The Kindness Project: Toxic Comment Classifier

Project Owner: Ting-Hsuan (Doris) Lin

Mentor: Dipanjan Sarker

Problems & Business Applications

- Problem:
  - Online profanity on communication platform can cause serious negative impacts to others and even make users leave the platform, leading to serious user drop.
  - However, the current solution to mitigate online profanity is through manual monitoring, which faces the challenges of high costs and low efficiency.

- Business Use Cases:
  - Companies with messaging features and online communication platform can reduce the cost of manual monitoring with machine learning solutions with below benefits:
    - Detect types of toxicity comment data to help online discussion platform improve user experience.
    - Automate manual labeling process by building multi-headed classifiers
    - Provide clear visualization for non-tech audience of content insights of different types of toxic comments

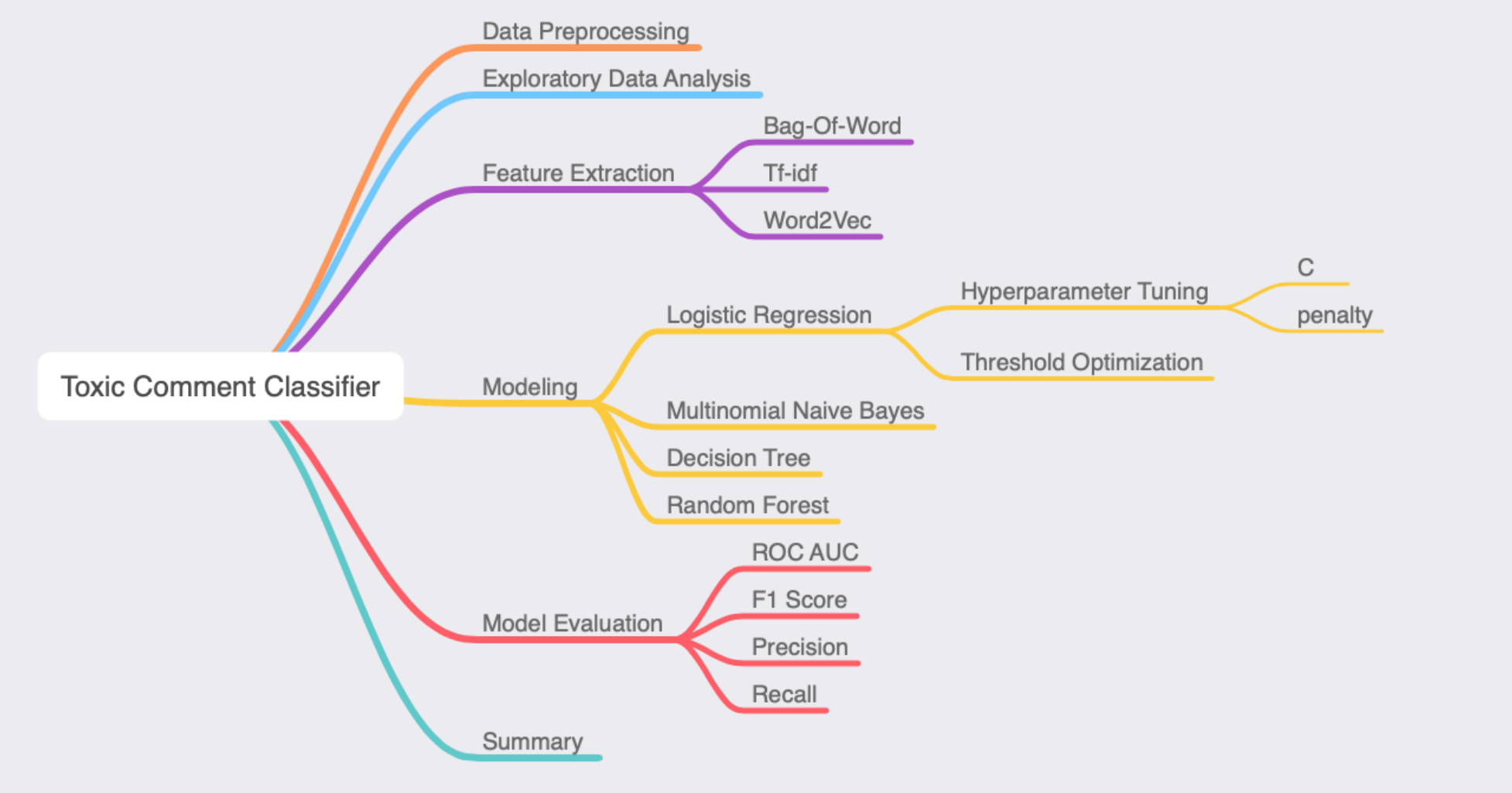
Objectives

- Toxic comment classifier is a natural language processing application that allows online platforms to detect the toxicity of a message based on its textual context, with the long term goal to improve the quality of online conversation. The project answers below questions:
  - What's the probability of each type of toxicity (toxic, severe toxic, obscene, threat, insult, identity hate) for each comment?
  - Which words/phrases frequently seen in toxic messages are key to the future prediction?
  - When do models make mistakes? If taking the cost of misclassification into account, what are tolerable mistakes and how should models adapt to it?

