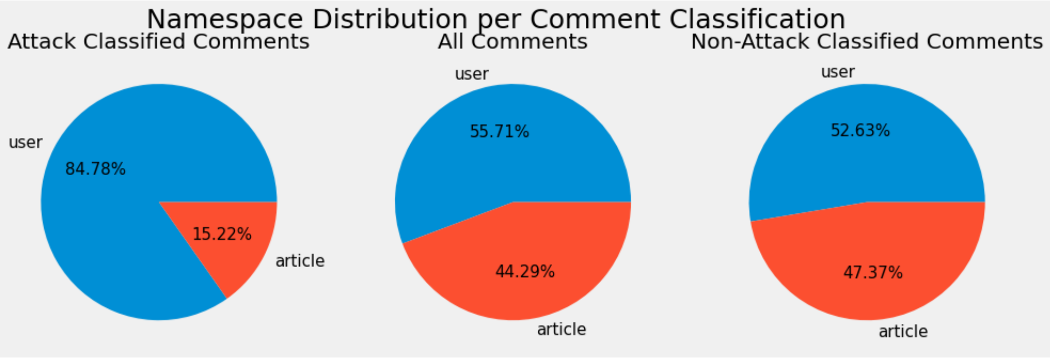
Identifying Personal Attacks in Wikipedia Comments

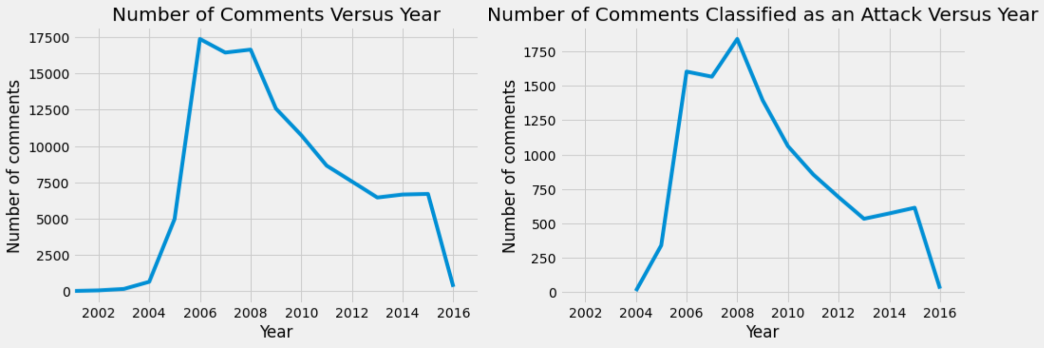
Danielle Mallare-Dani

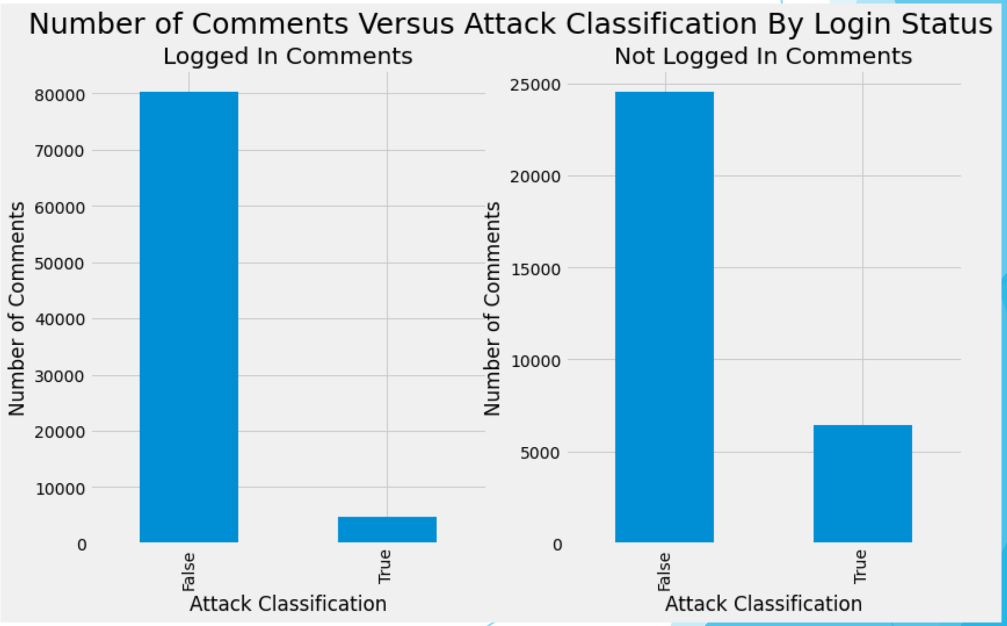
December 9th, 2021

Data Visualization

- To get a feel for the data and to understand how the features in the dataset impacted a comment's classification, I did an in-depth exploration of each feature in the dataset.
- The pie charts show that most comments that get classified as an attack are on a user.
- The bar charts illustrate the differences in comment distributions among users who are logged in or not.
- The year charts show that the distribution of all comments and attack labeled comments are very similar.

