Fight Online Abuse

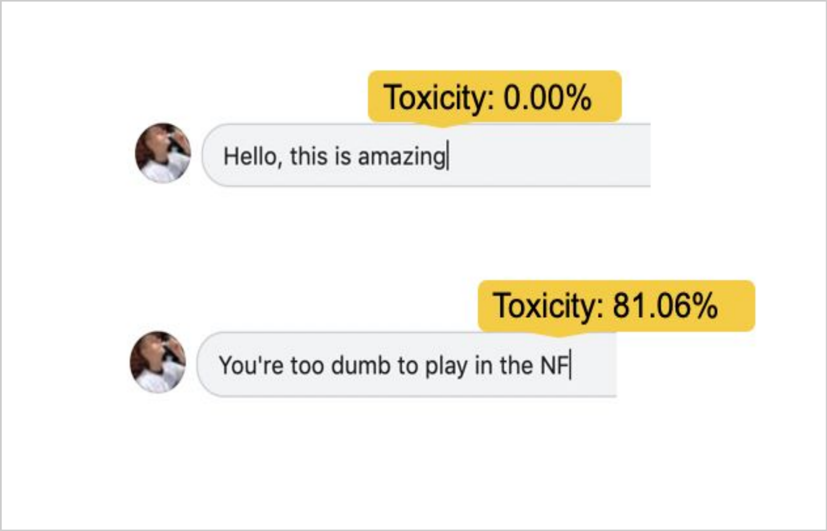
Cristobal Zamorano Astudillo, Nina Singiri, Nick Palmer, Preston Weber, Samantha Tang

DISCLAIMER

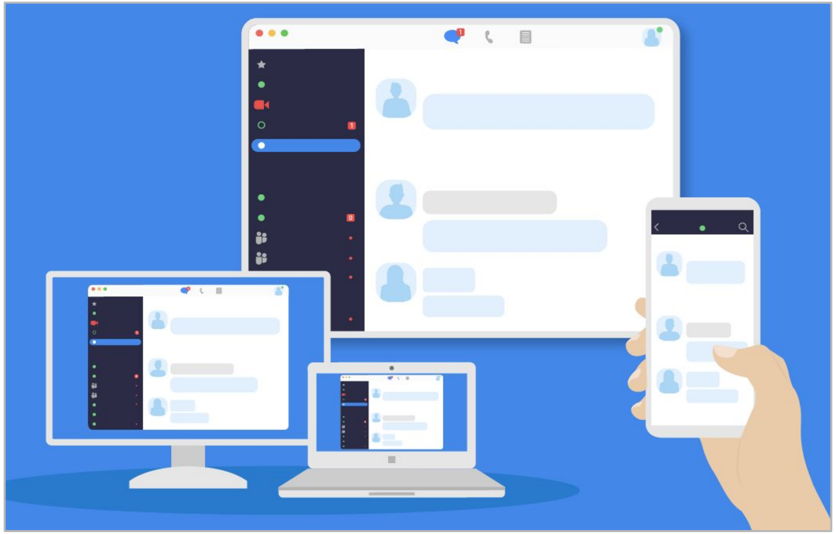
The following slides may contain or reference comments and/or words that may be offensive. Any material shown is meant only as a means to clarify our process, approach and machine learning methods.

Introduction: Project Context

Research Question: Are we able to identify toxic comments?



Motivation: Safer virtual community in chat rooms for online education, etc.



The Data

- A collection of Wikipedia comments from Kaggle
  - Train.csv - 159,571 rows
  - Test.csv - 63,978 rows
- The number of features used differs depending on how the models were created

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate	final_toxicity
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0	0

Approach #1 : Binary Classification

Ensemble-Based

Simple Combination: Choice of model and hyperparameters can differ in any way

Person 1's Model	1	0	1	1	1	0	1	1
Person 2's Model	1	1	1	1	1	0	1	0
Person 3's Model	0	0	0	1	1	1	1	0
Final Prediction	1	0	1	1	1	0	1	1

Methods and Approaches - Samantha

Preprocessing and Data Exploration:

- Cleaned up dataset (balanced)
  - Removed punctuation (except for quotes)
  - Removed special characters (e.g. #, @, %), numbers, and extra spaces
  - Lowercase all words
- Removed stop words using NLTK and tokenized words:
  - Replaced ambiguous / unknown words with placeholder value before tokenizing

```
Original Text:   hi quaranj curious think material wikipedia leading death threats hatecri
Tokenized Text:  [32, 1, 2411, 23, 424, 3, 2950, 506, 786, 1, 648, 1597, 424, 2222, 1, 156

Original length:  55
Tokenized length:  53
```

STOP words removed, so tokenized is smaller in length

cont.

