Utilizing Machine Learning for Content Moderation

Alex Gilmore, Maggy Foote, Tim Mandel, Matthew Perry

01

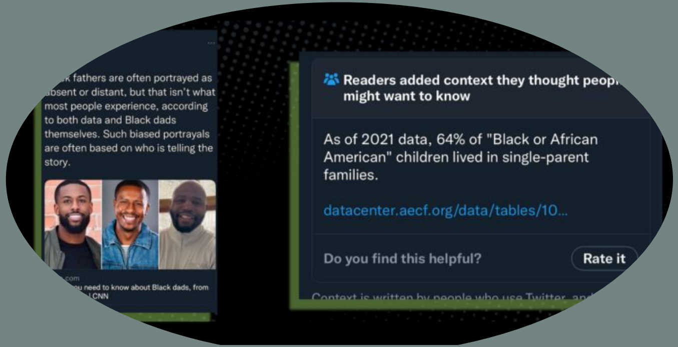
Background

Motivating Factors behind the Project

- Online Censorship is becoming increasingly more polarizing around the world. Social Media companies and lawmakers are trying to navigate the grey waters of false information, mature content for kids, and who ultimately gets to make the regulations.
- “China’s “Great Firewall” is one of the world’s most comprehensive internet censorship regimes, preventing citizens from accessing websites like Instagram, Wikipedia, and YouTube.” Which has seen VPN usage (despite being outlawed) doubled in the past year in China.
- Kids Online Safety Act (KOSA) is becoming more likely than ever before. There is a bill in both houses of congress in favor of regulations on the internet for when it comes to children, but that start a chain reaction of online restrictions?
  - What is a leading factor? “**Imagine you’re a teenager who just survived a school shooting, but the government says you can’t read the news about it on social media because it might make you depressed.**”

Community Notes on ‘X’ (formerly known as Twitter)

- ‘Community Notes’ under tweets that they deem are harmful, misleading, or missing important information. Are they working?
- “The program is severely hampered by the fact that for a Community Note to be public, it has to be generally accepted by a consensus of people from all across the political spectrum.”
- “Twitter waits until a similar number of people on the political right and left have agreed to attach a public Community Note to a tweet.”



New Landscape

Politics

Politicians are beginning to see use Social Media more and more to get their ‘message’ to the voter’s. As, we saw in 2016 and 2020 presidential elections, and will most likely continue in the future.

Voters

Us as Voter’s can feel like we’re close with these politicians due to being on Social Media with them. (“Only fingertips away”) It also becomes easier to engage with the opposite party.

Censorship

Companies are voter’s as well (or in other countries), do they get to determine what needs to be censored? How do they ensure to be unbiased in these censorship? What is eligible to be censored?

02

Objectives

Goals of this Project

- Build and implement a deep neural network machine learning model that can identify hate speech with at least 80% accuracy
- Use Pickle to apply the model to a separate data set
- Analyze and visualize the results

Process

01 Build the model

02 Optimize the Model

03 Apply the model to new data

04 Visualize the Results

03

About the Datasets

Hate Speech and Offensive Language

- Found on Kaggle
- Created by a company called Figure Eight, Inc.
- Created and operate an AI known as CrowdFlower
- Now called Appen

Hate Speech and Offensive Language

- Contained in a CSV
- Dataset contains ~24,700 real tweets
- CSV is separated into 7 columns

1. Index
2. Count
3. Hate_Speech
4. Offensive_Language
5. Neither
6. Class
7. Tweet

04

Preprocessing the Data

1. Remove usernames

```python
#remove usernames from dataset to enhance accuracy
for i in df.index:
    txt = df.loc[i]["tweet"]
    txt=re.sub(r'@[A-Z0-9a-z_:]+','',txt)#replace username-tags
    txt=re.sub(r'[RT]+','',txt)#replace RT-tags
    txt=re.sub('https?://[A-Za-z0-9./]+','',txt)#replace URLs
    df.at[i,"tweet"]=txt
```

