College of Engineering & Applied Science
UNIVERSITY OF COLORADO **BOULDER**

University of Colorado Boulder **Amulya Ambati**
CSCI 5922: Neural Nets and Deep Learning **Lab3**
Prof. Danna Gurari **March 13, 2024**

Q1

Introduction

Text classification, particularly in the domain of sentiment analysis, plays a crucial role in understanding public opinion and consumer sentiment towards various products, services, or experiences. Among the myriad of applications, analyzing movie reviews stands out as a prominent area of interest. With the proliferation of online platforms and social media, individuals express their opinions on movies through textual reviews, ratings, and comments, making movie review datasets a valuable resource for sentiment analysis.

In this context, the task of text classification involves categorizing movie reviews into predefined classes, typically representing sentiments such as positive, negative, or neutral. This classification enables filmmakers, production studios, and distributors to gauge audience reactions, identify trends, and make informed decisions regarding marketing strategies, content creation, and audience engagement.

The Dataset is downloaded from Kaggle resource(Link to the dataset). A Small snippet of the dataset is as follows.

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1

Train-Test Set

Performed a train-test split of 75-25 % on given train data. The size of the data post-split is as follows:

Train	30,000
Test	10,000

Data Processing

1. Text data usually contains punctuations and special characters which are generally not helpful in performing tasks like sentiment analysis. Hence as a first step of data preprocessing punctuations were removed.

2. Text usually contains lots of filler words or stop words which will be common across the whole English language and usually do not contain content. Hence removing this using a special package 'nltk' which has a set of all the common stop words present in English.

3. Post removing stop words, textvectorization was done. Since we cannot pass the words/tokens directly to the model as the model can only consume int/float values, words/tokens needs to be vectorized. Hence performing sparse way of encoding using textvectorization.

4. The target variable is also one-hot encoded with 2 unique classes, resulting in a 1d tensor of 2.

Modeling

Three different experiments were performed by using various algorithms of RNN, LSTM, and, GRU to find out how the performance varies across all three models.
RNNs are a class of neural networks designed for processing sequential data by maintaining an internal state (hidden state) that evolves as the network processes each element of the sequence. However, for longer sequences they undergo the issue of vanishing or exploding gradients. Hence, LSTMs and GRUs were introduced to tackle this problem. LSTM networks contain memory cells, which maintain a cell state and have three gating mechanisms: input gate, forget gate, and output gate. The input gate controls the flow of information into the memory cell, the forget gate regulates the retention of information in the cell state, and the output gate governs the flow of information from the cell state to the output.GRU simplifies the architecture of LSTM by combining the forget and input gates into a single update gate and merging the cell state and hidden state.
To compare all the 3, we are just varying a single layer in a neural network by keeping everything else constant.
Some of the hyperparameters that were used across all the 3 models while training are

· loss: cross-entropy loss

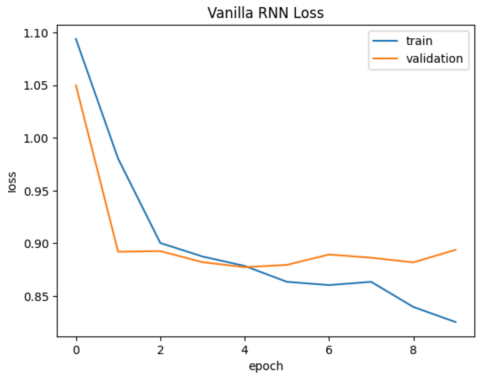
· batch size: 256

· epochs: 10

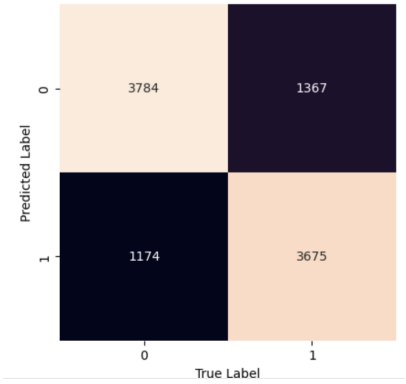
1) Model 1 - RNN

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 100)         100000

 simple_rnn (SimpleRNN)      (None, 32)                4256

 dense (Dense)               (None, 32)                1056

 dense_1 (Dense)             (None, 2)                 66

=================================================================
Total params: 105378 (411.63 KB)
Trainable params: 105378 (411.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The loss/learning curve for the above model is as follows:

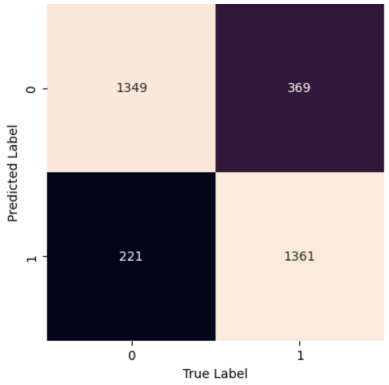


Post training the best model was considered and was tested upon.
The confusion matrix on the test dataset is as follows:

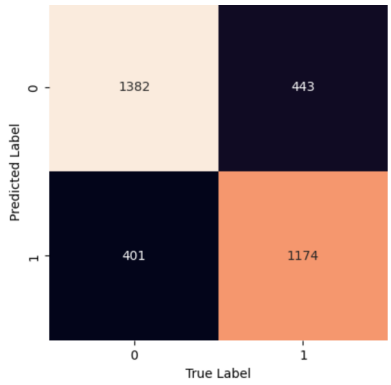


Another interesting exercise of checking model performance based on the length of test sentences was done, because rnn's can consume varying no.of tokens/words and gradients keep propagating accordingly. To perform this, the test set was divided into 3 sub-sets by sorting data based on length and dividing into 3 equal parts and tested on the above trained model.

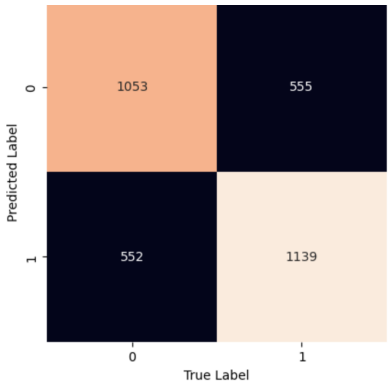
Test performance on sub test-data that has short length



Test performance on sub testdata that has medium length



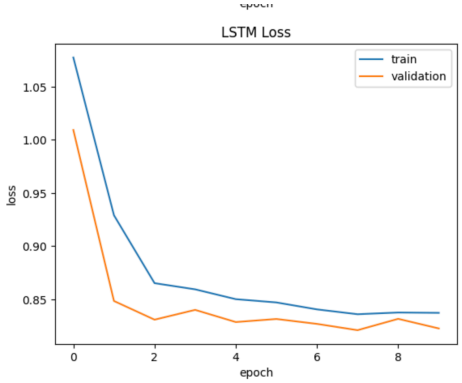
Test performance on sub test-data that has long length



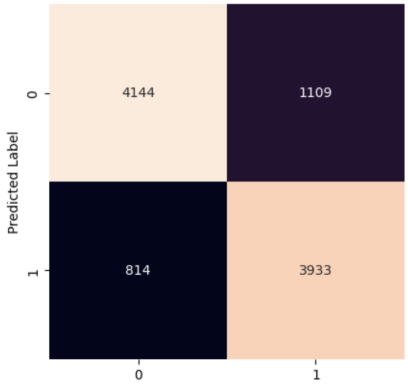
2) Model 2 - LSTM

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 100)         100000

 lstm (LSTM)                 (None, 32)                17024

 dense_4 (Dense)             (None, 32)                1056

 dense_5 (Dense)             (None, 2)                 66

=================================================================
Total params: 118146 (461.51 KB)
Trainable params: 118146 (461.51 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

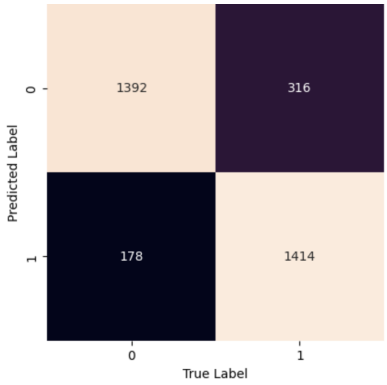
The loss/learning curve for the above model is as follows:



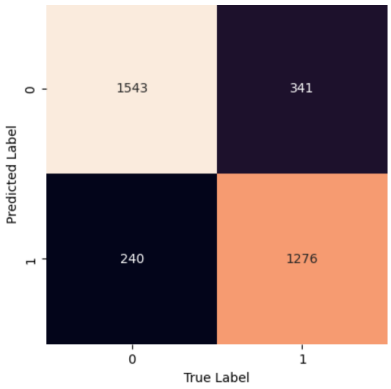
Post training the best model was considered and was tested upon.
The confusion matrix on the test dataset is as follows:



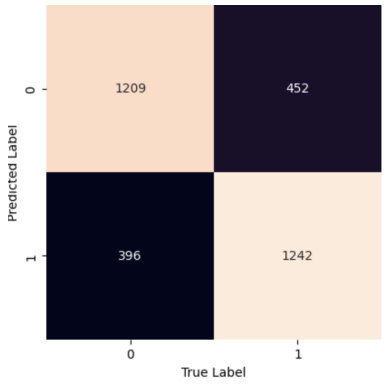
Another interesting exercise of checking model performance based on the length of test sentences was done, because rnn's can consume varying no.of tokens/words and gradients keep propagating accordingly. To perform this, the test set was divided into 3 sub-sets by sorting data based on length and dividing into 3 equal parts and tested on the above trained model.
Test performance on sub test-data that has short length



Test performance on sub test-data that has medium length



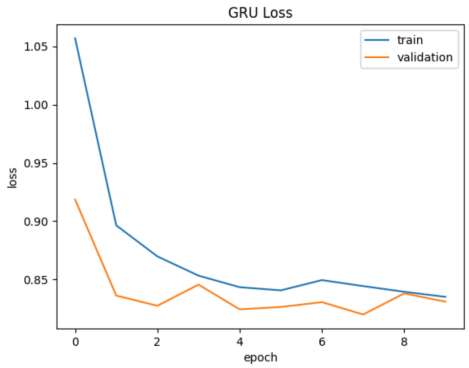
Test performance on sub test-data that has long length



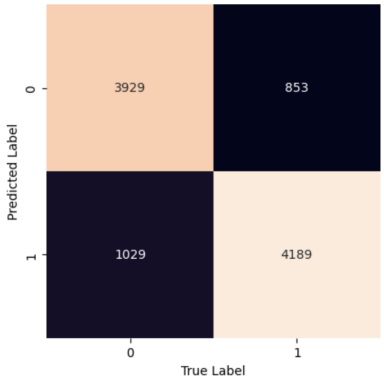
3) Model 3 - GRU

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 100)         100000

 gru (GRU)                   (None, 32)                12864

 dense_2 (Dense)             (None, 32)                1056

 dense_3 (Dense)             (None, 2)                 66

=================================================================
Total params: 113986 (445.26 KB)
Trainable params: 113986 (445.26 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

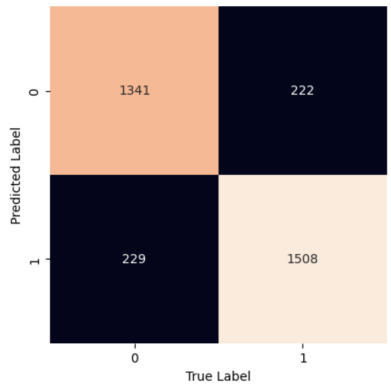
The loss/learning curve for the above model is as follows:



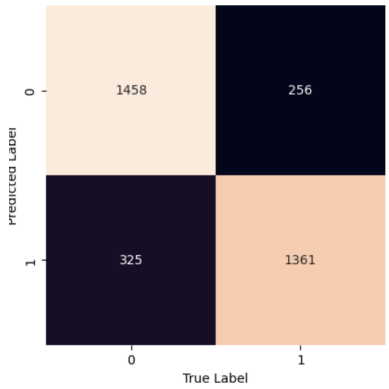
Post training the best model was considered and was tested upon.
The confusion matrix on the test dataset is as follows:



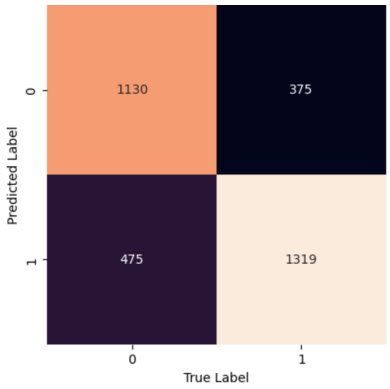
Another interesting exercise of checking model performance based on the length of test sentences was done, because rnn's can consume varying no.of tokens/words and gradients keep propagating accordingly. To perform this, the test set was divided into 3 sub-sets by sorting data based on length and dividing into 3 equal parts and tested on the above trained model.
Test performance on sub test-data that has short length



Test performance on sub test-data that has medium length



Test performance on sub test-data that has long length



Post comparing all these models, the model with the best test accuracy is selected as the final model, which is GRU.
The performance of the final model is as follows:

Data	Accuracy	loss
Train	90.04	0.85
Test	81.1	0.85

Observations/Patterns

Model	Training time(in s)	Train accuracy	Test accuracy	Train loss	Test loss
RNN	96	93.5	74.5	0.83	0.91
LSTM	353	89.63	80.76	0.85	0.84
GRU	385	90.46	81.1	0.85	0.85

As expected LSTM and GRU are taking higher training time than RNN, because of additional parameters and extra computations that happen inside the gates in LSTM and GRU. We can also see that LSTM and GRU are performing better than RNN on the whole test data. This behaviour is also expected, because RNN's usually give lower performance because of exploding/ vanishing gradients problem. Finally, GRU has the best accuracy among LSTM, RNN, and, GRU.