Trimming and padding words

```
txt = ["you suck"]
seq = tokenizer.texts_to_sequences(txt)
padded = pad_sequences(seq, maxlen=max_length)
pred = model.predict(padded)
labels = [0, 1]
print(pred, labels[np.argmax(pred)])
```

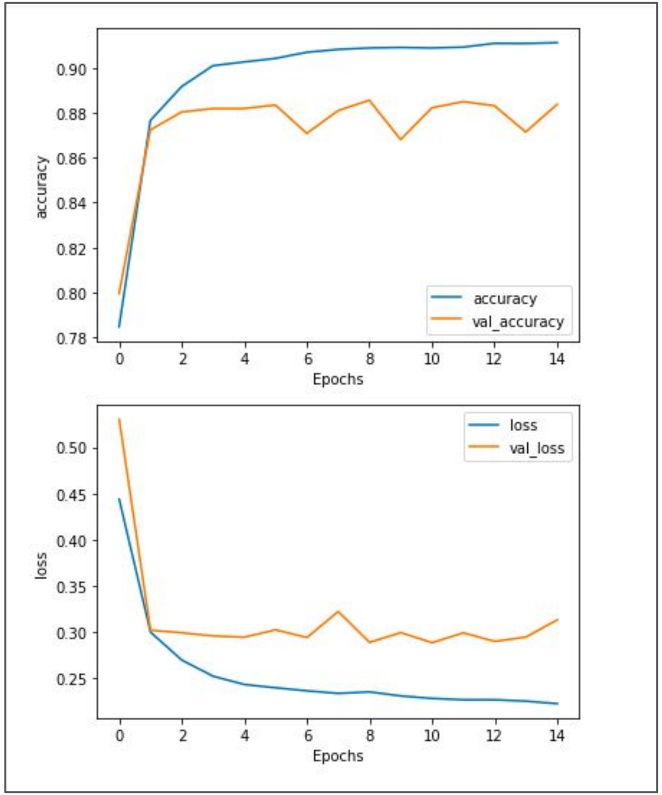
```
[[0.01067209 1.        ]] 1
```

```
# Note: too much padding can lead to worse results cuz
# bad content/words can be obscured by filler padding
padded
```

```
array([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0, 27, 20]], dtype=int32)
```

```
bidirectional_2 (Bidirection (None, 40)                6560
_________________________________________________________________
dense_4 (Dense)              (None, 20)                820
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 42
=================================================================
Total params: 107,422
Trainable params: 107,422
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
260/260 [==============================] - 8s 31ms/step - loss: 0.4096 - accuracy: 0.8045 - val_loss: 0.4071 - val_accuracy: 0.8418
Epoch 2/5
260/260 [==============================] - 7s 26ms/step - loss: 0.2211 - accuracy: 0.9112 - val_loss: 0.4002 - val_accuracy: 0.8293
Epoch 3/5
260/260 [==============================] - 7s 26ms/step - loss: 0.1886 - accuracy: 0.9268 - val_loss: 0.3991 - val_accuracy: 0.8268
Epoch 4/5
260/260 [==============================] - 7s 26ms/step - loss: 0.1676 - accuracy: 0.9360 - val_loss: 0.4006 - val_accuracy: 0.8247
Epoch 5/5
260/260 [==============================] - 7s 26ms/step - loss: 0.1553 - accuracy: 0.9412 - val_loss: 0.3787 - val_accuracy: 0.8524
```

Modeling



Methods and Approaches - Nick

Comment Attribute Analysis:

- Focused on using the raw data set to create attributes that captured the writing style of toxic comments rather than the actual words themselves; designed to locate potential toxicity that may not be detected in misspellings or uncommon slang

Profanity Probability Package:

- Leveraged an external package that outputs a probability that profanity is present for every comment
  - Linear SVM model trained on 200k human-labeled samples of clean and profane text strings

```
df['profanity_present+prob'] = predict_prob(df['comment_text'])
df['exclamation_marks'] = [np.count_nonzero('!' in x) / len(x) for x in df['comment_text']]
df['upper_case_proportion'] = [len(list(filter(lambda y: y.isupper(), x))) / len(x) for x in df['comment_text']]
```

cont.

