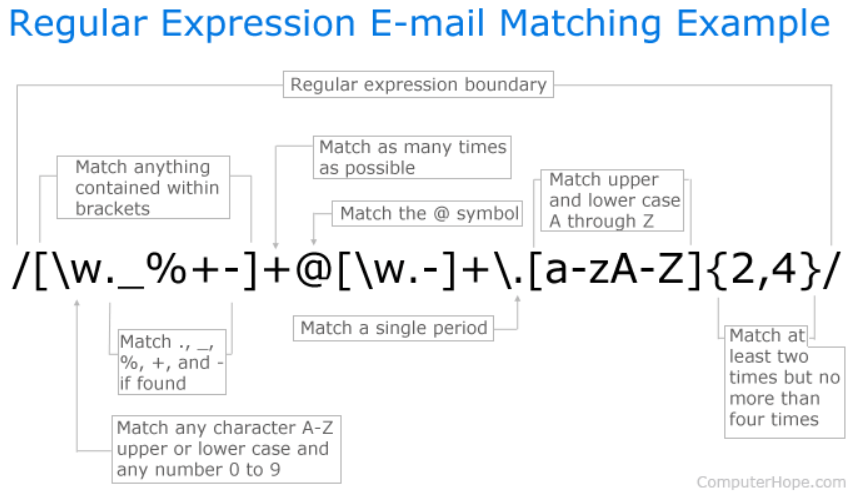
Password Strength Classification

## Introduction

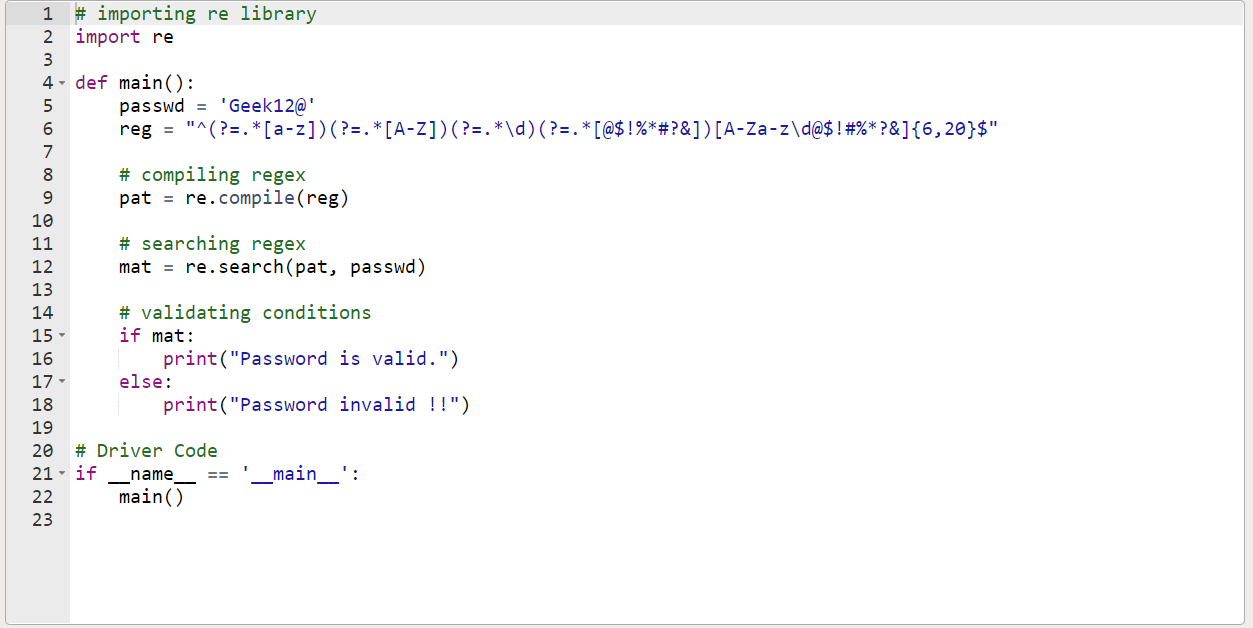
As passwords become more vulnerable in the modern world, we thought it would be ideal to focus on a machine learning model that classifies passwords based on their strength. Currently, several password meters give a strength output based on simple conditions. For example, whether a password contains a certain number of characters, whether a password contains special characters, and whether a password contains capital or lowercase lettering. Our goal was to improve on simple password strength meters and learn the fundamentals of machine learning in the process.