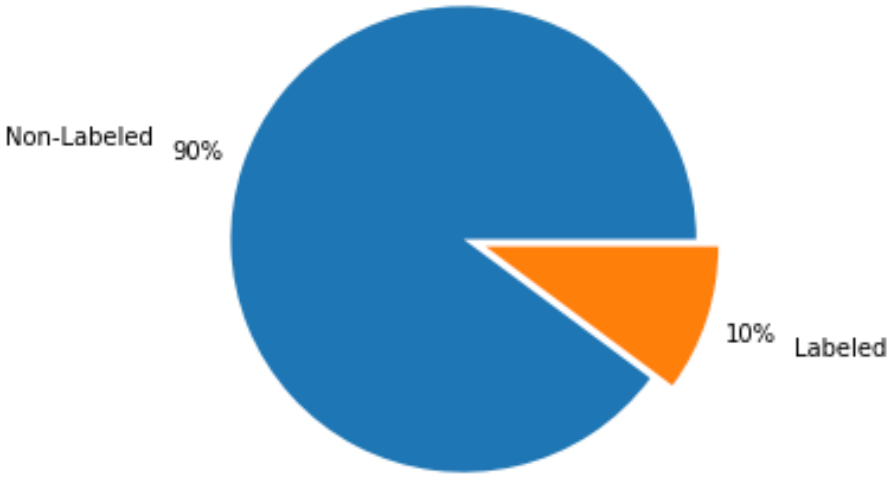
Data Source

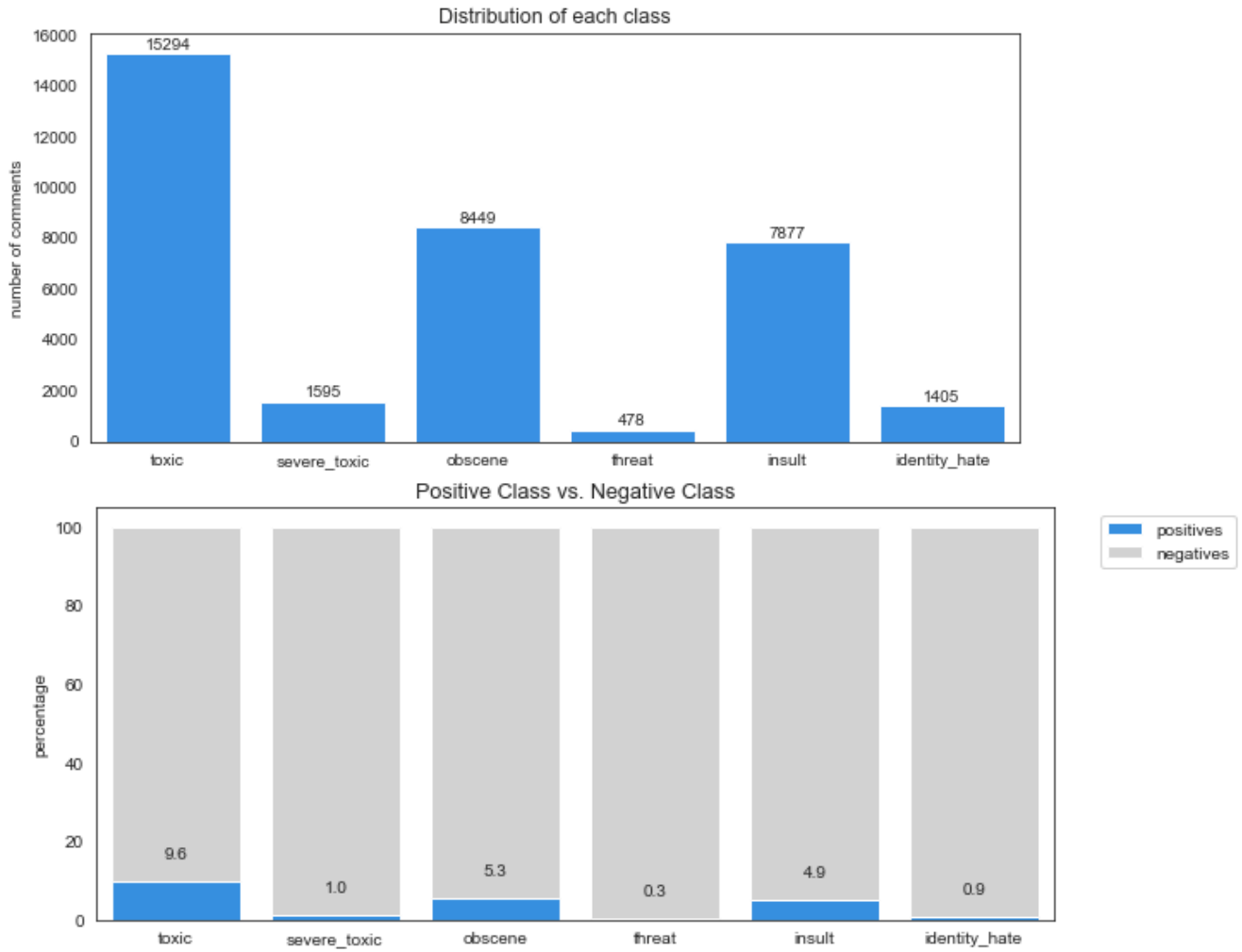
- The data covers 160K comments from English Wikipedia talk page edits, dating for 2004 to 2015, provided by Kaggle.
- Comments are labeled by 5K workers with one or more toxic types, including toxic, server toxic, obscene, threat, insult and identity hate.