Dataframe Construction + Train Test Split

id	profanity_present+prob	exclamation_marks	upper_case_proportion
0001ea8717f6de06	0.078170	0.000000	0.020833
000247e83dcc1211	0.333465	0.000000	0.031250
0002f87b16116a7f	0.014044	0.002232	0.031250
0003e1cccfd5a40a	0.004403	0.000000	0.079840
00059ace3e3e9a53	0.005857	0.000000	0.011976
...	...	...	...
fff8f64043129fa2	0.009015	0.001678	0.018456
fff9d70fe0722906	0.289365	0.000000	0.060773
fffa8a11c4378854	0.310736	0.012195	0.012195
fffac2a094c8e0e2	0.995592	0.000000	0.795620
fffb5451268fb5ba	0.082584	0.000000	0.036101

```
X = df.iloc[:,-3:]
Y = df['toxic']
X_train, X_test, y_train, y_test = train_test_split(X, Y,
                                                    test_size=0.2,
                                                    random_state=42)
```

Modeling

```
model = MLPClassifier(hidden_layer_sizes=(50,10), max_iter=100,
                      solver='sgd', verbose=0,  random_state=42)
model = model.fit(X_train,y_train)
```

- Neural Network approach with optimized hidden layer sizes and other hyperparameters
- SGD used as a loss function, final evaluation based off of pure accuracy score and F-1 score

```
accuracy_score(model.predict(X_test),y_test)
```

```
0.9695127682907724
```

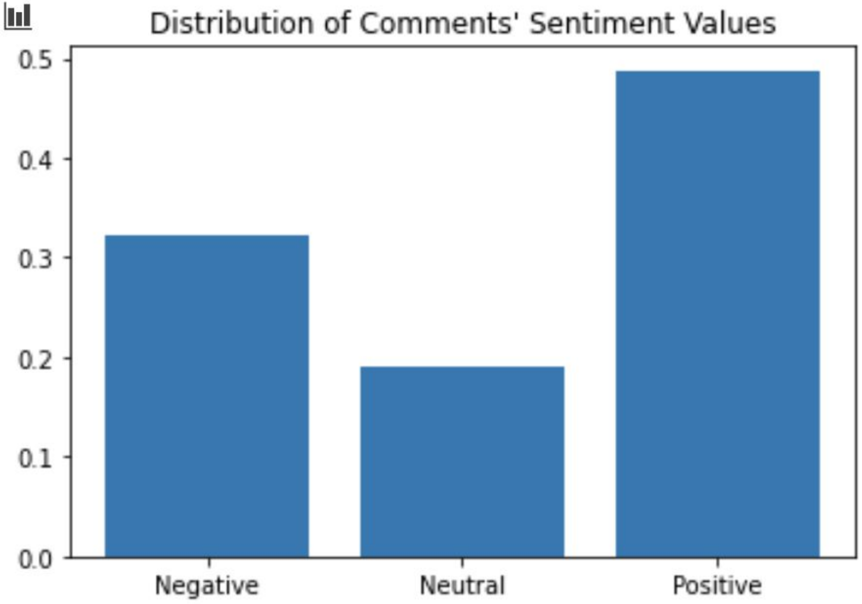
```
f1_score(model.predict(X_test),y_test)
```

```
0.847084708470847
```

Methods and Approaches - Nina

Feature Engineering:

- **Sentiment analysis:** used VADER (Valence Aware Dictionary for Sentiment Reasoning) to get sentiment score of comments
  - Scores range from -1 to 1
  - -1 = negative, 0= neutral, 1 = positive
- Word count
- Character Count
- Average Word Length



Dataframe:

