MPAA rating prediction

In [49]:
```
import pandas as pd
import numpy as np
import warnings
import re
import pickle
import seaborn as sns

%matplotlib inline
warnings.filterwarnings('ignore')
```

Import dataset :: using the rotten tomatoes dataset

In [50]:
```
dataset_df = pd.read_csv( './rtt_dataset/all_movie.csv' )
dataset_df = dataset_df[['Title', 'Rating', 'Description']]
dataset_df.columns = dataset_df.columns.str.strip()
dataset_df.head(4)
```

Out[50]:

	Title	Rating	Description
0	The Mummy: Tomb of the Dragon Emperor	PG-13	The Fast and the Furious director Rob Cohen co...
1	The Masked Saint	PG-13	The journey of a professional wrestler who bec...
2	Spy Hard	PG-13	Dead pan Leslie Nielsen stars as Dick Steele, ...
3	Der Baader Meinhof Komplex (The Baader Meinhof...	R	Director Uli Edel teams with screenwriter Bern...

In [51]:
```
# finding unique mpaa rating in the dataset
dataset_df['Rating'].unique()
```

Out[51]:
```
array(['PG-13 ', 'PG-13', 'R ', 'NR', 'PG', 'G', 'PG ', 'R', 'G ', 'NR ',
       'NC17', 'NC17 '], dtype=object)
```

In [52]:
```
# clean mpaa ratings
mpaa_fix = {
    'PG-13 ' : 'PG-13',
    'R ' : 'R',
    'PG ': 'PG',
    'G ' : 'G',
    'NR ': 'NR',
    'NC17 ': 'NC17'
}
for i, rating in dataset_df['Rating'].iteritems():
    if rating in mpaa_fix.keys():
        fix_val = mpaa_fix.get(rating)
        dataset_df.iloc[i]['Rating'] = fix_val
dataset_df['Rating'].unique()
```

Out[52]:
```
array(['PG-13', 'R', 'NR', 'PG', 'G', 'NC17'], dtype=object)
```

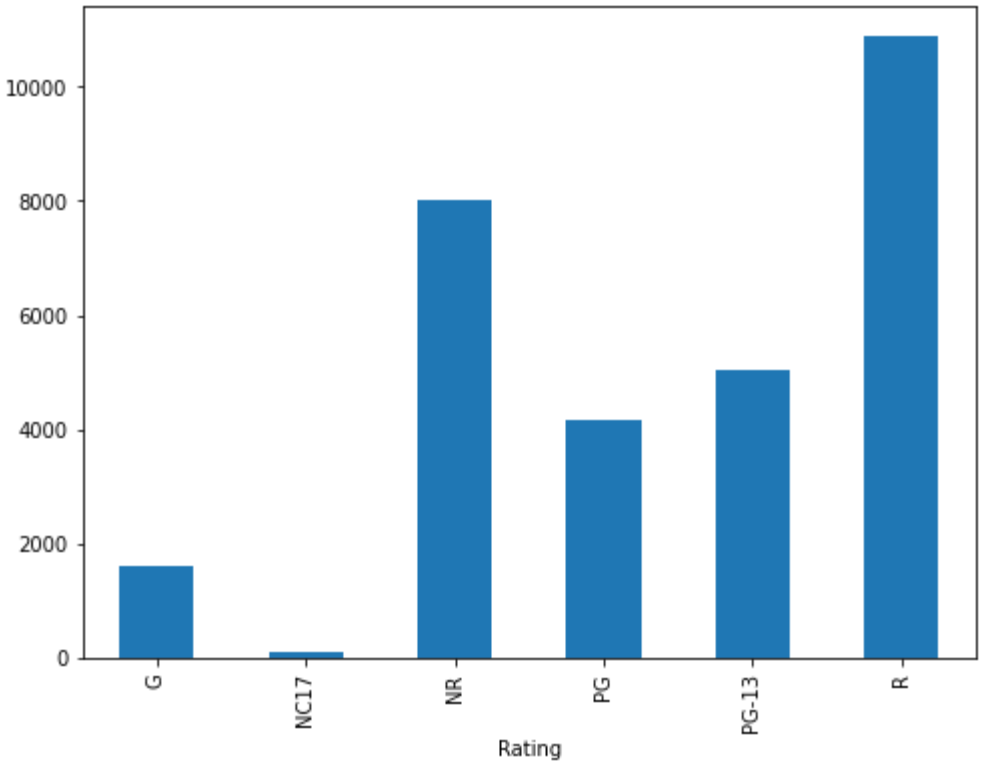
In [53]:
```
# Clean the movie description
def clean_description( text_str ):
    text = re.sub('[^a-zA-Z]',  ' ', text_str)
    text = re.sub(r'\s+[a-zA-Z]\s+', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    return text

for i, description in dataset_df['Description'].iteritems():
    dataset_df.iloc[i]['Description'] = clean_description( str(description) )
```