Model	Test sentence length	accuracy
RNN	short	82.2
RNN	medium	75.1
RNN	long	66.4
LSTM	short	85
LSTM	medium	82.9
LSTM	long	74.3
GRU	short	86.3
GRU	medium	82.9
GRU	long	74.2

For all the 3 models of RNN, LSTM, and GRU, the performance on sub test-sets based on sentence length is short>medium>long. However, if we look at the numbers in detail, we can see that the difference in performance between long and short is higher in RNN than compared to GRU and LSTM. We can see that for short sentences the performance is mostly same across RNN, LSTM, and GRU, the major difference is observed for long sentences. This can again be attributed to the fact that RNN's cannot handle sentences with longer length because of vanishing gradients.

Q2

The dataset and the train-test split is the same as Q1.

Data Processing

1. Text data usually contains punctuations and special characters which are generally not helpful in performing tasks like sentiment analysis. Hence as a first step of data preprocessing punctuations were removed.

2. Text usually contains lots of filler words or stop words which will be common across the whole English language and usually do not contain content. Hence removing this using a special package 'nltk' which has a set of all the common stop words present in English.

3. Post removing stop words, textvectorization was done. Since we cannot pass the words/tokens directly to the model as the model can only consume int/float values, words/tokens needs to be vectorized. For the word embeddings, we have considered dense representation using Glove50d and Glove100d which are pretrained embeddings and set to non-trainable.

4. The target variable is also one-hot encoded with 2 unique classes, resulting in a 1d tensor of 2.

Modeling

Here we are considering the best model from Q1, which is GRU model. And we are comparing the performance between Glove50d embeddings and Glove100d embeddings.

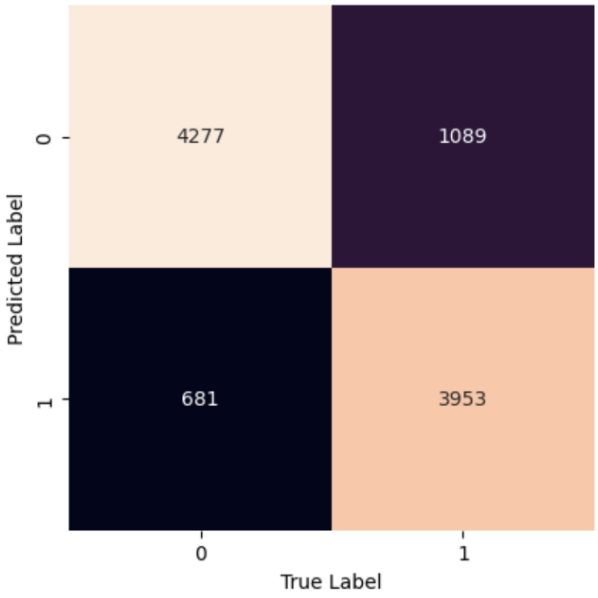
Some of the hyperparameters that were used across all the 3 models while training are

- loss: cross-entropy loss
- batch size: 256
- epochs: 10

Model architecture & parameters while considering Glove50d embeddings:

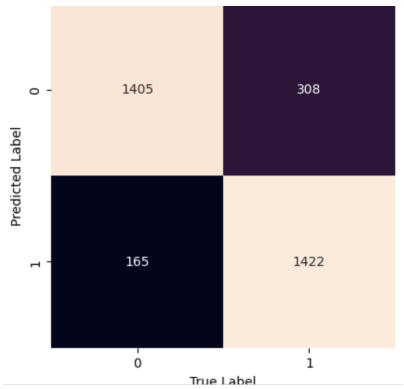
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 50)          500000

 gru_1 (GRU)                 (None, 32)                8064

 dense_2 (Dense)             (None, 32)                1056

 dense_3 (Dense)             (None, 2)                 66

=================================================================
Total params: 509186 (1.94 MB)
Trainable params: 9186 (35.88 KB)
Non-trainable params: 500000 (1.91 MB)
_________________________________________________________________
```

The Confusion matrix on the whole testset is as follows

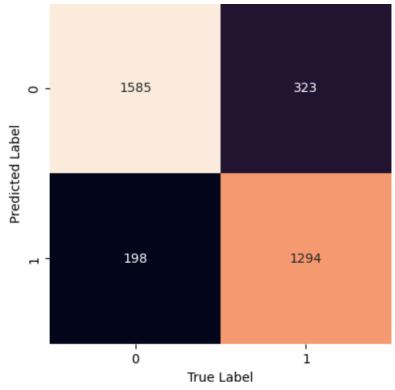


Another interesting exercise of checking model performance based on the length of test sentences was done, because gru's can consume varying no.of tokens/words and gradients keep propagating accordingly. To perform this, the test set was divided into 3 sub-sets by sorting data based on length and dividing into 3 equal parts and tested on the above trained model.

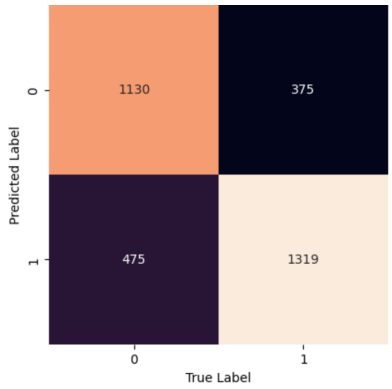
Test performance on sub test-data that has short length



Test performance on sub test-data that has medium length



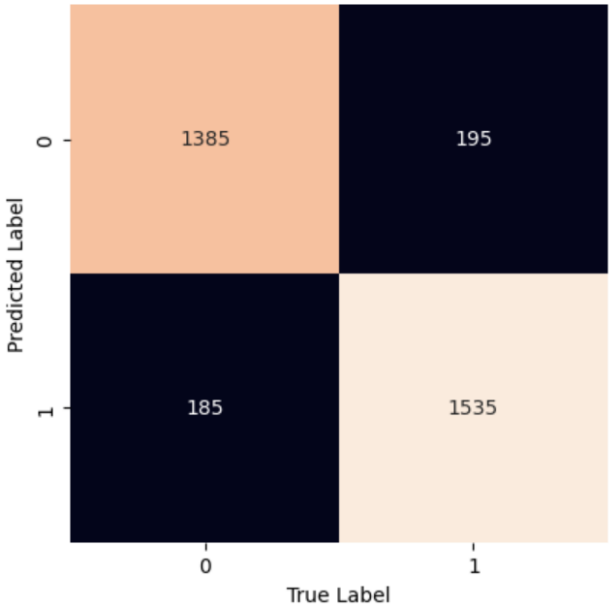
Test performance on sub test-data that has long length



Model architecture & parameters while considering Glove100d embeddings:

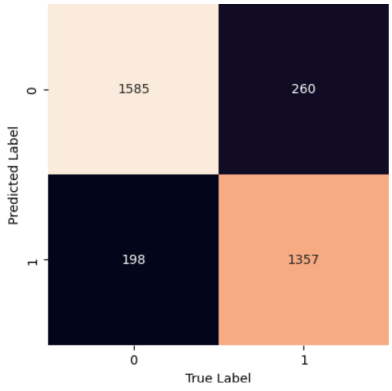
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 100)         1000000

 gru_2 (GRU)                 (None, 32)                12864

 dense_4 (Dense)             (None, 32)                1056

 dense_5 (Dense)             (None, 2)                 66

=================================================================
Total params: 1013986 (3.87 MB)
Trainable params: 13986 (54.63 KB)
Non-trainable params: 1000000 (3.81 MB)
_________________________________________________________________
```

The Confusion matrix on the whole testset is as follows

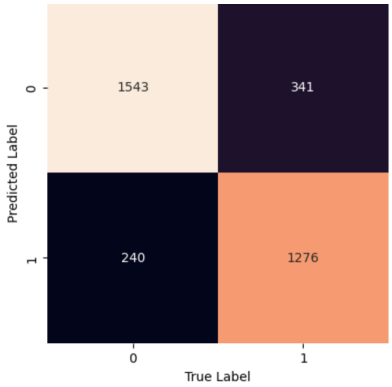


Another interesting exercise of checking model performance based on the length of test sentences was done, because gru's can consume varying no.of tokens/words and gradients keep propagating accordingly. To perform this, the test set was divided into 3 sub-sets by sorting data based on length and dividing into 3 equal parts and tested on the above trained model.

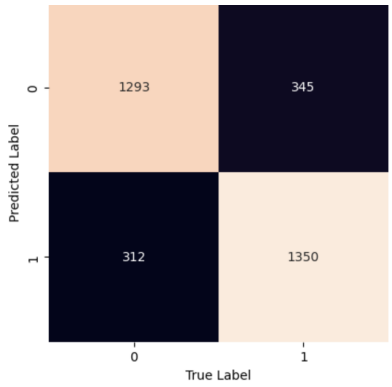
Test performance on sub test-data that has short length



Test performance on sub test-data that has medium length



Test performance on sub test-data that has long length



Model	Time taken for training(in s)	Train Accuracy	Test Accuracy
Glove50	150	83.4	82.4
Glove100	200	86.6	85.04

Observations/Patterns

Since Glove100d representation has more information and efficient than Glove50d, we are seeing better performance for gru model with Glove50 embeddings when compared to gru model with Glove100 embeddings. Also, time taken for Glove100d is little higher than Glove50d model because, Glove100d has little higher number of parameters.

Model	Test sentence length	accuracy
Glove50	short	85.6
Glove50	medium	84.6
Glove50	long	76.4
Glove100	short	88.4
Glove100	medium	86.5
Glove100	long	80

The difference is performance based on sentence length is same as what we observed in Q1, short>medium>long. This is because, shorter sentence have lower gradients to backprop and thereby more information retrival. Here aswell we are seeing Glove100 performance being greater than Glove50 across all the different sub testsets.

One more important comparison that can be made is model performance using sparse representation vs dense representation. Ideally, we would expect better performance for dense representation but here for sparse representation we are allowing updating the representation, while for dense representation using Glove we are fixing the embeddings, that might be the reason for lower performance using dense when compared to sparse representation.

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import nltk
from nltk.corpus import stopwords
import re
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, GRU, Dense
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
from keras.initializers import Constant
import time
```

In [2]:
```
data = pd.read_csv('/content/movie.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1

In [4]:
```
data['label'].value_counts(normalize=True)
```

Out[4]:
```
0    0.500475
1    0.499525
Name: label, dtype: float64
```

In [5]:
```
nltk.download('stopwords')
# We filter out the english language stopwrds

stop_words = stopwords.words('english')
print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [6]:
```
remove_chars = "@S+|https?:S+|http?:S|[^A-Za-z0-9]+"
```

In [7]:
```
def tokenize_text(text):
  # Text passed to the regex equatio
  text = re.sub(remove_chars, ' ', str(text).lower()).strip()
  # Empty list created to store final tokens
  tokens = []
  for token in text.split():
    # check if the token is a stop word or not
    if token not in stop_words:
        tokens.append(token)
  return " ".join(tokens)
```

In [8]:
```
data['tokenized_text'] = data['text'].apply(lambda x: tokenize_text(x))
```

In [9]:
```
data.head()
```

Out[9]:

	text	label	tokenized_text
0	I grew up (b. 1965) watching and loving the Th...	0	grew b 1965 watching loving thunderbirds mates...
1	When I put this movie in my DVD player, and sa...	0	put movie dvd player sat coke chips expectatio...
2	Why do people who do not know what a particula...	0	people know particular time past like feel nee...
3	Even though I have great interest in Biblical ...	0	even though great interest biblical movies bor...
4	Im a die hard Dads Army fan and nothing will e...	1	im die hard dads army fan nothing ever change ...

In [10]:
```
train,test = train_test_split(data, test_size=0.25, random_state=42)
```

In [11]:
```
train.reset_index(inplace=True, drop=True)
test.reset_index(inplace=True, drop=True)
```

In [12]:
```
test.head()
```

Out[12]:

	text	label	tokenized_text
0	The central theme in this movie seems to be co...	0	central theme movie seems confusion relationsh...
1	An excellent example of "cowboy noir", as it's...	1	excellent example cowboy noir called unemploye...
2	The ending made my heart jump up into my throa...	0	ending made heart jump throat proceeded leave ...
3	Only the chosen ones will appreciate the quali...	1	chosen ones appreciate quality story character...
4	This is a really funny film, especially the se...	1	really funny film especially second third four...

In [13]:
```
train.shape, test.shape
```

Out[13]:
```
((30000, 3), (10000, 3))
```

In [14]:
```
# tfidf = TfidfVectorizer(max_features=1000)
# tfidf.fit(train['tokenized_text'])
```

In [15]:
```
from keras.layers import TextVectorization

vectorizer = TextVectorization(max_tokens=1000, output_sequence_length=100)
vectorizer.adapt(train['tokenized_text'])
voc = vectorizer.get_vocabulary()
```

In [16]:
```
len(voc)
```

Out[16]:
```
1000
```

In [17]:
```
x_train = vectorizer(np.array([[s] for s in train['tokenized_text']])).numpy()
x_test  = vectorizer(np.array([[s] for s in test['tokenized_text']])).numpy()
```

In [18]:
```
enc = OneHotEncoder(handle_unknown='ignore')
enc.fit(np.array(train['label'].tolist()).reshape(-1,1))
y_train = enc.transform(np.array(train['label'].tolist()).reshape(-1,1)).toarray()
y_test = enc.transform(np.array(test['label'].tolist()).reshape(-1,1)).toarray()
```

In [19]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[19]:
```
((30000, 100), (30000, 2), (10000, 100), (10000, 2))
```

In [20]:
```
def model(input_dim,model_type,num_classes):
  model = Sequential()
  model.add(Embedding(input_dim = input_dim , output_dim = 100))
  if model_type == 'vanilla':
    model.add(SimpleRNN(32))
  elif model_type == 'GRU':
```