- Each row is a comment

	sentiment	word_count	char_count	avg_word_length
0	0.5574	43	201	4.674419
1	0.2263	15	71	4.733333
2	-0.1779	42	183	4.357143
3	0.5106	112	477	4.258929
4	0.6808	14	50	3.571429
...	...	...	...	...
159566	0.2263	47	225	4.787234
159567	-0.4767	18	65	3.611111
159568	-0.2960	11	62	5.636364
159569	0.3612	25	87	3.480000
159570	-0.7003	35	134	3.828571

159571 rows × 4 columns

Model:

- Gradient Boosting Classifier
  - Accuracy = 91%

```
x_train = train
y_train = train_df['final_toxicity']
X_train, X_val, Y_train, Y_val = train_test_split(x_train,y_train, random_state=42)

clf =GradientBoostingClassifier(learning_rate=0.05, max_depth=8, max_features=0.3,
min_samples_leaf=100, n_estimators=500, random_state = 42)
clf.fit(X_train,Y_train)

print('Accuracy on training---')
print(clf.score(X_val,Y_val))
```

```
Accuracy on training---
0.9142456069987216
```

Methods and Approaches - Preston

Feature Engineering:

- ***find_toxic_words***: Finding words with a high count in toxic comments in the training set (**toxic_count**):
  - Sorting **toxic_count** dictionary by top 500 most frequent words
  - Removing words from **toxic_count** that are also present in **nontoxic_count** dictionary (constructed in a similar manner to toxic_count)
- ***count_toxic_words****:* Count the number of times each word in **toxic_count** appear in each comment in the dataset
- ***df['Total Toxic']***: Sum the number of times a toxic word appears in a comment across all **toxic_count** words as **total toxic**

Feature Engineering

```
[ ] def find_toxic_words(df):

        # Using regex, pull out a new words column, the words from the body of each email
        df['words'] = df['comment_text'].replace(r'[^0-9a-zA-Z]',' ').str.split()

        # Split training DF into a DF of Toxic emails and a DF of Ham emails
        toxic_train = df[df['final_toxicity'] == 1]
        nontoxic_train = df[df['final_toxicity'] == 0]

        toxic_words = toxic_train['words']
        nontoxic_words = nontoxic_train['words']

        # Count the number of times a word appears in a Spam email or a Ham email
        toxic_count = dict()
        nontoxic_count = dict()

        for text in toxic_words:
            for word in text:
                if word in toxic_count:
                    toxic_count[word] += 1
                else:
                    toxic_count[word] = 1

        for text in nontoxic_words:
            for word in text:
                if word in nontoxic_count:
                    nontoxic_count[word] += 1
                else:
                    nontoxic_count[word] = 1

        # Sort word dictionaries to top 500 words
        sorted_toxic_words = sorted(toxic_count, key=toxic_count.__getitem__, reverse=True)[0:500]
        sorted_nontoxic_words = sorted(nontoxic_count, key=nontoxic_count.__getitem__, reverse=True)[0:500]

        # Remove words from spam word list that are in both top 500 lists
        for word in sorted_toxic_words:
            if word in sorted_nontoxic_words:
                sorted_toxic_words.remove(word)

        df.drop('words', axis = 1, inplace=True)

        return sorted_toxic_words
```

```
[ ] def count_toxic_words(df, words):
      for word in words:
        count = df['comment_text'].apply(lambda x: x).str.findall(word)
        for x in range(len(count)):
          count[x] = len(count[x])
        df[word] = count
      return df
```

```
[ ] df['total toxic'] = np.sum(df.iloc[:,8:290], axis = 1)
```

