Final Report

Cybersecurity Analytics- Predicting the CVSS (Common Vulnerability Scoring System) for future Vulnerability situations

Name-Manan Piyushkumar Ayde **(U31716415)**

- Project Description: Given a Cyber network vulnerability report with 23,918 entries, where each record represents a distinct occurrence of a vulnerability discovered on a device on a computer network. The response value, which goes from 1 to 10 and identifies the criticality level of an instance of a vulnerability detected on a device, is the Common Vulnerability Scoring System (CVSS). Here 1 denotes the least critical and 10 denotes the most critical vulnerability. Our objective is to follow a data-driven analytics method to predict the CVSS values in the testing data set to assess the relative criticality of future or unknown vulnerability situations.

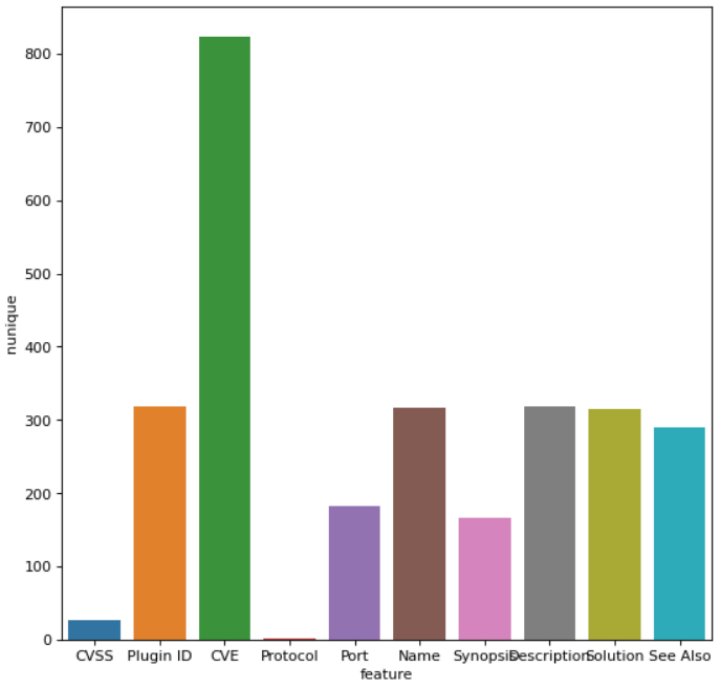
- Feature Extraction: There are 10 variables and 23918 entries in the data from which CVSS is the target variable. Initially, I combined the Name, Description, Synopsis, and Solution into 1 column and named that column “Combined” for training and testing data. Then after, I applied TFIDF vectorization on the newly combined created feature. I converted all the values in it into lowercase and applied the stop words= “English” parameter to ignore the commonly repeated characters in English.

- Data Visualization:

    1. Bar Graph: Below is the bar graph for each feature in the whole dataset. We have our target variable as CVSS (Common Vulnerability Scoring System). And rest of them are the dependent variables: Plugin ID, Port, Synopsis, Name, Protocol, Description, Solution and See Also. As shown in the figure below, we are computing the number of unique values in each of the features and presenting them in the form of a bar graph. We can also see

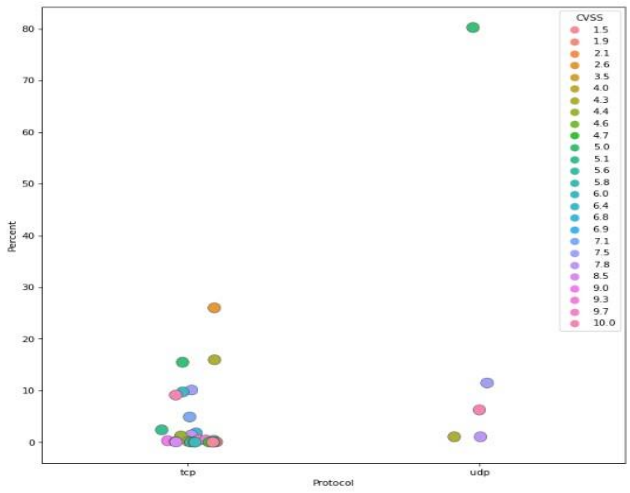
that we have the highest unique values in the 'CVE' column with 825 values. On the contrary, 'Protocol' has the lowest unique values with just 2 values which are TCP and UDP. Also, we can see there are so many unique values that are repeating in the CVE columns like we can see for 1st one in the figure which is repeating 1687 times which is too much as any value which is repeating for more than 500 times might not be good for dataset and to predict the target variable and those which are under 500 might be good but we cannot take a particular value from the "CVE" column and just replace it with missing values which can result in biased data and less accurate.

```
CVE-2015-2808    1687
CVE-2013-2566    1686
CVE-2016-2183    1525
CVE-2014-3566    1053
CVE-2015-4000     965
                 ...
CVE-2013-1896       1
CVE-2009-3095       1
CVE-2009-3094       1
CVE-2009-2699       1
CVE-2018-1788       1
Name: CVE, Length: 824, dtype: int64
```