```
    model.add(GRU(32))
  else:
    model.add(LSTM(32))
  model.add(Dense(32,activation='relu'))
  model.add(Dense(num_classes,activation='softmax'))
  model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['acc
  return model
```

In [21]:
```
len(voc)
```

Out[21]:
```
1000
```

In [22]:
```
rnn = model(len(voc),'vanilla',2)
```

In [23]:
```
rnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 100)         100000

 simple_rnn (SimpleRNN)      (None, 32)                4256

 dense (Dense)               (None, 32)                1056

 dense_1 (Dense)             (None, 2)                 66

=================================================================
Total params: 105378 (411.63 KB)
Trainable params: 105378 (411.63 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [24]:
```
start_time = time.time()
rnn = model(len(voc),'vanilla',2)
rnn_fitted = rnn.fit(x_train, y_train, epochs=10, validation_split=0.2,batch_size=256)
print('time taken to train the model:', time.time()-start_time)
```

```
Epoch 1/10
94/94 [==============================] - 8s 68ms/step - loss: 0.6818 - accuracy: 0.54
99 - val_loss: 0.5994 - val_accuracy: 0.7175
Epoch 2/10
94/94 [==============================] - 7s 75ms/step - loss: 0.4650 - accuracy: 0.79
60 - val_loss: 0.4117 - val_accuracy: 0.8220
Epoch 3/10
94/94 [==============================] - 6s 63ms/step - loss: 0.3728 - accuracy: 0.84
24 - val_loss: 0.4136 - val_accuracy: 0.8128
Epoch 4/10
94/94 [==============================] - 7s 75ms/step - loss: 0.3432 - accuracy: 0.85
73 - val_loss: 0.4081 - val_accuracy: 0.8187
Epoch 5/10
94/94 [==============================] - 6s 59ms/step - loss: 0.3230 - accuracy: 0.86
43 - val_loss: 0.4154 - val_accuracy: 0.8198
Epoch 6/10
94/94 [==============================] - 7s 76ms/step - loss: 0.2934 - accuracy: 0.87
92 - val_loss: 0.4541 - val_accuracy: 0.8267
Epoch 7/10
94/94 [==============================] - 7s 74ms/step - loss: 0.2597 - accuracy: 0.89
79 - val_loss: 0.4649 - val_accuracy: 0.8185
Epoch 8/10
94/94 [==============================] - 7s 76ms/step - loss: 0.2313 - accuracy: 0.91
10 - val_loss: 0.4874 - val_accuracy: 0.7993
Epoch 9/10
94/94 [==============================] - 7s 70ms/step - loss: 0.2098 - accuracy: 0.92
20 - val_loss: 0.5422 - val_accuracy: 0.8130
Epoch 10/10
94/94 [==============================] - 8s 82ms/step - loss: 0.1801 - accuracy: 0.93
85 - val_loss: 0.5730 - val_accuracy: 0.7992
time taken to train the model: 69.48656916618347
```

In [25]:
```
gru = model(len(voc),'GRU',2)
gru.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 100)         100000

 gru (GRU)                   (None, 32)                12864

 dense_4 (Dense)             (None, 32)                1056

 dense_5 (Dense)             (None, 2)                 66

=================================================================
Total params: 113986 (445.26 KB)
Trainable params: 113986 (445.26 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [26]:
```
start_time = time.time()
gru = model(len(voc),'GRU',2)
gru_fitted =gru.fit(x_train, y_train, epochs=10, validation_split=0.2)
print('time taken to train the model:', time.time()-start_time)
```

```
Epoch 1/10
750/750 [==============================] - 33s 41ms/step - loss: 0.6727 - accuracy:
0.5598 - val_loss: 0.6649 - val_accuracy: 0.5727
Epoch 2/10
750/750 [==============================] - 30s 40ms/step - loss: 0.5052 - accuracy:
0.7432 - val_loss: 0.4064 - val_accuracy: 0.8178
Epoch 3/10
750/750 [==============================] - 30s 41ms/step - loss: 0.3492 - accuracy:
0.8461 - val_loss: 0.3739 - val_accuracy: 0.8338
Epoch 4/10
750/750 [==============================] - 30s 41ms/step - loss: 0.3172 - accuracy:
0.8655 - val_loss: 0.3931 - val_accuracy: 0.8250
Epoch 5/10
750/750 [==============================] - 31s 41ms/step - loss: 0.2968 - accuracy:
0.8747 - val_loss: 0.3864 - val_accuracy: 0.8288
Epoch 6/10
750/750 [==============================] - 30s 39ms/step - loss: 0.2794 - accuracy:
0.8834 - val_loss: 0.3923 - val_accuracy: 0.8255
Epoch 7/10
750/750 [==============================] - 31s 41ms/step - loss: 0.2648 - accuracy:
0.8915 - val_loss: 0.4020 - val_accuracy: 0.8268
Epoch 8/10
750/750 [==============================] - 33s 44ms/step - loss: 0.2471 - accuracy:
0.8996 - val_loss: 0.4284 - val_accuracy: 0.8230
Epoch 9/10
750/750 [==============================] - 30s 40ms/step - loss: 0.2312 - accuracy:
0.9085 - val_loss: 0.4407 - val_accuracy: 0.8213
Epoch 10/10
750/750 [==============================] - 30s 40ms/step - loss: 0.2127 - accuracy:
0.9178 - val_loss: 0.4793 - val_accuracy: 0.8173
time taken to train the model: 324.2529835700989
```

In [27]:
```
lstm = model(len(voc),'lstm',2)
lstm.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, None, 100)         100000

 lstm (LSTM)                 (None, 32)                17024

 dense_8 (Dense)             (None, 32)                1056

 dense_9 (Dense)             (None, 2)                 66

=================================================================
Total params: 118146 (461.51 KB)
Trainable params: 118146 (461.51 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

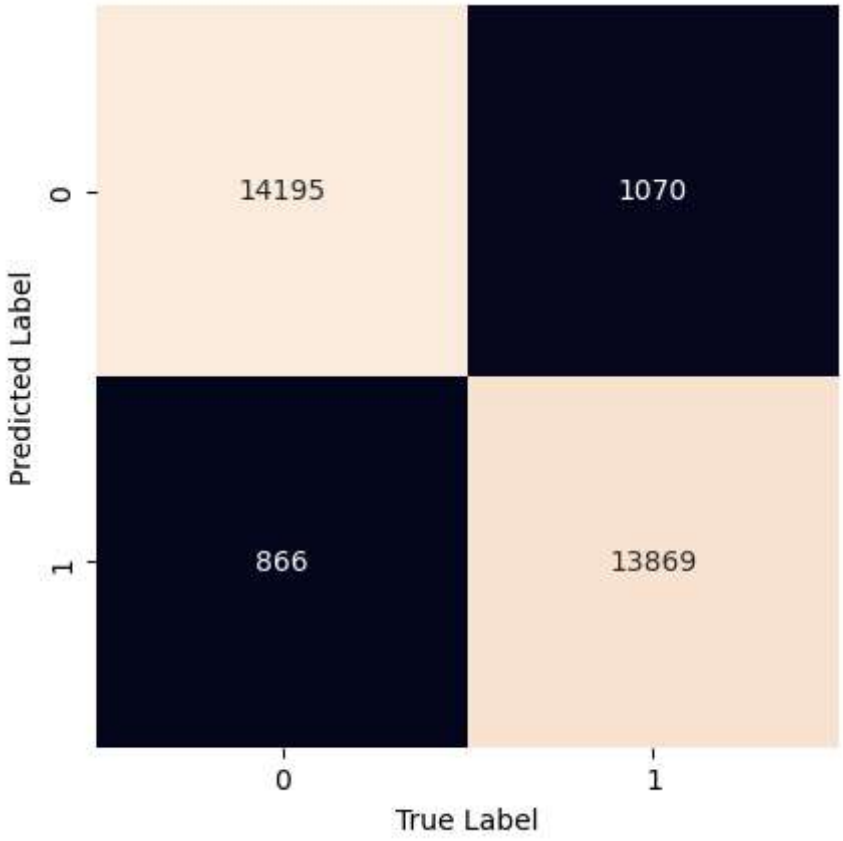
In [28]:
```
start_time = time.time()
lstm = model(len(voc),'lstm',2)
lstm_fitted=lstm.fit(x_train, y_train, epochs=10, validation_split=0.2)
print('time taken to train the model:', time.time()-start_time)
```

```
Epoch 1/10
750/750 [==============================] - 37s 47ms/step - loss: 0.5249 - accuracy:
0.7195 - val_loss: 0.4032 - val_accuracy: 0.8270
Epoch 2/10
750/750 [==============================] - 31s 42ms/step - loss: 0.3809 - accuracy:
0.8350 - val_loss: 0.4024 - val_accuracy: 0.8307
Epoch 3/10
750/750 [==============================] - 32s 43ms/step - loss: 0.3537 - accuracy:
0.8470 - val_loss: 0.3865 - val_accuracy: 0.8243
Epoch 4/10
750/750 [==============================] - 31s 41ms/step - loss: 0.3362 - accuracy:
0.8543 - val_loss: 0.3835 - val_accuracy: 0.8342
Epoch 5/10
750/750 [==============================] - 32s 42ms/step - loss: 0.3183 - accuracy:
0.8634 - val_loss: 0.4051 - val_accuracy: 0.8280
Epoch 6/10
750/750 [==============================] - 33s 44ms/step - loss: 0.3023 - accuracy:
0.8707 - val_loss: 0.4036 - val_accuracy: 0.8248
Epoch 7/10
750/750 [==============================] - 39s 52ms/step - loss: 0.2830 - accuracy:
0.8808 - val_loss: 0.4176 - val_accuracy: 0.8252
Epoch 8/10
750/750 [==============================] - 32s 42ms/step - loss: 0.2687 - accuracy:
0.8878 - val_loss: 0.4358 - val_accuracy: 0.8193
Epoch 9/10
750/750 [==============================] - 34s 46ms/step - loss: 0.2560 - accuracy:
0.8945 - val_loss: 0.4464 - val_accuracy: 0.8157
Epoch 10/10
750/750 [==============================] - 42s 56ms/step - loss: 0.2404 - accuracy:
0.9032 - val_loss: 0.4860 - val_accuracy: 0.8207
time taken to train the model: 384.3906238079071
```

In [26]:
```
def performance(model, x, y):

  loss,accuracy = model.evaluate(x,y)
  print(loss)
  print(accuracy)
  y_pred = model.predict(x)
  predicted_classes = np.argmax(y_pred, axis=1)
  true_classes = np.argmax(y, axis=1)

  conf_matrix = confusion_matrix(true_classes, predicted_classes)

  plt.figure()
  sns.heatmap(conf_matrix.T, square=True, annot=True, fmt='d', cbar=False)
  plt.xlabel("True Label")
  plt.ylabel("Predicted Label")
```

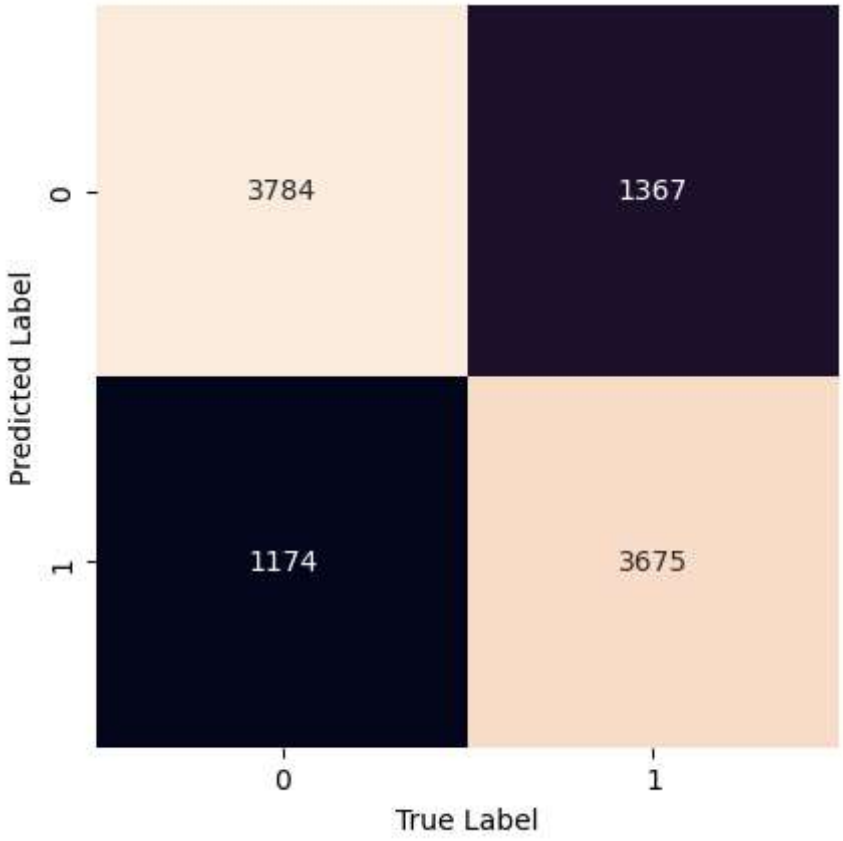
In [27]:
```
print('Confusion matrix of RNN model:')
performance(rnn,x_train,y_train)
```

```
Confusion matrix of RNN model:
938/938 [==============================] - 9s 10ms/step - loss: 0.2435 - accuracy: 0.
9355
0.2435411959886551
0.9354666471481323
938/938 [==============================] - 8s 8ms/step
```



