```
!pip install PyDrive
```

```
Requirement already satisfied: PyDrive in /usr/local/lib/python3.6/dist-packages (1.3.1)
Requirement already satisfied: google-api-python-client>=1.2 in /usr/local/lib/python3.6/dist-packages (from PyDri
Requirement already satisfied: PyYAML>=3.0 in /usr/local/lib/python3.6/dist-packages (from PyDrive) (3.13)
Requirement already satisfied: oauth2client>=4.0.0 in /usr/local/lib/python3.6/dist-packages (from PyDrive) (4.1.3
Requirement already satisfied: six<2dev,>=1.6.1 in /usr/local/lib/python3.6/dist-packages (from google-api-python-
Requirement already satisfied: httplib2<1dev,>=0.9.2 in /usr/local/lib/python3.6/dist-packages (from google-api-py
Requirement already satisfied: uritemplate<4dev,>=3.0.0 in /usr/local/lib/python3.6/dist-packages (from google-api
Requirement already satisfied: pyasn1>=0.1.7 in /usr/local/lib/python3.6/dist-packages (from oauth2client>=4.0.0->
Requirement already satisfied: rsa>=3.1.4 in /usr/local/lib/python3.6/dist-packages (from oauth2client>=4.0.0->PyD
Requirement already satisfied: pyasn1-modules>=0.0.5 in /usr/local/lib/python3.6/dist-packages (from oauth2client>
```

```
import os
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

```
%cd "gdrive/My Drive/project"
```

```
/content/gdrive/My Drive/project
```

```
!ls
!python
```

```
data              glove.6B.100d.txt   mltools            pre-trained
data_loader.py    glove6b100dtxt.zip  models             __pycache__
embeddings.h5     history_cnn.eps     nblr-svm           rnn
figure            lgbm                n_words_clean.eps  wiki.en.vec
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
>>> nltk.download('wordnet')
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
True
>>> nltk.download('stopwords')
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Import Data

```
import pandas as pd
import h5py
import numpy as np
from nltk.corpus import stopwords
import string
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad sequences
from nltk.stem.wordnet import WordNetLemmatizer
import data_loader
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
plt.set_cmap('RdYlBu')
```

```
Using TensorFlow backend.
<Figure size 432x288 with 0 Axes>
```

```
train, valid = data_loader.load_train_data('data/train.csv')
test = data_loader.load_test_data('data/test.csv','data/test_labels.csv')
```

```
list_classes = ["toxic", "severe_toxic", "obscene", "threat", "insult", "identity_hate"]
```

```
train_y = train[list_classes].values
valid_y = valid[list_classes].values
test_y = test[list_classes].values


train = train.fillna('')
valid = valid.fillna('')
test = test.fillna('')
```

Data Exploration

```
print(train.shape)
print(valid.shape)
print(test.shape)
```

```
(143645, 8)
(15926, 8)
(63978, 8)
```

```
print(train.dtypes)
```

```
id                object
comment_text      object
toxic              int64
severe_toxic       int64
obscene            int64
threat             int64
insult             int64
identity_hate      int64
dtype: object
```

```
print(train[0:5])
```

```
                        id  ... identity_hate
130060  b7bf5a6846bd456a  ...             0
```

```python
print(test[0:5])
```

```
                    id  ... identity_hate
5   0001ea8717f6de06  ...             0
7   000247e83dcc1211  ...             0
11  0002f87b16116a7f  ...             0
13  0003e1cccfd5a40a  ...             0
14  00059ace3e3e9a53  ...             0

[5 rows x 8 columns]
```

```python
# counting frequency of occurence of multi-labelled data
ct0,ct1,ct2 = 0,0,0
label = train[['toxic', 'severe_toxic' , 'obscene' , 'threat' , 'insult' , 'identity_hate']]
label = label.as_matrix()
for i in range(label.shape[0]):
    ct = np.count_nonzero(label[i])
    if ct :
        ct1 = ct1+1
    else:
        ct0 = ct0+1
    if ct>1 :
        ct2 = ct2+1
print("Train samples with no label:", ct0)
print("Train samples with atleast one label:", ct1)
print("Train samples with 2 or more labels", ct2)
```

```
Train samples with no label: 129006
Train samples with atleast one label: 14639
Train samples with 2 or more labels 8941
```

```python
# Explore the vocabulary
import collections
from tqdm import tqdm

x_train = train.comment_text.copy()
# Create a counter object for each dataset
word_counter = collections.Counter([word for sentence in tqdm(x_train, total=len(x_train)) \
                                                                  for word in sentence.split()])

print('{} words.'.format(len([word for sentence in x_train for word in sentence.split()])))
print('{} unique words.'.format(len(word_counter)))
print('10 Most common words in the dataset:')
print('"' + '" "'.join(list(zip(*word_counter.most_common(10)))[0]) + '"')
```