2. Scatter Plot: As we saw in the above visualization that 'Protocol' has the lowest unique values which are TCP and UDP. This graph depicts different vulnerabilities in either both

of the given values in percentage. For instance, we can say that there are more than 75% of UDP values have a vulnerability of 4.7.



- Data Pre-processing and data splitting techniques:

  As mentioned in feature extraction, I combined the Name, Description, Synopsis, and Solution into one data frame and named that column “Combined” by keeping the space between each of them. Also, Performed the same code on testing data. Now this means we have 11 columns in the dataset including the new one. Now, as a part of data pre-processing, I decided to create a new data frame in which I removed all the remaining columns by just keeping the combined column. Now to convert text into numerical data, I used TFIDF vectorization using tokens. Through this, I was able to convert the “Combined” column into a meaningful form that a machine can understand.

  Talking about data-splitting techniques, it was quite simple. I split my dataset into an 80-20% ratio by keeping a random state of 1234.

- Description of the model(s) used for analysis and justify the following:

  I. Model Selection:

     A. Decision Tree Regressor

     B. Random Forest Regressor

     C. K Nearest Neighbours Regressor

     D. Adaboost Regressor

  II. Resampling Method used during Training:

     Resampling Method: After generating the data in the form of visualization, I was able to know that we were given unbalanced data which means there are way fewer data available that have low vulnerability scores that cause poor model training so thought of applying random oversample as some of the target variables has 0 corresponding data available. For this reason, I may not also use SMOTE. But after applying this balancing data technique, I was not getting the desired results and the KNN model was performing the best and had a drastic difference in RMSE scores between random forest and KNN which is, in my opinion, skeptical and got the high error on actual test data.

  III. Performance metric used to gauge the prediction accuracy of the model:

     Initially, I believed that selecting the models might affect the prediction accuracy of the model. But as the project proceeded, I realized that it is necessary to understand the data and how to apply data pre-processing on the dataset. Moreover, using Tfidf vectorization and dropping the columns which were not contributing to getting good accuracy for the model or the ones

which were of no use made them the performance metric which helped in getting better accuracy for the model.

IV. Insights about why the selected model produces such results:

A. Decision Tree Regressor: When using decision tree regression, properties of an object are observed, and a model is trained using a tree's structure to forecast data in the future and create useful continuous output. After applying this model, I got an RMSE value of 0.07. The reason is that Decision trees intend to computationally do well in large data sets and also do not require as much data preparation as compared to other models.

B. Random Forest Regressor: The most crucial aspect of this model is that, like previous models, it does not overfit the data. Furthermore, the features chosen for each tree in this model are chosen at random, preventing the trees from becoming too deep and allowing them to focus on the chosen feature. This results in various trees that are created with various sub-features from the features. Because it takes into account all features at once and builds deeper trees that may overfit the data, the decision tree differs significantly from the random forest in this regard. For instance, in my opinion, the same thing has happened with my decision tree model as well, and there is a significant likelihood that the model has been overfitted because it is generating fewer errors than Random Forest, which is not feasible in the majority of circumstances. Hence, I got the RMSE

value of 0.09 which is quite less than the Decision Trees model but this might be a good accuracy score for the random forest.

C. K-Nearest Neighbours Regressor: KNN is a great model and is best when has to adjust with the neighbor parameter. So, when new values are imputed in the dataset, KNN can successfully and comfortably adjust based on the n_neighbors' parameter the dataset has. But it is not a good option when you go for missing data as KNN cannot handle them unless the process of imputation is applied. We can see from the error we got and it is expected as we have around 10,000 missing data from the whole dataset.

D. AdaBoost Regressor: Adaboost is easy to use and it can increase the accuracy of weak machine-learning models. But this model requires quality data for training as it does not perform well for the noisy data and outliers which got me the result for the Adaboost around 4.62 RMSE.