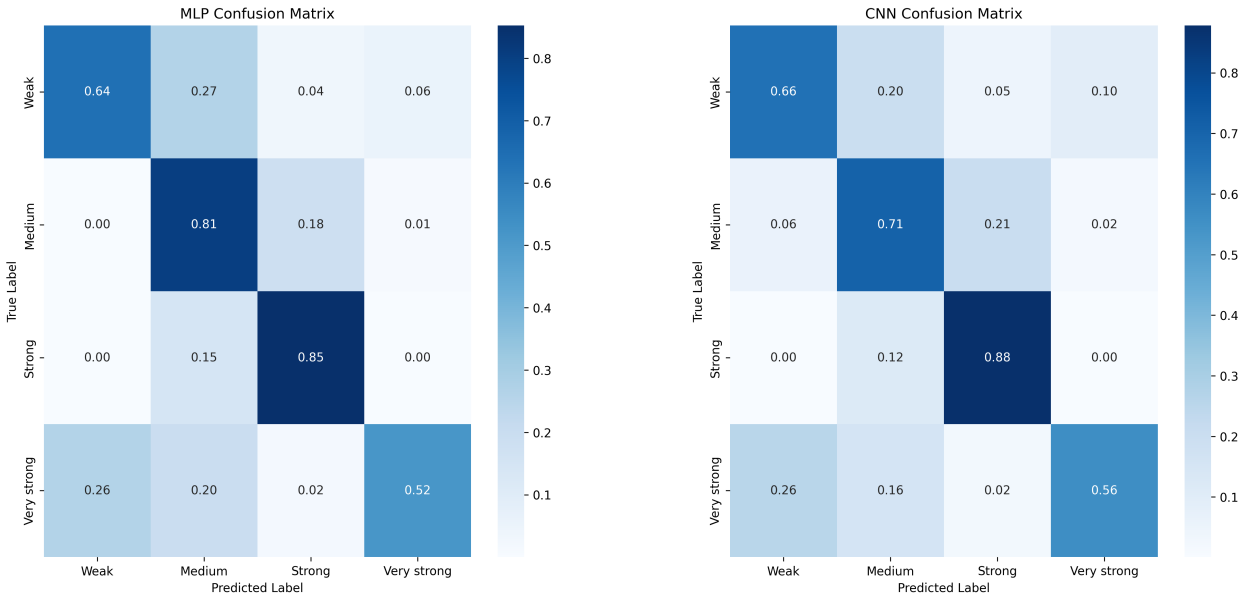
	Precision	Recall	F1-score	Support
Weak	0.67	0.66	0.66	23211
Medium	0.60	0.71	0.65	23365
Strong	0.76	0.88	0.81	23155
Very strong	0.83	0.56	0.67	23509
Accuracy			0.70	93240
Macro avg	0.71	0.70	0.70	93240
Weighted avg	0.71	0.70	0.70	93240

Table 2: CNN model metrics.

More broadly, the differences in precision and recall between classes for both models suggest that the underlying features and patterns that distinguish these classes are not equally identifiable by the models. The higher recall for the strong class could be a result of more easily discernible patterns that the both models can detect. Meanwhile the lower recall for the very strong class might be due to more complex patterns that the models struggled to recognize. We will look at the confusion matrices to better understand the misclassifications of the model.

Figure 5 illustrates the confusion matrices for the MLP and the CNN model. The matrices reaffirmed that both models could predict the strong class particularly well. For the MLP model, it had 19740 true positives for the strong class, with only 2 misclassified as weak, 3405 misclassified as medium, and 8 misclassified as very strong. The CNN model had 20331 true positives for the strong class, with 72 misclassified as weak, 2730 misclassified as medium, and 22 misclassified as very strong. Since the medium class had the highest number of misclassifications in both models for the strong class, it can be concluded that the models struggled a bit with differentiating the strong class from the medium class. A similar trend holds for the CNN model, which also revealed that the model has a bit of trouble discriminating between the medium class and the strong class.