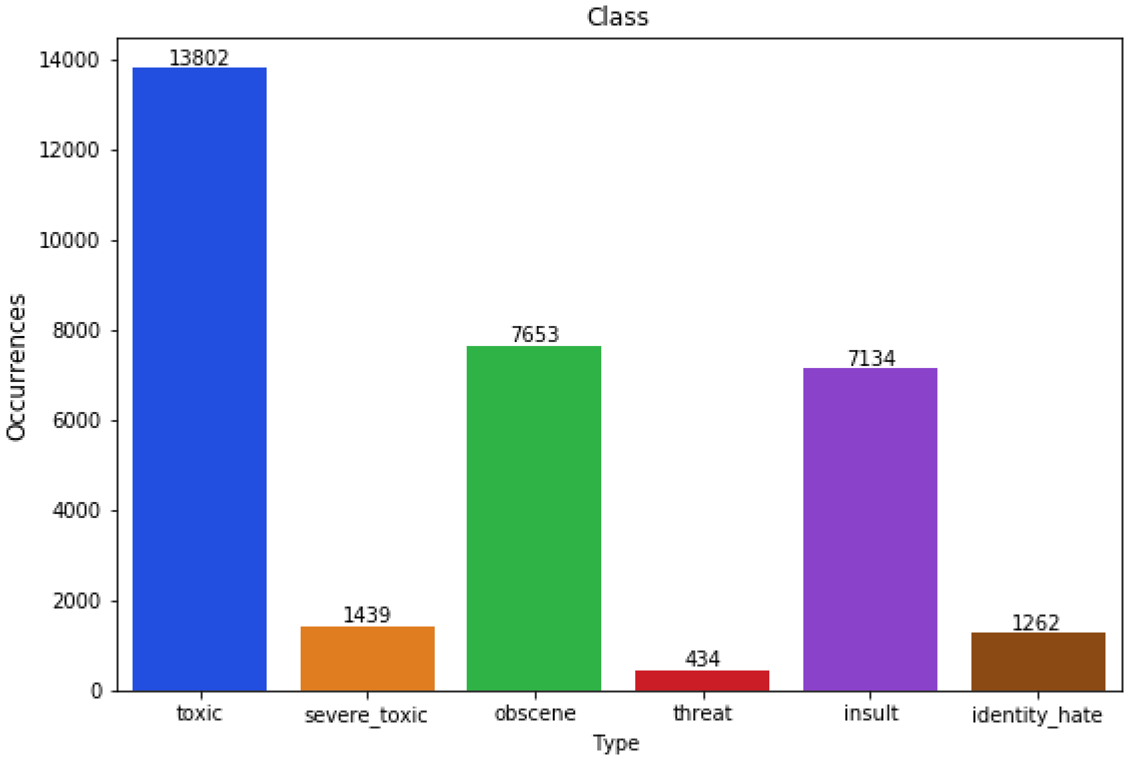
```
100%|██████████| 143645/143645 [00:01<00:00, 93796.01it/s]
9655027 words.
496117 unique words.
10 Most common words in the dataset:
"the" "to" "of" "and" "a" "I" "is" "you" "that" "in"
```

One problem here is that we are counting uppercase words as different from lower case words and a bunch of other symbols that aren't really useful for our goal. A data cleanup will be done in the next step.

```
# visualizing the comment size
comment = train['comment_text']
comment = comment.as_matrix()
x = [len(comment[i]) for i in range(comment.shape[0])]
print('average length of comment: {:.3f}'.format(sum(x)/len(x)) )
bins = [1,200,400,600,800,1000,1200]
plt.hist(x, bins=bins)
plt.xlabel('Length of comments')
plt.ylabel('Number of comments')
plt.axis([0, 1200, 0, 90000])
plt.grid(True)
plt.show()
```

