```
!pip install emoji
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from tqdm import tqdm
from wordcloud import WordCloud
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
import string
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.metrics import roc_auc_score,roc_curve,auc,confusion_matrix,classification_re
from sklearn.manifold import TSNE
from sklearn import preprocessing
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
import re
from gensim.models import KeyedVectors
from wordcloud import WordCloud
from scipy.sparse import hstack
from sklearn.manifold import TSNE
from sklearn.preprocessing import Normalizer
import plotly.offline as py
import plotly
from plotly.offline import *
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
from sklearn.naive_bayes import MultinomialNB
import emoji
from tqdm.notebook import tqdm
tqdm.pandas()

import tensorflow
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Embedding, SpatialDropout1D, add, concat
from tensorflow.compat.v1.keras.layers import CuDNNLSTM, Bidirectional, GlobalMaxPooling1D
from tensorflow.keras.preprocessing import text, sequence
from gensim.models import KeyedVectors
from tensorflow.keras.utils import plot_model
!pip install pyLDAvis
from pprint import pprint

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
from gensim.parsing.preprocessing import STOPWORDS
```

```
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *

# spacy for lemmatization
import spacy

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt
%matplotlib inline

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERRO
import nltk
nltk.download('wordnet')
from gensim.models import LdaModel
import tensorflow as tf
from textblob import TextBlob, Word, Blobber
from sklearn.preprocessing import StandardScaler

!pip show tensorflow
!pip install plot_model
!pip install tensorboardcolab
%load_ext tensorboard
!rm -rf ./logs/
import warnings
warnings.filterwarnings("ignore")
```

```
Collecting emoji
  Downloading https://files.pythonhosted.org/packages/40/8d/521be7f0091fe0f2ae690cc04
     |████████████████████████████████| 51kB 1.2MB/s
Building wheels for collected packages: emoji
  Building wheel for emoji (setup.py) ... done
  Created wheel for emoji: filename=emoji-0.5.4-cp36-none-any.whl size=42176 sha256=c
  Stored in directory: /root/.cache/pip/wheels/2a/a9/0a/4f8e8cce8074232aba240caca3fad
Successfully built emoji
Installing collected packages: emoji
Successfully installed emoji-0.5.4
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnin
  import pandas.util.testing as tm
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
Collecting pyLDAvis
  Downloading https://files.pythonhosted.org/packages/a5/3a/af82e070a8a96e13217c8f362
     |████████████████████████████████| 1.6MB 2.7MB/s
Requirement already satisfied: wheel>=0.23.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: numpy>=1.9.2 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: scipy>=0.18.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: pandas>=0.17.0 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: joblib>=0.8.4 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: jinja2>=2.7.2 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: numexpr in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: pytest in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from
Collecting funcy
  Downloading https://files.pythonhosted.org/packages/ce/4b/6ffa76544e46614123de31574
     |████████████████████████████████| 552kB 14.9MB/s
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.6/dis
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: atomicwrites>=1.0 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: more-itertools>=4.0.0 in /usr/local/lib/python3.6/dist
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: pluggy<0.8,>=0.5 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: py>=1.5.0 in /usr/local/lib/python3.6/dist-packages (f
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (
Building wheels for collected packages: pyLDAvis, funcy
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473

Enter your authorization code:
··········
Mounted at /content/drive
```