Project Structure



Imbalance in toxic and non-toxic volume





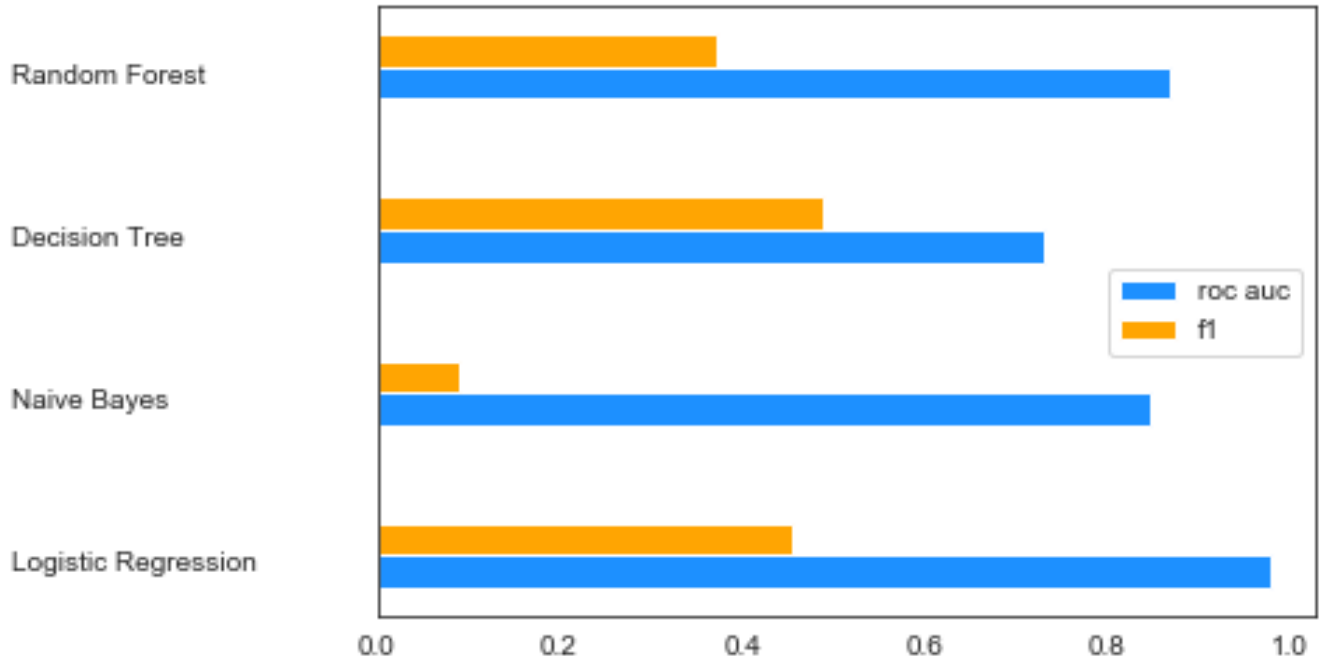
Frequent words reflects common sense

ROC AUC & F1 score serve as key metrics

- The imbalance issue of this dataset and cost of misclassification are considered when choosing suitable metrics for model evaluation.
- ROC AUC and F1 score are the main metrics since both of them are insensitive to the issue of data imbalance and can provide a general summary of the model overall performance.
- Precision and Recall on the other hand provide additional information as secondary metrics when there is a tie. Since the outcome of false alarm is more expensive than miss detection, the false negative error is more tolerable than false positive error. Models with higher precision at the expense of slightly low recall are preferred.