Perhaps the most interesting takeaway from Figure 5 is the last row of both matrices. The MLP model correctly classified very strong passwords only 52% of the time. It misclassified strong passwords as very strong only 2% of the time, and misclassified weak passwords as very strong 26% of the time. The CNN model showed a similar trend. It correctly classified very strong passwords 56% of the time, misclassified strong passwords as very strong 2% of the time, and misclassified weak passwords as very strong 26% of



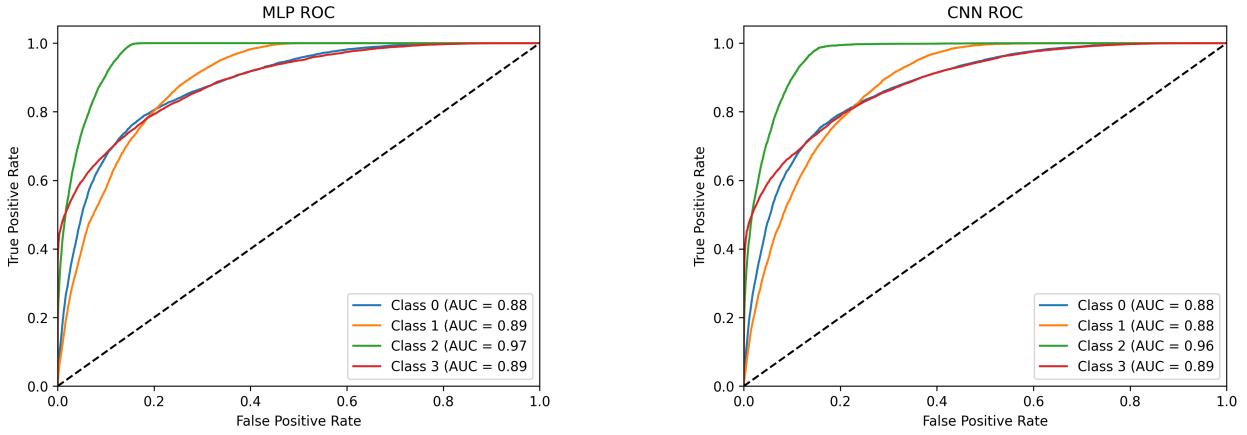
(a) MLP confusion matrix.

(b) CNN confusion matrix.

Figure 5: Confusion matrices for supervised models.

the time. This might seem paradoxical at first. Intuitively, one would expect the models to struggle to differentiate between similar classes (i.e., weak vs medium, strong vs very Strong). However, this is not the case for weak passwords. For the MLP model, it correctly classified weak passwords 64% of the time, and misclassified weak passwords as very strong only 6% of the time. For the CNN model, it correctly classified weak passwords 66% of the time, and misclassified weak passwords as very strong only 10% of the time.

The underlying features used to classify password strength could be the root of this interesting observation. It is quite possible that there are specific features in both weak and very strong passwords that could have caused the model to confuse the two classes. For example, very strong passwords might contain unexpected combinations of characters that make them seem more like weak passwords to the models. Further investigation into the feature importance and model architecture could help in understanding this behavior.



(a) MLP ROC curve.

(b) CNN ROC curve.

Figure 6: ROC curves for supervised models.

Figure 6 displays the Receiver Operating Characteristic (ROC) curves for both the MLP and the CNN

models. Both models had a macro-average Area Under the Curve (AUC) of 0.90, which suggests that they were both able to effectively discriminate between the different password strengths. Moreover, they performed similarly across all classes. In both the MLP and the CNN, it can be seen that the smallest AUC was 0.88, which was the weak class for the MLP and was both the weak class and the medium class for the CNN. For the MLP, the AUC of the medium class was 0.89. The very strong class had an AUC of 0.89 for both the MLP and CNN. The strong class had the highest AUC in both models, with a AUC of 0.97 in the MLP and a AUC of 0.96 in the CNN. 1 and Table 2, which also suggested that the model is able to identify those in the strong class. This was consistent from the observations made from Table 1 and Table 2, since both of these tables suggested that both models do a good job at identify strong passwords.