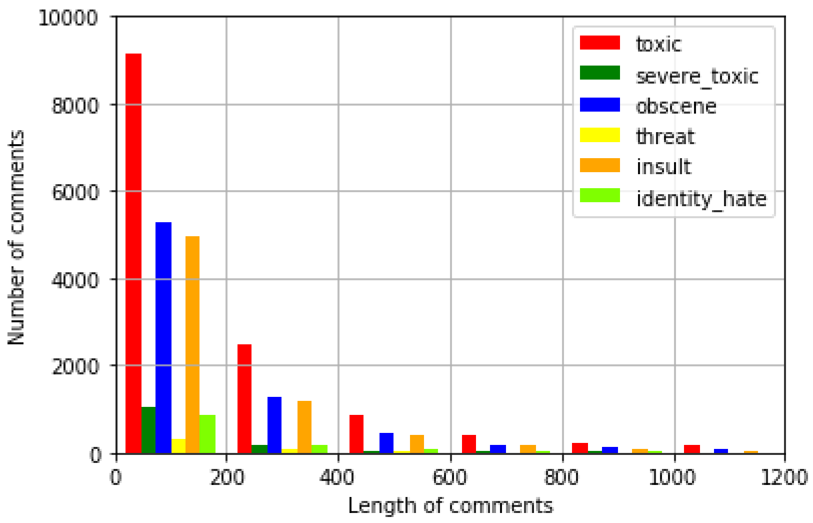
```
    average length of comment: 393.746
```

```
import seaborn as sns
# visualizing the no. of comments of each category
palette= sns.color_palette("bright")
x=train.iloc[:,2:].sum()
plt.figure(figsize=(9,6))
ax= sns.barplot(x.index, x.values, palette=palette)
plt.title("Class")
plt.ylabel('Occurrences', fontsize=12)
plt.xlabel('Type ')
rects = ax.patches
xlabels = x.values
for rect, lbl in zip(rects, xlabels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 10, lbl,
            ha='center', va='bottom')

plt.show()
```



```
# No. of comments of each type grouped by lengths
```