## The Problem and Objective

Surely there are more advanced forms of password strength validation. However, the most common form, regex, uses a string of characters to parse through text and find a specific condition. Regex validation is conditional and is too simple for modern passwords. Having a password that contains special characters and a certain number of letters and numbers is important. However, how will a user know how their password stands when compared to other users’ passwords? We hope to improve on these simple password strength meters with machine learning models. As seen below, the most common password strength meters use regex, or regular expressions, to estimate the strength of a given password.



(Hope)



(dkp1903)



(user2412839user2412839)

## Related Work

We tried to focus on interesting topics that gave us the foundation of machine learning as well as the opportunity to expand on a topic. We thought the article, “Data Science for Cybersecurity — Password Strength Meter” by Dwi Gustin Nurdialit, was a simple and fascinating introduction to classification and text processing with a machine learning focus (Nurdialit).

In the article, the author starts by explaining the need for a more sophisticated password meter and how current password meters are handled by organizations. The author continues by laying out each category of her project. The categories include data preparation, building a classification model, evaluation, and a conclusion of their work(Nurdialit).

We used this article for the foundation of our work in password classification. The author layed out their work in a way where it was easy to dive deep into complex ideas. The combination of structure, detailed explanations, and content made this the perfect introduction to a machine learning concept (Nurdialit).

##### Dwi Gustin Nurdialit’s Implementation

* Data Preparation
  + Visualize Data
  + Array Conversion
  + Vectorization of Arrays
* Building Classification Machine Learning Model
  + Splitting Data
  + Training the Model
  + Evaluating the Model
  + Tuning the Model
* Evaluation
  + Classification Report Evaluation
  + Testing a Prediction
* Conclusion
  + Summary and Future Improvements

## The Implementation

##### Initial Idea

The first step we decided to take was to learn about the different methods used for testing a password’s strength. As discovered, regex is one of the primary methods for password classification. Improving on the strength classification process, we wanted to try and classify passwords with a machine learning model. We started by using a dataset with over 600,000 passwords. Each password was classified into a strength category 0,1,and 2. The strengths were calculated using a tool from Georgia Tech University called PARS. PARS uses a combination of the most popular password meters to determine strength(“ECE CAP PARS – An ECE Research Project”).

##### Data Preprocessing

Next, because passwords are text, we needed to know the fundamentals of text classification. Text classification usually involves the context of the data. In our case, passwords are supposed to be irrelevant contextually. Usually with recommendation systems, chat bots, and all other text based machine learning projects the goal is to remove anything that would counteract the text contextually. Removing things like numbers and uppercase characters increases the accuracy of a text based model’s results. This is an important note in our case because we didn’t want to remove the special characters, upper case letters, numbers, or any part of the password. We wanted the password in its original format. Similar to Dwi’s implementation the only text processing we did was tokenization. Tokenization is the process of cutting each password into characters. Cutting the passwords into characters allows the model to evaluate each character individually as its original format. Furthermore, uppercase and lowercase letters of the same letter are seen as different tokens because we didn’t change the data in any way.

Computers have an easier time evaluating information in arrays. An array is similar to a list, but a list cannot be manipulated arithmetically (Bolaji). After manipulating the two columns (passwords and password strength) into arrays, we wanted to turn each array into a vector. A vector, in our case, is used to quantify each token. There are a few ways to vectorize text data without context. We decided to use the TF-IDF Vectorizer (Term Frequency-Inverse Document Frequency) and Count Vectorizer (One-Hot Encoding) vectorizers (Brownlee). The TF-IDF vectorizer gives the frequency of the token compared to the entire document and the Count vectorizer simply counts each token(Heidenreich).