7.4 Performance of Unsupervised Models

We assessed the performance of our k-means algorithm through interpreting the Sillhouette score, the Davies-Bouldin score, the adjusted rand score, the normalized mutual information score, and the adjusted mutual information score.

7.4.1 Two Clusters

We first performed k-means with $k = 2$. Table 3 shows us the distribution of the zxcvbn classes within each cluster. Cluster 1 had less observations than cluster 0, and mainly consisted of passwords of the weak class, but had a relatively small size size. Cluster 0 contained a mix of all four classes.

Cluster	Weak	Medium	Strong	Very Strong
0	109681	114497	114905	115078
1	6869	2053	1645	1472

Table 3: Distribution of zxcvbn classes within each cluster for $k = 2$.

We performed dimension reduction through principal component analysis (PCA) so that we could visualize the clusters in a two-dimensional space. Figure 7 visualizes the clusters through PCA. The clusters do not appear to be distinct on the plot, which suggests that k-means may not have found any significant clusters in the data, at least in the first two principal components. It is clear that none of the clusters overlap with each other and are separated; however, the clusters are not really that far from each other. It is possible that underlying patterns in the password strength classes lie in higher dimensions or are not linearly separable.

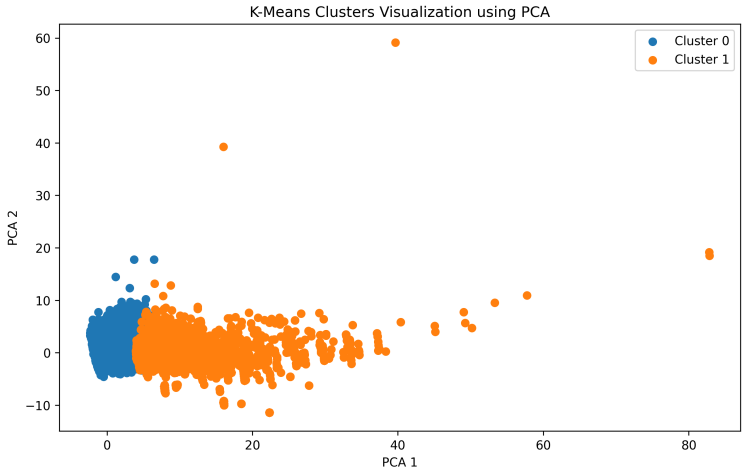


Figure 7: K-means clusters visualization using PCA.

Table 4 reveals different metrics that assessed the performance of our algorithm. The silhouette score of 0.7321 suggested that the clusters are reasonably cohesive, which indicates better clustering. The Davies-

Bouldin score of 0.9053 was relatively low, indicating that the clusters are not too dispersed and have a reasonable separation from each other. However, the Adjusted Rand Index (ARI) of 0.0004 was very low, implying that the clusters do not align well with the true password strength classes. Similarly, the Normalized Mutual Information (NMI) and Adjusted Mutual Information (AMI) scores of 0.0088 also indicated weak correspondence between the clusters and the true classes. Moreover, the Silhouette score and the Davies-Bouldin score both did not reasonably aligned with our findings from our PCA plot. This signaled that perhaps in a higher dimension, the clusters are more cohesive and distinct. The ARI, NMI, and AMI match better with our findings from our PCA plot.

Metric	Value
Silhouette Score	0.7321
Davies-Bouldin Score	0.9053
Adjusted Rand Index	0.0004
Normalized Mutual Information	0.0088
Adjusted Mutual Information	0.0088

Table 4: K-means clustering performance metrics for $k = 2$.

Overall, although the silhouette score and Davies-Bouldin score suggested that clusters appear to be well separated and cohesive, they do not seem to effectively capture the underlying password strength classes, suggesting that the k-means algorithm with $k = 2$ may not be the best approach for classifying password strength. This makes sense given that there are four actual classes in the dataset, and there are only two clusters here.