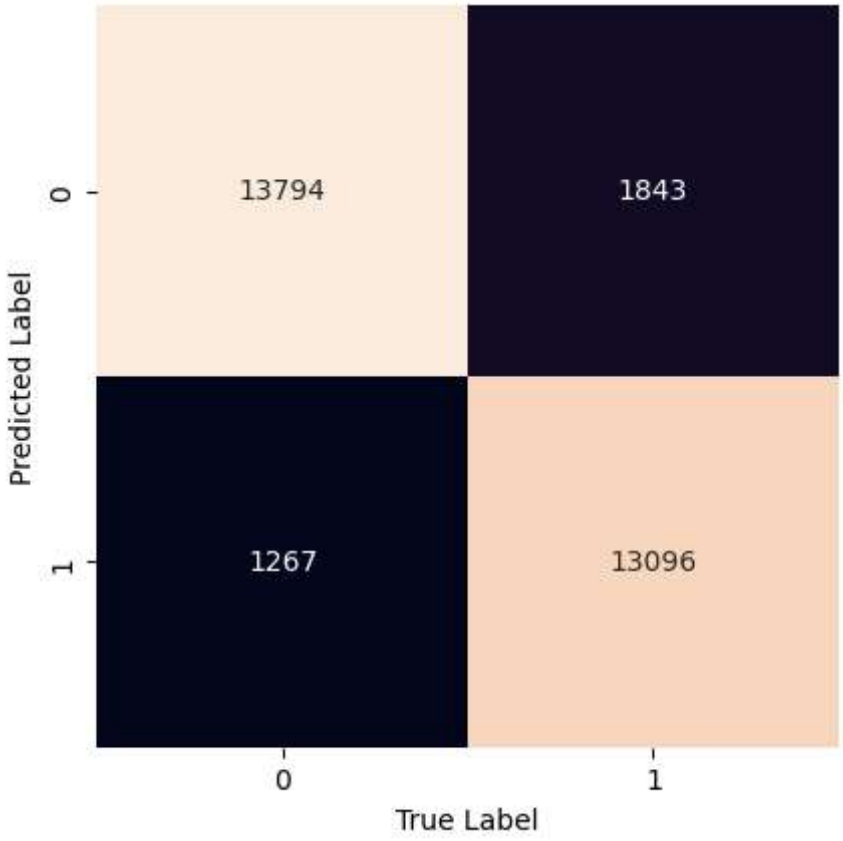
In [28]:
```
print('Confusion matrix of RNN model:')
performance(rnn,x_test,y_test)
```

```
Confusion matrix of RNN model:
313/313 [==============================] - 3s 11ms/step - loss: 0.9698 - accuracy: 0.
7459
0.9697715044021606
0.7458999752998352
313/313 [==============================] - 2s 8ms/step
```



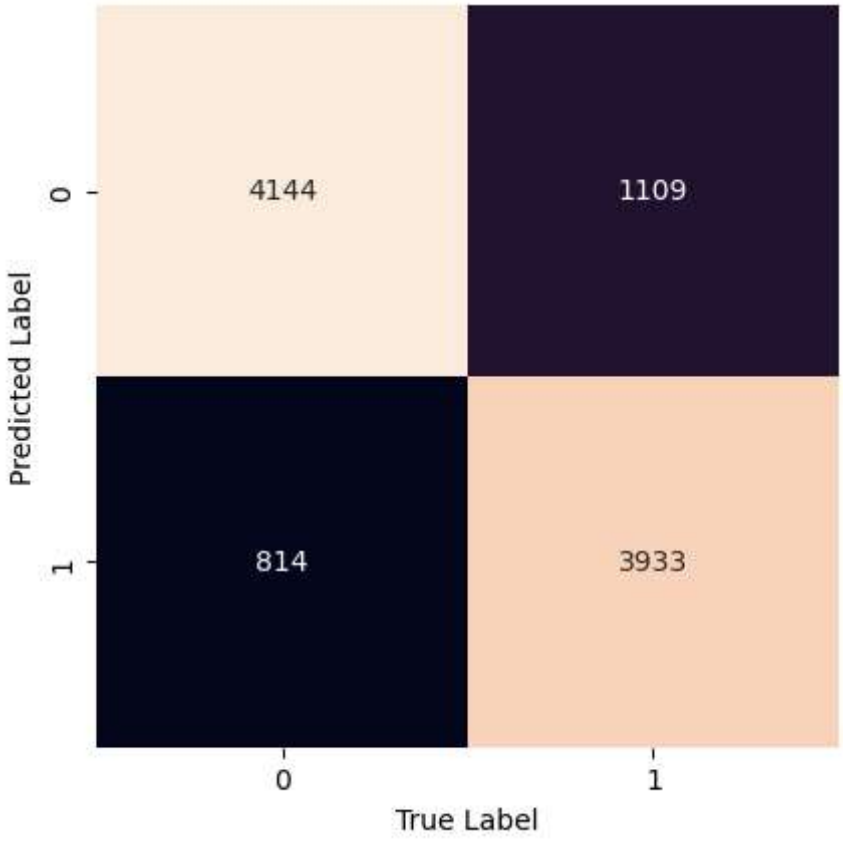
In [29]:
```
print('Confusion matrix of LSTM model:')
performance(lstm,x_train,y_train)
```

```
Confusion matrix of LSTM model:
938/938 [==============================] - 13s 14ms/step - loss: 0.2620 - accuracy:
0.8963
0.2620275616645813
0.8963333368301392
938/938 [==============================] - 14s 14ms/step
```



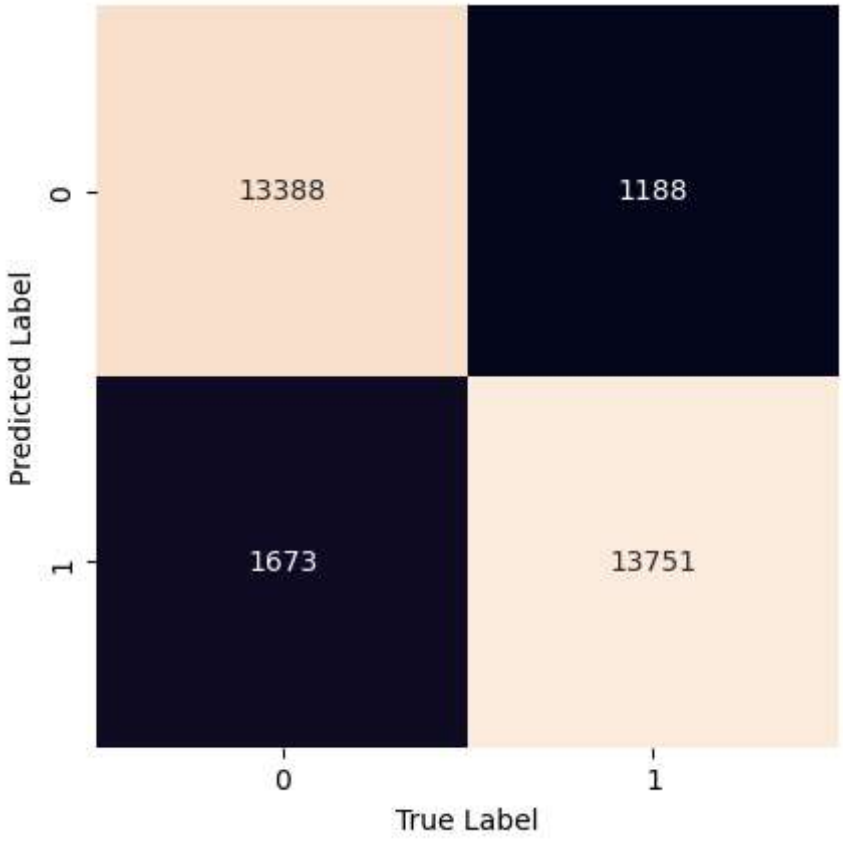
In [30]:
```
print('Confusion matrix of LSTM model:')
performance(lstm,x_test,y_test)
```

```
Confusion matrix of LSTM model:
313/313 [==============================] - 4s 13ms/step - loss: 0.4733 - accuracy: 0.
8077
0.47330960631370544
0.807699978351593
313/313 [==============================] - 4s 13ms/step
```



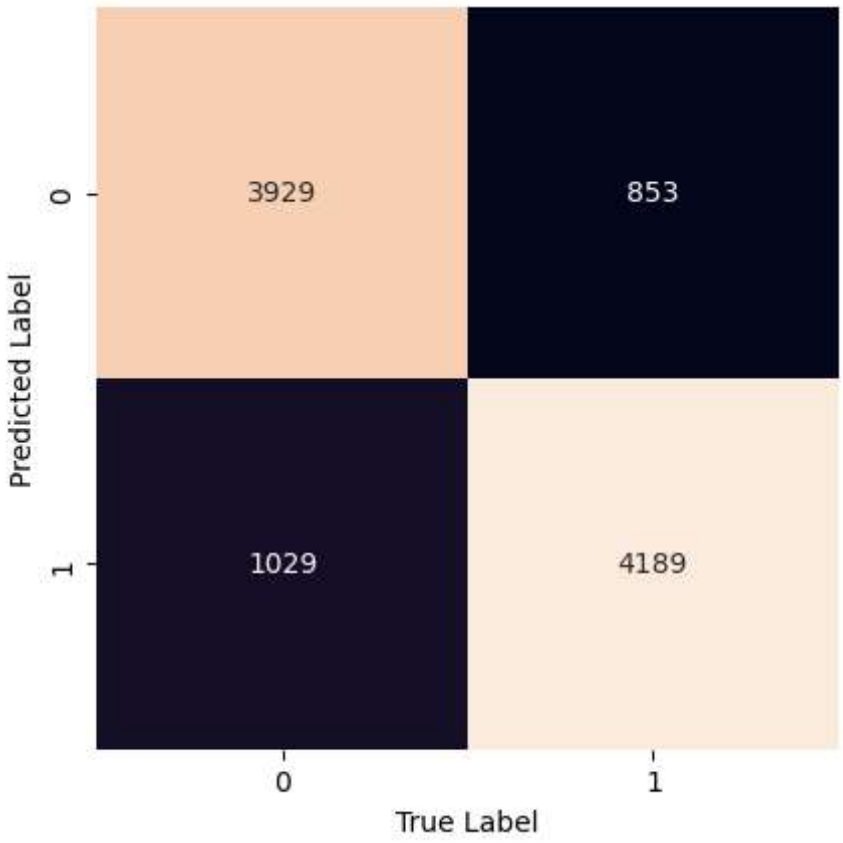
In [31]:
```
print('Confusion matrix of GRU model:')
performance(gru,x_train,y_train)
```

```
Confusion matrix of GRU model:
938/938 [==============================] - 12s 13ms/step - loss: 0.2495 - accuracy:
0.9046
0.2494727522134781
0.9046333432197571
938/938 [==============================] - 13s 13ms/step
```



In [32]:
```
print('Confusion matrix of GRU model:')
performance(gru,x_test,y_test)
```

```
Confusion matrix of GRU model:
313/313 [==============================] - 4s 12ms/step - loss: 0.4655 - accuracy: 0.
8118
0.465484619140625
0.8118000030517578
313/313 [==============================] - 4s 14ms/step
```



In [33]:
```
test['no_of_tokens'] = test['tokenized_text'].apply(lambda x : len(x.split(' ',-1)))
```

In [34]:
```
test.head()
```

Out[34]:

	text	label	tokenized_text	no_of_tokens
0	The central theme in this movie seems to be co...	0	central theme movie seems confusion relationsh...	148
1	An excellent example of "cowboy noir", as it's...	1	excellent example cowboy noir called unemploye...	241
2	The ending made my heart jump up into my throa...	0	ending made heart jump throat proceeded leave ...	29
3	Only the chosen ones will appreciate the quali...	1	chosen ones appreciate quality story character...	67
4	This is a really funny film, especially the se...	1	really funny film especially second third four...	63

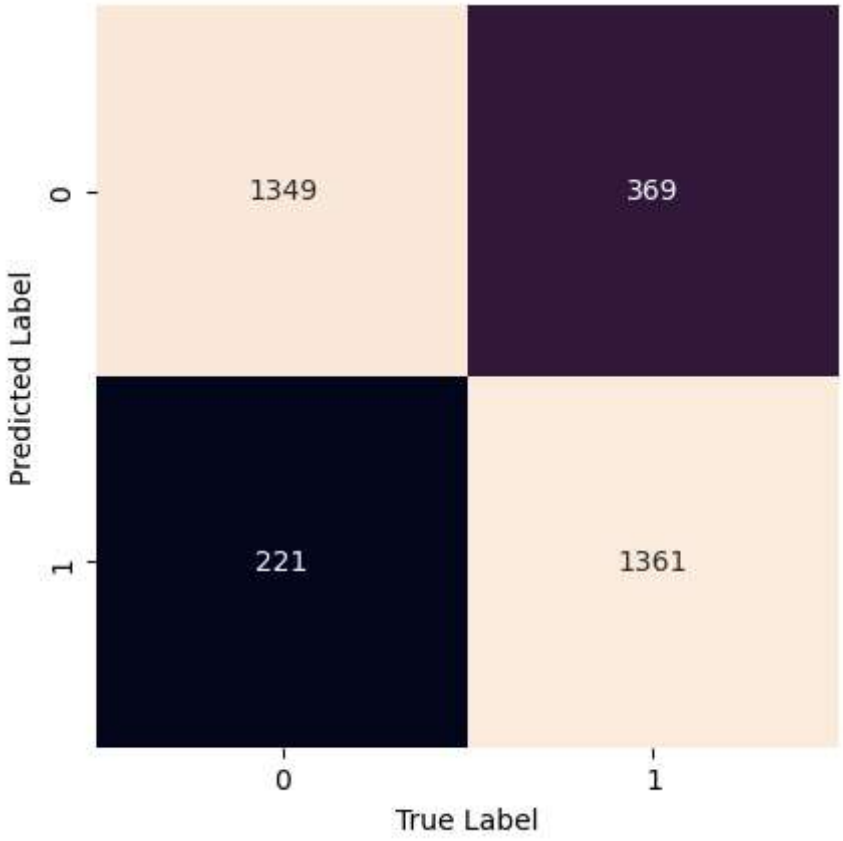
In [35]:
```
sorted_test = test.sort_values('no_of_tokens')
n = len(test)
short_indices = sorted_test.index[0:int(0.33*n)]
short_x_test = x_test[short_indices]
short_y_test = y_test[short_indices]

medium_indices = sorted_test.index[int(0.33*n):int(0.67*n)]
medium_x_test = x_test[medium_indices]
medium_y_test = y_test[medium_indices]
```

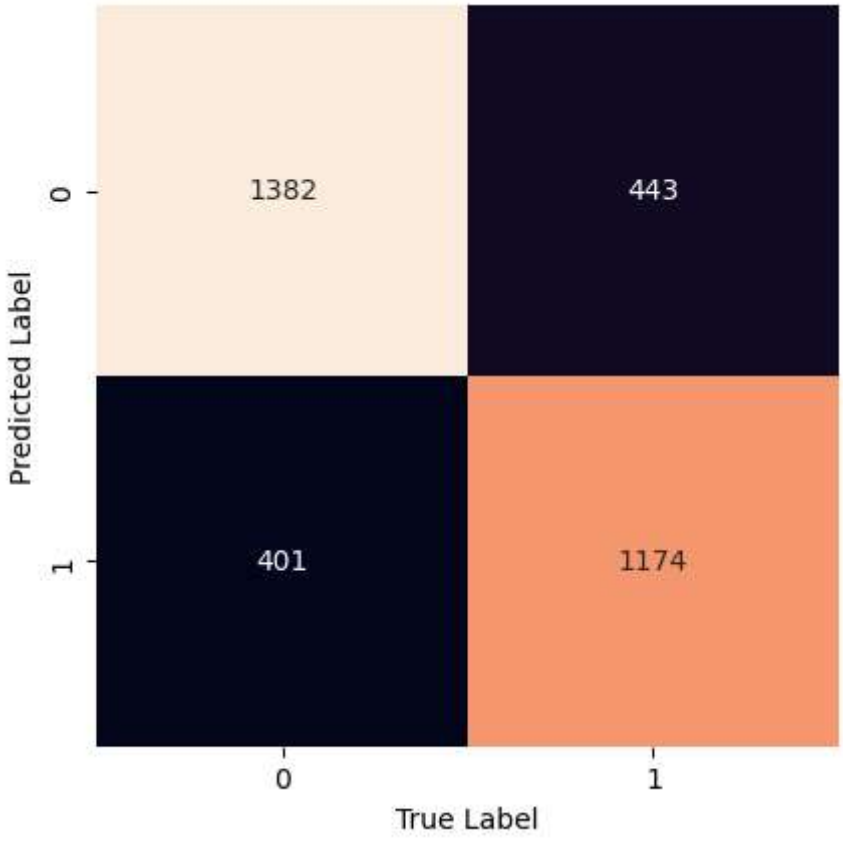
```
long_indices = sorted_test.index[int(0.67*n):-1]
long_x_test = x_test[long_indices]
long_y_test = y_test[long_indices]
```

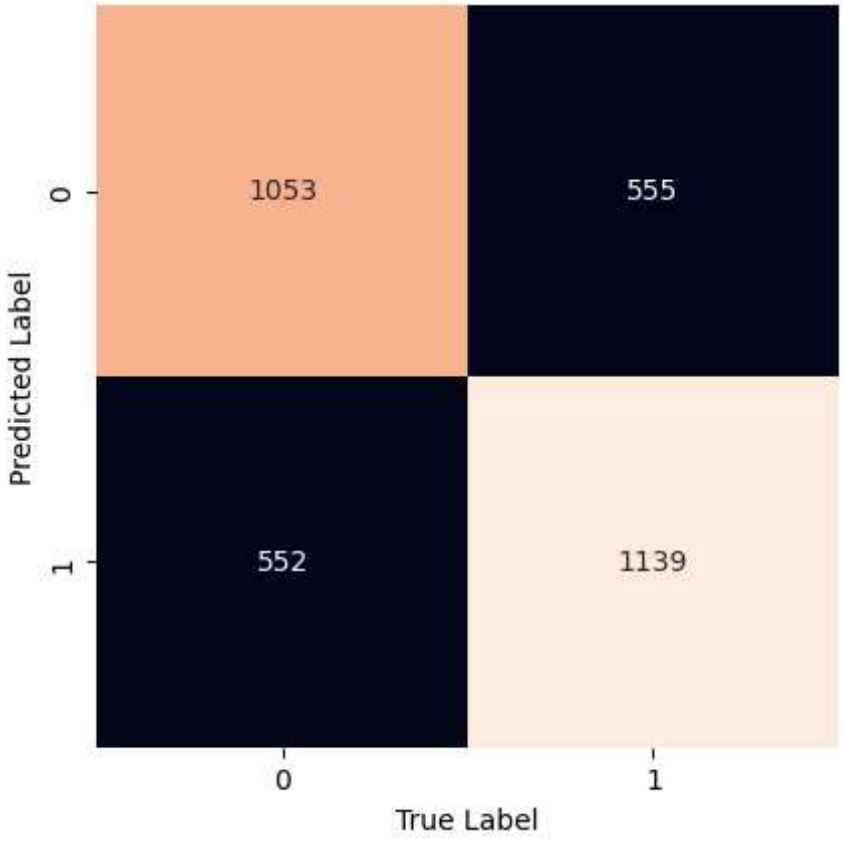
In [37]:

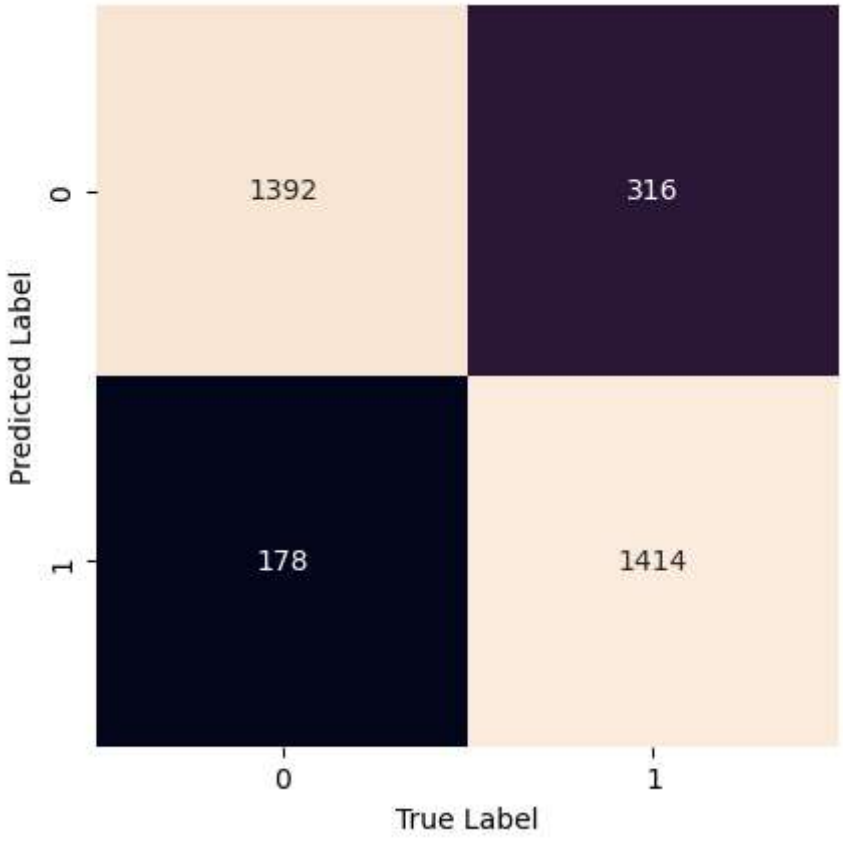
```
for model in ('rnn','lstm','gru'):
  for data in ('short','medium','long'):
    if data == 'short':
      x = short_x_test
      y = short_y_test
    elif data == 'medium':
      x = medium_x_test
      y = medium_y_test
    else:
      x = long_x_test
      y = long_y_test

    if model == 'rnn':
      trained_model = rnn
    elif model == 'lstm':
      trained_model = lstm
    else:
      trained_model = gru

    performance(trained_model, x,y)
```

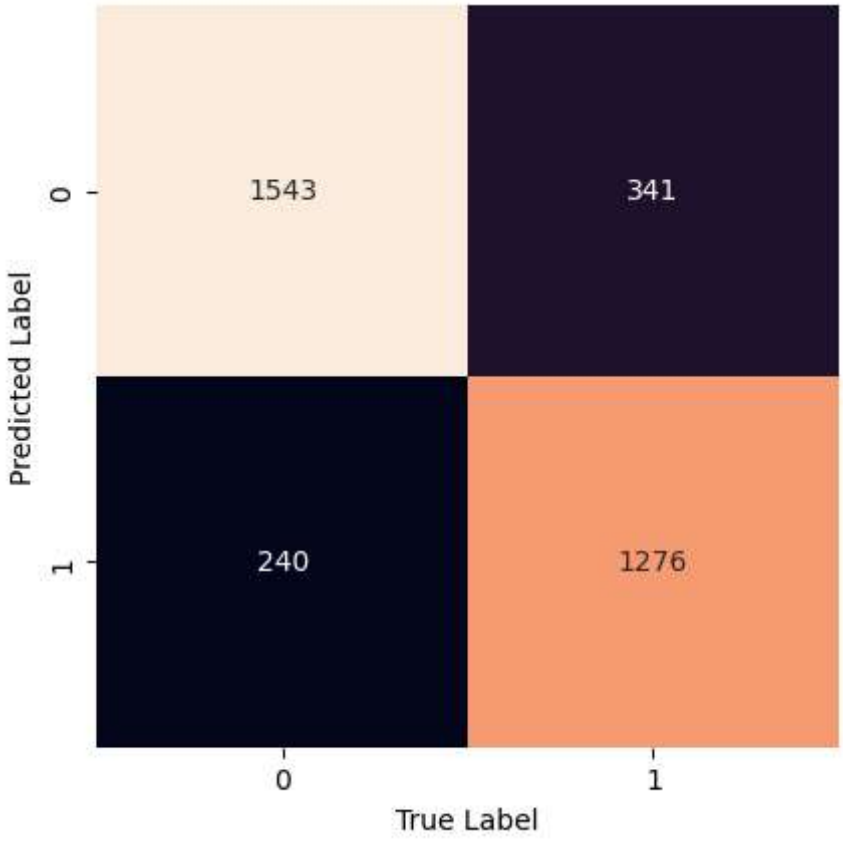
```
104/104 [==============================] - 1s 8ms/step - loss: 0.6324 - accuracy: 0.8
212
0.6324098706245422
0.821212112903595
104/104 [==============================] - 1s 13ms/step
107/107 [==============================] - 1s 8ms/step - loss: 0.9169 - accuracy: 0.7
518
0.9169014096260071
0.751764714717865
107/107 [==============================] - 1s 7ms/step
104/104 [==============================] - 1s 8ms/step - loss: 1.3620 - accuracy: 0.6
644
1.3620173931121826
0.6644437909126282
104/104 [==============================] - 1s 7ms/step
104/104 [==============================] - 1s 12ms/step - loss: 0.3587 - accuracy: 0.
8503
0.3586712181568146
0.850303053855896
104/104 [==============================] - 1s 12ms/step
107/107 [==============================] - 1s 12ms/step - loss: 0.4216 - accuracy: 0.
8291
0.42159804701805115
0.8291176557540894
107/107 [==============================] - 2s 18ms/step
104/104 [==============================] - 1s 13ms/step - loss: 0.6414 - accuracy: 0.
7430
0.6414075493812561
0.7429524064064026
104/104 [==============================] - 1s 11ms/step
104/104 [==============================] - 1s 11ms/step - loss: 0.3308 - accuracy: 0.
8633
0.3308359682559967
0.8633333444595337
104/104 [==============================] - 1s 12ms/step
107/107 [==============================] - 1s 12ms/step - loss: 0.4136 - accuracy: 0.
8291
0.4135618209838867
0.8291176557540894
107/107 [==============================] - 2s 16ms/step
104/104 [==============================] - 2s 16ms/step - loss: 0.6538 - accuracy: 0.
7423
0.6537901163101196
0.7423461675643921
104/104 [==============================] - 1s 12ms/step
```