In [54]:
```
pickle.dump(dataset_df, open('./data/dataset_df_cleaned_n_sample.pkl', 'wb'))
```

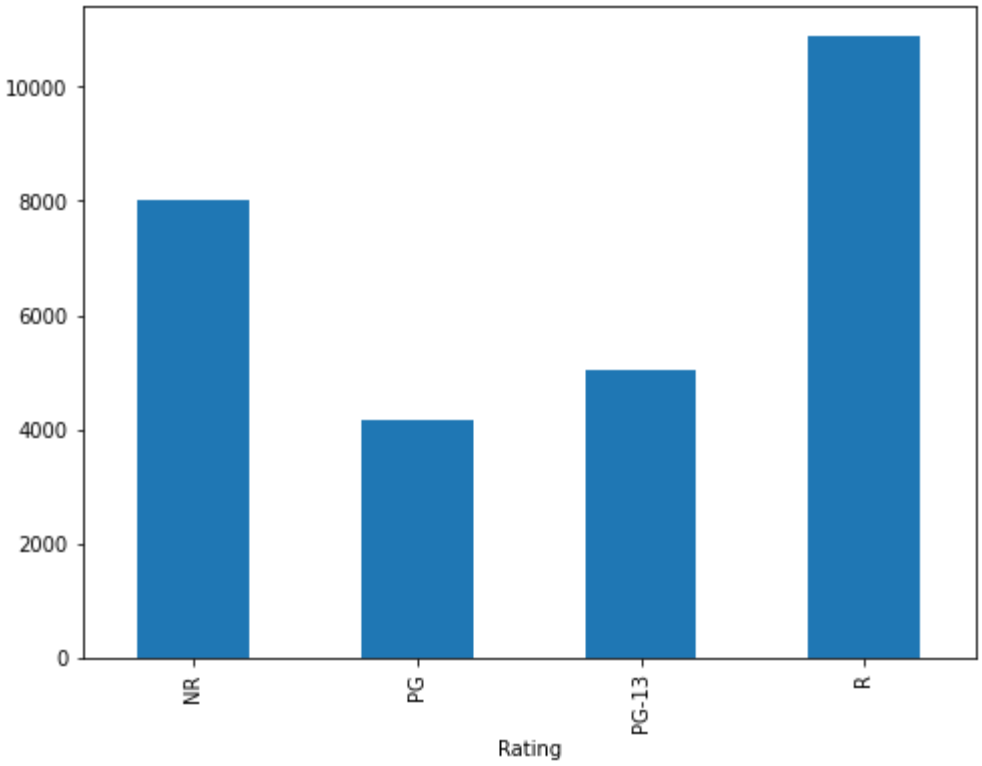
Data Distribution

In [55]:
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,6))
dataset_df.groupby('Rating')['Description'].count().plot.bar(ylim=0)
plt.show()
```



In [56]:
```
## drop G, NC-17
dataset_df = dataset_df[dataset_df.Rating != 'NC17']
dataset_df = dataset_df[dataset_df.Rating != 'G']
```

In [57]:
```
fig = plt.figure(figsize=(8,6))
dataset_df.groupby('Rating')['Description'].count().plot.bar(ylim=0)
plt.show()
```



Use same amount of labels

In [58]:
```
PG_count = dataset_df[dataset_df.Rating == 'PG'].shape
PG_count
```

Out[58]: (4172, 3)

In [59]:
```
dataset_df = dataset_df.sample(frac=1).groupby('Rating').head(4172)
dataset_df
```

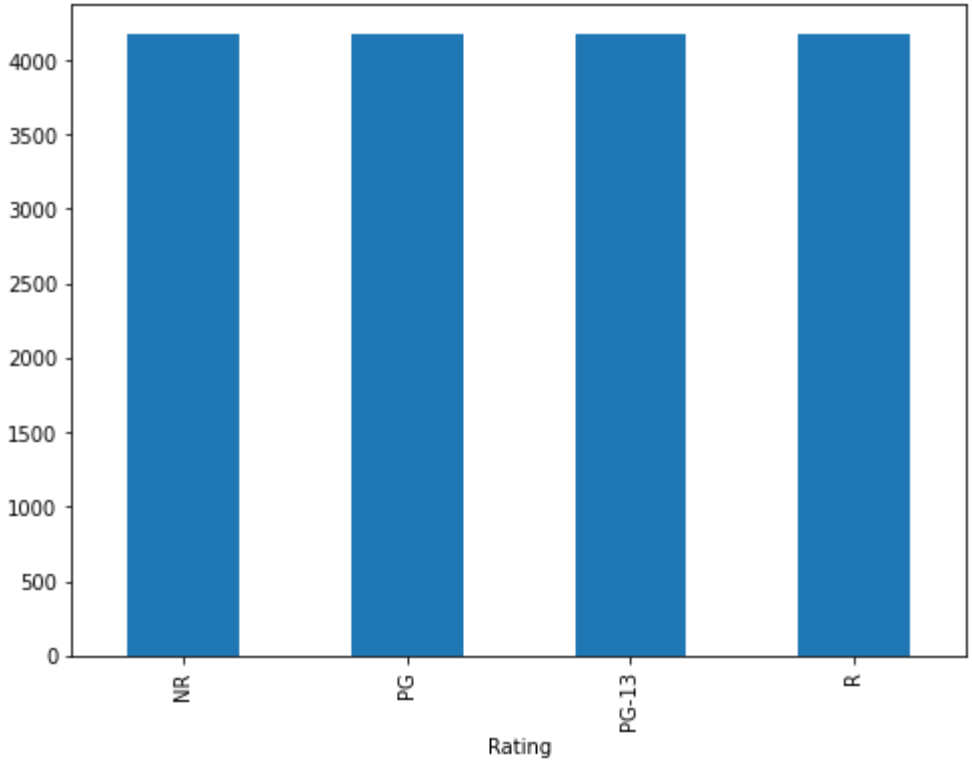
Out[59]:

	Title	Rating	Description
5801	Billy Jack Goes to Washington	PG	The fourth film starring Tom Laughlin as Billy...
23621	Scream 4	R	In Scream Sidney Prescott now the author of se...

	Title	Rating	Description
3543	Wizards	PG	In this animated futuristic tale pair of twins...
9338	The Big Day (Jour De Fete)	NR	In Jacques Tati charming and essentially plotl...
6532	The Mummy's Hand	NR	The first of four loose sequels to the origina...
...	...	...	...
7230	To Catch a Thief	PG	A jewel thief is at large on the Riviera and a...
595	Thunderball	PG	Thunderball finds James Bond matching wits wit...
28151	The Last Tycoon	PG	Based on an unfinished novel by Scott Fitzgera...
23078	Muppets Most Wanted	PG	Disney Muppets Most Wanted takes the entire Mu...