Modeling

```
model = Sequential()
model.add(Dense(200, input_dim=x_train.shape[1], activation='relu'))
model.add(Dense(100, activation='sigmoid'))
model.add(Dense(25, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
model.fit(x_train,y_train, epochs=20)
print('Accuracy of Training Set: ', model.evaluate(x_train,y_train)[1])
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 200)               56600
_________________________________________________________________
dense_13 (Dense)             (None, 100)               20100
_________________________________________________________________
dense_14 (Dense)             (None, 25)                2525
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 26
=================================================================
Total params: 79,251
Trainable params: 79,251
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
961/961 [==============================] - 2s 2ms/step - loss: 0.4307 - accuracy: 0.7967
Epoch 2/20
961/961 [==============================] - 2s 2ms/step - loss: 0.3409 - accuracy: 0.8488
Epoch 3/20
961/961 [==============================] - 2s 2ms/step - loss: 0.3277 - accuracy: 0.8551
Epoch 4/20
961/961 [==============================] - 2s 2ms/step - loss: 0.3200 - accuracy: 0.8575
Epoch 5/20
961/961 [==============================] - 2s 2ms/step - loss: 0.3158 - accuracy: 0.8613
Epoch 6/20
961/961 [==============================] - 2s 2ms/step - loss: 0.3126 - accuracy: 0.8627
```

Results: Models

	precision	recall	f1-score	accuracy
Samantha	0.38	0.86	0.52	0.85
Nick	0.86	0.48	0.62	0.90
Preston	0.32	0.89	0.48	0.81
Nina	0.56	0.33	0.42	0.91
Combined / Majority Vote	0.61	0.72	0.66	0.93

Findings

- Different approaches resulted in different combination of scores (see results chart)
- Our ensembling technique was able to be significantly above our individual accuracies
- Comments are labeled as toxic for a variety of reasons and multiple models allow us to capture this
- Ethics:
  - Censorship vs. Freedom of Speech
- Next Steps:
  - Taking it further, we can construct a breakdown of toxicity to identify levels and types of toxic comments

Approach #2 : Multilabel Classifier

Model Selection

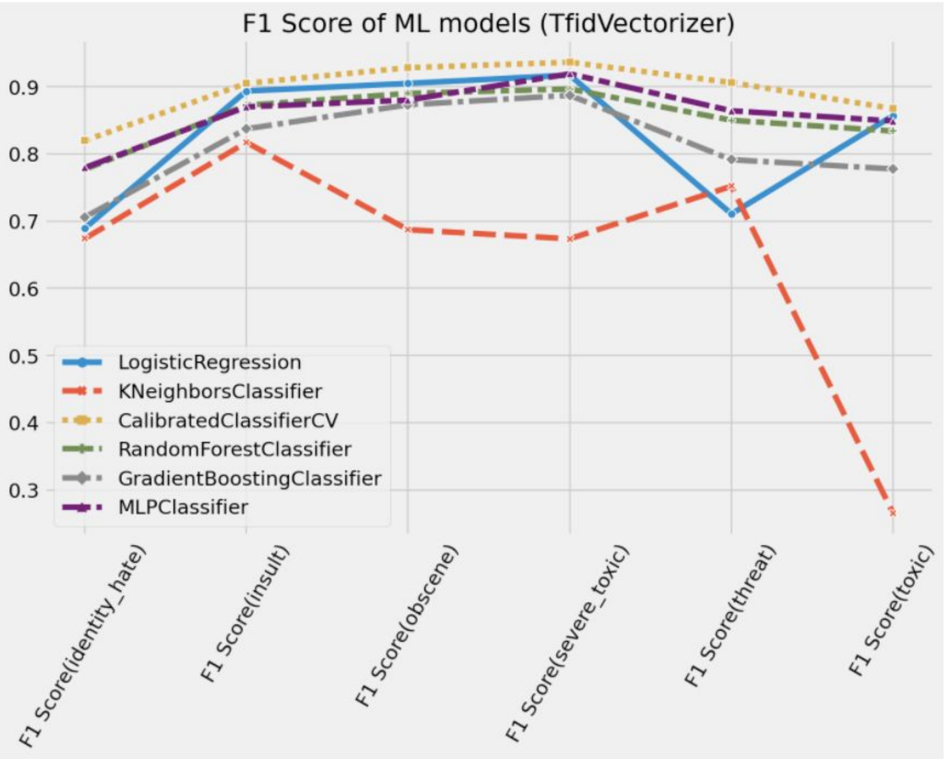
1. Select a list of Base Models (no hyperparameter tuning)
    a. Logistic Regression
    b. Random Forest
    c. LinearSVC
    d. KNeighborsClassifier
    e. GradientBoostingClassifier
    f. MLPClassifier
2. Train-Split against each toxic as target
    a. Need to clean data
    b. Split data into respective toxic feature volume
3. F1 Score Models Benchmark
4. Three ways to Evaluate them
    a. Self-made Cross-Validation Function -> Results were not very clear to me
    b. 2 different Vectorizers for text data