Finally, in the last step of data preprocessing, we had to balance out the password dataset. If we implemented the model in its current state, the passwords with a strength of 1 would skew the model and lead to inaccurate predictions. There are a few methods to accomplish this task. We could synthetically add data with a strength of 0 and 2 using SMOTE (Synthetic Minority Oversampling Technique). We could methodically decrease the amount of data in the 1 category with a method called ENN (Edited Nearest Neighbor). Finally, we could use the class weights of each category to keep the data integrity as well as balance the data. Ultimately, preserving data integrity with class weights ended up being the most practical method.

##### Model Picking, Training, and Testing

After processing the data and implementing all of the basic text processing prerequisites, we split the data into training and test sets. Splitting the data for training and testing is essential. You cannot test the data on the same data the model was trained on. The end results will be inaccurate if so. We also picked several models to classify the data. Browsing the Scikit learn library, we found several models that included the “balance” parameter that takes into account the weight of each class to balance out the skewed data. We used all of the most popular text classification models that included the “balance” parameter. The models used for training, testing, and fitting were as follows: Logistic Regression, Decision Tree, Random Forest, and Linear SVC. We chose to use Linear SVC instead of SVC for its ease-of-use with larger datasets.

After picking the models, we decided to cross validate each to find the highest performers. Cross-validation uses all parts of the data by splitting the data into five parts. One of the five parts is the testing data. The tested segment is then given an accuracy score. Eventually, all parts of the data will be tested with this method. Taking the mean of the five scores will give an accurate cross-validation of the entire dataset(“Classification: Accuracy | Machine Learning | Google Developers”).

We tested the models across both vector types to analyze the difference in output. The Random Forest model was the highest performer on the TF-IDF vectorized data. The Logistic Regression model was the highest performer on the Count vectorized data.

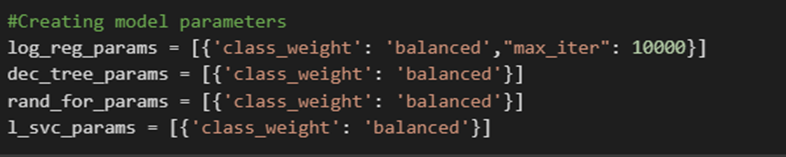
##### Live Prediction

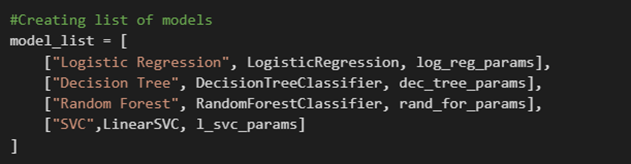
The final part of the project was to implement a small widget that can classify a password once submitted. We wrote a function that takes a password input and predicts the strength as a number (0,1, and 2). The strength is then output as a “meter” that grows or shrinks based on the password input. The password’s submit button disables upon click to prevent any password attacks over a short period of time. It is disable for three seconds and then enabled to continue password submissions.

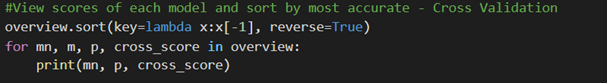
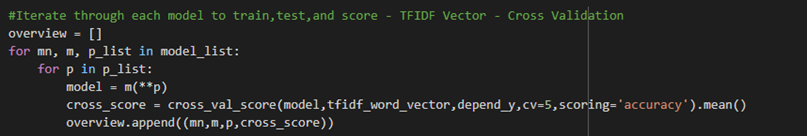
The Results

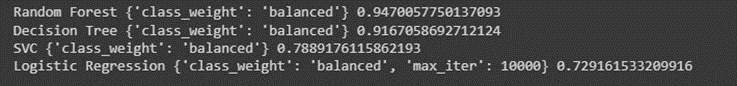
The rigorous validation and testing of both TF-IDF vector models and the Count vector models gives a well-rounded assessment of the accuracy of all models and all vectors tested. As stated above, the highest performing model for the TF-IDF vector was the Random Forest model. The highest performing model for the Count Vector was the Logistic Regression model.