1042	The Deep	PG	Peter Benchley who wrote Jaws also wrote The D...

In [60]:

```
fig = plt.figure(figsize=(8,6))
dataset_df.groupby('Rating')['Description'].count().plot.bar(ylim=0)
plt.show()
```



Training model

In [61]:
```
training_df = pickle.load( open('./data/dataset_df_cleaned_n_sample.pkl', 'rb') )
```

In [62]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.preprocessing.text import one_hot, Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Dense, Flatten, LSTM, Embedding, GlobalMaxPooling1D
```

In [63]:
```
_y = training_df['Rating']
encoder = LabelEncoder()
Y = encoder.fit_transform(_y)
Y
```

Out[63]:
```
array([4, 4, 4, ..., 5, 4, 4])
```

In [64]:
```
X = training_df['Description']
X
```

Out[64]:
```
0        The Fast and the Furious director Rob Cohen co...
1        The journey of professional wrestler who becom...
2        Dead pan Leslie Nielsen stars as Dick Steele a...
3        Director Uli Edel teams with screenwriter Bern...
4        One of cluster of late films about the Vietnam...
                               ...
29805    Filmed at least nine times over the last nine ...
29806    Fred MacMurray stars in this Walt Disney comed...
29807    A resident of rd century Earth becomes involve...
29808    Supernova chronicles the search and rescue pat...
29809    For years there have been documented cases of ...
Name: Description, Length: 29810, dtype: object
```

Train, test split

In [65]:
```
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.3, random_state=40 )
```

In [66]:
```
Y_train = to_categorical(Y_train)
Y_test  = to_categorical(Y_test)
```