7.4.2 Four Clusters

Next, we performed k-means with $k = 4$. Table 5 shows the distribution of password strengths within each of the four clusters. Cluster 0 contained a large number of strong and very strong passwords, with a lower count of medium and weak passwords. Cluster 1 had a more balanced distribution of password strengths, with the highest count in the medium category. Cluster 2 primarily consisted of weak passwords, with a smaller number of medium and strong passwords, and the least number of very strong passwords. Cluster 3 had a larger number of medium and weak passwords, and fewer strong and very strong passwords.

Cluster	**Medium**	**Strong**	**Very Strong**	**Weak**
0	9310	41703	98096	6519
1	13791	15421	11152	14200
2	902	801	636	4671
3	92547	58625	6666	91160

Table 5: Distribution of zxcvbn classes within each cluster for $k = 4$.

Similarly to when we were analyzing the case where $k = 2$, we performed dimension reduction through PCA so that we can visualize the clusters in a two-dimensional space. Figure 8 visualizes the clusters through PCA. The clusters do not appear to be distinct on the plot, which suggests that k-means may not have found any meaningful clusters in the data, at least in the first two principal components.

We performed dimension reduction through Principal component analysis (PCA) so that we can visualize the clusters in a two-dimensional space. Figure 7 visualizes the clusters through PCA. The clusters do not appear to be distinct on the plot, which suggests that k-means may have struggled to cluster the data, at least in the first two principal components. The clusters appear to slightly overlap with each other. Just like when $k = 2$, it is possible that underlying patterns in the password strength classes lie in higher dimensions or are not linearly separable.

Table 6 reveals different metrics that assess the performance of our algorithm. Our performance metrics are different than what Table 4 revealed for when $k = 2$. The silhouette score of 0.2351 is lower, suggesting that the clusters are less well separated and cohesive. The Davies-Bouldin score of 1.2822 is higher than with

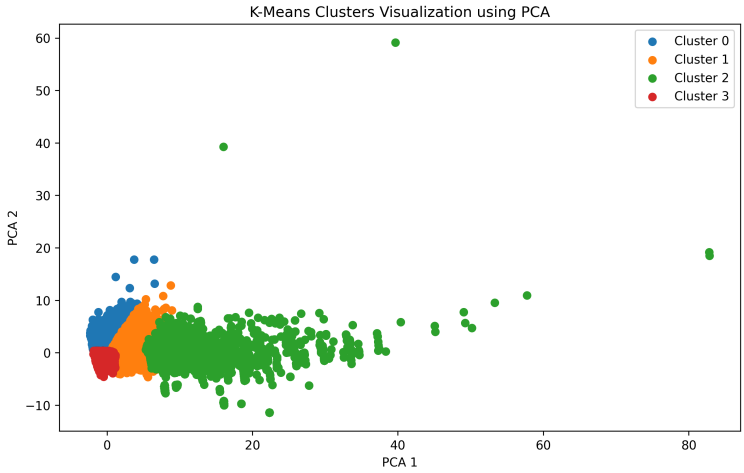


Figure 8: K-means clusters visualization using PCA.

$k = 2$, indicating that the clusters are more dispersed and less separated. However, the Adjusted Rand Index (ARI) of 0.2077 shows a significant improvement compared to $k = 2$, suggesting that the clusters align better with the true password strength classes. Similarly, the Normalized Mutual Information (NMI) and Adjusted Mutual Information (AMI) scores of 0.2227 also indicated a better correspondence between the clusters and the true classes when compared to $k = 2$. In summary, although the clusters were less well separated and cohesive when $k = 4$, they seem to capture the underlying password strength classes more effectively than when $k = 2$. This makes sense give that the actual dataset has four classes, which suggests that k-means clustering with $k = 4$ might provide a better representation of password strength classes, despite the trade-off in cluster quality.