##### TF-IDF Vector - Cross Validation Results

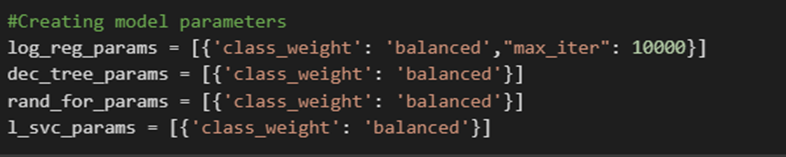


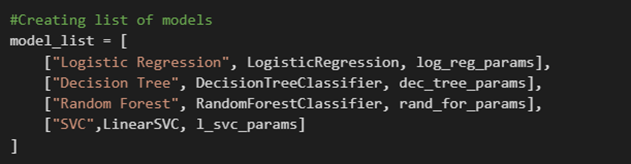


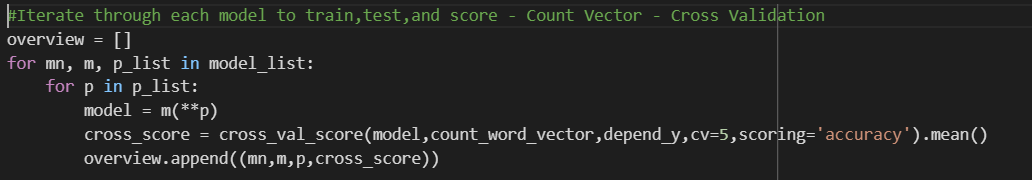


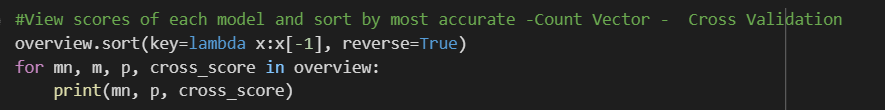


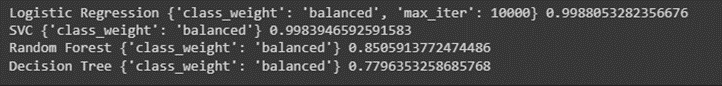
##### Count Vector - Cross Validation Results









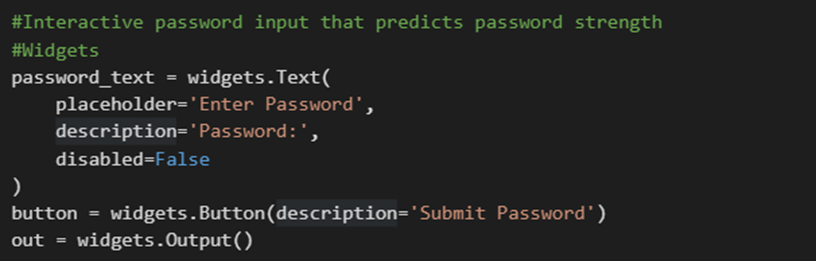


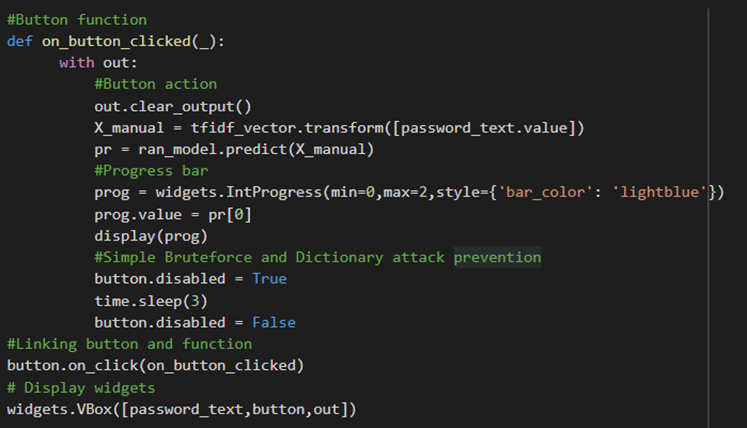
## Discussion

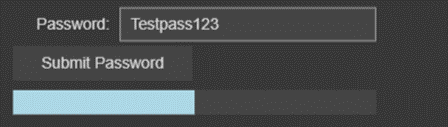
## We wanted to improve on Dwi’s password classification project by implementing new vectors, models, validation processes, and widget implementation to show a real world example. The goal was to experiment with the process of machine learning and gain fundamental knowledge along the way. Let it be known, we attempted to implement other vectors such as the Hash vector. However, the process of running a hash for each token took a large amount of resources. The large amount of resources used caused us to abandon the Hash vector.