In [67]:
```
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(X_train)
```

In [68]:
```
X_train = tokenizer.texts_to_sequences(X_train)
X_test  = tokenizer.texts_to_sequences(X_test)
```

In [69]:
```
vocabulary_size = len(tokenizer.word_index) + 1
maxlen = 400 # Only concider first 400 words of each description

X_train = pad_sequences( X_train, maxlen=maxlen )
X_test  = pad_sequences( X_test, maxlen=maxlen )
```

Use GloVe : Global Vectors for Word Representation

https://nlp.stanford.edu/projects/glove/

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

In [70]:
```
glove_file = open('./glove.6B/glove.6B.100d.txt', encoding='utf8')
```

In [71]:
```
embedd_dictionary = dict()
for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dims = np.asarray(records[1:], dtype='float32')
    embedd_dictionary[word] = vector_dims

glove_file.close()
```

In [72]:
```
embedd_matrix = np.zeros((vocabulary_size, 100))
for word, index in tokenizer.word_index.items():
    embedd_vector = embedd_dictionary.get(word)
    if embedd_vector is not None:
        embedd_matrix[index] = embedd_vector
```

Build the NN Model

In [73]:
```
model = Sequential()
model.add(Embedding(input_dim=vocabulary_size, output_dim=100, weights=[embedd_matrix], trainable=False))
model.add(LSTM(units=128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(6, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         5635900
_________________________________________________________________
lstm (LSTM)                  (None, 128)               117248
_________________________________________________________________
dense (Dense)                (None, 6)                 774
=================================================================
Total params: 5,753,922
Trainable params: 118,022
Non-trainable params: 5,635,900
_________________________________________________________________
```