Metric	Value
Silhouette Score	0.2351
Davies-Bouldin Score	1.2822
Adjusted Rand Index	0.2077
Normalized Mutual Information	0.2227
Adjusted Mutual Information	0.2227

Table 6: K-means clustering performance Metrics for $k = 4$.

In conclusion, the analysis with K-means clustering using $k = 4$ provided a more accurate representation of the password strength classes compared to when $k = 2$. The distribution of password strengths within each cluster, as shown in Table 5, indicates that the algorithm managed to separate the passwords into different groups with varying degrees of strength. However, the PCA visualization in Figure 8 suggests that the clusters are not clearly distinct in the first two principal components, possibly indicating that the underlying patterns in the password strength classes lie in higher dimensions or are not linearly separable. Our metrics in Table 6 suggests there is a trade-off in cluster quality. All in all, it appears that k-means with $k = 4$ provided a more meaningful representation of the clusters, despite the cluster quality not being as good when $k = 2$.

7.5 Feature Importance Analysis

In this section, we will look for the most influential features.

7.5.1 Supervised Models

Table 7 ranks the importance of features for the MLP model. Four-gram frequency held the highest rank with an importance of 125.99, followed closely by Levenshtein distance, with an importance of 123.45. Character repetition weight sum had the lowest importance of 25.43. Table 8 ranks the importance of features for the CNN. Shannon entropy had the highest importance with an importance of 0.293, followed by character frequency ratio with an importance of 0.152. Password length to unique ratio had a score of 0, indicating that it had no importance in the model.

Ranking	Feature	Importance
1	Four-gram Frequency	125.99
2	Levenshtein Distance	123.45
3	Trigram Frequency	115.98
4	Shannon Entropy	106.79
5	Password Length to Unique Ratio	106.11
6	Bigram Frequency	72.76
7	Character Frequency Ratio	46.56
8	Most Common Character Type	41.93
9	Character Repetition Weight Sum	25.43

Table 7: MLP feature importance ranking

Ranking	Feature	Importance
1	Shannon Entropy	0.293
2	Character Frequency Ratio	0.152
3	Most Common Character Type	0.125
4	Levenshtein Distance	0.100
5	Character Repetition Weight Sum	0.038
6	Bigram Frequency	0.034
7	Trigram Frequency	0.007
8	Four-gram Frequency	0.002
9	Password Length to Unique Ratio	0.000

Table 8: CNN feature importance ranking.

It is interesting to note that the features for the MLP model held more importance than in the CNN models. For example, Shannon entropy held the highest rank in the CNN, but it also only had an importance score of 0.293, which was lower than any of the importance scores for the MLP model. This implies that these features may not hold as much influence in the CNN. Moreover, two of the three n-grams made it to the top three most important features of the MLP model, but all of the n-grams were on the bottom half of the rankings for the CNN. Shannon entropy and the Levenshtein distance were in the top four features for both models. Despite having different feature importance rankings, both models were able to achieve comparable accuracy. This demonstrates that there are different ways for the models to learn effectively from the same data, hence their different rankings of features.

7.5.2 Unsupervised Models

Let us first consider the case when $k = 2$. By examining the centroids of each cluster in Table 9, we can see that some features appear to be more important in distinguishing between the clusters. In Cluster 0, n-grams have the highest value, which means that cluster, on average, contains passwords with higher values of the n-grams. Password length to unique ratio has the lowest value, indicating that this cluster contains passwords that have low values of this metric. The variety of centroid values in this cluster do not really signal whether this cluster contains a majority of a specific category. This makes sense, given that we saw in Section 7.4.1 that this cluster contains a mix of passwords from all four categories.

In general, Cluster 1 contains metrics with lower values. Unlike Cluster 0, the n-grams hold the lowest values. Character repetition weight sum and password length to unique ratio were the highest values in this cluster. It is interesting to note that they also were the lowest values in Cluster 0. Moreover, as seen in section 7.4.1., this cluster 1 contained a small number of passwords, but they were mostly weak passwords. This suggests that perhaps a large character repetition weight sum and password length to unique ratio is actually associated with weaker passwords.