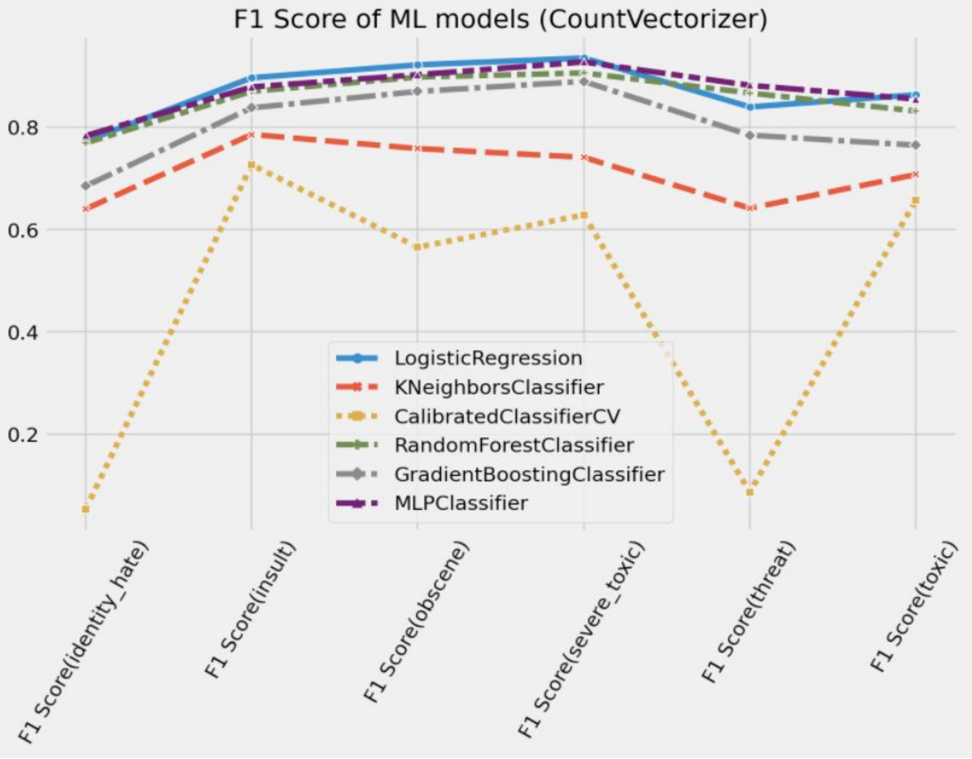
CountVectorizer V/S TfidfVectorizer

For the modeling I decided to use two methods for converting text data into vectors so that the computer can understand text data.

Idea is to compare the **same models, but with different text data vector pre processing,** and see if there is a difference in terms of F1 Score performance against all features.

- **CountVectorizer:** count number of times a word appear in the text
  - Bias in favour of most frequent words
- **TfidVectorizer:** calculates the overall text weightage of a word
  - Know how often a word appears, can help us to penalize the word when Cross-Validating the model.





From above, we can do a final round between the following winners:

LinearSVC (CalibratedClassifierCV) + TfidVectorizer

V/S

MLPClassifier/Logistic Reg. + CountVectorizer

Demo

Enjoy! :)

Future Work

- Get more data that has more valuable information.
    - Add some background context of each comment. **[Main Constraint in this Project]**
    - **SFU Opinion and Comments Corpus** interesting comment based dataset corpus. Similar to the Kaggle Toxic Competition
- Learn and use from more computational expensive models such as the IBM Research Approach
- Use more Cross-Validation methods
- Use other Python frameworks such as **Spacy**

Real-World Application

- ➔ Education:
  - Integrate into chat rooms / online forums to moderate discussion
  - Cyberbullying prevention
- ➔ Social Media:
  - Flagging, censoring or alerting people of abusive comments
- ➔ Digital Civic Engagement
  - Unicef and other organizations are working on improve social interactions with the existing systems we have right now.
- ➔ ... anything related to preserving the safety of the online community :)

Thank you!

Questions?