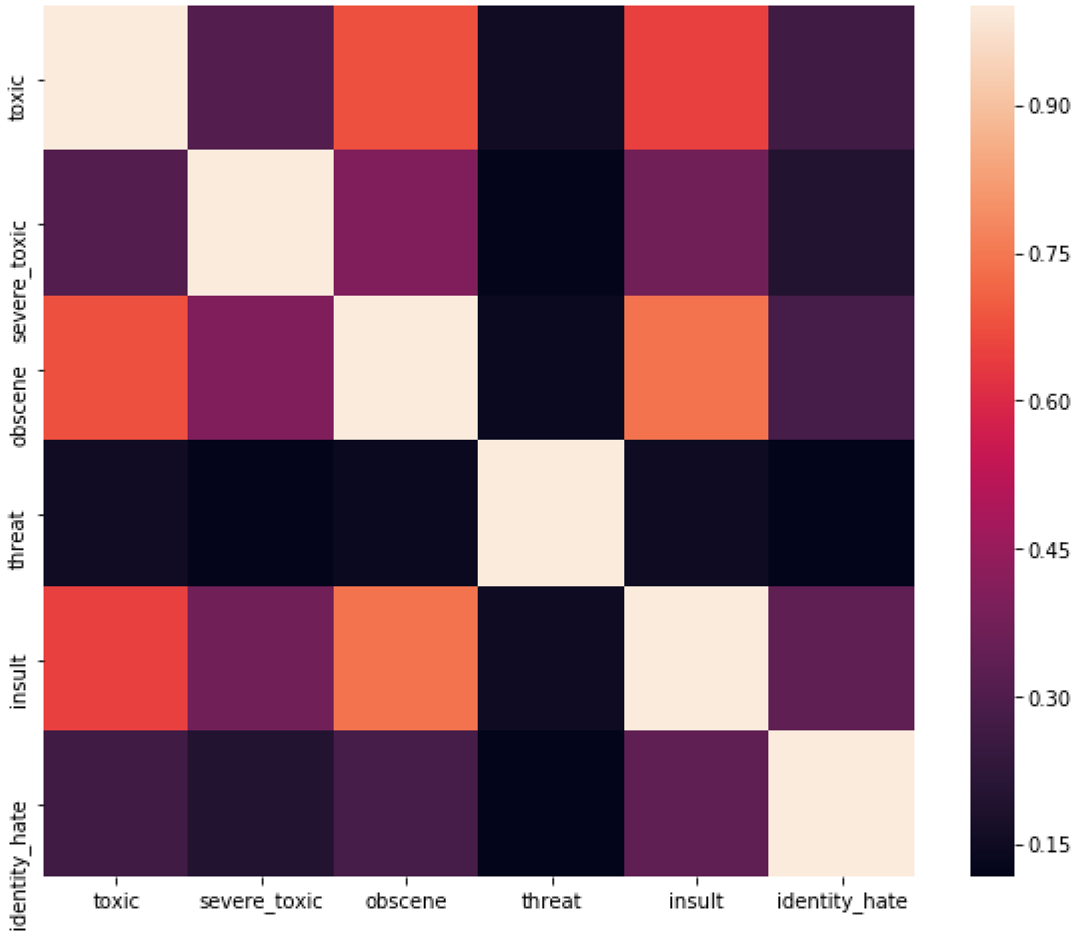
```
y = np.zeros(label.shape)
for ix in range(comment.shape[0]):
    l = len(comment[ix])
    if label[ix][0] :
        y[ix][0] = l
    if label[ix][1] :
        y[ix][1] = l
    if label[ix][2] :
        y[ix][2] = l
    if label[ix][3] :
        y[ix][3] = l
    if label[ix][4] :
        y[ix][4] = l
    if label[ix][5] :
        y[ix][5] = l

labelsplt = ['toxic','severe_toxic','obscene','threat','insult','identity_hate']
color = ['red','green','blue','yellow','orange','chartreuse']
plt.hist(y,bins = bins,label = labelsplt,color = color)
plt.axis([0, 1200, 0, 10000])
plt.xlabel('Length of comments')
plt.ylabel('Number of comments')
plt.legend()
plt.grid(True)
plt.show()
```



```
# correlation matrix between features
f, ax = plt.subplots(figsize=(10, 8))
corr = train.corr()
```

```
corr.style.background_gradient()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
            square=True, ax=ax, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb89c89b860>
```



Prepare the Data

Cleaning the data

```
"""this function receives comments and returns clean word-list
  split words by witespace, remove punctuations, change letters to lower case,
  remove words that are not alphanumeric, remove 1-letter words
"""
def pre_process(word_text):
    tokens = word_text.split()
    table = str.maketrans({key: None for key in string.punctuation})
    tokens = [token.translate(table) for token in tokens]
    tokens = [token for token in tokens if token.isalpha()]
    tokens = [token.lower() for token in tokens]
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]
    tokens = [token for token in tokens if len(token) > 1]
    lem = WordNetLemmatizer()
    tokens = [lem.lemmatize(token,"v") for token in tokens]
    sentence = ' '.join(tokens)
    return sentence
```

```
"""clean comment_text from the trainning set and testing set"""
train_comment_text = train.comment_text.copy()
valid_comment_text = valid.comment_text.copy()
test_comment_text = test.comment_text.copy()

train_text_processed = [pre_process(comment) for comment in train_comment_text]
valid_text_processed = [pre_process(comment) for comment in valid_comment_text]
test_text_processed = [pre_process(comment) for comment in test_comment_text]
```

```
print('The 0th comment text in unprocessed training set:')
print(train_comment_text.iloc[0])
print('\n')
print('The 0th comment text in clean training set:')
print(train_text_processed[0])
print('\n')
print('The 0th comment text in unprocessed validation set:')
print(valid_comment_text.iloc[0])
print('\n')
print('The 0th comment text in clean validation set:')
print(valid_text_processed[0])
print('\n')
print('The 0th comment text in unprocessed test set:')
print(test_comment_text.iloc[0])
print('\n')
print('The 0th comment text in clean test set:')
print(test_text_processed[0])
```

```
The 0th comment text in unprocessed training set:
"
 Oppose. WP:MOSTM, the guideline covering trademarks and brands, explicitly states not to do this. We don't need a


The 0th comment text in clean training set:
oppose wpmostm guideline cover trademark brand explicitly state dont need article title like realtor time etc desp


The 0th comment text in unprocessed validation set:
Risk factors

The role of chlamydia should be discussed, as with the increasing ectopic pregnancy rate with the increasing chlam


The 0th comment text in clean validation set:
risk factor role chlamydia discuss increase ectopic pregnancy rate increase chlamydia incidence iuds also link inc


The 0th comment text in unprocessed test set:
Thank you for understanding. I think very highly of you and would not revert without discussion.


The 0th comment text in clean test set:
```

```
df_train = pd.DataFrame(data={"comment_text": train_text_processed})
df_train.to_csv("data/cleaned_train.csv", sep=',',index=False)

df_valid = pd.DataFrame(data={"comment_text": valid_text_processed})
df_valid.to_csv("data/cleaned_valid.csv", sep=',',index=False)

df_test = pd.DataFrame(data={"comment_text": test_text_processed})
df_test.to_csv("data/cleaned_test.csv", sep=',',index=False)
```