Cluster	0	1	2	3	4	5	6	7	8
0	0.057	0.121	0.136	0.112	0.013	-0.058	-0.002	-0.039	-0.093
1	-2.119	-4.570	-5.100	-4.177	-0.465	2.194	0.129	1.495	3.527

Table 9: K-means centroids for $k = 2$.

Now we will consider $k = 4$. Table 10 contains the centroids for the clusters. The features with the highest centroids in Cluster 0 were Shannon entropy and Levenshtein distance. Recall from Section 7.4.2 that Cluster 0 mostly contained very strong and strong passwords. Thus, this suggests that strong and very strong passwords are associated with high values of Shannon entropy and Levenshtein distance.

Section 7.4.2 showed us that Cluster 1 contained a variety of password strengths. It is interesting to highlight that most of the centroids were negative here, except for character repetition weight sum, character frequency ratio, and password length to unique ratio. In the case of $k = 2$, we saw that Cluster 1, which had a majority of weak passwords, also had a large character weight repetition sum and password length to unique ratio. In this case, these large values gave us a cluster of mixed passwords. Cluster 1 in the case of $k = 2$ did have a relatively small sample size though, so further research could be done to examine if these features are actually associated to weaker passwords or not.

Similarly to cluster 1 when $k = 2$, cluster 2 when $k = 4$ had a small number of observations and primarily consisted of weak passwords. Moreover, both of these clusters actually shared a similar trend in centroids. As mentioned in the previous paragraph, further investigation could be done to see if this trend is only held because of its small sample size, or if it is possible to see this occur with a larger sample size.

Cluster 3 had high centroids for the n-grams, similarly to cluster 0 when $k = 2$. However, they did not share the lowest centroids. As seen in section 7.4.2, this cluster mostly had weak and medium passwords. This may suggest that higher values of n-grams are associated with weak and/or medium passwords.

Cluster	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0.825	0.202	0.166	0.113	0.536	-0.053	0.887	0.328	0.020
1	-1.084	-1.390	-0.433	-0.007	-0.081	0.696	-0.020	1.125	1.301
2	-2.368	-5.239	-6.605	-6.520	-0.587	2.904	0.241	1.660	4.207
3	-0.211	0.326	0.177	0.115	-0.301	-0.200	-0.556	-0.499	-0.415

Table 10: K-means centroids for $k = 4$.

8 Ethics, Limitations, and Implications

In this section, we will discuss the ethics, limitations, and implicaitons of our research.

8.1 Ethics

Passwords are crucial for protecting the sensitive information of users. The main goal of AI-driven password evaluation should be to help people protect their data and privacy. Any password evaluation method should prioritize user security, and should not be used for malicious activities. As we have shown in this report, AI can be used to evaluate password strengths. However, it can also be exploited by hackers for password cracking. Thus, researchers in this field should keep this in mind to prevent unethical applications of their work.

8.2 Limitations

There are several limitations in this paper that should be acknowledged:

- Time and resources were limited, so only a small amount of hyperparameter combinations were considered. Hyperparameter tuning is an exhaustive process; an in-depth exploration of hyperparameters may discover hyperparameters that could lead to better results.
- The dataset is from 2015. It is possible that the consensus around password strengths have changed since then, causing the passwords that are used now to be different than what was used in that time. In order words, there may have been a shift in how users choose and select their passwords, thus it is important to consider that this dataset is older and may not reflect passwords that are used today.
- The study focused solely on the zxcvbn algorithm for password strength assessment, which does not represent the entire landscape of password strength evaluation techniques. There are a multitude of password strength evaluation algorithms that could be analyzed. Future research should consider other algorithms to provide a more comprehensive understanding of password strength classification.
- The order of inputs for the CNN was not experimented with, which could have potentially impacted the performance of the model. Since CNNs take sequential data into account, future research should investigate the effect of varying input order on the model's performance.