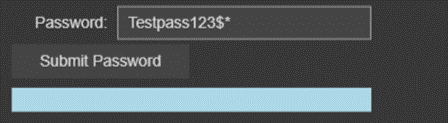
## Dwi’s project consisted of only one type of vector, the TF-IDF vector. We implemented both the TF-IDF vector and the Count vector to possibly return different results. Along with the new token vector, we used a different model, the Linear SVC model. The Linear SVC model was a reliable support vector model for bigger datasets. We also performed cross-validation on all models instead of just one model. Dwi’s project focused on the validation of the highest performing model. Last but not least, the live classification of a password. We wanted to show a unique example of password classification in real time with widgets. The implementation of a widget with a submit button turned out to be a fantastic way to prevent password attacks. Adding a delay to the button on-click allowed us to expand deeper into security aspects of the project.

##### Live Password Classification









## Summary, Limitations, and Conclusion

In summary, I believe we achieved the goal of expanding on the code and methods to see how different methods output different results. Surprisingly, we achieved a higher accuracy with the Count vector rather than the TF-IDF vector. Because the dataset was so large, it prevented us from testing different vectors and hypertuning the parameters to possibly improve the model’s accuracy. Overall, the model performed quite well regardless of the vector or model we used.

## Github Source

<https://github.com/RayThibodeaux/password-strength>

## References

Bolaji. “List vs Array — Data Types. Some Days Back, I Was Working on A… | by Bolaji | Backticks & Tildes | Medium.” Medium, Backticks & Tildes, 6 May 2018, <https://medium.com/backticks-tildes/list-vs-array-python-data-type-40ac4f294551>.

Brownlee, Jason. “A Gentle Introduction to Vectors for Machine Learning - MachineLearningMastery.Com.” MachineLearningMastery.Com, https://www.facebook.com/MachineLearningMastery/, 1 Feb. 2018, <https://machinelearningmastery.com/gentle-introduction-vectors-machine-learning/#:~:text=A%20vector%20is%20a%20tuple,the%20numbers%20in%20the%20list>.

“Classification: Accuracy | Machine Learning | Google Developers.” Google Developers, https://developers.google.com/machine-learning/crash-course/classification/accuracy#:~:text=Accuracy%20%3D%20Number%20of%20correct%20predictions,%2C%20and%20FN%20%3D%20False%20Negatives. Accessed 28 Nov. 2022.

dkp1903. “Password Validation in Python.” Geeksforgeeks, GeeksforGeeks, 29 Dec. 2020, <https://www.geeksforgeeks.org/password-validation-in-python/>.

“ECE CAP PARS – An ECE Research Project.” ECE CAP PARS – An ECE Research Project, https://www.pars.gatech.edu/. Accessed 28 Nov. 2022.

Heidenreich, Hunter. “Natural Language Processing: Count Vectorization with Scikit-Learn.” Towards Data Science, medium, 23 Aug. 2018, <https://towardsdatascience.com/natural-language-processing-count-vectorization-with-scikit-learn-e7804269bb5e>.

Hope, Computer. “What Is a Regex (Regular Expression)?” Computer Hope’s Free Computer Help, Computer Hope, https://www.computerhope.com/jargon/r/regex.htm. Accessed 28 Nov. 2022.

Nurdialit, Dwi Gustin. “Data Science for Cybersecurity — Password Strength Meter | by Dwi Gustin Nurdialit | Analytics Vidhya | Medium.” Medium, Analytics Vidhya, 7 May 2021, <https://medium.com/analytics-vidhya/data-science-for-cybersecurity-password-strength-meter-b933b96bff32>.

user2412839user2412839 15111. “Python - Checking the Strength of a Password (How to Check Conditions) - Stack Overflow.” Stack Overflow, https://stackoverflow.com/questions/16709638/checking-the-strength-of-a-password-how-to-check-conditions. Accessed 28 Nov. 2022.