Tokenizing and embedding

Tokenizing

```
embedding_dim = 300
```

```
# Tokenize and Pad

# Create tokenizer
tokenizer = Tokenizer()

# Fit and run tokenizer
tokenizer.fit_on_texts(train_text_processed + valid_text_processed  + test_text_processed)
tokenized_train = tokenizer.texts_to_sequences(train_text_processed)
tokenized_valid = tokenizer.texts_to_sequences(valid_text_processed)
tokenized_test = tokenizer.texts_to_sequences(test_text_processed)
word_index = tokenizer.word_index

# Extract variables
vocab_size = len(word_index)
print('Vocab size: {}'.format(vocab_size))
longest = max(len(seq) for seq in tokenized_train)
print("Longest comment size: {}".format(longest))
average = np.mean([len(seq) for seq in tokenized_train])
print("Average comment size: {}".format(average))
stdev = np.std([len(seq) for seq in tokenized_train])
print("Stdev of comment size: {}".format(stdev))
max_len = int(average + stdev * 3)
print('Max comment size: {}'.format(max_len))
print()

# Pad sequences
processed_X_train = pad_sequences(tokenized_train, maxlen=max_len, padding='post', truncating='post')
processed_X_valid = pad_sequences(tokenized_valid, maxlen=max_len, padding='post', truncating='post')
processed_X_test = pad_sequences(tokenized_test, maxlen=max_len, padding='post', truncating='post')

# Sample tokenization
for sample_i, (sent, token_sent) in enumerate(zip(train_text_processed[:2], tokenized_train[:2])):
    print('Sequence {}'.format(sample_i + 1))
    print('  Input:  {}'.format(sent))
    print('  Output: {}'.format(token_sent))
```

```
Vocab size: 287514
Longest comment size: 1250
Average comment size: 23.820040437407845
```

Embedding - Fasttext

```python
embedding_dim = 300

# Get embeddings
embeddings_index = {}
f = open('wiki.en.vec', encoding="utf8")
for line in f:
    values = line.rstrip().rsplit(' ', embedding_dim)
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found {} word vectors.'.format(len(embeddings_index)))
```

```
Found 2519371 word vectors.
```

```python
# Build embedding matrix
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector


# Save embeddings
with h5py.File('embeddings.h5', 'w') as hf:
    hf.create_dataset("fasttext",  data=embedding_matrix)
```

Creating model and training model

```python
# Load embeddings
with h5py.File('embeddings.h5', 'r') as hf:
    embedding_matrix = hf['fasttext'][:]
```