8.3 Implications

This study has several implications for future research and practice in the field of password security:

- The results suggest that deep learning, specifically MLPs and CNNs, can be effectively used for password strength classification. This opens up new avenues for research in leveraging deep learning techniques for improving password security.
- CNNs are normally thought of a technique that is used for image classification. The success of the CNN model in this study implies that more research should be conducted to understand the underlying reasons for its performance and identify areas for further improvement.
- The models' struggle with classifying very strong passwords as weak highlights the need for continued analysis in this area. It would be interesting to know if this problem is unique to the zxcvbn algorithm, or if other password strength evaluation algorithms also experience a similar issue.

9 Conclusion

The insights derived from both supervised and unsupervised models complement each other. By definition, supervised models are trained on pre-labeled data. In this case, the zxcvbn algorithm was used to classify the password strengths. The supervised methods provided more interpretable results. For example, it was clear to see which features were seen as more important for either. Basically, supervised methods are useful for understanding which features are essentially for classifying password strength. Unsupervised approaches provided a more exploratory understanding of the data, since the data is unlabeled. Patterns within the different clusters could be detected, which can be used to better understand the similarities and differences across password strengths. This could be especially crucial for our supervised models. Although the supervised models, overall, had a high accuracy, they struggled with classifying very strong passwords from weak passwords. An in-depth analysis on the underlying structures that make up these passwords could be performed, and thus potentially help fix this issue that the supervised models experienced. But this is not the only way that unsupervised methods relate to supervised methods. Based on the findings of k-means, there was not an explicit linear relationship between centroids and password strengths. Moreover, this could be demonstrated in the PCA plots, since k-means struggled to separate the clusters in a two-dimensional space. K-means suggested that the patterns in the features was non-trivial, which implied that deep learning

could be appropriate since they are generally used for higher level data since they can detect subtle patterns. Just as we saw, both deep learning models were able to classify password strengths fairly well. In short, both supervised approaches and unsupervised approaches offer valuable insights to password strength evaluation that complement each other. Together, both approaches can be used to understand the underlying patterns within passwords, and which of these patterns are important for assessing pattern strengths.

This was touched upon earlier, but research must be done to investigate why both supervised models misclassified very strong passwords as weak. Although the converse did not hold, misclassifcations could cause a serious security threat. Therefore, this issue must be investigated further. Moreover, it was interesting to see a CNN perform almost as well as a MLP. CNNs are associated with image classification since they take in sequential data. Further research should be done to examine the use of CNNs as more sophisticated password evaluation metrics. Perhaps there are some sequential relationships between the features themselves that CNNs are able to capture. As the field of password evaluation evolves, researchers must be careful to ensure their research is not used maliciously by those who seek to access sensitive data of users worldwide.

10 References

[1] Drapkin, Aaron. "Data Breaches That Have Happened in 2023 so Far - Updated List." Tech.co, 2 May 2023, https://tech.co/news/data-breaches-updated-list.

[2] Wheeler, Dan. "Zxcvbn: Realistic Password Strength Estimation." Dropbox, https://dropbox.tech/security/zxcvbn-realistic-password-strength-estimation.

A Model Architecture and Hyperparameters

This appendix will provide extended details on the models.

A.1 MLP

This section will describe the architecture and hyperparameters of the MLP model.

A.1.1 Architecture

Table 11 provides a detailed overview of the architecture of the MLP model.

Layer	Type	Description
h_0	Input	$h_0 = x$
h_1	Fully Connected	512 nodes, ReLU activation: $h_1 = f_1(W_1 \cdot h_0 + b_1)$
h_2	Dropout	Dropout rate of 0.1: $h_2 = d_2(h_1)$
h_3	Fully Connected	256 nodes, ReLU activation: $h_3 = f_3(W_3 \cdot h_2 + b_3)$
h_4	Fully Connected	128 nodes, ReLU activation: $h_4 = f_4(W_4 \cdot h_3 + b_4)$
h_5	Dropout	Dropout rate of 0.1: $h_5 = d_5(h_4)$
h_6	Fully Connected	64 nodes, ReLU activation, L1 regularization: $h_6 = f_6(W_6 \cdot h_5 + b_6)$
h_7	Fully Connected	32 nodes, ReLU activation: $h_7 = f_7(W_7 \cdot h_6 + b_7)$
h_8	Dropout	Dropout rate of 0.1: $h_8 = d_8(h_7)$
h_9	Fully Connected	16 nodes, ReLU activation: $h_9 = f_9(W_9 \cdot h_8 + b_9)$
h_{10}	Fully Connected	8 nodes, ReLU activation: $h_{10} = f_{10}(W_{10} \cdot h_9 + b_{10})$
h_{11}	Output	3 nodes, softmax activation: $h_{11} = f_{11}(W_{11} \cdot h_{10} + b_{11})$

Table 11: Architecture of the MLP model.

A.1.2 Hyperparameters

The following is a list of hyperparameters of the model:

Shape hyperparameters and their values:

- $L = 8$ (number of fully connected layers)
- $D_{in} = 8$ (input dimension)
- $D_{out} = 3$ (output dimension)
- k_i: [512, 256, 128, 64, 32, 16, 8] (number of nodes for fully connected layers 1-7)
- d_i: [0.1, 0.1, 0.1] (dropout rates for dropout layers 1-3)

Initialization hyperparameters:

- Initialization method: Glorot Uniform
- Weight initialization range: $[-\sqrt{\frac{6}{k_{i-1}+k_i}}, \sqrt{\frac{6}{k_{i-1}+k_i}}]$
- Bias initialization: zeros

A.2 CNN

This section will describe the architecture and the hyperparameters of the CNN model.

A.2.1 Architecture

Table 12 provides a detailed overview of the architecture of the MLP model.

Layer	**Description**
Input	$h_0 = x$
1D Convolutional (Conv1)	32 filters, kernel size 3, ReLU activation
Max Pooling (Pool1)	Pool size 2
1D Convolutional (Conv2)	8 filters, kernel size 2, ReLU activation
Flatten	$h_4 = \text{Flatten}(h_3)$
Fully Connected (FC1)	64 nodes, ReLU activation
Output	3 nodes, softmax activation

Table 12: Architecture of the CNN model.

A.2.2 Hyperparameters

The following is a list of hyperparameters of the model:

Shape hyperparameters and their values:

- $L = 9$ (number of layers, including dense and dropout layers)
- $D_{in} = 8$ (input dimension)
- $D_{out} = 3$ (output dimension)
- k_i: [512, 256, 128, 64, 32, 16, 8] (number of nodes for dense layers 1-7)
- d_i: [0.1, 0.1, 0.1] (dropout rates for dropout layers 1-3)

Initialization hyperparameters:

- Initialization method: Glorot Uniform
- Weight initialization range: $[-\sqrt{\frac{6}{k_{i-1}+k_i}}, \sqrt{\frac{6}{k_{i-1}+k_i}}]$
- Bias initialization: zeros

Total parameters:

- Total parameters: 179,347
- Trainable parameters: 179,347
- Non-trainable parameters: 0

A.3 K-means

A.3.1 Hyperparameters

The following is a list of hyperparameters of the k-means algorithm:

- k: number of clusters (selected based on the elbow method)
- Initialization method: K-means++
- Maximum iterations: 300
- Convergence tolerance: $1e^{-4}$

B Feature Numbers to Feature Names

In our report, for some of the visualizations and tables, we referenced the features by numbers rather than their actual names to make the tables and visualizations more concise. Table 13 provides this mapping.

Number	Feature
0	Shannon Entropy
1	Bigram Frequency
2	Trigram Frequency
3	Four-gram Frequency
4	Levenshtein Distance
5	Character Repetition Weight Sum
6	Most Common Character Type
7	Character Frequency Ratio
8	Password Length to Unique Ratio

Table 13: Mapping of feature numbers to feature names.