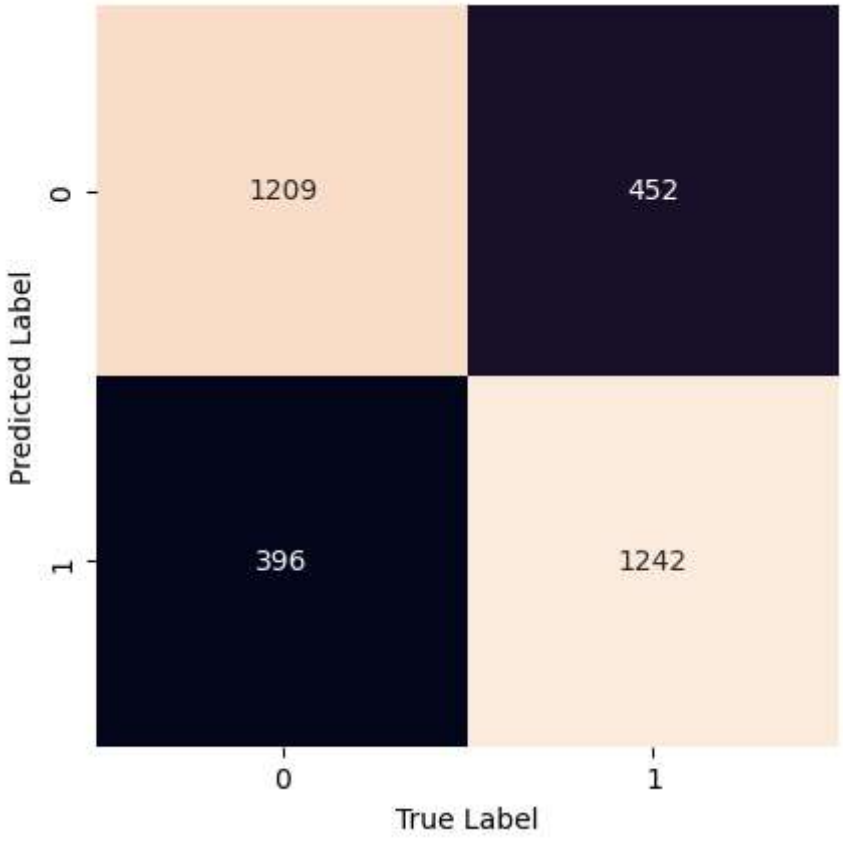

1349
369
221
1361
0
1
Predicted Label
0
1
True Label

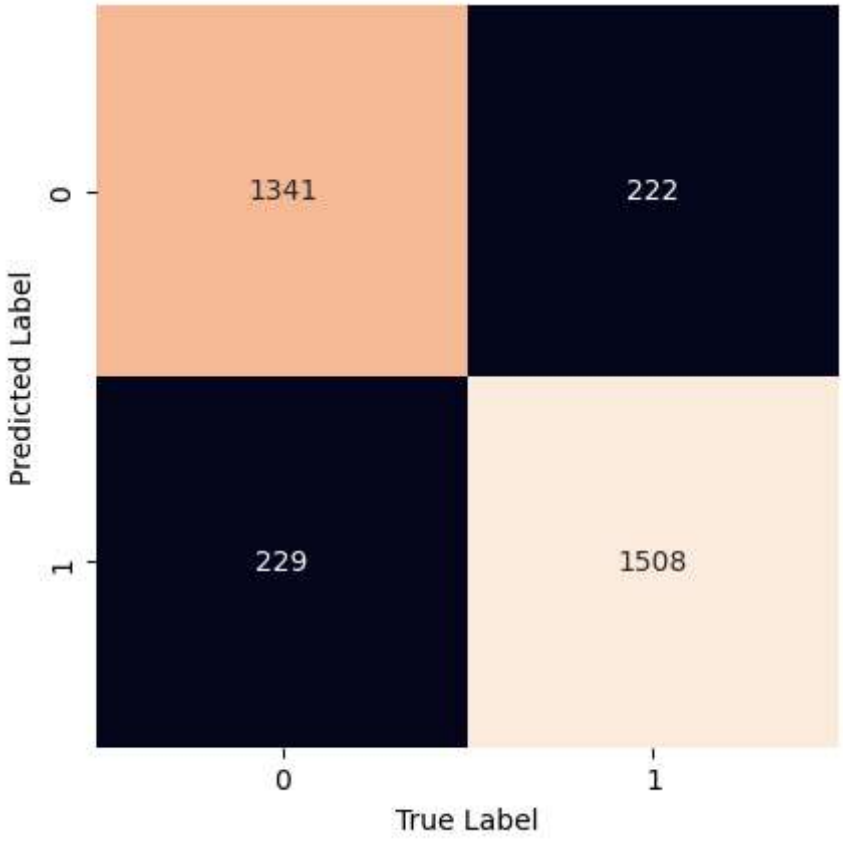

1382
443
401
1174
0
1
Predicted Label
0
1
True Label

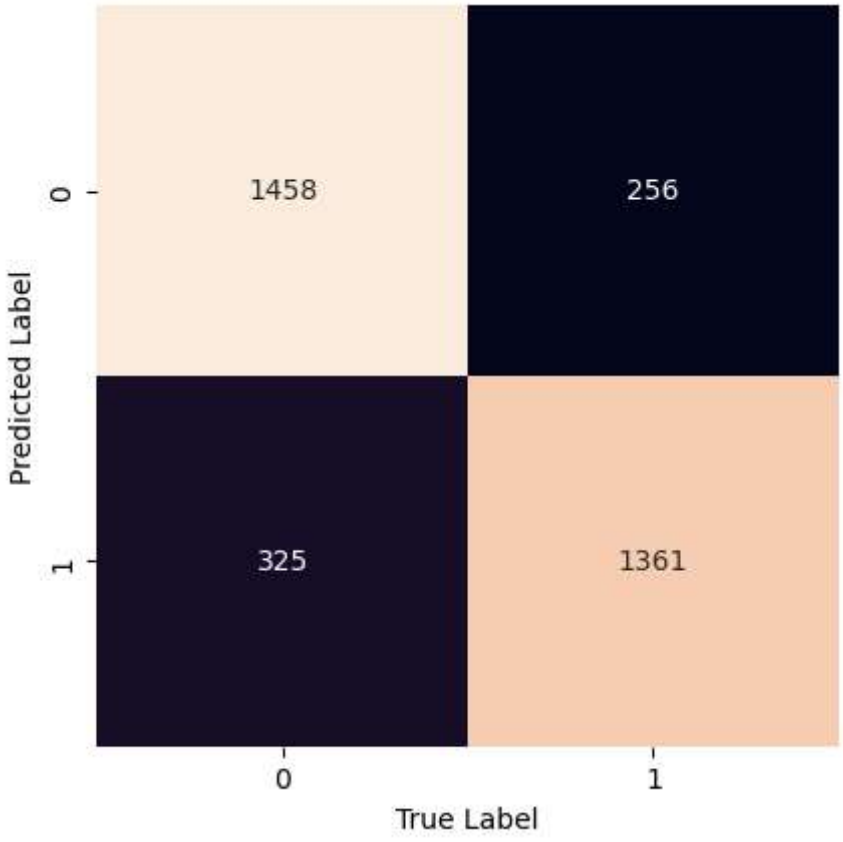

1053
555
552
1139
0
1
Predicted Label
0
1
True Label

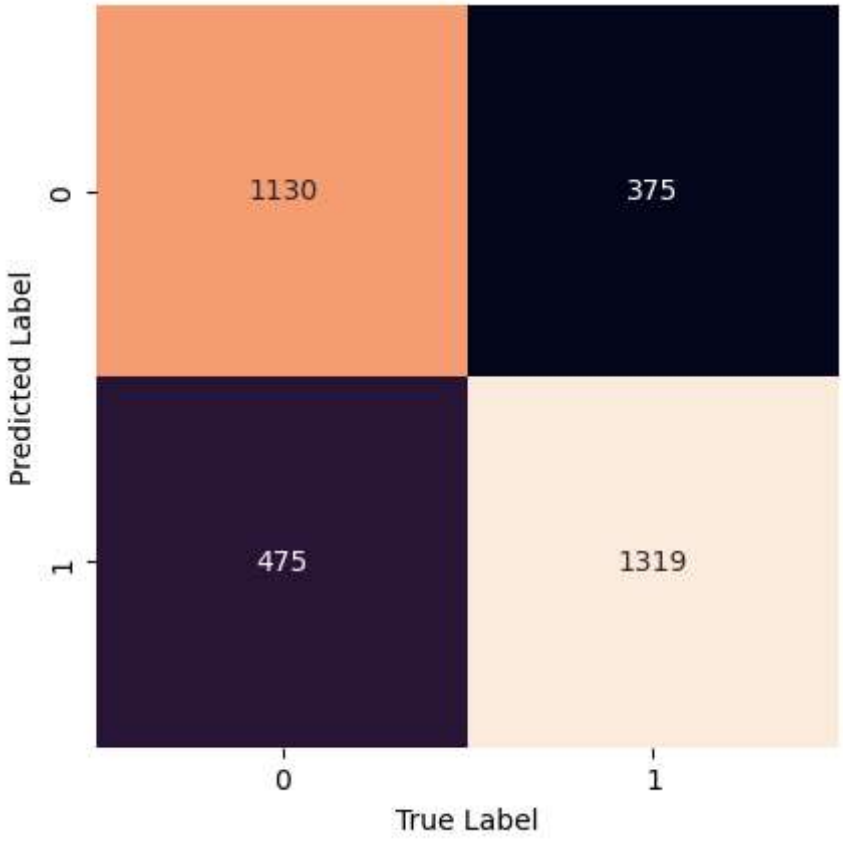

1392
316
178
1414
0
1
Predicted Label
0
1
True Label


1543
341
240
1276
0
1
Predicted Label
0
1
True Label


1209
452
396
1242
0
1
Predicted Label
0
1
True Label


1341
222
229
1508
0
1
Predicted Label
0
1
True Label


1458
256
325
1361
0
1
Predicted Label
0
1
True Label



In [ ]:

In [31]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import nltk
from nltk.corpus import stopwords
import re
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, GRU, Dense
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
from keras.initializers import Constant
import numpy as np
```

In [32]:
```
data = pd.read_csv('/content/movie.csv')
```

In [33]:
```
data.head()
```

Out[33]:

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1

In [34]:
```
data['label'].value_counts(normalize=True)
```

Out[34]:
```
0    0.500475
1    0.499525
Name: label, dtype: float64
```

In [35]:
```
nltk.download('stopwords')
# We filter out the english language stopwrds

stop_words = stopwords.words('english')
print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [36]:
```
remove_chars = "@S+|https?:S+|http?:S|[^A-Za-z0-9]+"
```

In [37]:
```
def tokenize_text(text):
  # Text passed to the regex equatio
  text = re.sub(remove_chars, ' ', str(text).lower()).strip()
  # Empty list created to store final tokens
  tokens = []
  for token in text.split():
    # check if the token is a stop word or not
    if token not in stop_words:
        tokens.append(token)
  return " ".join(tokens)
```

In [38]:
```
data['tokenized_text'] = data['text'].apply(lambda x: tokenize_text(x))
```

In [39]:
```
data.head()
```

Out[39]:

	text	label	tokenized_text
0	I grew up (b. 1965) watching and loving the Th...	0	grew b 1965 watching loving thunderbirds mates...
1	When I put this movie in my DVD player, and sa...	0	put movie dvd player sat coke chips expectatio...
2	Why do people who do not know what a particula...	0	people know particular time past like feel nee...
3	Even though I have great interest in Biblical ...	0	even though great interest biblical movies bor...
4	Im a die hard Dads Army fan and nothing will e...	1	im die hard dads army fan nothing ever change ...

In [40]:
```
train,test = train_test_split(data, test_size=0.25, random_state=42)
```

In [41]:
```
train.reset_index(inplace=True, drop=True)
test.reset_index(inplace=True, drop=True)
```

In [42]:
```
test.head()
```

Out[42]:

	text	label	tokenized_text
0	The central theme in this movie seems to be co...	0	central theme movie seems confusion relationsh...
1	An excellent example of "cowboy noir", as it's...	1	excellent example cowboy noir called unemploye...
2	The ending made my heart jump up into my throa...	0	ending made heart jump throat proceeded leave ...
3	Only the chosen ones will appreciate the quali...	1	chosen ones appreciate quality story character...
4	This is a really funny film, especially the se...	1	really funny film especially second third four...

In [43]:
```
train.shape, test.shape
```

Out[43]:
```
((30000, 3), (10000, 3))
```

In [44]:
```
from keras.layers import TextVectorization

vectorizer = TextVectorization(max_tokens=10000, output_sequence_length=100)
vectorizer.adapt(train['tokenized_text'])
voc = vectorizer.get_vocabulary()
```

In [45]:
```
x_train = vectorizer(np.array([[s] for s in train['tokenized_text']])).numpy()
x_test  = vectorizer(np.array([[s] for s in test['tokenized_text']])).numpy()

# y_train = np.array(train['label'])
# y_test = np.array(test['label'])
```

In [46]:
```
enc = OneHotEncoder(handle_unknown='ignore')
enc.fit(np.array(train['label'].tolist()).reshape(-1,1))
y_train = enc.transform(np.array(train['label'].tolist()).reshape(-1,1)).toarray()
y_test = enc.transform(np.array(test['label'].tolist()).reshape(-1,1)).toarray()
```

In [47]:
```
x_train
```

Out[47]:
```
array([[4933, 1120,  950, ...,    0,    0,    0],
       [ 241,  216, 8904, ...,    0,    0,    0],
       [  14,  457,  178, ...,  216,   26,    6],
       ...,
       [2390,  362,   13, ...,   23,    1, 4093],
       [ 420, 7989,    5, ...,  487, 3603, 1313],
       [   3,  365, 1294, ...,    0,    0,    0]])
```

In [48]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[48]:
```
((30000, 100), (30000, 2), (10000, 100), (10000, 2))
```

In [22]:
```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

```
--2024-03-13 15:59:32--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2024-03-13 15:59:32--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2024-03-13 15:59:33--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:44
3... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.09MB/s    in 2m 38s

2024-03-13 16:02:12 (5.19 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]
```

In [49]:
```
path_to_glove_file = "glove.6B.50d.txt"

embeddings_dict_50 = {}

with open(path_to_glove_file) as f:
  for line in f:
    word, coefs = line.split(maxsplit=1)
    coefs = np.fromstring(coefs, 'f', sep=' ')
    embeddings_dict_50[word] = coefs

print("Found {} word vectors. Vector length is {}.".format(len(embeddings_dict_50), le
```

```
Found 400000 word vectors. Vector length is 50.
```

In [50]:
```
path_to_glove_file = "glove.6B.100d.txt"

embeddings_dict_100 = {}

with open(path_to_glove_file) as f:
  for line in f:
    word, coefs = line.split(maxsplit=1)
    coefs = np.fromstring(coefs, 'f', sep=' ')
    embeddings_dict_100[word] = coefs

print("Found {} word vectors. Vector length is {}.".format(len(embeddings_dict_100), l
```

```
Found 400000 word vectors. Vector length is 100.
```

In [51]:
```
hits = 0

embedding_matrix_50 = np.zeros((len(voc), 50))
for ind, word in enumerate(voc):
  if word in embeddings_dict_50.keys():
    embedding_matrix_50[ind] = embeddings_dict_50[word].copy()
    hits += 1
```

```
print('converted {} words output {} ({} misses)'.format(hits, len(voc), len(voc)-hits)
```

```
converted 9995 words output 10000 (5 misses)
```

In [52]:
```
hits = 0

embedding_matrix_100 = np.zeros((len(voc), 100))
for ind, word in enumerate(voc):
  if word in embeddings_dict_100.keys():
    embedding_matrix_100[ind] = embeddings_dict_100[word].copy()
    hits += 1

print('converted {} words output {} ({} misses)'.format(hits, len(voc), len(voc)-hits)
```

```
converted 9995 words output 10000 (5 misses)
```

In [53]:
```
def gru_model(input_dim,output_dim,num_classes,embedding_matrix):
  model = Sequential()
  model.add(Embedding(input_dim = input_dim , output_dim = output_dim,embeddings_initi
  model.add(GRU(32))
  model.add(Dense(32,activation='relu'))
  model.add(Dense(num_classes,activation='softmax'))
  model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['acc
  return model
```

In [54]:
```
gru_50 = gru_model(len(voc),50,2,embedding_matrix_50)
gru_50.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 50)          500000

 gru_1 (GRU)                 (None, 32)                8064

 dense_2 (Dense)             (None, 32)                1056

 dense_3 (Dense)             (None, 2)                 66

=================================================================
Total params: 509186 (1.94 MB)
Trainable params: 9186 (35.88 KB)
Non-trainable params: 500000 (1.91 MB)
_________________________________________________________________
```

In [55]:
```
gru_fitted_50 = gru_50.fit(x_train, y_train, epochs=10, validation_split=0.2,batch_siz
```