RNN with LSTM

Creating Model

```
#### RNN with LSTM

import keras.backend
from keras.models import Sequential
from keras.layers import Dense, Conv1D, MaxPooling1D
from keras.layers import Dropout, GlobalMaxPooling1D, BatchNormalization
from keras.layers import Bidirectional
from keras.layers.embeddings import Embedding
from keras.optimizers import Nadam
from keras.layers.recurrent import LSTM

# Initate model
model = Sequential()

# Add Embedding layer
model.add(Embedding(vocab_size + 1, embedding_dim, weights=[embedding_matrix], input_length=max_len, trainable=True))

# Add Recurrent layer
model.add(LSTM(60, return_sequences=True, name='lstm_layer'))
model.add(Conv1D(filters=128, kernel_size=5, padding='same', activation='relu'))
model.add(MaxPooling1D(3))
model.add(GlobalMaxPooling1D())
model.add(BatchNormalization())

# Add fully connected layers
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(6, activation='sigmoid'))

# Summarize the model
model.summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W0614 02:38:38.917062 139681009899392 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/kera

W0614 02:38:38.961061 139681009899392 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/kera

W0614 02:38:38.967788 139681009899392 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/kera

W0614 02:38:38.978593 139681009899392 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/kera

W0614 02:38:38.979368 139681009899392 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/kera

W0614 02:38:44.129183 139681009899392 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/kera

W0614 02:38:44.494319 139681009899392 deprecation.py:506] From /usr/local/lib/python3.6/dist-packages/keras/backen
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 189, 300)          86254500  
_________________________________________________________________
lstm_layer (LSTM)            (None, 189, 60)           86640     
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 189, 128)          38528     
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 63, 128)           0         
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512       
_________________________________________________________________
dense_1 (Dense)              (None, 50)                6450      
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0         
```

Using binary crossentropy as the loss function and clipping gradients to avoid any explosions.

```
=================================================================
```

```python
def loss(y_true, y_pred):
    return keras.backend.binary_crossentropy(y_true, y_pred)

lr = .0001
model.compile(loss=loss, optimizer=Nadam(lr=lr, clipnorm=1.0),
              metrics=['binary_accuracy'])
```

```
W0614 02:38:44.555711 139681009899392 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/kera

W0614 02:38:44.567229 139681009899392 deprecation.py:323] From /usr/local/lib/python3.6/dist-packages/tensorflow/p
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

Training Model

```
# Evaluation Metric
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from keras.callbacks import Callback

class RocAucEvaluation(Callback):
    def __init__(self, filepath, validation_data=(), test_data=(), interval=1, max_epoch = 100):
        super(Callback, self).__init__()
        # Initialize state variables
        print("After init")
        self.interval = interval
        self.filepath = filepath
        self.stopped_epoch = max_epoch
        self.best = 0
        self.X_val, self.y_val = validation_data
        self.y_pred = np.zeros(self.y_val.shape)
        self.X_test, self.y_test = test_data
        self.y_test_pred = np.zeros(self.y_test.shape)

    def on_epoch_end(self, epoch, logs={}):
        print("Epoch end")
        if epoch % self.interval == 0:
            y_pred = self.model.predict_proba(self.X_val, verbose=0)
            current = roc_auc_score(self.y_val, y_pred)
            logs['roc_auc_val'] = current
            y_test_pred = self.model.predict_proba(self.X_test, verbose=0)
            current_test = roc_auc_score(self.y_test, y_test_pred)
            print("Test: {:.5f} Val:{:.5f}".format(current_test, current))

            if current > self.best: #save model
                print(" - AUC - improved from {:.5f} to {:.5f}".format(self.best, current))
                self.best = current
                self.y_pred = y_pred
                self.stopped_epoch = epoch+1
                self.model.save(self.filepath, overwrite=True)
            else:
                print(" - AUC - did not improve")


# Training

from keras.callbacks import EarlyStopping, ModelCheckpoint

print("Starting to train model...")
file_path = 'rnn/rnn_model_best.hdf5'
RocAuc_checkpoint = RocAucEvaluation(filepath=file_path,validation_data=(processed_X_valid, valid_y), test_data=(processed_
early_stop = EarlyStopping(monitor="roc_auc_val", mode="max", patience=3)
callbacks_list = [RocAuc_checkpoint, early_stop]