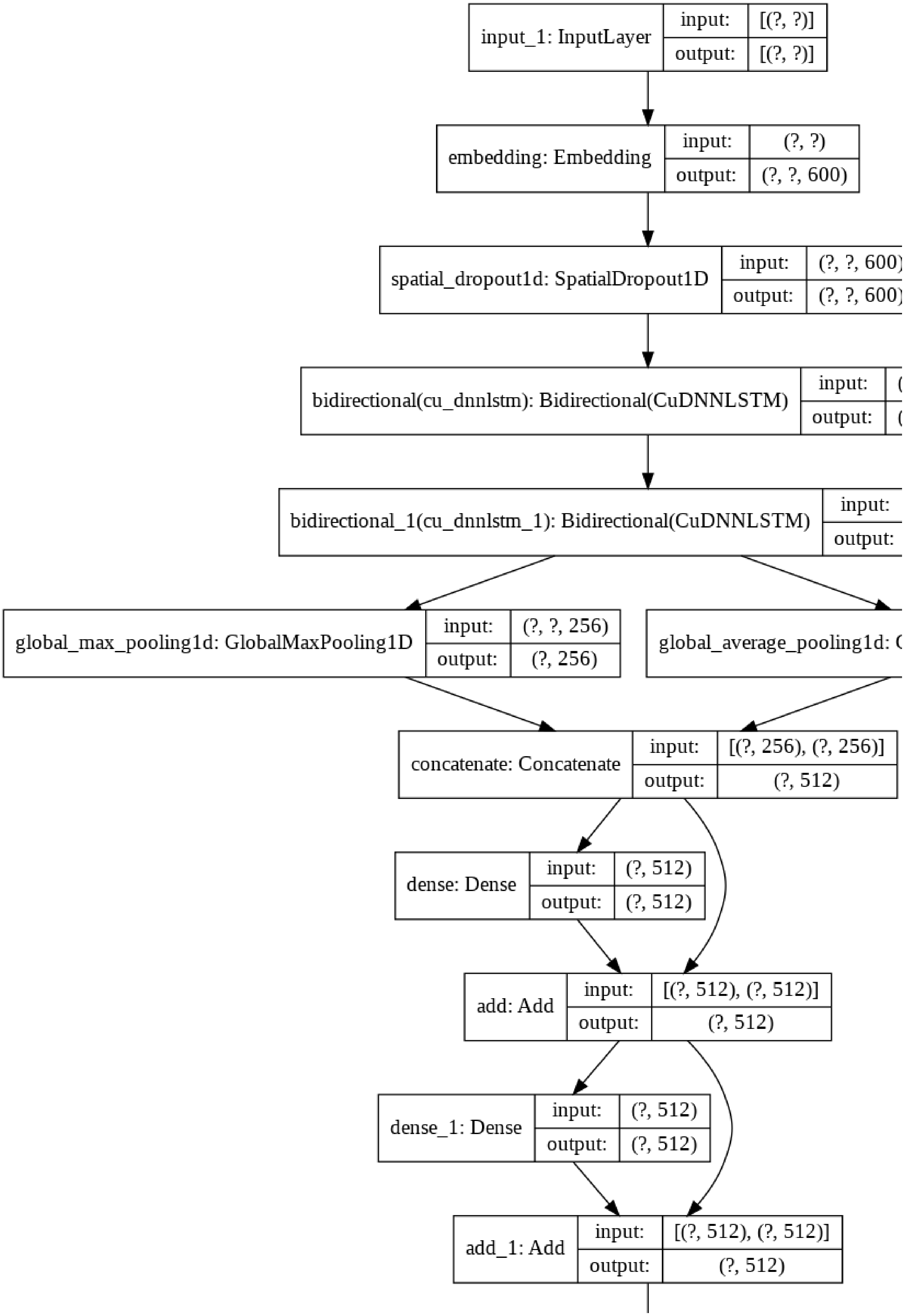
```
Name: tensorflow
```

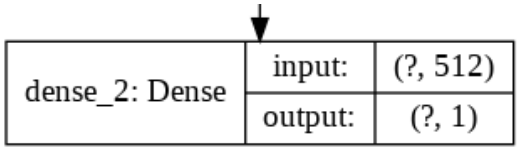
```
# Loading the train data into pandas dataframe
train = pd.read_csv('/content/drive/My Drive/train.csv')
# Loading the test data into pandas dataframe
test = pd.read_csv('/content/drive/My Drive/test.csv')
```

```
License: Apache 2.0
```