```
Epoch 1/10
94/94 [==============================] - 16s 129ms/step - loss: 0.6892 - accuracy: 0.
5286 - val_loss: 0.6809 - val_accuracy: 0.5440
Epoch 2/10
94/94 [==============================] - 9s 91ms/step - loss: 0.6039 - accuracy: 0.66
89 - val_loss: 0.5231 - val_accuracy: 0.7473
Epoch 3/10
94/94 [==============================] - 14s 149ms/step - loss: 0.4882 - accuracy: 0.
7727 - val_loss: 0.4972 - val_accuracy: 0.7673
Epoch 4/10
94/94 [==============================] - 9s 94ms/step - loss: 0.4466 - accuracy: 0.79
53 - val_loss: 0.4421 - val_accuracy: 0.7953
Epoch 5/10
94/94 [==============================] - 11s 112ms/step - loss: 0.4227 - accuracy: 0.
8073 - val_loss: 0.4322 - val_accuracy: 0.7968
Epoch 6/10
94/94 [==============================] - 12s 131ms/step - loss: 0.4052 - accuracy: 0.
8160 - val_loss: 0.4197 - val_accuracy: 0.8067
Epoch 7/10
94/94 [==============================] - 7s 74ms/step - loss: 0.3951 - accuracy: 0.82
15 - val_loss: 0.4104 - val_accuracy: 0.8152
Epoch 8/10
94/94 [==============================] - 11s 118ms/step - loss: 0.3894 - accuracy: 0.
8252 - val_loss: 0.4136 - val_accuracy: 0.8080
Epoch 9/10
94/94 [==============================] - 12s 126ms/step - loss: 0.3785 - accuracy: 0.
8310 - val_loss: 0.4113 - val_accuracy: 0.8148
Epoch 10/10
94/94 [==============================] - 8s 83ms/step - loss: 0.3722 - accuracy: 0.83
40 - val_loss: 0.4001 - val_accuracy: 0.8223
```

In [56]:
```
gru_100 = gru_model(len(voc),100,2,embedding_matrix_100)
gru_100.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 100)         1000000

 gru_2 (GRU)                 (None, 32)                12864

 dense_4 (Dense)             (None, 32)                1056

 dense_5 (Dense)             (None, 2)                 66

=================================================================
Total params: 1013986 (3.87 MB)
Trainable params: 13986 (54.63 KB)
Non-trainable params: 1000000 (3.81 MB)
_________________________________________________________________
```

In [57]:
```
gru_fitted_100 = gru_100.fit(x_train, y_train, epochs=10, validation_split=0.2,batch_s
```

```
Epoch 1/10
94/94 [==============================] - 19s 171ms/step - loss: 0.6888 - accuracy: 0.
5243 - val_loss: 0.6775 - val_accuracy: 0.5512
Epoch 2/10
94/94 [==============================] - 15s 164ms/step - loss: 0.5820 - accuracy: 0.
6926 - val_loss: 0.4955 - val_accuracy: 0.7708
Epoch 3/10
94/94 [==============================] - 13s 143ms/step - loss: 0.4468 - accuracy: 0.
7995 - val_loss: 0.4370 - val_accuracy: 0.8010
Epoch 4/10
94/94 [==============================] - 13s 136ms/step - loss: 0.3980 - accuracy: 0.
8231 - val_loss: 0.3965 - val_accuracy: 0.8237
Epoch 5/10
94/94 [==============================] - 18s 189ms/step - loss: 0.3663 - accuracy: 0.
8382 - val_loss: 0.3835 - val_accuracy: 0.8280
Epoch 6/10
94/94 [==============================] - 19s 204ms/step - loss: 0.3509 - accuracy: 0.
8479 - val_loss: 0.3946 - val_accuracy: 0.8257
Epoch 7/10
94/94 [==============================] - 13s 141ms/step - loss: 0.3374 - accuracy: 0.
8534 - val_loss: 0.3633 - val_accuracy: 0.8398
Epoch 8/10
94/94 [==============================] - 14s 145ms/step - loss: 0.3274 - accuracy: 0.
8608 - val_loss: 0.3778 - val_accuracy: 0.8373
Epoch 9/10
94/94 [==============================] - 14s 145ms/step - loss: 0.3201 - accuracy: 0.
8615 - val_loss: 0.3553 - val_accuracy: 0.8448
Epoch 10/10
94/94 [==============================] - 13s 141ms/step - loss: 0.3124 - accuracy: 0.
8665 - val_loss: 0.3510 - val_accuracy: 0.8478
```

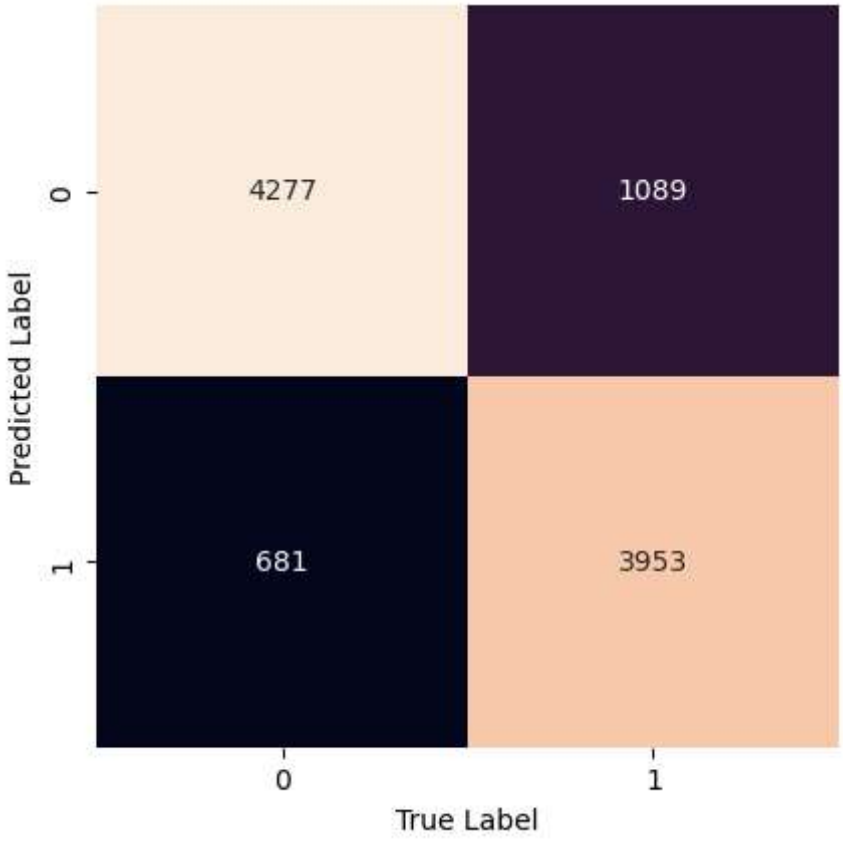
In [58]:

```
def performance(model, x, y):
  loss,accuracy = model.evaluate(x,y)
  print(loss)
  print(accuracy)

  y_pred = model.predict(x)
  predicted_classes = np.argmax(y_pred, axis=1)
  true_classes = np.argmax(y, axis=1)

  conf_matrix = confusion_matrix(true_classes, predicted_classes)
  plt.figure()
  sns.heatmap(conf_matrix.T, square=True, annot=True, fmt='d', cbar=False)
  plt.xlabel("True Label")
  plt.ylabel("Predicted Label")
```

In [59]:

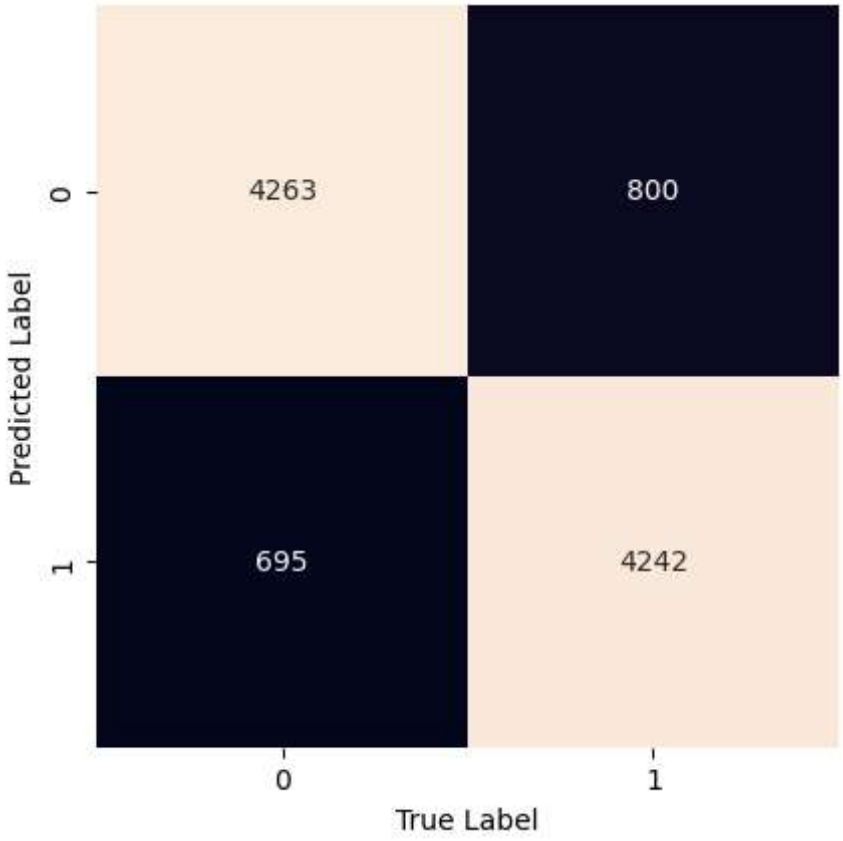
```
print('Confusion matrix of GRU 50 model:')
performance(gru_50,x_test,y_test)
```

```
Confusion matrix of GRU 50 model:
313/313 [==============================] - 5s 16ms/step - loss: 0.3948 - accuracy: 0.
8230
0.3948245942592621
0.8230000138282776
313/313 [==============================] - 6s 16ms/step
```



In [60]:
```
print('Confusion matrix of GRU 100 model:')
performance(gru_100,x_test,y_test)
```

```
Confusion matrix of GRU 100 model:
313/313 [==============================] - 4s 12ms/step - loss: 0.3433 - accuracy: 0.
8505
0.3433462977409363
0.8504999876022339
313/313 [==============================] - 4s 10ms/step
```



In [61]:
```
test['no_of_tokens'] = test['tokenized_text'].apply(lambda x : len(x.split(' ',-1)))
```

In [62]:
```
test.head()
```

Out[62]:

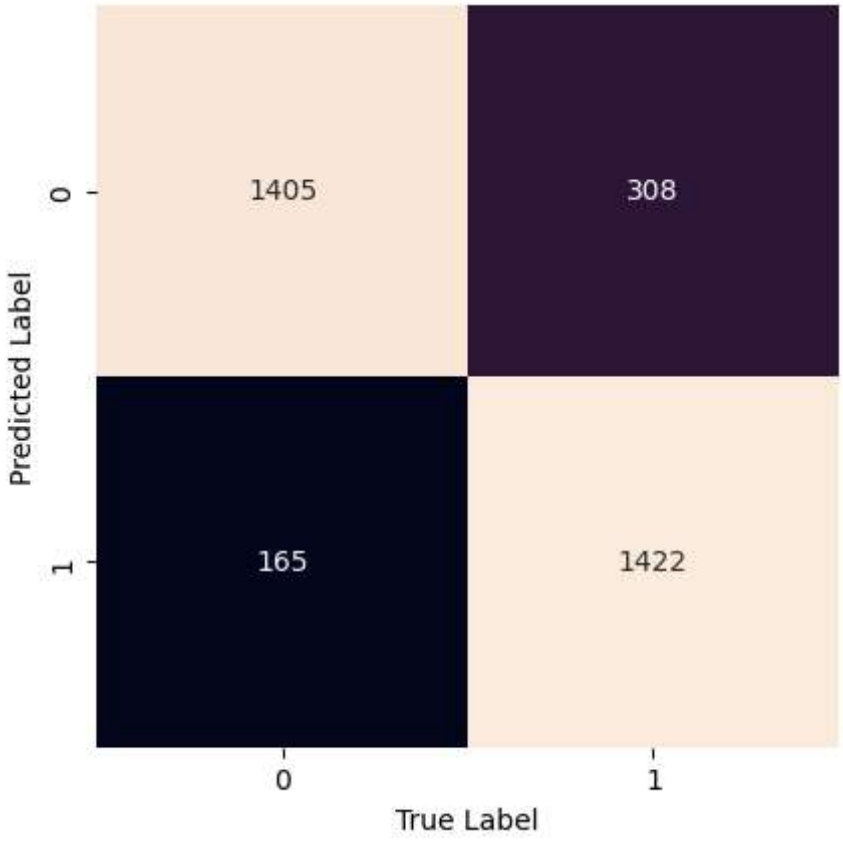
	text	label	tokenized_text	no_of_tokens
0	The central theme in this movie seems to be co...	0	central theme movie seems confusion relationsh...	148
1	An excellent example of "cowboy noir", as it's...	1	excellent example cowboy noir called unemploye...	241
2	The ending made my heart jump up into my throa...	0	ending made heart jump throat proceeded leave ...	29
3	Only the chosen ones will appreciate the quali...	1	chosen ones appreciate quality story character...	67
4	This is a really funny film, especially the se...	1	really funny film especially second third four...	63

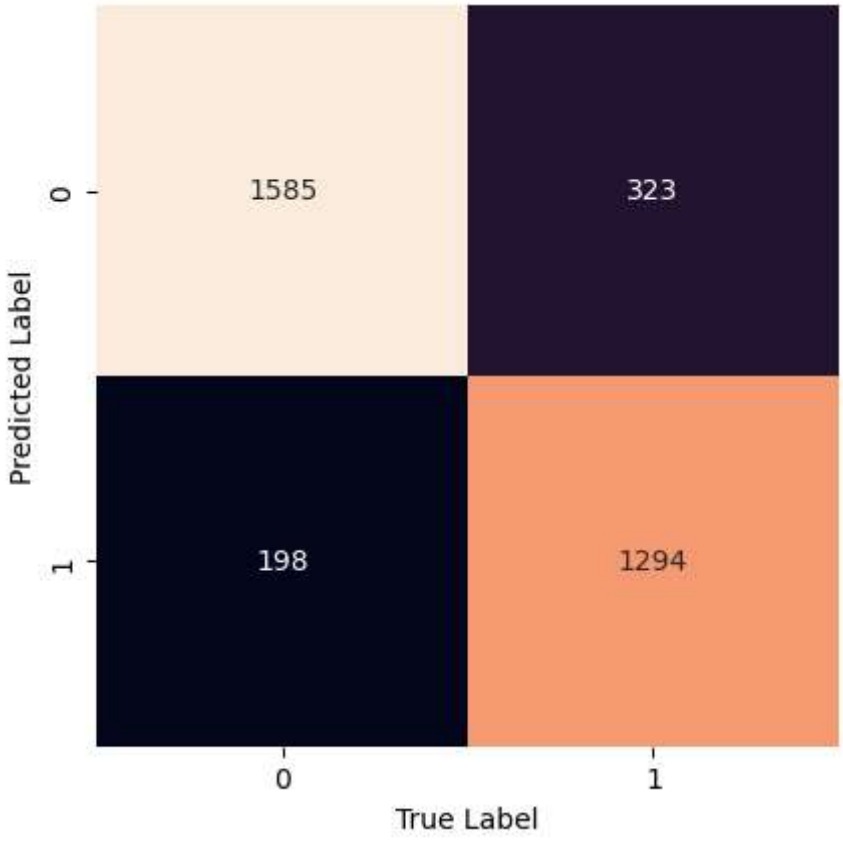
In [63]:
```
sorted_test = test.sort_values('no_of_tokens')
n = len(test)
short_indices = sorted_test.index[0:int(0.33*n)]
short_x_test = x_test[short_indices]
short_y_test = y_test[short_indices]

