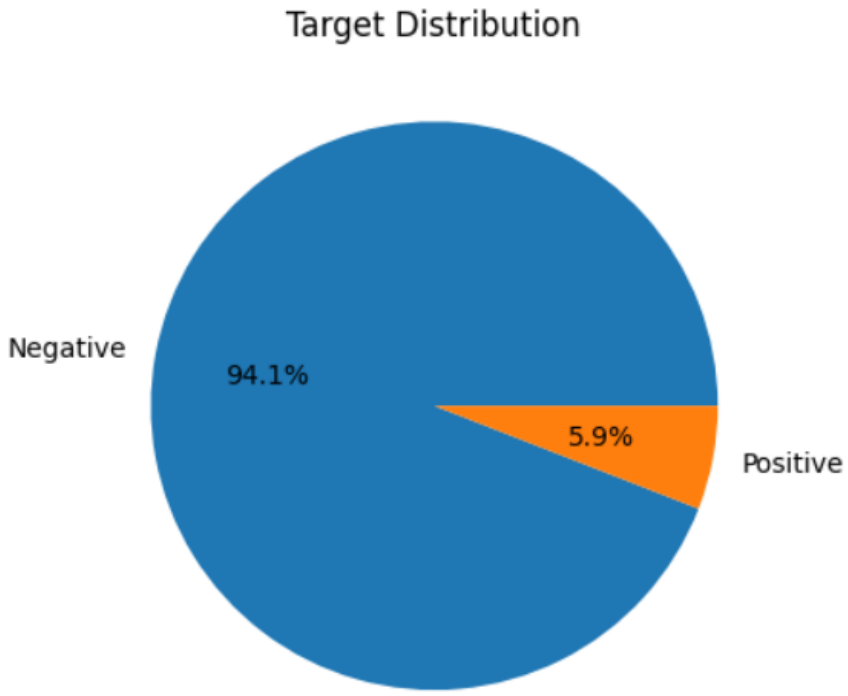
Problem Statement:
With the increasing use of online platforms for communication, the prevalence of toxic language and online abuse has become a major concern. To promote a safe and inclusive online environment, there is a need for an efficient and effective toxic words classification model that can identify and categorize toxic language in comments. The aim of this capstone project is to develop a model that can accurately classify comments into toxic and non-toxic categories, which can be used to improve content moderation, sentiment analysis, and chatbot performance.

Data Descritpion:

The dataset contains a total of 1,803,874 examples. The target variable is binary, with 1 indicating a toxic comment and 0 indicating a non-toxic comment.

The distribution of the target variable is highly imbalanced, with only 5.90% of the examples being toxic and 94.10% being non-toxic. This means that the dataset is heavily skewed towards non-toxic comments, which can pose a challenge for machine learning models that are trained on this data.

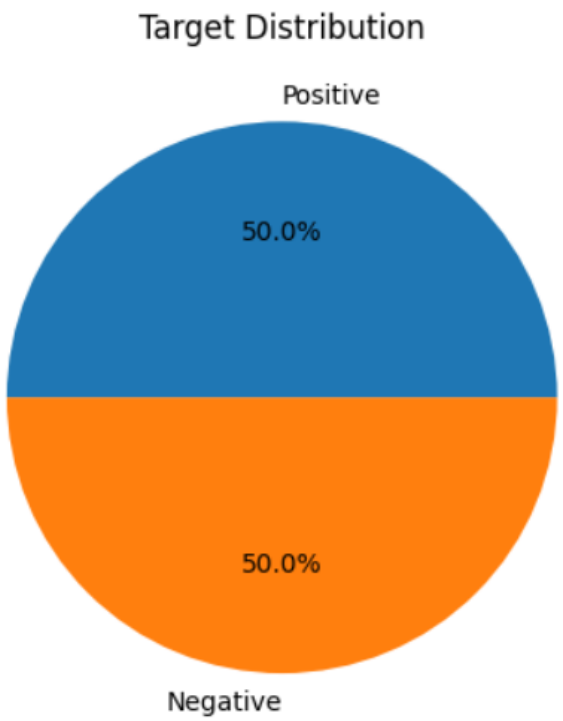



```
Number of positive examples: 106438 (5.90%)
Number of negative examples: 1698436 (94.10%)
```

Data Preprocessing:

At begining, a portion of the negative class is randomly sampled to create a balanced dataset. The sampling is done in such a way that the number of negative comments is equal to the number of positive comments in the original dataset. This is done to ensure that the dataset is balanced, meaning that each class has an equal number of examples.

This process is important because it allows us to create a more representative dataset that can be used to train machine learning models for tasks such as toxic comment classification. By balancing the dataset, we can ensure that the models are not biased towards one class, which can lead to poor performance on the other class.




```
Number of positive examples: 106438 (5.90%)
Number of negative examples: 106438 (5.90%)
```

Next, I begin to preprocess text data. The preprocessing steps applied to the text data are as follows:

1. Remove unwanted characters: Any characters that are not alphabets are removed from the text data.

2. Convert to lowercase: The text data is converted to lowercase to avoid having multiple copies of the same word.
3. Tokenization: The text data is split into individual words.
4. Stopword removal: Stopwords are common words in a language that do not add much meaning to a sentence. These words are removed from the text data to reduce noise.
5. Stemming: Words are reduced to their base or root form. This is done to reduce the dimensionality of the data.
6. Lemmatization: Words are transformed into their base form based on the context of the sentence. This is done to obtain more meaningful features.
7. Join the words: The processed words are joined back together to form a clean sentence.

After the preprocessing steps, the text data is transformed into numerical features using the TfidfVectorizer. The TfidfVectorizer computes the TF-IDF (Term Frequency-Inverse Document Frequency) values for each word in the text data. The TF-IDF values represent the importance of each word in a document relative to the entire corpus of documents. The resulting feature vectors are used as input to machine learning models for classification. Finally, the data is spilted into train-test set.

Traditional models

I use default parameters at first. Here is the summary of the result.

	Accuracy	Precision	Recall	F1	Time taken (s)
Logistic regression	0.827508	0.856931	0.786526	0.820220	0.106007
Naive Bayes	0.785771	0.762875	0.829671	0.794872	0.005503
Random forest	0.826123	0.849540	0.792864	0.820224	53.947193
SVM	0.834437	0.860438	0.798592	0.828362	7.712046

Based on the results, the SVM model performs the best in terms of accuracy, precision, and F1 score, with an accuracy of 0.834, precision of 0.860, and F1 score of 0.829.
The logistic regression and random forest models also perform well, with accuracy scores of 0.827 and 0.826, respectively. However, the Naive Bayes model has the lowest accuracy score of 0.786.

When considering precision, the SVM and logistic regression models perform the best, with precision scores of 0.860 and 0.857, respectively. The Naive Bayes model has the lowest precision score of 0.763.

For recall, the Naive Bayes model performs the best with a score of 0.830, followed closely by the SVM model with a score of 0.799. The random forest model has the lowest recall score of 0.793.

In terms of F1 score, the SVM model again performs the best with a score of 0.828, followed by the logistic regression and random forest models with scores of 0.820.

Finally, the time taken to train the models varies significantly, with the random forest model taking the longest at 53.95 seconds, while the Naive Bayes model takes the shortest time at only 0.0055 seconds followed by logistic regression with training time of 0.1060 seconds. The SVM model takes a moderate amount of time at 7.71 seconds.

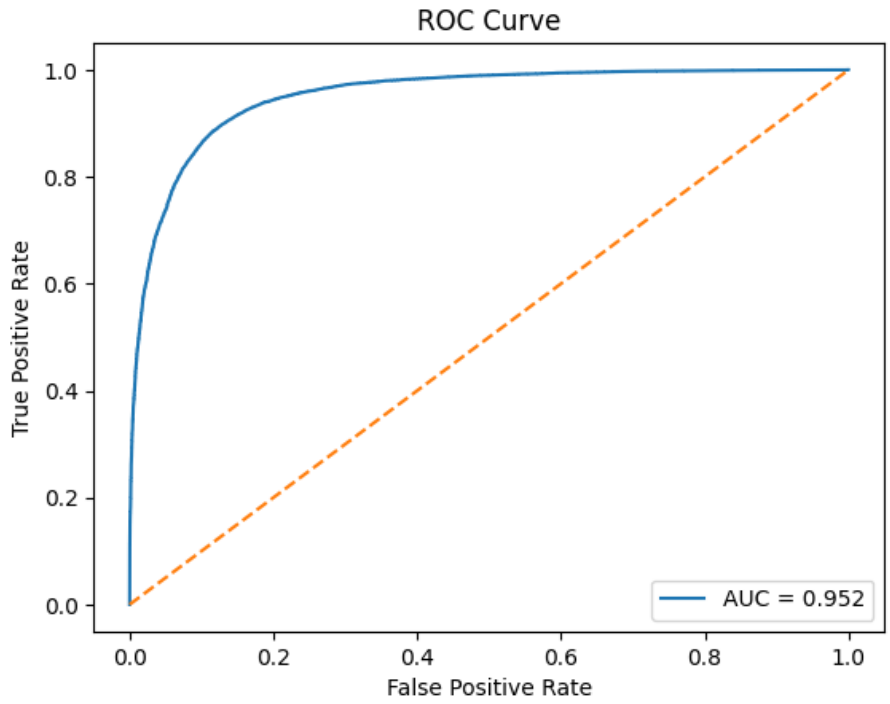
Tune Logistic regression

Based on these results, logistic regression was chosen as the model to be tuned due to its near-best performance and fast training time.

I then use gridsearchcv to find best hyperparameters [penalty,C] of logistic regression.
The results show that the model achieved an accuracy of 0.8819, precision of 0.8953, recall of 0.8651, and F1 score of 0.8799. The model had a training time of 1.2644 seconds.Compared to the previous logistic regression model, which had an accuracy of 0.8275, precision of 0.8569, recall of 0.7865, and F1 score of 0.8202, the tuned logistic regression model achieved a significant improvement in accuracy and recall, while maintaining a high precision and F1 score.

	lr_all_train_tune	lr
Accuracy	0.881905	0.827508
Precision	0.895335	0.856931
Recall	0.865070	0.786526
F1	0.879943	0.820220
time_taken(s)	1.264386	0.106007

Additionally, the ROC curve for the logistic regression model is shown below, with an AUC score of 0.952. This suggests that the model has good discriminative ability and performs well in distinguishing between positive and negative comments.



Overall, the logistic regression model with tuned hyperparameters has shown to be an effective approach for sentiment analysis on the dataset.

Neural network,CNN,and LSTM

Next, i use Neural network,CNN,and LSTM. I stil preprocessed the text data using a function called preprocess_text(), which removes unwanted characters, converts all text to lowercase, tokenizes the text into words, removes stop words, stems the words using the Porter stemmer, and lemmatizes the stemmed words using the WordNet lemmatizer. Afterwards, instead of using TfidfVectorizer,I tokenize the text data and convert it to sequences, and then padded the sequences to ensure they all have the same length using the pad_sequences() function. I customize layers for these three models.

To prevent overfitting, I set up early stopping with a patience of 1. Finally, Itrained the model on the preprocessed text data for 10 epochs with a batch size of 64, and recorded the training time.

Here is the result.

	Accuracy	Precision	Recall	F1	time taken(s)
Neural Newtork	0.879557	0.896740	0.857901	0.876891	9.350226
CNN	0.883150	0.886070	0.879369	0.882707	17.893144
LSTM	0.889445	0.875153	0.908493	0.891511	70.261034
logistic_regression	0.881905	0.895335	0.865070	0.879943	1.264386

Overall, the LSTM model achieved the best performance with an accuracy of 0.8894, followed by the CNN model with an accuracy of 0.8832. The Logistic Regression model achieved an accuracy of 0.8819, and the Neural Network achieved an accuracy of 0.8796. The LSTM model took the longest time to train, with a time of 70.26 seconds, followed by the CNN model with a time of 17.89 seconds. The Logistic Regression model took the shortest time to train, with a time of 1.26 seconds, and the Neural Network took a time of 9.35 seconds.

Tune CNN

Based on these results, CNN was chosen as the model to be tuned due to its near-best performance and fast training time.

I use GridSearchVv and KerasClassifier together to test best combination of 'optimizer': ['adam', 'rmsprop', 'sgd'] and 'learning_rate': [0.001,0.001, 0.01, 0.1]. The best hyperrameters are 'learning_rate': 0.001 and 'optimizer': 'adam'. The accuracy on the valid set is 0.882. The result does not improve.

I then use this best parameters to create the best model and train on 10 bootstrap dataset. Here is the result after ensemble.

	cnn_ensemble
Accuracy	0.886767
Precision	0.886822
Recall	0.886697
F1	0.886759

The accuracy increases from 0.883150 to 0.8866767. The precision increases from 0.886070 to 0.886822. The recall score increases from 0.879369 to 0.886697. The f1 score increases from 0.882707 t0 0.886759. The improvement is very limited.

Pretrained Models:GloVe

I tried to use pre-trained word embeddings GloVe. These pre-trained embeddings are trained on very large datasets and may provide better performance on my task.

```
# Load pre-trained GloVe embeddings
word_vectors = KeyedVectors.load_word2vec_format('glove.6B.100d.txt', binary=False)

# Tokenize the text and convert it to sequences
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(train_text)
train_seq = tokenizer.texts_to_sequences(train_text)
val_seq = tokenizer.texts_to_sequences(val_text)

# Pad the sequences so they all have the same length
maxlen = 100
train_seq = pad_sequences(train_seq, maxlen=maxlen)
val_seq = pad_sequences(val_seq, maxlen=maxlen)

# Map vocabulary to pre-trained embeddings
num_words = min(10000, len(tokenizer.word_index)+1)
embedding_matrix = np.zeros((num_words, 100))
for word, i in tokenizer.word_index.items():
    if i >= num_words:
        break
    if word in word_vectors:
        embedding_matrix[i] = word_vectors[word]


# Create CNN model using pre-trained embeddings
model = Sequential()
model.add(Embedding(input_dim=num_words, output_dim=100, input_length=maxlen, weights=[embedding_matrix]))
model.add(Conv1D(filters=32, kernel_size=5, activation='relu'))
model.add(Conv1D(filters=64, kernel_size=5, activation='relu'))
model.add(Conv1D(filters=128, kernel_size=5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

# Compile and fit the model
learning_rate = 0.001
optimizer = Adam(learning_rate=learning_rate)
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
model.fit(train_seq, train_target, validation_data=(val_seq, val_target), epochs=10, batch_size=64, verbose=1, callbacks=[early_stopping])
```

The accuracy score is 0.8885, which increase a little from the previous ensemble result 0.8866.

Interpret the GloVe model

```
# get the word-to-index mappings
word_index = tokenizer.word_index

# get the probability of each word being toxic
weights = model.layers[0].get_weights()[0]
word_prob = dict(zip(word_index.keys(), weights[:, 0]))

# sort the words by the probability of being toxic
sorted_words = sorted(word_prob.items(), key=lambda x: x[1], reverse=True)

# print the top 20 most toxic words
for i in range(20):
    print(sorted_words[i])
```

✓ 0.0s

```
('untrustworthi', 1.6873754)
('uncomfort', 1.5301968)
('boot', 1.505801)
('scratch', 1.4772482)
('mathemat', 1.4629576)
('railroad', 1.4444764)
('congreg', 1.4242818)
('accumul', 1.400033)
('ray', 1.3844199)
('effect', 1.3692124)
('given', 1.3606409)
('gomer', 1.3530343)
('requir', 1.3435355)
('scofflaw', 1.3414085)
('intox', 1.3394476)
('hart', 1.337225)
('pervas', 1.3231249)
('stake', 1.3137982)
('shambl', 1.3057103)
('frickin', 1.3044283)
```

I extracts the top 20 most toxic words from the trained model. The sorted_words list is created by sorting the word_prob dictionary in descending order based on the probability of being toxic. The resulting list contains tuples of the form (word, probability).From the top 20 most toxic words, we can see that some words may not necessarily be toxic in general, but they may be used in a toxic context. For example, words like "mathemat" and "given" are not inherently toxic, but they can be used in a toxic manner. Additionally, some words are not commonly used in everyday language, such as "scofflaw" and "shambl". Overall, these top 20 most toxic words may not necessarily represent the most commonly used toxic words in real-world scenarios, but they provide some insights into the types of words that the model has learned to associate with toxicity.

Conclusion

This project aimed to develop an efficient and effective toxic words classification model that can identify and categorize toxic language in comments. I have tried SVM, logistic regression, Naive Bayes , random forest, neural network, CNN, LSTM, CNN with GloVe embedding. Among them, CNN with GloVe embedding has the overall best performance with Accuracy 0.8885,precisioni 0.8814, recall 0.8986, f1 0.8894 and training time 17. LSTM has slightly better accuracy but much longer training time. Logistic regression has slightly lower accuracy but much short training time. Due to my hardware limit, I did not explore more hypermaraters and other pre training models. I believe there are still a lot improvement that can be done to deeping learning models. Overall, This model can be used to improve content moderation, sentiment analysis, and chatbot performance, promoting a safe and inclusive online environment.