2. Remove capitalization and punctuation

```python
#lower case all words before any preprocessing
df["tweet"] = df["tweet"].str.lower()

#removing punctuation
punctuations_list = string.punctuation
def remove_punctuations(text):
    temp = str.maketrans('', '',punctuations_list)
    return text.translate(temp)
df['tweet'] = df['tweet'].apply(lambda x: remove_punctuations(x))
df.tail()
```

3. Remove Unnecessary Text

```
#remove words that add no value to the tweet
# https://www.geeksforgeeks.org/hate-speech-detection-using-deep-learning/
def remove_stopwords(text):
    stop_words = stopwords.words('english')

    imp_words = []
    #store the important words
    for word in str(text).split():
        if word not in stop_words:
            #it's recommended to lemmatize the word as well
            #splitting it into its root which will provide a more accurate measure of which words are hate speech
            lemmatizer =WordNetLemmatizer()
            lemmatizer.lemmatize(word)
            imp_words.append(word)
    output = " ".join(imp_words)
    return output
df['tweet'] = df['tweet'].apply(lambda text: remove_stopwords(text))
df.head()
```

05

Training The Model

First Model (non-DNN)

```
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state= 1)
```

```
# Instantiate a StandardScaler instance
scaler = StandardScaler()

# Fit the training data to the standard scaler
X_scaler = scaler.fit(X_train)

# Transform the training data using the scaler
X_train_scaled = X_scaler.transform(X_train)

# Transform the testing data using the scaler
X_test_scaled = X_scaler.transform(X_test)
```

```
# Instantiate the KNeighborsClassifier model with n_neighbors = 3
knn = KNeighborsClassifier(n_neighbors=2)
```

```
# Train the model using the training data
knn.fit(X_train_scaled, y_train)
```

	Precision	**Recall**	**f1-Score**
0	0.04	0.04	0.07
1	1	0.84	0.91
Accuracy			**0.84**
Micro avg	0.52	0.87	0.49
Weighted avg	0.99	0.84	0.91

DNN Model 1

Layer 1 Units	16
Input_dim	# columns after removing y values
Layer 2 Units	1
Layer 2 Activation	Relu
Dropout	0.25
Output Layer Units	1
Output Layer Activation:	Sigmoid
Epochs	15

Accuracy: 84.26%
Loss: 0.96

06

Optimizing the Model using DNN

DNN Model 2: Utilizing KerasTuner to find best hyperparameters

Layer 1 Units	91
Layer 1 Activation	Relu
Layer 2 Units	51
Layer 2 Activation	Relu
Dropout	0.25
Output Layer Units	1
Output Layer Activation:	Sigmoid
Epochs	15

Accuracy: 84.25%
Loss: 0.96

DNN Model 3: Normalized Data

Layer 1 Units	16
Layer 1 Activation	Relu
Layer 2 Units	16
Layer 2 Activation	Relu
Dropout	None
Output Layer Units	1
Output Layer Activation:	Sigmoid
Epochs	50

Accuracy: 83.74%

Loss: 0.45

Layer 1 Units	16
Layer 1 Activation	Relu
Layer 2 Units	16
Layer 2 Activation	Relu
Dropout	0.5
Output Layer Units	1
Output Layer Activation:	Sigmoid
Epochs	50

DNN Model 4: Increased Dropout

Accuracy: 84.25%

Loss: 0.63

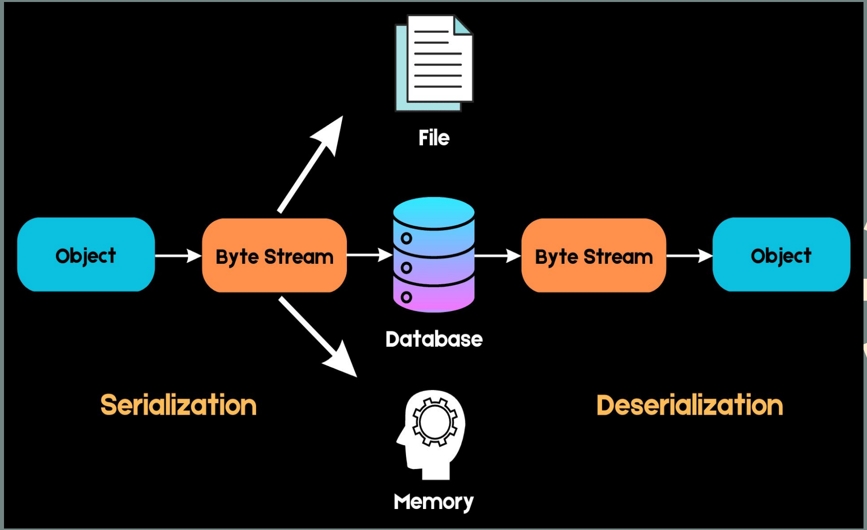
07

Applying the Model

Attempt 1: Pickle

Pickle:

- Built in python module that transforms complex objects into byte streams and then transform the byte stream into an object (called Object Serialization)

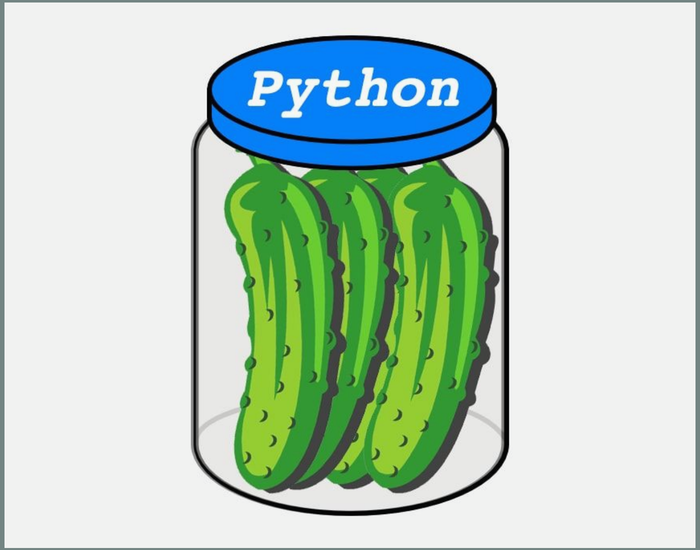


Attempt 1: Pickle

Used For:

- Storing high level objects data memory (serialization) and then loading them back into high-level objects (deserialization)

- Appling trained learning models onto new data sets



Roadblocks:

- When working with large data structures or models, Pickle slows down considerably and can override memory capabilities.

```
44]: import pickle

     # save the iris classification model as a pickle file
     fit_model_pkl_file = "fit_model.pkl"

     with open(fit_model_pkl_file, 'wb') as file:
         pickle.dump(knn, file)

---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
Cell In[44], line 7
      4 fit_model_pkl_file = "fit_model.pkl"
      6 with open(fit_model_pkl_file, 'wb') as file:
----> 7     pickle.dump(knn, file)

MemoryError:
```

Attempt 2: Keras.model's load_model

- Keras has its own method for saving models, which is much less of a memory burden

```python
nn_model.save('my_model.h5')
```

```python
from keras.models import load_model
model = load_model('my_model.h5')
```

Roadblocks: Incompatible Architecture

- Data type of testing must be the same as data that the model was trained on

- The shape of the new data must be compatible with the shape of the training data, with an identical number of features.

```
nn_model.save('my_model.h5')
```

```
from keras.models import load_model
model = load_model('my_model.h5')
```

Roadblocks: In Our Code

- Pd.dummies turned every tweet into a column, which means a lot of columns

- The dataset that we tried to apply the model on was too dissimilar to the dataset it was trained on

- Rerunning the model took a considerable amount of time

[11]:

professor bet hes starbucks queer	05professor charlie sheen could coach better crusty dusty yeah hired bryan price pitching coach	05professor thats happens getting pussy pussy twitter seems like clear cut choice hahahaha	0beythelau huntermoore bitch gross fuck touchin saggy titties	0ddment could fuck bowl graham cracker cereal	0hallis0n bitch caught body week ago everybody catching bullet holes	0o faggot	...	zooyorkinit youre frumpy bitch	zp3 little girl tom sawyer gets trapped cave along injun joe almost die injun joe dies find treasure	zramsin betodavidthomas go fuck self pussy 128536	zrexxx slowdollaz take bitch recieve head mom	zt0mm lucky monkey	ztsupreme uncr4fted yopapichulo doesnt matter joke fact still remains hes faggot pretending girl	b
0	0	0	0	0	0	0	...	0	0	0	0	0	0	
0	0	0	0	0	0	0	...	0	0	0	0	0	0	
0	0	0	0	0	0	0	...	0	0	0	0	0	0	
0	0	0	0	0	0	0	...	0	0	0	0	0	0	
0	0	0	0	0	0	0	...	0	0	0	0	0	0	

```
[12]: X.shape
[12]: (24783, 24699)
```

Roadblocks: How We Tried to Fix It

- Find a simpler data set
- Make sure the pre-processing of the new dataset is identical to the old set
- Examine the shape of the new dataset's features and compare

```
X_test_scaled.shape
```

```
(6196, 24699)
```

```
X_tweets.shape
```

```
(24783, 24136)
```

Roadblocks: How We Tried to Fix It

- Add more columns to the new dataset

```
additional_length = 241
X_tweets_extended = np.append(X_tweets, np.full(additional_length, np.nan))
```

```
X_tweets_extended.head()
```

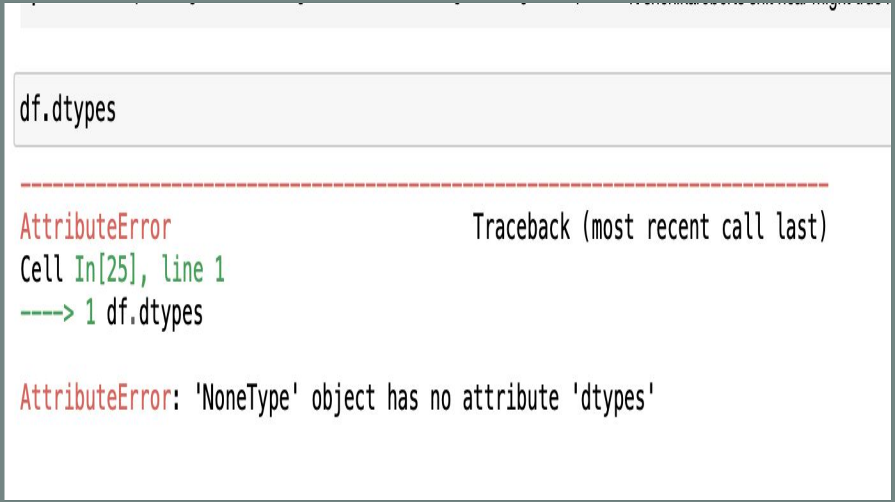
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[58], line 1
----> 1 X_tweets_extended.head()

AttributeError: 'numpy.ndarray' object has no attribute 'head'
```

```
missing_columns = 241
for col in missing_columns:
    X_tweets[col] = 0
```

```
---------------------------------------------
TypeError
Cell In[57], line 2
      1 missing_columns = 241
----> 2 for col in missing_columns:
      3     X_tweets[col] = 0

TypeError: 'int' object is not iterable
```

Roadblocks: How We Tried to Fix It

- Deleting columns from the training dataset

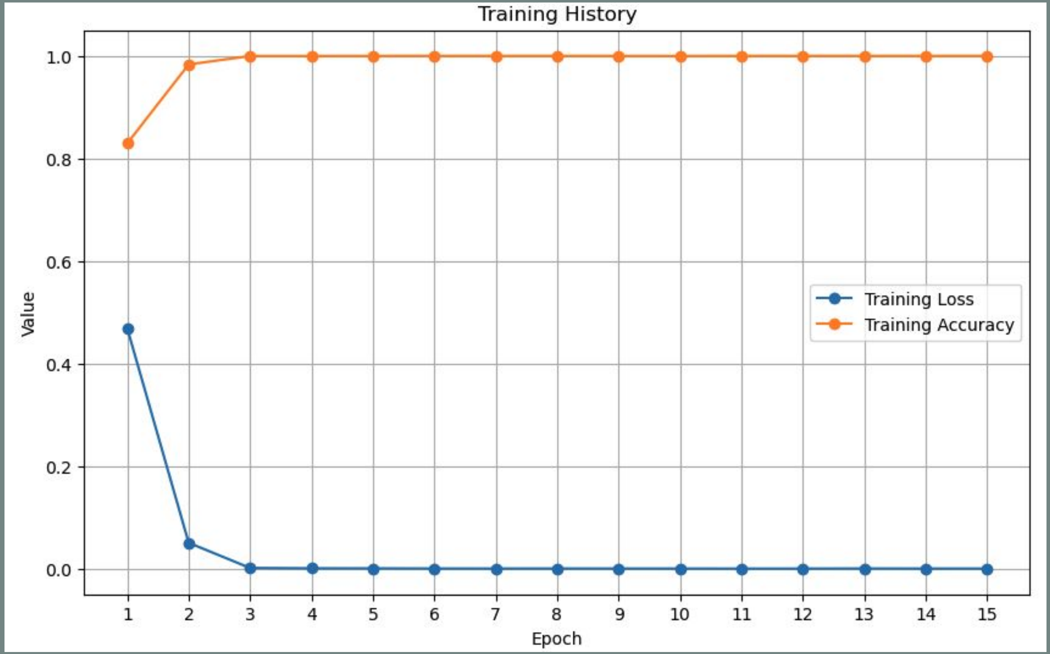


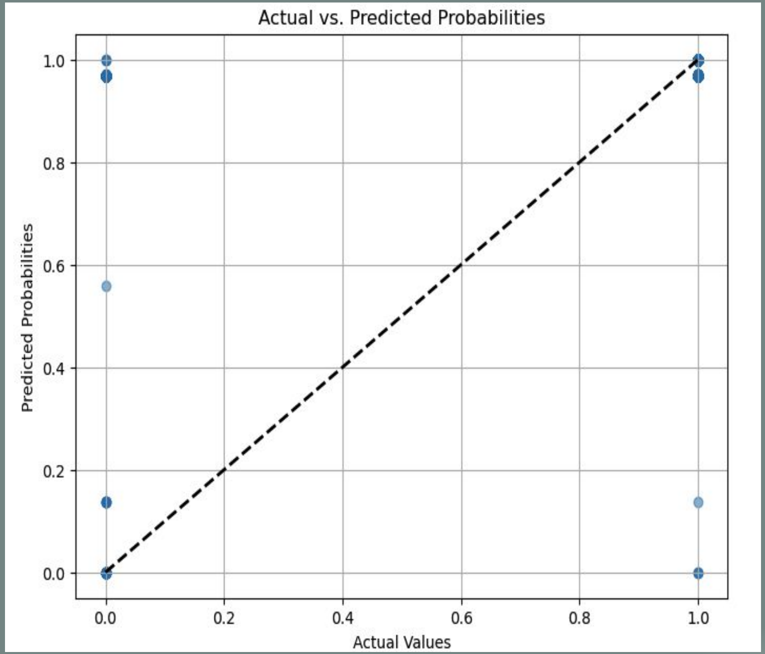
```
df.dtypes

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[25], line 1
----> 1 df.dtypes

AttributeError: 'NoneType' object has no attribute 'dtypes'
```

08

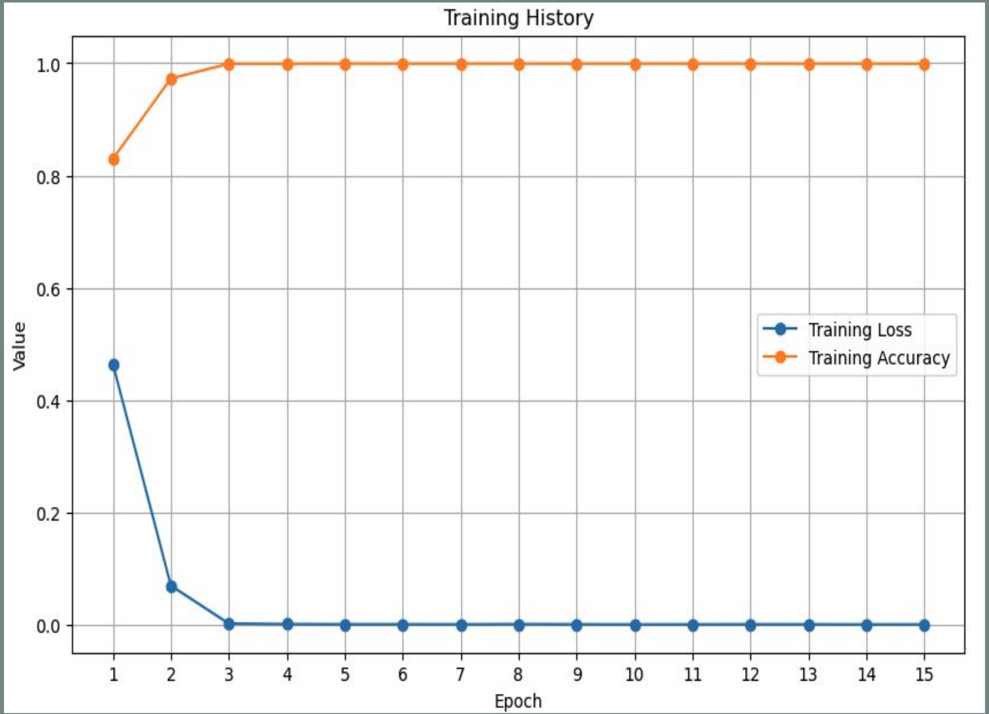
Results

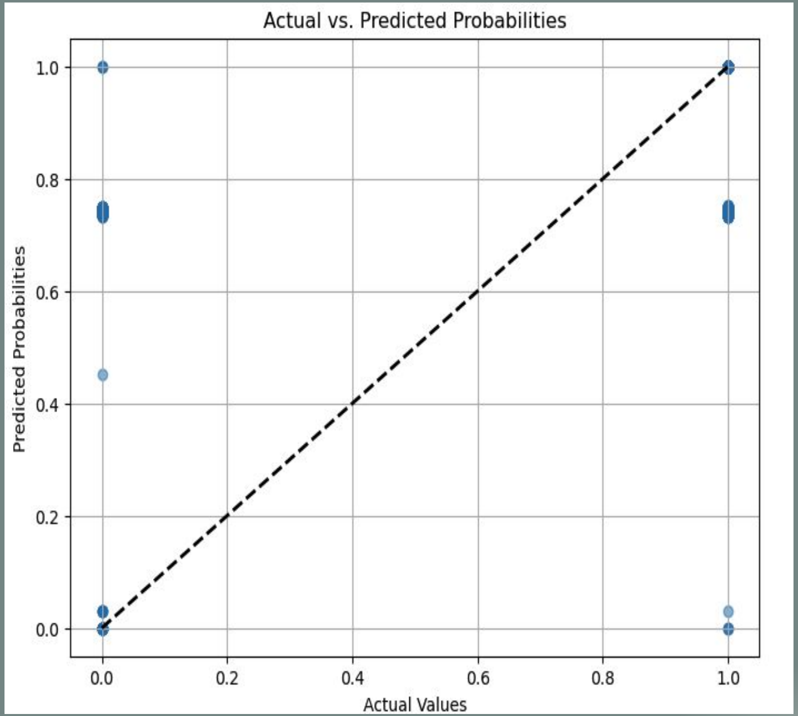
First Model


Training History
Value
Epoch
Training Loss
Training Accuracy
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


Actual vs. Predicted Probabilities
Predicted Probabilities
Actual Values
0.0
0.2
0.4
0.6
0.8
1.0

Final Model


Training History
Value
Epoch
1.0
0.8
0.6
0.4
0.2
0.0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Training Loss
Training Accuracy


Actual vs. Predicted Probabilities
Predicted Probabilities
Actual Values
1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.2 0.4 0.6 0.8 1.0

09

Limitations and Considerations

Size of Dataset & High Loss

- One of the issues that we confronted was the dataset not being large enough to accommodate all instances of offensive language / hate speech
- Despite multiple rounds of cleaning, the data was still registering with a high loss.
  - Could be due to the anomalous nature of tweets
  - Unless you know what someone might have said, it's very difficult to filter out words that are not normalized

Questions?