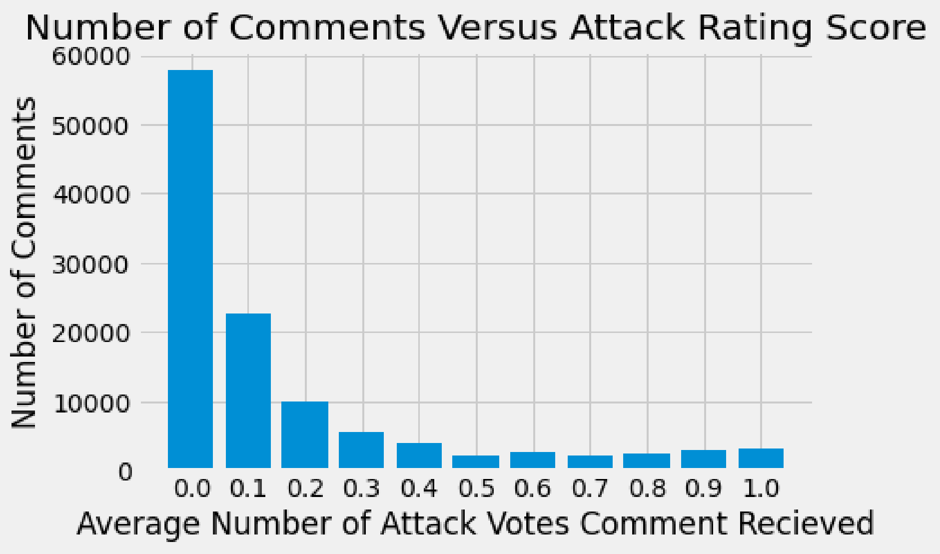






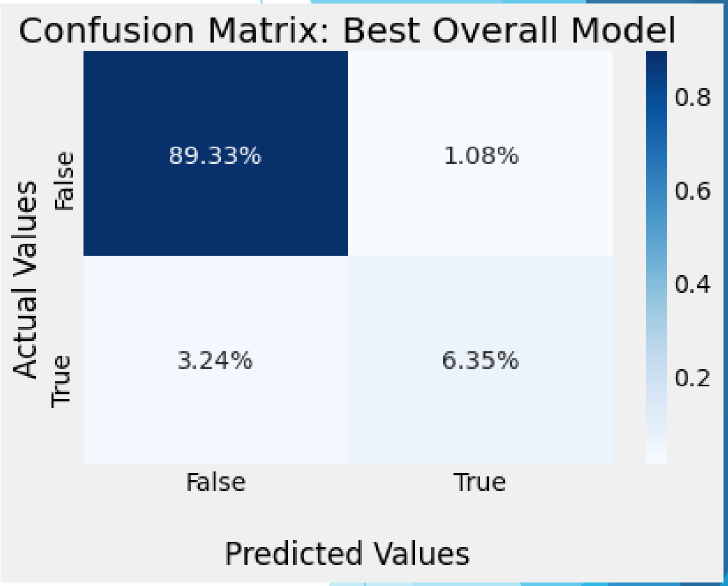
Comment Cleaning & Feature Extraction

- The attack information from different annotators was examined (see the bar chart).
- The final decision was to label a comment as an attack if the comment had an average number of ratings greater than 0.6. That is, at least 7 of the 10 annotators rated it as a personal attack.
- To determine how to clean the comment text, I did an initial simple clean of the text, generated a bag-of-words representation, and analyzed the representation. This led to a final clean that included:
    - Removal of all numbers
    - Removal of all punctuation
    - Removal of some of the HTML style elements. For example, “cellpadding”
- The following features were considered for use:
    - Login status, was used to train the models
    - Namespace, was used to train the models
    - Various combinations of character and word n-gram tf-idf vectorizers were experimented with
    - Year was also considered, but not utilized to train the models



Modeling the Data

- Evaluating model performance:
  - Average macro precision, recall, and f1-score
    - Precision: Proportion of comments classified as attacks that were correct
    - Recall: Proportion of offensive comments that were identified correctly
  - Confusion matrix converted to percentages and displayed as a heat map
    - Understand how well the model does with labeling both types of comments
- K-Fold cross-validation was used with k = 4.
  - Generally led to an approximately one percent increase in recall
  - Generally led to a half percent increase in precision
- Best results obtained from each model (without hyperparameter tuning)



Model	Precision	Recall	F1-Score
Linear SVC	0.9130	0.8293	0.8650
Logistic Regression	0.9157	0.7937	0.8414
Random Forest	0.9223	0.6465	0.7015

Hyperparameter Tuning

- The following hyperparameter tuning was conducted for each model. The best parameters are highlighted in bold.
- Tuning yielded between a 0.5-0.75% increase in average macro f1-score.

Model	Parameter 1	Parameter 2	Parameter 3
Linear SVC	C [0.1, **1**, 10,100]	Class Weight [‘**balanced**’, None]	N/A
Logistic Regression	C [0.1, **1**, 10, 100]	Class Weight [‘**balanced**’, None]	N/A
Random Forest	Number of Estimators [**10**, 50, 100]	Max Depth [2, 5, 10, **None**]	Criterion [‘**entropy**’, ‘gini’]

- I also examined the following parameters associated with the word n-gram tf-idf vectorizer:
  - max_df
  - max_features
  - Usage of stop words

Conclusion

- Best overall model: Linear SVC
  - Features: Login status, namespace, and tf-idf unigram and bigram word and character vectorizers
  - Parameters set:

```
model__C: 1
model__class_weight: 'balanced'
preprocessor__comments__word__max_df: 0.8
preprocessor__comments__word__max_features: None
preprocessor__comments__word__stop_words: None
```

- Performance of the best overall model versus strawman:

Model	Precision	Recall	F1-Score
Best Linear SVC	0.88	0.87	0.87
Strawman	0.80	0.79	0.80

- Optimizations to my code included:
  - Creation of custom functions to allow me to systematically train, test, and conduct hyperparameter tuning and generate results in a consistent format.
  - Use of *RandomizedSearchCV* instead of exhaustive *GridSearchCV* with large search spaces