In [74]:
```
hist = model.fit(X_train, Y_train, batch_size=128, epochs=40, validation_split=0.30, verbose=1)
```

```
Epoch 1/40
115/115 [==============================] - 76s 642ms/step - loss: 1.4920 - accuracy: 0.3615 - val_loss: 1.43
05 - val_accuracy: 0.3971
Epoch 2/40
115/115 [==============================] - 74s 641ms/step - loss: 1.3835 - accuracy: 0.4085 - val_loss: 1.32
15 - val_accuracy: 0.4344
Epoch 3/40
115/115 [==============================] - 72s 630ms/step - loss: 1.3201 - accuracy: 0.4406 - val_loss: 1.31
28 - val_accuracy: 0.4485
Epoch 4/40
115/115 [==============================] - 72s 630ms/step - loss: 1.2818 - accuracy: 0.4600 - val_loss: 1.27
61 - val_accuracy: 0.4659
Epoch 5/40
115/115 [==============================] - 72s 630ms/step - loss: 1.2419 - accuracy: 0.4852 - val_loss: 1.21
55 - val_accuracy: 0.4871
Epoch 6/40
115/115 [==============================] - 71s 614ms/step - loss: 1.2107 - accuracy: 0.5002 - val_loss: 1.18
61 - val_accuracy: 0.5071
```

```
Epoch 7/40
115/115 [==============================] - 69s 605ms/step - loss: 1.1605 - accuracy: 0.5181 - val_loss: 1.20
44 - val_accuracy: 0.4868
Epoch 8/40
115/115 [==============================] - 73s 636ms/step - loss: 1.1490 - accuracy: 0.5302 - val_loss: 1.18
36 - val_accuracy: 0.5103
Epoch 9/40
115/115 [==============================] - 74s 644ms/step - loss: 1.1229 - accuracy: 0.5482 - val_loss: 1.14
77 - val_accuracy: 0.5269
Epoch 10/40
115/115 [==============================] - 75s 656ms/step - loss: 1.0768 - accuracy: 0.5696 - val_loss: 1.13
00 - val_accuracy: 0.5435
Epoch 11/40
115/115 [==============================] - 71s 614ms/step - loss: 1.0424 - accuracy: 0.5882 - val_loss: 1.11
83 - val_accuracy: 0.5513
Epoch 12/40
115/115 [==============================] - 72s 624ms/step - loss: 1.0190 - accuracy: 0.6024 - val_loss: 1.11
61 - val_accuracy: 0.5470
Epoch 13/40
115/115 [==============================] - 74s 645ms/step - loss: 0.9627 - accuracy: 0.6211 - val_loss: 1.10
31 - val_accuracy: 0.5542
Epoch 14/40
115/115 [==============================] - 75s 654ms/step - loss: 0.9354 - accuracy: 0.6372 - val_loss: 1.07
39 - val_accuracy: 0.5672
Epoch 15/40
115/115 [==============================] - 86s 753ms/step - loss: 0.9188 - accuracy: 0.6440 - val_loss: 1.11
39 - val_accuracy: 0.5550
Epoch 16/40
115/115 [==============================] - 84s 727ms/step - loss: 0.8727 - accuracy: 0.6672 - val_loss: 1.04
99 - val_accuracy: 0.5884
Epoch 17/40
115/115 [==============================] - 82s 711ms/step - loss: 0.8178 - accuracy: 0.6841 - val_loss: 1.05
10 - val_accuracy: 0.5894
Epoch 18/40
115/115 [==============================] - 82s 712ms/step - loss: 0.7870 - accuracy: 0.7005 - val_loss: 1.03
06 - val_accuracy: 0.6039
Epoch 19/40
115/115 [==============================] - 77s 668ms/step - loss: 0.7525 - accuracy: 0.7171 - val_loss: 1.00
46 - val_accuracy: 0.6154
Epoch 20/40
115/115 [==============================] - 73s 634ms/step - loss: 0.7301 - accuracy: 0.7268 - val_loss: 1.01
91 - val_accuracy: 0.6210
Epoch 21/40
115/115 [==============================] - 76s 658ms/step - loss: 0.6973 - accuracy: 0.7409 - val_loss: 1.02
88 - val_accuracy: 0.6162
```

```
Epoch 22/40
115/115 [==============================] - 75s 653ms/step - loss: 0.6710 - accuracy: 0.7506 - val_loss: 0.98
73 - val_accuracy: 0.6411
Epoch 23/40
115/115 [==============================] - 72s 629ms/step - loss: 0.6353 - accuracy: 0.7664 - val_loss: 0.99
19 - val_accuracy: 0.6430
Epoch 24/40
115/115 [==============================] - 73s 638ms/step - loss: 0.6199 - accuracy: 0.7731 - val_loss: 0.97
78 - val_accuracy: 0.6513
Epoch 25/40
115/115 [==============================] - 74s 647ms/step - loss: 0.5878 - accuracy: 0.7837 - val_loss: 1.02
81 - val_accuracy: 0.6505
Epoch 26/40
115/115 [==============================] - 75s 652ms/step - loss: 0.5625 - accuracy: 0.7996 - val_loss: 0.95
84 - val_accuracy: 0.6668
Epoch 27/40
115/115 [==============================] - 79s 692ms/step - loss: 0.5344 - accuracy: 0.8051 - val_loss: 0.98
48 - val_accuracy: 0.6702
Epoch 28/40
115/115 [==============================] - 72s 626ms/step - loss: 0.5236 - accuracy: 0.8102 - val_loss: 1.01
97 - val_accuracy: 0.6679
Epoch 29/40
115/115 [==============================] - 73s 637ms/step - loss: 0.4978 - accuracy: 0.8216 - val_loss: 0.99
57 - val_accuracy: 0.6750
Epoch 30/40
115/115 [==============================] - 71s 620ms/step - loss: 0.4633 - accuracy: 0.8351 - val_loss: 0.99
60 - val_accuracy: 0.6783
Epoch 31/40
115/115 [==============================] - 75s 651ms/step - loss: 0.4585 - accuracy: 0.8365 - val_loss: 0.99
88 - val_accuracy: 0.6916
Epoch 32/40
115/115 [==============================] - 75s 655ms/step - loss: 0.4511 - accuracy: 0.8394 - val_loss: 0.98
53 - val_accuracy: 0.6949
Epoch 33/40
115/115 [==============================] - 74s 647ms/step - loss: 0.4263 - accuracy: 0.8481 - val_loss: 0.98
67 - val_accuracy: 0.6975
Epoch 34/40
115/115 [==============================] - 73s 634ms/step - loss: 0.4182 - accuracy: 0.8548 - val_loss: 0.96
12 - val_accuracy: 0.7015
Epoch 35/40
115/115 [==============================] - 75s 657ms/step - loss: 0.3914 - accuracy: 0.8614 - val_loss: 0.99
55 - val_accuracy: 0.7005
Epoch 36/40
115/115 [==============================] - 72s 625ms/step - loss: 0.3897 - accuracy: 0.8582 - val_loss: 1.01
89 - val_accuracy: 0.7037
```

```
Epoch 37/40
115/115 [==============================] - 72s 629ms/step - loss: 0.3743 - accuracy: 0.8687 - val_loss: 1.00
39 - val_accuracy: 0.7123
Epoch 38/40
115/115 [==============================] - 73s 631ms/step - loss: 0.3783 - accuracy: 0.8697 - val_loss: 0.99
35 - val_accuracy: 0.7114
Epoch 39/40
115/115 [==============================] - 73s 635ms/step - loss: 0.3302 - accuracy: 0.8847 - val_loss: 1.02
14 - val_accuracy: 0.7120
Epoch 40/40
115/115 [==============================] - 72s 624ms/step - loss: 0.3354 - accuracy: 0.8826 - val_loss: 0.99
```