Deep Learning model 1: Using Only Text Feature

Downloading: https://files.pythonhosted.org/packages/62/78/c096/e50391a/c07002c5e7bca






```
#https://www.kaggle.com/thousandvoices/simple-lstm
EMBEDDING_FILES = [
    '/content/drive/My Drive/jigsaw/crawl-300d-2M.gensim',
    '/content/drive/My Drive/jigsaw/glove.840B.300d.gensim'
]
NUM_MODELS = 2
BATCH_SIZE = 512
LSTM_UNITS = 128
DENSE_HIDDEN_UNITS = 4 * LSTM_UNITS
EPOCHS = 4
MAX_LEN = 220
IDENTITY_COLUMNS = ['male', 'female', 'homosexual_gay_or_lesbian', 'christian', 'jewish',

AUX_COLUMNS = ['target', 'severe_toxicity', 'obscene', 'identity_attack', 'insult', 'threa
TEXT_COLUMN = 'comment_text'
TARGET_COLUMN='target'
CHARS_TO_REMOVE = '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n“”’\'∞θ÷α•à−β∅³π‘₹´°£€\×™√²—'

X_train, X_te = train_test_split(train, test_size=0.1,random_state=42)
X_train, X_cv = train_test_split(X_train, test_size=0.1,random_state=42)

x_train = X_train[TEXT_COLUMN].astype(str)
y_train = X_train[TARGET_COLUMN].values


x_cv = X_cv[TEXT_COLUMN].astype(str)
y_cv = X_cv[TARGET_COLUMN].values

x_te = X_te[TEXT_COLUMN].astype(str)
y_te = X_te[TARGET_COLUMN].values

for column in IDENTITY_COLUMNS + [TARGET_COLUMN]:
    X_train[column] = np.where(X_train[column] >= 0.5, True, False)
    X_cv[column] = np.where(X_cv[column] >= 0.5, True, False)
    X_te[column] = np.where(X_te[column] >= 0.5, True, False)


def build_matrix(word_index, path):
    embedding_index = KeyedVectors.load(path, mmap='r')
    embedding_matrix = np.zeros((len(word_index) + 1, 300))
    for word, i in word_index.items():
        for candidate in [word, word.lower()]:
            if candidate in embedding_index:
                embedding_matrix[i] = embedding_index[candidate]
                break
    return embedding_matrix


tokenizer = text.Tokenizer(filters=CHARS_TO_REMOVE, lower=False)
tokenizer.fit_on_texts(list(x_train))
```

```
tokenizer.fit_on_texts(list(x_train))

x_train = tokenizer.texts_to_sequences(x_train)
x_cv = tokenizer.texts_to_sequences(x_cv)
x_te = tokenizer.texts_to_sequences(x_te)

x_train = sequence.pad_sequences(x_train, maxlen=MAX_LEN)
x_cv = sequence.pad_sequences(x_cv, maxlen=MAX_LEN)
x_te = sequence.pad_sequences(x_te, maxlen=MAX_LEN)


#pickle.dump(tokenizer,open("/content/drive/My Drive/jigsaw/dltokenizer","wb"))
sample_weights = np.ones(len(x_train), dtype=np.float32)
sample_weights += X_train[IDENTITY_COLUMNS].sum(axis=1)
sample_weights += X_train[TARGET_COLUMN] * (~X_train[IDENTITY_COLUMNS]).sum(axis=1)
sample_weights += (~X_train[TARGET_COLUMN]) * X_train[IDENTITY_COLUMNS].sum(axis=1) * 5
sample_weights /= sample_weights.mean()

embedding_matrix = np.concatenate([build_matrix(tokenizer.word_index, f) for f in EMBEDDIN


  def build_model(embedding_matrix):
    words = Input(shape=(None,))
    x = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)(wo
    x = SpatialDropout1D(0.2)(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)

    hidden = concatenate([
        GlobalMaxPooling1D()(x),
        GlobalAveragePooling1D()(x),
    ])
    hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
    hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
    result = Dense(1, activation='sigmoid')(hidden)

    model = Model(inputs=words, outputs=result)
    model.compile(loss='binary_crossentropy', optimizer='adam')
    #plot_model(model, to_file='/content/drive/My Drive/jigsaw/model1.png', show_shapes=Tr
    return model


!pip show tensorflow
!pip install plot_model
!pip install tensorboardcolab
%load_ext tensorboard
!rm -rf ./logs/
import warnings
warnings.filterwarnings("ignore")
```

```
Name: tensorflow
Version: 2.2.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: tensorboard, absl-py, keras-preprocessing, opt-einsum, h5py, grpcio, numpy
```

```
checkpoint_predictions = []
weights = []
checkpoint = tensorflow.keras.callbacks.ModelCheckpoint('/content/drive/My Drive/jigsaw/Mo
log_dir="/content/drive/My Drive/jigsaw/Model11/logs/fit/" + datetime.now().strftime("%Y%m
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, w

model = build_model(embedding_matrix)
model.fit(x_train, y_train,batch_size=BATCH_SIZE,epochs=5,verbose=2,validation_data=(x_cv,
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoa
2020-05-31 10:31:57,422 : WARNING : `write_grads` will be ignored in TensorFlow 2.0 f
Epoch 1/5

Epoch 00001: val_loss improved from inf to 0.24552, saving model to /content/drive/My
2856/2856 - 829s - loss: 0.4226 - val_loss: 0.2455
Epoch 2/5

Epoch 00002: val_loss improved from 0.24552 to 0.24449, saving model to /content/driv
2856/2856 - 813s - loss: 0.4088 - val_loss: 0.2445
Epoch 3/5

Epoch 00003: val_loss improved from 0.24449 to 0.24227, saving model to /content/driv
2856/2856 - 813s - loss: 0.4039 - val_loss: 0.2423
Epoch 4/5

Epoch 00004: val_loss did not improve from 0.24227
2856/2856 - 809s - loss: 0.3998 - val_loss: 0.2475
Epoch 5/5

Epoch 00005: val_loss did not improve from 0.24227
2856/2856 - 808s - loss: 0.3963 - val_loss: 0.2436
<tensorflow.python.keras.callbacks.History at 0x7fa320058208>
```

```
%tensorboard --logdir='/content/drive/My Drive/jigsaw/Model11/logs/fit'
```

```
# load the model
model1=tensorflow.keras.models.load_model('/content/drive/My Drive/jigsaw/Model11.hdf5')

predictions=model1.predict(x_te, batch_size=2048).flatten()
MODEL_NAME = 'score'
X_te[MODEL_NAME] = predictions
bias_metrics_df = compute_bias_metrics_for_model(X_te, IDENTITY_COLUMNS, MODEL_NAME, TARGE
get_final_metric(bias_metrics_df, calculate_overall_auc(X_te, MODEL_NAME))
```

```
0.9319064728629746
```

Deep Learning Model 2 : Text feature + Additional Features

Additional Features:

- topic features
- positive word count
- negative word count
- sentiment of each comment
- word count,character count

Train cv split

```
X_train, X_te = train_test_split(train, test_size=0.1,random_state=42)

X_train, X_cv = train_test_split(X_train, test_size=0.1,random_state=42)
```

Topic modeling (Unsupervised Clustering Method)

- LDA (Latent Dirichlet Allocation) is an unsupervised machine-learning model that automatic and to derive hidden patterns exhibited by a text corpus. Thus, assisting better decision mak
- we will model our clean_text into 5 different topics and then take these topics as features.

```
#https://github.com/sonalijathar01/Toxic-comment-classification/blob/master/Jigsaw_UnInten
%%time
stemmer = SnowballStemmer("english")
def lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))

# Tokenize and lemmatize
def preprocess(text):
    '''this is for preprocessing text using gensim.utils.simple_preprocess() function'''
    result=[]
    for token in gensim.utils.simple_preprocess(text) :
        if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 3:
            result.append(lemmatize_stemming(token))

    return result


data = X_train.clean_text.values.tolist()
processed_docs = []
for doc in data:
    processed_docs.append(preprocess(doc))
```

```
CPU times: user 11min 3s, sys: 2.11 s, total: 11min 5s
```

Create the Dictionary and Corpus needed for Topic Modeling

```
%%time
# Create Dictionary
dictionary = gensim.corpora.Dictionary(processed_docs)

# Create Corpus
texts = processed_docs

# Term Document Frequency
# Gensim creates a unique id for each word in the document. The produced corpus shown abov
corpus = [dictionary.doc2bow(text) for text in texts]

# View
print(corpus[:1])
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]]
CPU times: user 1min 12s, sys: 1.14 s, total: 1min 13s
Wall time: 1min 13s
```

```
pickle.dump(dictionary,open('/content/drive/My Drive/jigsaw/dictionary','wb'))
#dictionary=pickle.load(open('/content/drive/My Drive/jigsaw/dictionary','rb'))
```

```
# Human readable format of corpus (term-frequency)
[[(dictionary[id], freq) for id, freq in cp] for cp in corpus[:1]]
```

```
[[('alaska', 1),
  ('career', 1),
  ('collegi', 1),
  ('good', 1),
  ('gymnast', 1),
  ('luck', 1),
  ('repres', 1),
  ('thank', 1)]]
```

Building topic model

```
#this code took me 10 hours to run

# Build LDA model
lda_model = LdaModel(corpus=corpus, id2word=dictionary, num_topics=5, random_state=100, up
                                           chunksize=100, passes=10, alpha='auto', per_wor
```

```
#pickle.dump(lda_model,open('/content/drive/My Drive/jigsaw/ldamodelall','wb'))
lda_model=pickle.load(open('/content/drive/My Drive/jigsaw/ldamodelall','rb'))
```

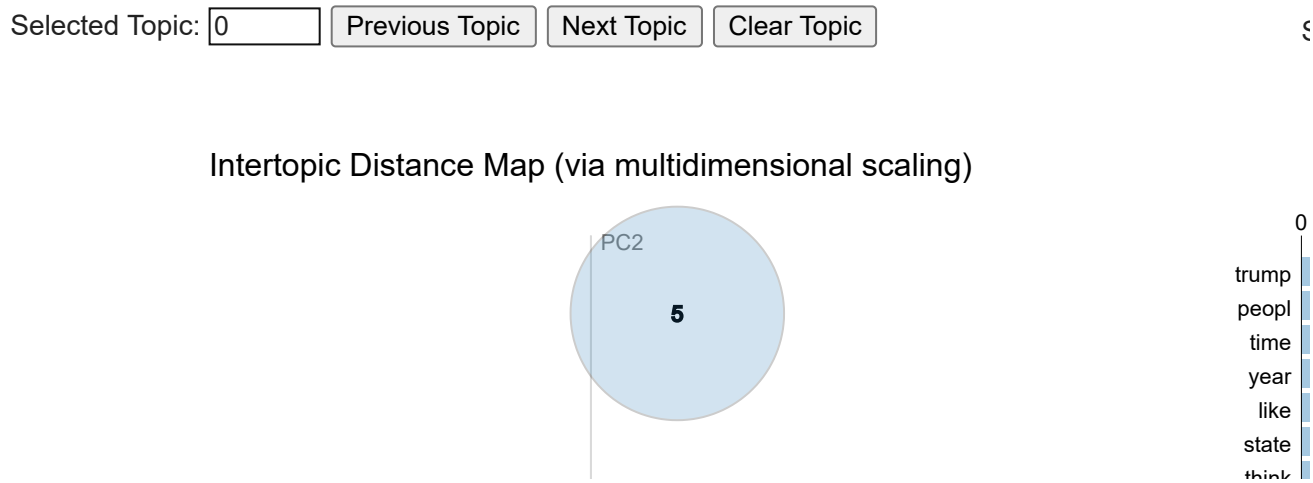
```
from gensim.models import CoherenceModel

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=processed_docs, dictionary=dic
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Coherence Score:  0.4943335322897525
```

```
# Print the Keyword in the 5 topics
print(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0, '0.012*"life" + 0.011*"million" + 0.011*"question" + 0.009*"women" + 0.009*"chur
```

```
# Visualize the topics on train
 pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, dictionary)
vis
```