Logistic Regression outperforms others

- Applying Tf-idf to transform content into numerical values to four models.



Model Tuning

- Logistic regression might be prone to be overfitting when there are more independent features than volume of data. In order to avoid that, I focus more on hyper parameters “C” and “penalty” when tuning the model.
- It turns out the model with default hyper parameter setting is the best-performed one. Compared to the benchmark model, which guess the majority class all the time, the model improves by 18% ROC AUC and by 47% F1 score.

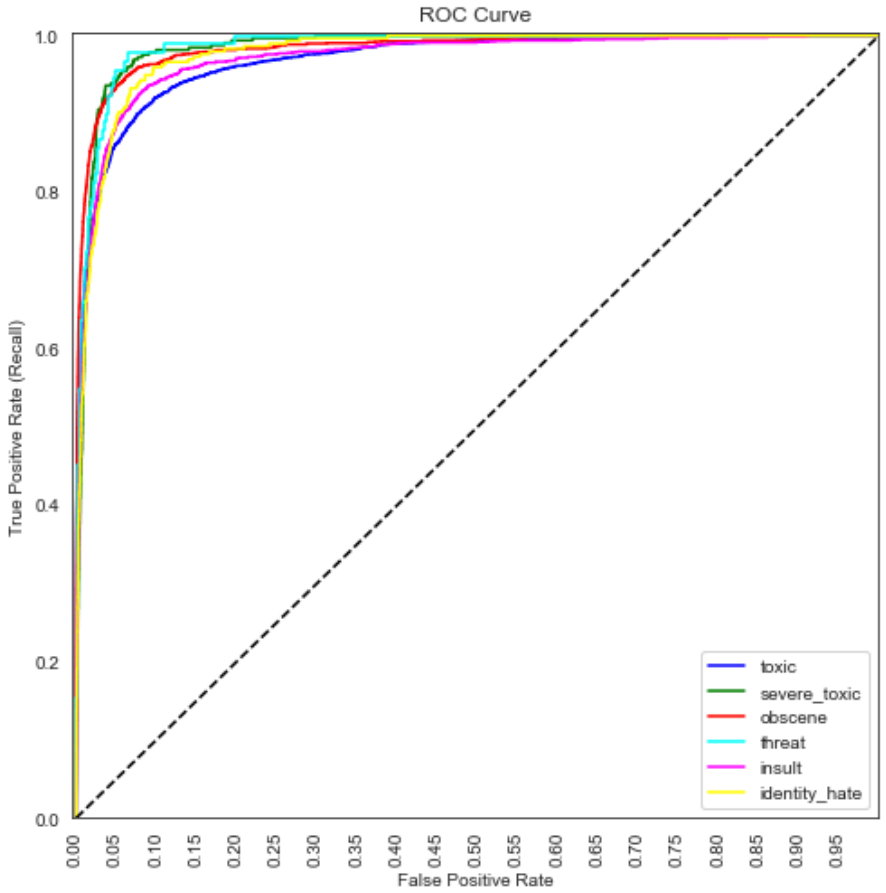
Benchmark model

```
average ROC AUC: 0.5
average f1 score: 0.0
average precison score: 0.0
average recall score: 0.0
```

Optimized model

```
average ROC AUC: 0.6835835414562642
average f1 score: 0.476909265770046 94
average precison score: 0.738031923260121
average recall score: 0.37001984298529633
```

Threshold optimization to reduce false alarms



- The default threshold at 0.5 has already provided good precision score. We can be very confident to trust the result when model predicts a sample Positive. The high precision reduces the risk when we take a suspected comment down from the platform, while it is indeed neutral.
- But 0.4 gives a good balance between precision and recall. While we don't want to piss off the user due to false alarm, we don't want to miss capturing toxic messages and cause negative impact on user behaviors. Thus, 0.4 is the recommended threshold value.

Next Step

- Random forest being the second best-performed model in the project. However, it requires expensive computational power for hyper parameter tuning, I have not gotten a chance to optimize the model. “XGBoost model”, also a tree model but is faster to run, might be an alternative model to use.