In [75]:

```
loss, acc = model.evaluate(X_test, Y_test, verbose=1)
print('loss: ', loss)
print('accuracy: ', acc)
```

```
280/280 [==============================] - 15s 54ms/step - loss: 1.0078 - accuracy: 0.7178
loss:  1.0077930688858032
accuracy:  0.7177680730819702
```

Save model and tokenizer

In [76]:

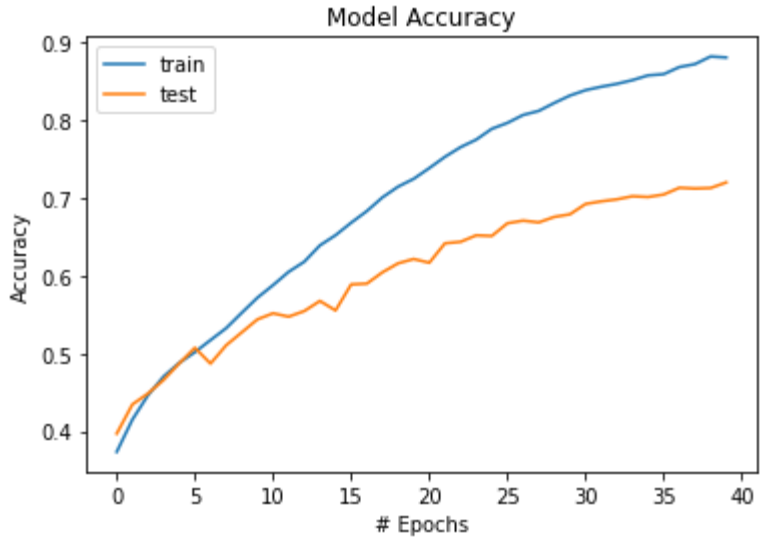
```
# save model
model.save('./data/mpaa_classifier_n_sample.h5')

# save word tokenizer
pickle.dump(tokenizer, open('./data/tokenizer_n_sample.pkl', 'wb'))
```

Metrics

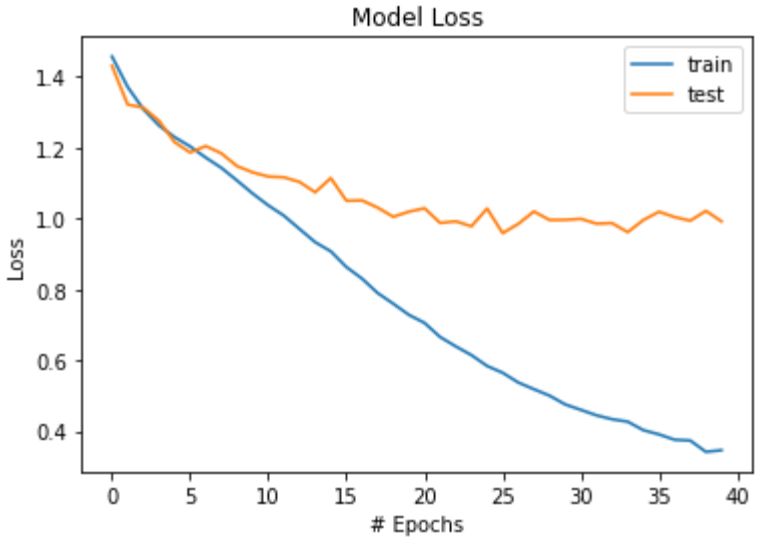
In [77]:

```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('# Epochs')
plt.legend(['train', 'test'])
plt.show()
```



In [78]:

```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('# Epochs')
plt.legend(['train', 'test'])
plt.show()
```


Model Loss
train
test
Loss
Epochs
1.4
1.2
1.0
0.8
0.6
0.4
0
5
10
15
20
25
30
35
40

In [ ]: