Preprocessing

```
import numpy as np
import pandas as pd
import nltk
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from nltk.stem import WordNetLemmatizer
```

```
# nltk.download('punkt')
# nltk.download('wordnet')
```

```
data = pd.read_csv('updated_hate_speech2.csv',engine='python')
X = data['Content'].values
y = data['Label'].values

# We did a 80/20 split for training and testing. We later split the training set into training and validation
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.2, random_state=42)
```

Naive Bayes

We chose a Multinomial Naive Bayes because it works best with discrete features such as word counts or frequencies

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from keras.preprocessing.text import Tokenizer
```

Saved successfully! ×

```
process the data to create vectors of the word frequencies
p_words='english')

nb_X_train = vectorizer.fit_transform(train_X)
nb_X_test = vectorizer.transform(test_X)
```

```
naive = MultinomialNB()
naive.fit(nb_X_train, train_y)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
# Evaluate the model
y_pred = naive.predict(nb_X_test)
y_pred = (y_pred > 0.5).astype('int32')

misclassified_samples = 0
for i in range(len(y_pred)):
    if y_pred[i] != test_y[i]:
        original_sentence = vectorizer.inverse_transform(nb_X_test[i])[0]
        print("Sentence: ", " ".join(original_sentence))
        print("Actual Label: ", test_y[i])
        print("Predicted Label: ", y_pred[i])
        print(" ")
        misclassified_samples += 1
    if misclassified_samples >= 5:
        break

print('Accuracy: %.3f' % accuracy_score(test_y, y_pred))

print('Precision: %.3f' % precision_score(test_y, y_pred))
print('Recall: %.3f' % recall_score(test_y, y_pred))

print('F1: %.3f' % f1_score(test_y, y_pred))
```

```
Sentence:  really point people longer hate fuck bitch
Actual Label:  0
Predicted Label:  1

Sentence:  trump rick love listening interview fuck
Actual Label:  1
Predicted Label:  0

Sentence:  plumber make kill family
Actual Label:  0
Predicted Label:  1

Sentence:  whatsoever welsh visit totally source seoul sentiments saying reliable regard reference prove professional positi
Actual Label:  1
Predicted Label:  0

Sentence:  wow word wikipedia violation views view users user undeniable truth tracks template surprised suggest stop star s
Actual Label:  1
Predicted Label:  0

Accuracy: 0.800
Precision: 0.771
Recall: 0.780
F1: 0.776
```

CNN

```
from keras.models import Sequential
from keras.layers import Conv1D, GlobalMaxPooling1D, Dense, Dropout, Flatten
from keras.utils import pad_sequences
from keras.layers import Embedding, Conv1D, MaxPooling1D, Dense, Dropout, Flatten
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from keras.callbacks import EarlyStopping


train_X, val_X, train_y, val_y = train_test_split(train_X, train_y, test_size=0.25, random_state=42)
```

Saved successfully! ×

```
                                r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        embedding = np.asarray(values[1:], dtype='float32')
        word_embeddings[word] = embedding


tokenizer = Tokenizer()
#This tokenizes the text and counts the frequency of each token
tokenizer.fit_on_texts(train_X)
#create a vocabulary of the most frequently occurring words in the training data
cnn_X_train = tokenizer.texts_to_sequences(train_X)
cnn_X_val = tokenizer.texts_to_sequences(val_X)
cnn_X_test = tokenizer.texts_to_sequences(test_X)


# We need to pad the sequences here so they have the right shape
maxlen = 100
cnn_X_train = pad_sequences(cnn_X_train, padding='post', maxlen=maxlen)
cnn_X_val = pad_sequences(cnn_X_val, padding='post', maxlen=maxlen)
cnn_X_test = pad_sequences(cnn_X_test, padding='post', maxlen=maxlen)


#Making the matrix for the embedding layer
word_index = tokenizer.word_index
embedding_dim = 100
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))

for word, i in word_index.items():
    embedding_vector = word_embeddings.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
    else:
        # If word is not in pre-trained embeddings, use random vector
        embedding_matrix[i] = np.random.normal(scale=0.6, size=(embedding_dim,))


cnn = Sequential()
cnn.add(Embedding(input_dim=len(word_index) + 1, output_dim=100, input_length=maxlen, weights=[embedding_matrix], trainable=False
cnn.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
cnn.add(MaxPooling1D(pool_size=2))
cnn.add(Flatten())
cnn.add(Dense(units=250, activation='relu'))
cnn.add(Dropout(rate=0.2))
cnn.add(Dense(units=1, activation='sigmoid'))


cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
cnn.fit(cnn_X_train, train_y, epochs=10, batch_size=64, validation_data=(cnn_X_val, val_y), callbacks=[EarlyStopping(patience=3)]
```

```
Epoch 1/10
1681/1681 [==============================] - 53s 30ms/step - loss: 0.4766 - accuracy: 0.7673 - val_loss: 0.4433 - val_accura
Epoch 2/10
1681/1681 [==============================] - 48s 28ms/step - loss: 0.4155 - accuracy: 0.8062 - val_loss: 0.4180 - val_accura
Epoch 3/10
1681/1681 [==============================] - 47s 28ms/step - loss: 0.3848 - accuracy: 0.8248 - val_loss: 0.4210 - val_accura
Epoch 4/10
1681/1681 [==============================] - 54s 32ms/step - loss: 0.3548 - accuracy: 0.8402 - val_loss: 0.4250 - val_accura
Epoch 5/10
1681/1681 [==============================] - 48s 29ms/step - loss: 0.3218 - accuracy: 0.8578 - val_loss: 0.4462 - val_accura
<keras.callbacks.History at 0x7f071d062fd0>
```

```
y_pred = cnn.predict(cnn_X_test)
y_pred = (y_pred > 0.5).astype('int32')

misclassified_samples = 0
for i in range(len(y_pred)):
    if y_pred[i] != test_y[i]:
        original_sentence = tokenizer.sequences_to_texts([cnn_X_test[i]])[0]
        actual_label = test_y[i]
        predicted_label = y_pred[i][0]
        print("Sentence: ", original_sentence)
        print("Actual Label: ", actual_label)
        print("Predicted Label: ", predicted_label)
        print(" ")
        misclassified_samples += 1
    if misclassified_samples >= 5:
```

Saved successfully!

```
print('Accuracy: %.3f' % accuracy_score(test_y, y_pred))

print('Precision: %.3f' % precision_score(test_y, y_pred))

print('Recall: %.3f' % recall_score(test_y, y_pred))

print('F1: %.3f' % f1_score(test_y, y_pred))
```

```
1121/1121 [==============================] - 12s 11ms/step
Sentence:  i really hate being a bitch to people but at this point i no longer give a fuck
Actual Label:  0
Predicted Label:  1

Sentence:  what are you doing afghan faggot i am bored louis watching netflix
Actual Label:  1
Predicted Label:  0

Sentence:  how to play these niggas and bitches that be snakes
Actual Label:  1
Predicted Label:  0

Sentence:  and you are in what position to give out orders dictate and judge exactly none none whatsoever you are only makin
Actual Label:  1
Predicted Label:  0

Sentence:  bolest reba recite
Actual Label:  1
Predicted Label:  0

Accuracy: 0.805
Precision: 0.795
Recall: 0.754
F1: 0.774
```

RNN

```
import numpy as np
from keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, GRU, Bidirectional
from sklearn.metrics import classification_report
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from keras.callbacks import EarlyStopping
```

```
train_X, val_X, train_y, val_y = train_test_split(train_X, train_y, test_size=0.25, random_state=42)
```

```
word_embeddings = {}
with open('glove.6B.100d.txt', 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        embedding = np.asarray(values[1:], dtype='float32')
        word_embeddings[word] = embedding
```

```
# Could include num_words = 500
tokenizer = Tokenizer()
#This tokenizes the text and counts the frequency of each token
tokenizer.fit_on_texts(train_X)
#create a vocabulary of the most frequently occurring words in the training data
rnn_X_train = tokenizer.texts_to_sequences(train_X)
rnn_X_val = tokenizer.texts_to_sequences(val_X)
rnn_X_test = tokenizer.texts_to_sequences(test_X)
```

```
# We need to pad the sequences here so they have the right shape
maxlen = 100
rnn_X_train = pad_sequences(rnn_X_train, padding='post', maxlen=maxlen)
rnn_X_val = pad_sequences(rnn_X_val, padding='post', maxlen=maxlen)
rnn_X_test = pad_sequences(rnn_X_test, padding='post', maxlen=maxlen)
```

Saved successfully!

```
#Making the matrix for the embedding layer
[illegible]x
[illegible]
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))

for word, i in word_index.items():
    embedding_vector = word_embeddings.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
    else:
        # If word is not in pre-trained embeddings, use random vector
        embedding_matrix[i] = np.random.normal(scale=0.6, size=(embedding_dim,))
```

```
rnn = Sequential()
rnn.add(Embedding(len(word_index) + 1, embedding_dim, input_length=maxlen,
                    weights=[embedding_matrix], trainable=False))
rnn.add(LSTM(64))
rnn.add(Dense(1, activation='sigmoid'))
```

```
rnn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
rnn.fit(rnn_X_train, train_y, epochs=10, batch_size=64, validation_data=(rnn_X_val, val_y), callbacks=[EarlyStopping(patience=3)]
```

```
Epoch 1/10
1681/1681 [==============================] - 221s 130ms/step - loss: 0.6140 - accuracy: 0.6410 - val_loss: 0.5078 - val_accu
Epoch 2/10
1681/1681 [==============================] - 217s 129ms/step - loss: 0.4715 - accuracy: 0.7708 - val_loss: 0.4659 - val_accu
Epoch 3/10
1681/1681 [==============================] - 197s 117ms/step - loss: 0.4271 - accuracy: 0.8010 - val_loss: 0.4165 - val_accu
Epoch 4/10
1681/1681 [==============================] - 217s 129ms/step - loss: 0.4031 - accuracy: 0.8145 - val_loss: 0.4004 - val_accu
Epoch 5/10
1681/1681 [==============================] - 217s 129ms/step - loss: 0.3849 - accuracy: 0.8247 - val_loss: 0.4354 - val_accu
Epoch 6/10
1681/1681 [==============================] - 215s 128ms/step - loss: 0.3700 - accuracy: 0.8330 - val_loss: 0.3843 - val_accu
Epoch 7/10
1681/1681 [==============================] - 214s 127ms/step - loss: 0.3577 - accuracy: 0.8399 - val_loss: 0.3841 - val_accu
```

```
Epoch 8/10
1681/1681 [==============================] - 216s 129ms/step - loss: 0.3443 - accuracy: 0.8467 - val_loss: 0.3903 - val_accu
Epoch 9/10
1681/1681 [==============================] - 217s 129ms/step - loss: 0.3353 - accuracy: 0.8518 - val_loss: 0.3876 - val_accu
Epoch 10/10
1681/1681 [==============================] - 217s 129ms/step - loss: 0.3243 - accuracy: 0.8573 - val_loss: 0.3980 - val_accu
<keras.callbacks.History at 0x7f0720405310>
```

```
y_pred = rnn.predict(rnn_X_test)
y_pred = (y_pred > 0.5).astype('int32')

misclassified_samples = 0
for i in range(len(y_pred)):
    if y_pred[i] != test_y[i]:
        original_sentence = tokenizer.sequences_to_texts([rnn_X_test[i]])[0]
        actual_label = test_y[i]
        predicted_label = y_pred[i][0]
        print("Sentence: ", original_sentence)
        print("Actual Label: ", actual_label)
        print("Predicted Label: ", predicted_label)
        print(" ")
        misclassified_samples += 1
    if misclassified_samples >= 5:
        break

from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Accuracy: %.3f' % accuracy_score(test_y, y_pred))

print('Precision: %.3f' % precision_score(test_y, y_pred))

print('Recall: %.3f' % recall_score(test_y, y_pred))

print('F1: %.3f' % f1_score(test_y, y_pred))
```

```
1121/1121 [==============================] - 25s 22ms/step
Sentence:  how do you make a plumber cry you kill his family
Actual Label:  0
Predicted Label:  1

Sentence:  how to play these niggas and bitches that be snakes
Actual Label:  1
Predicted Label:  0

                   at position to give out orders dictate and judge exactly none none whatsoever you are only makin
Saved successfully!


Sentence:  views on wikipedia especially your foul slimy racist view that manchus are ethnically chinese you have proven you
Actual Label:  1
Predicted Label:  0

Sentence:  she got arrested for domestic violence against him ugh bitch
Actual Label:  0
Predicted Label:  1

Accuracy: 0.829
Precision: 0.806
Recall: 0.809
F1: 0.808
```

Combined CNN-LSTM

We wanted to combine the CNN and RNN as we believe it will capture both short-distance and long-distance dependencies

```
import numpy as np
from keras.preprocessing.text import Tokenizer
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation, Conv1D, MaxPooling1D, Bidirectional
from keras.models import Model
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical
import pandas as pd
from keras.utils import pad_sequences
from sklearn.metrics import classification_report, accuracy_score
from keras.models import Sequential
from keras.layers import Flatten
```

```
train_X, val_X, train_y, val_y = train_test_split(train_X, train_y, test_size=0.25, random_state=42)


word_embeddings = {}
with open('glove.6B.100d.txt', 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        embedding = np.asarray(values[1:], dtype='float32')
        word_embeddings[word] = embedding


# Could include num_words = 500
tokenizer = Tokenizer()
#This tokenizes the text and counts the frequency of each token
tokenizer.fit_on_texts(train_X)
vocab_size = len(tokenizer.word_index) + 1
#create a vocabulary of the most frequently occurring words in the training data
combined_X_train = tokenizer.texts_to_sequences(train_X)
combined_X_val = tokenizer.texts_to_sequences(val_X)
combined_X_test = tokenizer.texts_to_sequences(test_X)


# We need to pad the sequences here so they have the right shape
maxlen = 100
combined_X_train = pad_sequences(combined_X_train, padding='post', maxlen=maxlen)
combined_X_val = pad_sequences(combined_X_val, padding='post', maxlen=maxlen)
combined_X_test = pad_sequences(combined_X_test, padding='post', maxlen=maxlen)


#Making the matrix for the embedding layer
word_index = tokenizer.word_index
embedding_dim = 100
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))

for word, i in word_index.items():
    embedding_vector = word_embeddings.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
    else:
        # If word is not in pre-trained embeddings, use random vector
        embedding_matrix[i] = np.random.normal(scale=0.6, size=(embedding_dim,))
```

Saved successfully!

```
embedding_layer = Embedding(vocab_size, embedding_dim, weights=[embedding_matrix], input_length=maxlen, trainable=False)(inputs)
conv_layer = Conv1D(filters=64, kernel_size=3, padding='valid', activation='relu')(embedding_layer)
pooling_layer = MaxPooling1D(pool_size=2)(conv_layer)

lstm_layer = Bidirectional(LSTM(64))(pooling_layer)
fc_layer = Dropout(0.5)(lstm_layer)

outputs = Dense(1, activation='sigmoid')(fc_layer)
CNNLSTM = Model(inputs=inputs, outputs=outputs)


CNNLSTM.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# I used early stopping here to prevent overfitting since this model is prone to overfitting
CNNLSTM.fit(combined_X_train, train_y, epochs=10, batch_size=128, validation_data=(combined_X_val, val_y), callbacks=[EarlyStoppi
```

```
Epoch 1/10
841/841 [==============================] - 193s 224ms/step - loss: 0.4821 - accuracy: 0.7639 - val_loss: 0.4308 - val_accura
Epoch 2/10
841/841 [==============================] - 192s 229ms/step - loss: 0.4178 - accuracy: 0.8069 - val_loss: 0.4041 - val_accura
Epoch 3/10
841/841 [==============================] - 191s 227ms/step - loss: 0.3940 - accuracy: 0.8204 - val_loss: 0.3968 - val_accura
Epoch 4/10
841/841 [==============================] - 186s 222ms/step - loss: 0.3772 - accuracy: 0.8295 - val_loss: 0.3893 - val_accura
Epoch 5/10
841/841 [==============================] - 187s 223ms/step - loss: 0.3636 - accuracy: 0.8381 - val_loss: 0.3983 - val_accura
Epoch 6/10
841/841 [==============================] - 193s 229ms/step - loss: 0.3493 - accuracy: 0.8458 - val_loss: 0.3879 - val_accura
Epoch 7/10
841/841 [==============================] - 192s 229ms/step - loss: 0.3388 - accuracy: 0.8507 - val_loss: 0.4032 - val_accura
Epoch 8/10
841/841 [==============================] - 192s 229ms/step - loss: 0.3307 - accuracy: 0.8547 - val_loss: 0.3925 - val_accura
Epoch 9/10
841/841 [==============================] - 192s 228ms/step - loss: 0.3208 - accuracy: 0.8602 - val_loss: 0.3930 - val_accura
<keras.callbacks.History at 0x7f071d7b8910>
```

```
y_pred = CNNLSTM.predict(combined_X_test)
y_pred = (y_pred > 0.5).astype('int32')

misclassified_samples = 0
for i in range(len(y_pred)):
    if y_pred[i] != test_y[i]:
        original_sentence = tokenizer.sequences_to_texts([combined_X_test[i]])[0]
        actual_label = test_y[i]
        predicted_label = y_pred[i][0]
        print("Sentence: ", original_sentence)
        print("Actual Label: ", actual_label)
        print("Predicted Label: ", predicted_label)
        print(" ")
        misclassified_samples += 1
    if misclassified_samples >= 5:
        break

from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Accuracy: %.3f' % accuracy_score(test_y, y_pred))

print('Precision: %.3f' % precision_score(test_y, y_pred))

print('Recall: %.3f' % recall_score(test_y, y_pred))

print('F1: %.3f' % f1_score(test_y, y_pred))
```

```
1121/1121 [==============================] - 24s 21ms/step
Sentence:  thanks to covid the olympics are cancelled and these were the first olympics where men could compete as women bec
Actual Label:  1
Predicted Label:  0

Sentence:  how do you make a plumber cry you kill his family
Actual Label:  0
Predicted Label:  1

Sentence:  how to play these niggas and bitches that be snakes
Actual Label:  1
Predicted Label:  0

Sentence:  and you are in what position to give out orders dictate and judge exactly none none whatsoever you are only makin
Actual Label:  1
Predicted Label:  0

                                e
```

Saved successfully!

```
Accuracy: 0.825
Precision: 0.791
Recall: 0.820
F1: 0.806
```

Outside Testing

HSD Dataset

```
import random
from sklearn.feature_extraction.text import TfidfVectorizer


outside_data = pd.read_csv('merged_hate.csv',engine='python')
test_X = outside_data['contents'].values
new_test_y = outside_data['label'].values
```

Naive Bayes

```
# Fit vectorizer on testing data
nb_new_test_X = vectorizer.transform(test_X)


# Evaluate the model
y_pred = naive.predict(nb_new_test_X)
y_pred = (y_pred > 0.5).astype('int32')
```

```
print('Accuracy: %.3f' % accuracy_score(new_test_y, y_pred))

print('Precision: %.3f' % precision_score(new_test_y, y_pred))

print('Recall: %.3f' % recall_score(new_test_y, y_pred))

print('F1: %.3f' % f1_score(new_test_y, y_pred))
```

```
    Accuracy: 0.763
    Precision: 0.752
    Recall: 0.783
    F1: 0.767
```

CNN

```
cnn_X_new_test = tokenizer.texts_to_sequences(test_X)
cnn_X_new_test = pad_sequences(cnn_X_new_test, padding='post', maxlen=maxlen)

# make predictions on the test data
y_pred = cnn.predict(cnn_X_new_test)
y_pred = (y_pred > 0.5).astype('int32')

# evaluate the model's performance
print('Accuracy: %.3f' % accuracy_score(new_test_y, y_pred))

print('Precision: %.3f' % precision_score(new_test_y, y_pred))

print('Recall: %.3f' % recall_score(new_test_y, y_pred))

print('F1: %.3f' % f1_score(new_test_y, y_pred))
```

```
    75/75 [==============================] - 1s 9ms/step
    Accuracy: 0.757
    Precision: 0.785
    Recall: 0.707
    F1: 0.744
```

RNN

Saved successfully! ×

```
                                   _to_sequences(test_X)
                                   nn_X_new_test, padding='post', maxlen=maxlen)

y_pred = rnn.predict(rnn_X_new_test)
y_pred = (y_pred > 0.5).astype('int32')

from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Accuracy: %.3f' % accuracy_score(new_test_y, y_pred))

print('Precision: %.3f' % precision_score(new_test_y, y_pred))

print('Recall: %.3f' % recall_score(new_test_y, y_pred))

print('F1: %.3f' % f1_score(new_test_y, y_pred))
```

```
    75/75 [==============================] - 2s 21ms/step
    Accuracy: 0.793
    Precision: 0.805
    Recall: 0.773
    F1: 0.788
```

Combined CNN-LSTM

```
combined_X_test = tokenizer.texts_to_sequences(test_X)
combined_X_test = pad_sequences(combined_X_test, padding='post', maxlen=maxlen)

y_pred = CNNLSTM.predict(combined_X_test)
y_pred = (y_pred > 0.5).astype('int32')

from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Accuracy: %.3f' % accuracy_score(new_test_y, y_pred))

print('Precision: %.3f' % precision_score(new_test_y, y_pred))
print('Recall: %.3f' % recall_score(new_test_y, y_pred))

print('F1: %.3f' % f1_score(new_test_y, y_pred))
    75/75 [==============================] - 1s 18ms/step
    Accuracy: 0.787
    Precision: 0.791
    Recall: 0.780
    F1: 0.786
```

Davidson Hate Speech Dataset

```
davidson_data = pd.read_csv('davidson_data.csv',engine='python')
test_X = davidson_data['tweet'].values
new_test_y = davidson_data['class'].values
```

Naive Bayes

```
# Fit vectorizer on testing data
nb_new_test_X = vectorizer.transform(test_X)

# Evaluate the model
y_pred = naive.predict(nb_new_test_X)
y_pred = (y_pred > 0.5).astype('int32')
print('Accuracy: %.3f' % accuracy_score(new_test_y, y_pred))

print('Precision: %.3f' % precision_score(new_test_y, y_pred))

print('Recall: %.3f' % recall_score(new_test_y, y_pred))

print('F1: %.3f' % f1_score(new_test_y, y_pred))

    Accuracy: 0.801
    Precision: 0.753
    Recall: 0.897
    F1: 0.819
```

CNN

Saved successfully!

```
_to_sequences(test_X)
cnn_X_new_test = pad_sequences(cnn_X_new_test, padding='post', maxlen=maxlen)

# make predictions on the test data
y_pred = cnn.predict(cnn_X_new_test)
y_pred = (y_pred > 0.5).astype('int32')

# evaluate the model's performance
print('Accuracy: %.3f' % accuracy_score(new_test_y, y_pred))

print('Precision: %.3f' % precision_score(new_test_y, y_pred))

print('Recall: %.3f' % recall_score(new_test_y, y_pred))

print('F1: %.3f' % f1_score(new_test_y, y_pred))

    90/90 [==============================] - 0s 5ms/step
    Accuracy: 0.760
    Precision: 0.757
    Recall: 0.766
    F1: 0.761
```

RNN

```
rnn_X_new_test = tokenizer.texts_to_sequences(test_X)
rnn_X_new_test = pad_sequences(rnn_X_new_test, padding='post', maxlen=maxlen)

y_pred = rnn.predict(rnn_X_new_test)
y_pred = (y_pred > 0.5).astype('int32')

from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Accuracy: %.3f' % accuracy_score(new_test_y, y_pred))
print('Precision: %.3f' % precision_score(new_test_y, y_pred))

print('Recall: %.3f' % recall_score(new_test_y, y_pred))

print('F1: %.3f' % f1_score(new_test_y, y_pred))
```

```
    90/90 [==============================] - 2s 21ms/step
    Accuracy: 0.792
    Precision: 0.781
    Recall: 0.811
    F1: 0.796
```

Combined CNN-LSTM

```
combined_X_test = tokenizer.texts_to_sequences(test_X)
combined_X_test = pad_sequences(combined_X_test, padding='post', maxlen=maxlen)

y_pred = CNNLSTM.predict(combined_X_test)
y_pred = (y_pred > 0.5).astype('int32')

from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
print('Accuracy: %.3f' % accuracy_score(new_test_y, y_pred))

print('Precision: %.3f' % precision_score(new_test_y, y_pred))

print('Recall: %.3f' % recall_score(new_test_y, y_pred))

print('F1: %.3f' % f1_score(new_test_y, y_pred))
```

```
    90/90 [==============================] - 2s 20ms/step
    Accuracy: 0.790
    Precision: 0.770
    Recall: 0.827
    F1: 0.797
```

Saved successfully! ×

Colab paid products - Cancel contracts here

✓ 2s completed at 9:01 PM ● ×