hist = model.fit(processed_X_train, train_y, epochs=5, batch_size=64, shuffle=False, validation_data=(processed_X_valid, va
                 callbacks = callbacks_list, verbose=1)
best_score = min(hist.history['val_loss'])
print("Final ACC. {}".format(best_score))
```

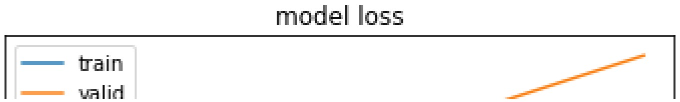
```
Starting to train model...
After init
Train on 143645 samples, validate on 15926 samples
Epoch 1/5
143645/143645 [==============================] - 892s 6ms/step - loss: 0.0449 - binary_accuracy: 0.9832 - val_loss
Epoch end
Test: 0.97252 Val:0.98014
 - AUC - improved from 0.00000 to 0.98014
Epoch 2/5
143645/143645 [==============================] - 900s 6ms/step - loss: 0.0393 - binary_accuracy: 0.9848 - val_loss
Epoch end
Test: 0.97244 Val:0.98004
 - AUC - did not improve
Epoch 3/5
143645/143645 [==============================] - 859s 6ms/step - loss: 0.0350 - binary_accuracy: 0.9863 - val_loss
Epoch end
Test: 0.97199 Val:0.97886
 - AUC - did not improve
Epoch 4/5
143645/143645 [==============================] - 852s 6ms/step - loss: 0.0317 - binary_accuracy: 0.9876 - val_loss
Epoch end
Test: 0.97009 Val:0.97731
 - AUC - did not improve
Final ACC. 0.04671059962809759
```

```
evaluation_cost = hist.history['val_loss']
evaluation_accuracy = hist.history['val_binary_accuracy']
training_cost = hist.history['loss']
training_accuracy = hist.history['binary_accuracy']

np.save("rnn/evaluation_cost.npy", evaluation_cost)
np.save("rnn/evaluation_accuracy.npy", evaluation_accuracy)
np.save("rnn/training_cost.npy", training_cost)
np.save("rnn/training_accuracy.npy", training_accuracy)
```

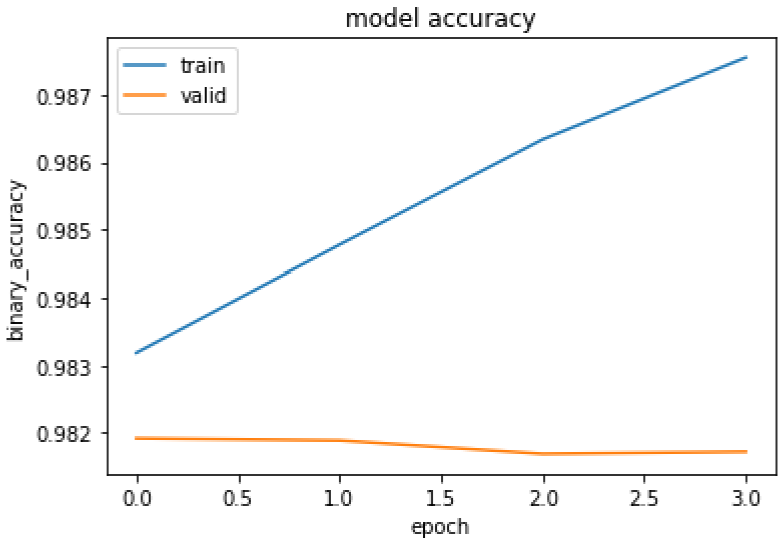
```
model.load_weights('rnn/rnn_model_best.hdf5')
print("Predicting results...")
test_predicts_path = "rnn/rnn_test_predicts.npy"
test_pred = model.predict(processed_X_test, batch_size=1024, verbose=1)
np.save(test_predicts_path, test_pred)
```

```
Predicting results...
63978/63978 [==============================] - 12s 184us/step
```

```
# Visualize history of loss
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```



```
# Visualize history of accuracy
plt.plot(hist.history['binary_accuracy'])
plt.plot(hist.history['val_binary_accuracy'])
plt.title('model accuracy')
plt.ylabel('binary_accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```



Submission

```
preds_loaded = np.load("rnn/rnn_test_predicts.npy")
predictions = pd.DataFrame(preds_loaded)
sample_submission = pd.read csv("data/sample_submission.csv")
sample_submission[list_classes] = predictions
sample_submission.to_csv("rnn/submission.csv", index=False)
```

```
## test roc auc score
from sklearn.metrics import roc_auc_score
class_names = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
```

```
preds_loaded = np.load("rnn/rnn_test_predicts.npy")
predictions = pd.DataFrame(preds_loaded)
test = data_loader.load_test_data('data/test.csv','data/test_labels.csv')
roc_auc_scores_test = 0
for class_name in class_names:
    score = roc_auc_score(test[class_name], predictions[class_name])
    roc_auc_scores_test += score
    print(score)
print("ROC AUC Test score:", roc_auc_scores_test/6)
```