medium_indices = sorted_test.index[int(0.33*n):int(0.67*n)]
medium_x_test = x_test[medium_indices]
medium_y_test = y_test[medium_indices]
```

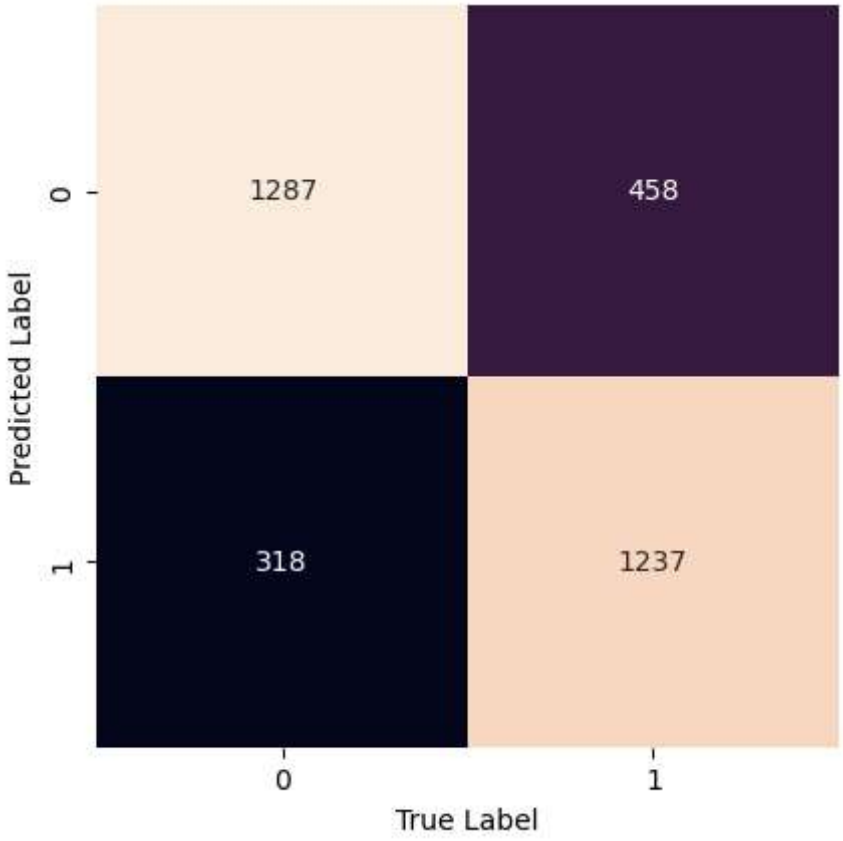
```
long_indices = sorted_test.index[int(0.67*n):n]
long_x_test = x_test[long_indices]
long_y_test = y_test[long_indices]
```

In [64]:
```
for data in ('short','medium','long'):
  print(f'Confusion matrix on {data} data')
  if data == 'short':
    x = short_x_test
    y = short_y_test
  elif data == 'medium':
    x = medium_x_test
    y = medium_y_test
  else:
    x = long_x_test
    y = long_y_test
  performance(gru_50, x,y)
```

```
Confusion matrix on short data
104/104 [==============================] - 2s 18ms/step - loss: 0.3445 - accuracy: 0.
8567
0.3444814085960388
0.8566666841506958
104/104 [==============================] - 3s 28ms/step
Confusion matrix on medium data
107/107 [==============================] - 2s 19ms/step - loss: 0.3592 - accuracy: 0.
8468
0.3591828942298889
0.8467646837234497
107/107 [==============================] - 1s 9ms/step
Confusion matrix on long data
104/104 [==============================] - 1s 10ms/step - loss: 0.4819 - accuracy: 0.
7648
0.4818896949291229
0.7648484706878662
104/104 [==============================] - 1s 9ms/step
```

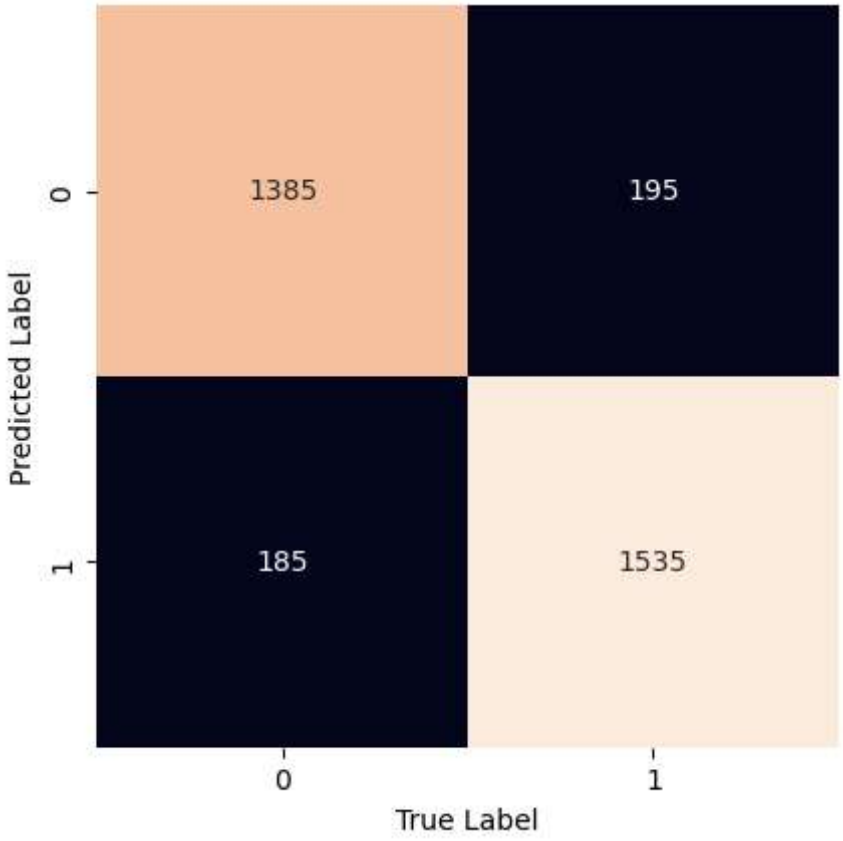

1405
308
165
1422
0
1
Predicted Label
0
1
True Label

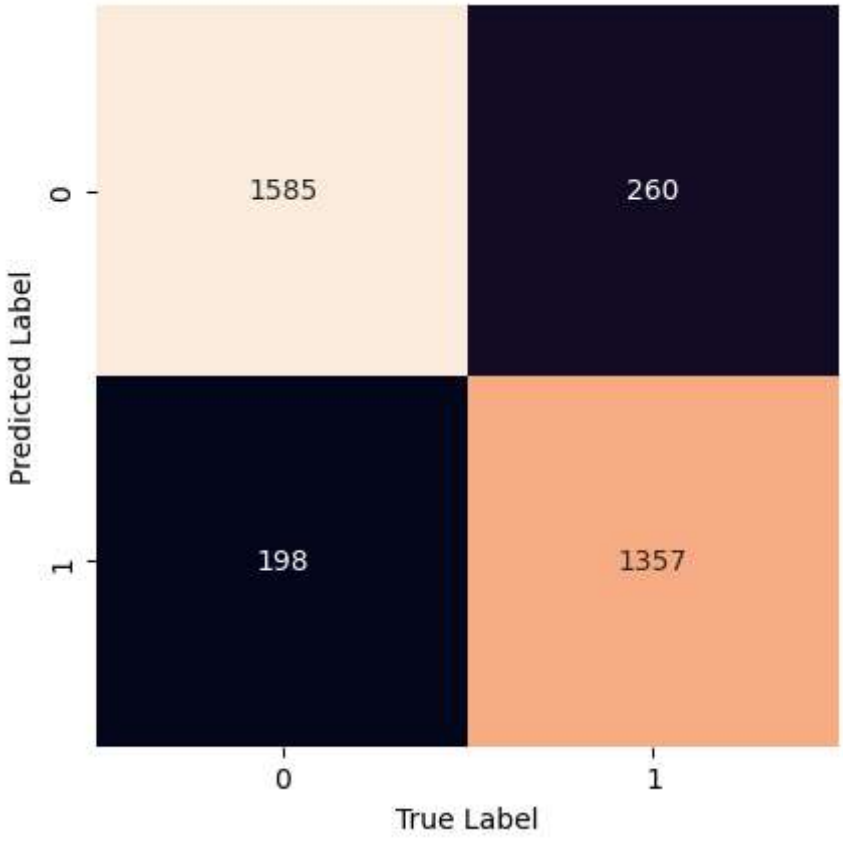

1585
323
198
1294
0
1
Predicted Label
0
1
True Label

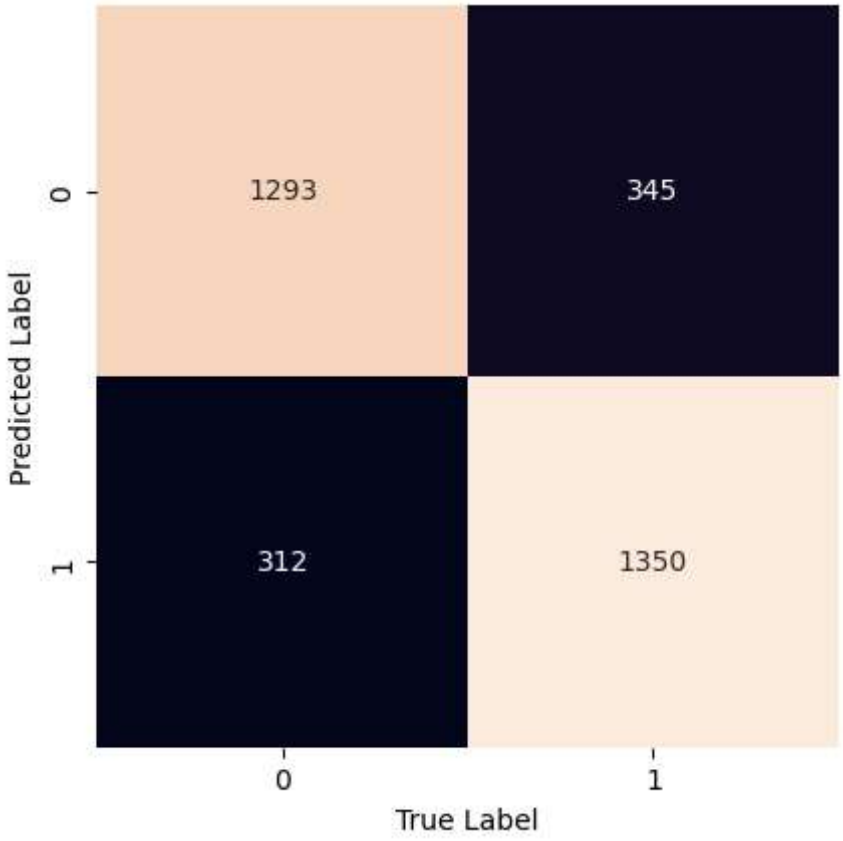


In [65]:
```
for data in ('short','medium','long'):
  print(f'Confusion matrix on {data} data')
  if data == 'short':
    x = short_x_test
    y = short_y_test
  elif data == 'medium':
    x = medium_x_test
    y = medium_y_test
  else:
    x = long_x_test
    y = long_y_test
  performance(gru_100, x,y)
```

```
Confusion matrix on short data
104/104 [==============================] - 1s 11ms/step - loss: 0.2773 - accuracy: 0.
8848
0.27731049060821533
0.8848484754562378
104/104 [==============================] - 1s 10ms/step
Confusion matrix on medium data
107/107 [==============================] - 2s 14ms/step - loss: 0.3188 - accuracy: 0.
8653
0.31879889965057373
0.8652940988540649
107/107 [==============================] - 2s 17ms/step
Confusion matrix on long data
104/104 [==============================] - 2s 17ms/step - loss: 0.4347 - accuracy: 0.
8009
0.4346729516983032
0.8009091019630432
104/104 [==============================] - 2s 18ms/step
```


1385
195
185
1535
0
1
Predicted Label
0
1
True Label


1585
260
198
1357
0
1
Predicted Label
0
1
True Label



In [ ]: