

**BLM3120 – Bilgiye Erişim ve Arama Motorları**

Prof. Dr. Mehmet Sıddık AKTAŞ

Predicting Password Strength

PROJECT REPORT

|  |  |
| --- | --- |
| Hüsamettin IŞIKTAŞ | Ege Alp HEYİK |
| 21011014 | 21011041 |
| [husamettin.isiktas@std.yildiz.edu.tr](mailto:husamettin.isiktas@std.yildiz.edu.tr) | [alp.heyik@std.yildiz.edu.tr](mailto:alp.heyik@std.yildiz.edu.tr) |

1. **Introduction**

In today's digital era, securing sensitive information has become increasingly critical. Passwords are often the first line of defense against unauthorized access. However, weak or predictable passwords remain a significant vulnerability, making them a prime target for cyberattacks. Predicting password strength is important to improving security measures about the importance of creating robust passwords.

By leveraging supervised machine learning algorithms, we aim to develop a predictive model that evaluates the strength of passwords based on their characteristics. Such a model can be used to guide users in creating stronger passwords.

Through this project, we hope to provide insights into the advantages and limitations of different machine learning algorithms for this task and contribute to the password security research field.

1. **Dataset Description**

‘Password Strength and Vulnerability’ Dataset (can be found in: <https://www.kaggle.com/datasets/utkarshx27/passwords>.) is used for this project.

**A screenshot of a computer

Description automatically generated**

1. **Experimental Setup & Data Preperation**

Dataset has 9 features. We inspected the dataset and made some experiments and analysis on it. We did correlation analysis and inspected the distribution of the dataset. Here is initial columns of dataset:

**#   Column Non-Null Count   Dtype**

-   ------             --------------   ----- -----

**0**   rank 500 non-null   float64

**1**   password         500 non-null   object

**2**   offline\_crack\_sec   500 non-null   float64

**3**   strength           500 non-null   float64

**4**   font\_size           500 non-null   float64

**5**   category           500 non-null   object

**6**   value                     500 non-null float64

**7**   time\_unit               500 non-null   object

**8**   rank\_alt           500 non-null   float64

After some inspection we noticed variable *offline\_crack\_sec* can be calculated with *“value x time\_unit”*. Therefore, *value* and *time\_unit* variables are dropped. With correlation analysis we noticed *rank\_alt* variable is identical to the rank variable, because of that we dropped it. Then there is 6 columns left.

A red and blue squares with white text

Description automatically generated

After that we needed more columns for train a machine learning model on this dataset. Because features were not enough and some features like *font\_size* and *offline\_crack\_sec* is not seemed computable and properly explained. Also *font\_size* and *strength* had a very high positive correlation. Generally high positive correlation causes wrong predictions. Since it weighs more than the others over strength column, it may cause misleading decisions.

So we decided to extract features from words. Our code takes a word and extract features below:

**length**: The total number of characters in the password.

**unique\_chars**: The count of unique characters in the password

**uppercase\_ratio**: The number of uppercase letters in the password.

**lowercase\_ratio**: The number of lowercase letters in the password.

**digit\_ratio**: The number of numerical digits (0–9) in the password.

**special\_ratio**: The count of special characters (e.g., !@#$%^&\*() in the password.)

After that, 6 new columns are added to dataset. With these new features we noticed *font\_size* was a combination of these features. So we dropped *font\_size* column as well. And the last we decoded category column because it was a categorical data and we needed to make it numerical for trainin. So there was 11 columns at the end of these processes. The final dataset’ columns are below.

**#   Column Non-Null Count   Dtype**

-   ------             --------------   ----- -----

**0**   rank 500 non-null   float64

**1**   password         500 non-null   object

**2**   offline\_crack\_sec   500 non-null   float64

**3**   strength           500 non-null   float64

**4**   length             500 non-null   float64

**5**   unique\_chars       500 non-null float64

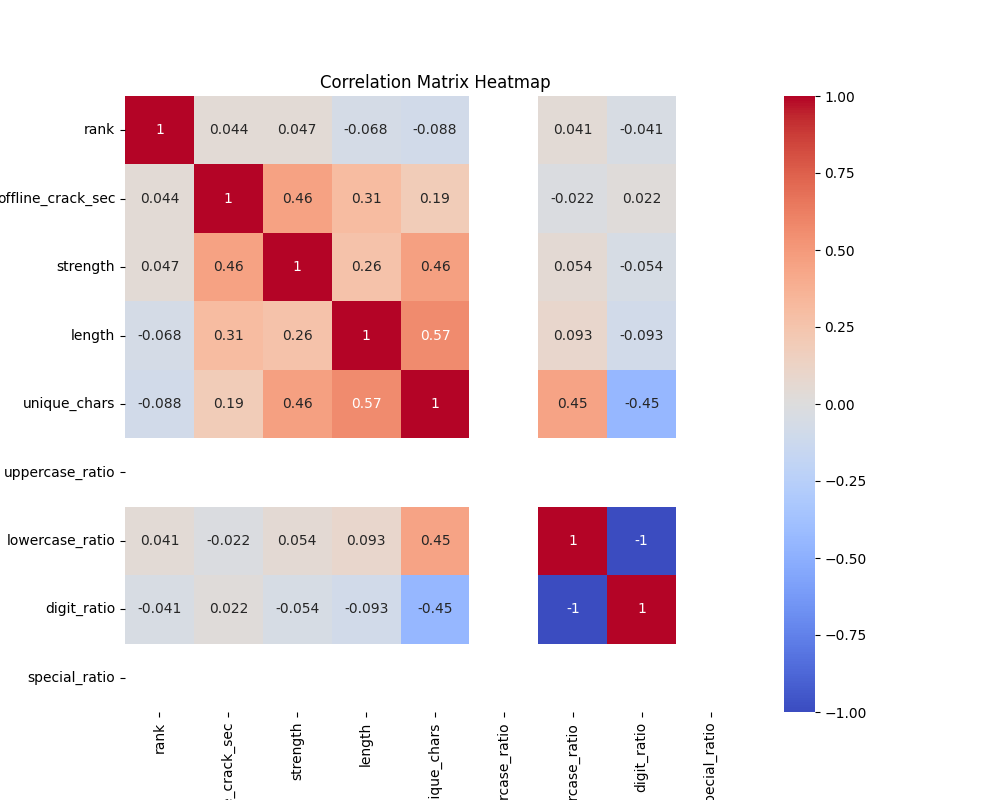
**6**   uppercase\_ratio     500 non-null   float64

**7**   lowercase\_ratio     500 non-null   float64

**8**   digit\_ratio       500 non-null   float64

**9**   special\_ratio       500 non-null   float64

**10**   category\_encoded   500 non-null   int32



The variables *rank* and *offline\_crack\_sec* are only available in dataset. But desired application will take the password from the user and calculate the strength. Since there needs to be a calculation in new passwords given by user, this 2 variables needs to be addressed to a value.

After some calculations about correlation of the variables, it can be seen that:

* *Offline\_crack\_sec* and *unique\_chars* variables are quite similar on strength and have 0.57 correlation ratio.
* *Rank* variable has almost no impact to strength.

Given that, we can generate the value of *offline\_crack\_sec* by multiplying the value of *unique\_chars* with the correlation ratio of 0.57 and we can randomly generate *rank* variable.

Data is split into training and testing subsets using an 80-20 ratio to allow model evaluation on unseen data.

1. **Analysis Of The Dataset**

This section of the report includes some graphs made with matplotlib library. This graphs are created for clear and better understanding of the dataset and the distrubition of data.

**A graph of a password cracking time analysis

Description automatically generated**

Passwords in the “nerdy-pop” category is more resistant to cracking. Other categories (like “animal,” or “password-related”) have much shorter average cracking times, indicating simpler or more predictable passwords.

A bar graph with blue squares

Description automatically generated

Users tend to choose passwords based on names, likely because they are memorable. However, these may also be more predictable and easier to crack. Rarely used categories like “rebellious-rude” or “food” might contribute to more unique passwords, but their rarity indicates they are less preferred.

A graph of a diagram

Description automatically generated with medium confidence

While longer passwords tend to be stronger, length alone does not guarantee security.

1. **Model Training**

Three algorithms are trained on the same dataset to compare their performance:

* 1. **Decision Tree Classifier:**

• Decision trees split the data at each node based on a feature’s information gain (using the *entropy* criterion).

• The tree is pruned with a *max\_depth=5* to avoid overfitting, and *min\_samples\_split=5* ensures that a node is split only when it has at least 5 samples.

• Training involves recursively partitioning the training data and finding the optimal splits to predict password strength.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

* 1. **Naive Bayes:**

• Naive Bayes is a probabilistic classifier that assumes all features are conditionally independent.

• The GaussianNB implementation models feature distributions as Gaussian (normal).

• It uses Bayes’ Theorem to calculate the probability of a password belonging to each strength class and selects the class with the highest probability.

• As can see below, var\_smoothing parameter is set to 1e-05 for better accuracy.

metin, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. **Logistic Regression:**

• Logistic regression models the relationship between features and the target variable as a logistic (sigmoid) function.

• Before training, the features are standardized using StandardScaler to ensure equal importance for all features.

• It uses regularization (penalty='l2') to prevent overfitting and a high regularization strength (C=1000) to prioritize fitting the data closely.

|  |  |
| --- | --- |
|  |  |
|  |  |

1. **Comparison of Algorithms**
   1. **Decision Tree:**

Decision tree handles non-linear relationships well. It’s also easy to implement and interpret due to it’s tree structure. But it’s tend to overfit the dataset. It does not generalize as well as other algorithms.

* 1. **Naive Bayes:**

Works well in small datasets and fast to train and predict. But this algorithm assumes that all the features are independent which is generally not the case in real life examples.

* 1. **Logistic Regression:**

Performs well in linear distributed data. Though it is really simple to implement, this algorithm works surprisingly well for most of real case scenarios. But it overfits on high-dimensional data and it is not capable of solving non-linear problems.

1. **Project Structure**

Here is the github link for the project code:

<https://github.com/husamettyn/password_ml_search_engines_project/tree/main>

There are both source code and screenshots for graphs created with matplotlib. “analysis” file has the screenshots. “src” file has source code of the project.

**Train.py**: This file is responsible for loading the dataset from utils.py, making feature and target selection, splitting the data and training the models for 3 algorithms.

**Utils.py**: This file is responsible from importing the dataset and feature extraction.

**Feature\_analysis.py**: This file makes feature analysis, feature correalation, feature-target relationships and visulize all this analysis.

**Model\_analysis.py**: This script is used to analyze the performance of different models on the passwords dataset. It compares the performance of Decision Tree and Naive Bayes classifiers on the dataset.

It also plots the results of GridSearchCV for all parameters of the classifiers.

**Main.py**: This main file of the project loads pre-trained models from train.py. It uses gradio for User Interface. It predicts samples from a subset of dataset using the model, makes category analysis by retrieving the strongest and weakest passwords for each category, and make extraction and encode feature for passwords that user entered.

1. **Conclusion**

This project aimed to explore and compare three machine learning algorithms—Decision Tree, Naive Bayes, and Logistic Regression—in predicting password strength based on various password features.

* 1. Dataset Insights:

The dataset did not provided a rich variety of password characteristics led us to work on a smaller dataset. The dataset allowed us to extract meaningful features such as password length.

* 1. Algorithm Comparison:

• Decision Tree Classifier excelled in handling non-linear relationships but was prone to overfitting, especially in smaller datasets. Despite pruning techniques, its performance was outpaced by other models in terms of generalizability.

• Naive Bayes performed remarkably well on smaller datasets and demonstrated quick training and prediction times. However, its assumption of feature independence limited its ability to capture the interdependencies between password features.

• Logistic Regression proved to be a robust and simple model for this task. It achieved consistent performance across various datasets, albeit with limitations in capturing non-linear patterns.

The models’ accuracies, based on the testing data, were as follows:

**Logistic Regression (Accuracy: 0.75)**

Logistic Regression achieved the highest accuracy in this task due to its ability to model linear relationships between features and the target variable effectively. Password strength characteristics, such as length, digit ratio, and special character usage, often have predictable, proportional effects on the outcome, making Logistic Regression a good fit. Regularization during training helped avoid overfitting, so the model worked well on new data. This mix of simplicity, reliability, and fit for the dataset made it perform the best.

**Naive Bayes (Accuracy: 0.71)**

Naive Bayes did fairly well by using probabilities to classify passwords. It is simple and works quickly, which is great for small datasets like this one. However, it assumes that features are independent, which isn’t always true. Even with this issue, it performed decently because many features had clear, distinct patterns that suited its probabilistic approach.

**Decision Tree (Accuracy: 0.63)**

The Decision Tree had the lowest accuracy because it had trouble finding general patterns in the dataset. While it’s good at handling complex relationships, it often overfitted the training data, especially with small datasets. Even with techniques to limit overfitting, like pruning, it likely memorized the training data instead of learning general rules. The lack of feature scaling and its sensitivity to small changes in the data also made its predictions less reliable.

* 1. Practical Applications:

The trained models can be incorporated into real-world applications to provide users with instant feedback on password strength. Such tools can guide users toward creating stronger passwords, thereby enhancing digital security.

* 1. Final Thoughts

This project underscores the potential of machine learning in enhancing password security by identifying patterns and vulnerabilities in password creation. By comparing different algorithms, we gained valuable insights into their strengths, weaknesses, and suitability for this task. The findings not only contribute to the growing field of password security research but also pave the way for practical implementations that can help users adopt safer online practices.

1. **References**

1. Kaggle Dataset: “Password Strength and Vulnerability” dataset, available at <https://www.kaggle.com/datasets/utkarshx27/passwords>.

2. Scikit-learn Documentation: <https://scikit-learn.org/stable/>.

3. Gradio Library: <https://gradio.app/>.

4. Understanding Decision Trees: https://medium.com/@MrBam44/decision-trees-91f61a42c724

5. Naive Bayes Classifier: <https://machinelearningmastery.com/naive-bayes-for-machine-learning/>.

6. Logistic Regression: <https://medium.com/data-science-group-iitr/logistic-regression-simplified-9b4efe801389>

7. Python Matplotlib Library: <https://matplotlib.org/stable/index.html>.