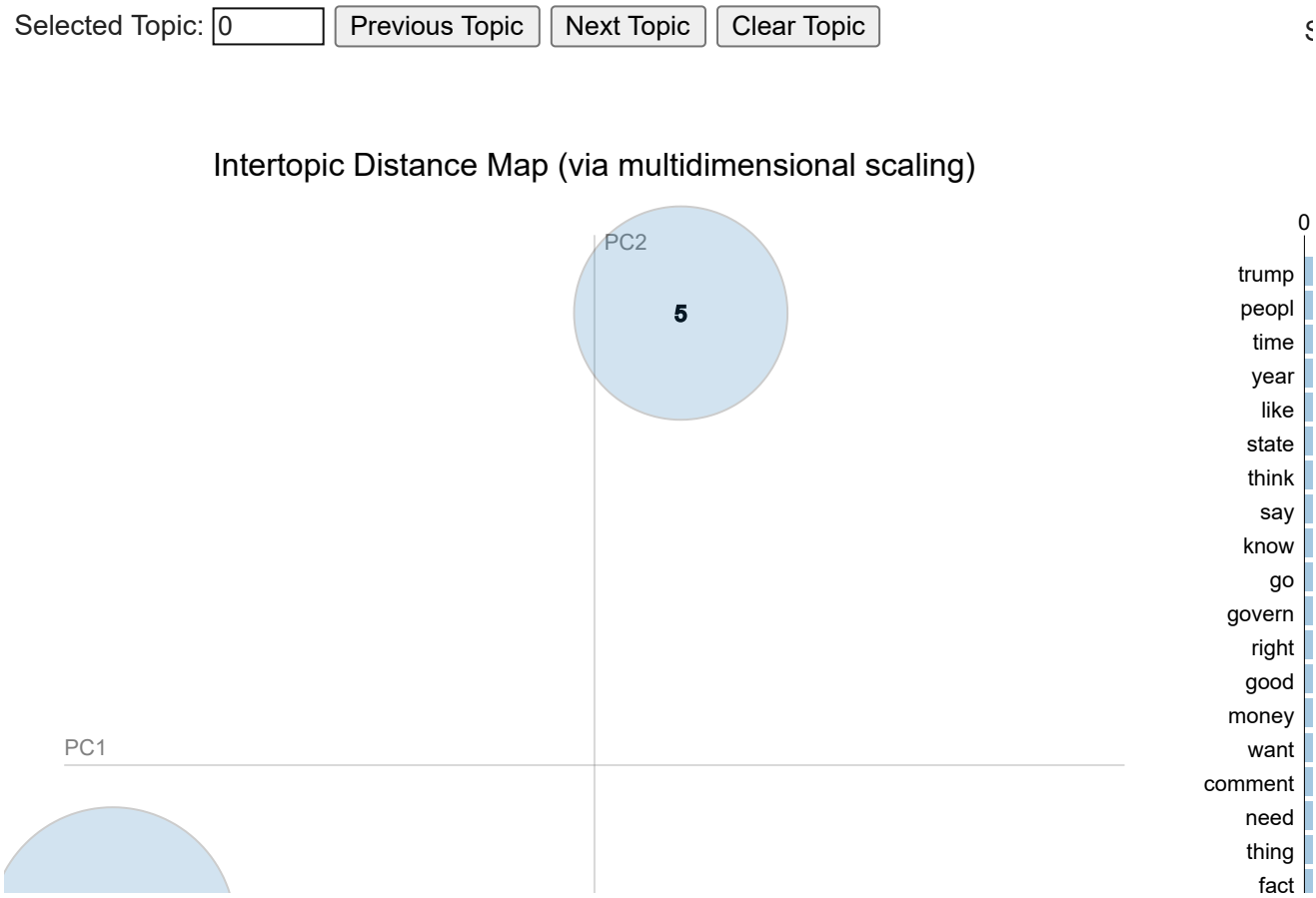


CV data preprocessing

```
%%time
data1 = X_cv.clean_text.values.tolist()
processed_docs1 = []
for doc in data1:
    processed_docs1.append(preprocess(doc))

print(processed_docs1[:1])
corpus1= [dictionary.doc2bow(text) for text in processed_docs1]
```

```
[['fals', 'doctrin', 'attack', 'catholic', 'contain', 'articl']]
CPU times: user 1min 19s, sys: 386 ms, total: 1min 19s
Wall time: 1min 19s
```

```
vis = pyLDAvis.gensim.prepare(lda_model, corpus1, dictionary)
vis
```



Test Data Preprocessing

look

```
%%time
data11 = X_te.clean_text.values.tolist()
processed_docs11 = []
for doc in data11:
    processed_docs11.append(preprocess(doc))

print(processed_docs11[:1])
corpus11= [dictionary.doc2bow(text) for text in processed_docs11]
```

```
[['breath', 'fresh', 'embrac', 'common', 'sens', 'valu', 'instead', 'leadership', 'ca
CPU times: user 1min 24s, sys: 393 ms, total: 1min 25s
Wall time: 1min 25s
```

Actual test data

```
#will use for submission
%%time
data_test = test.clean_text.values.tolist()
processed_docs_test = []
for doc in data_test:
    processed_docs_test.append(preprocess(doc))

print(processed_docs_test[:1])
corpus_test= [dictionary.doc2bow(text) for text in processed_docs_test]
```

```
[['integr', 'mean', 'debt', 'appli', 'presid', 'trump']]
CPU times: user 45.9 s, sys: 238 ms, total: 46.2 s
Wall time: 46.2 s
```

Converting Topics to Feature Vectors

```
# For train vectors
train_vecs = []

for i in range(len(X_train)):
    top_train_topics = lda_model.get_document_topics(corpus[i], minimum_probability=0.0)
    topic_train_vec = [top_train_topics[i][1] for i in range(5)]
    train_vecs.append(topic_train_vec)
# Printing top five train vectors
train_vecs[:5]
```

```
[[0.2572527, 0.116501085, 0.15880889, 0.23540507, 0.23203227],
 [0.1085511, 0.12808271, 0.23760512, 0.205904, 0.31985703],
 [0.10038725, 0.13197713, 0.24514905, 0.28512138, 0.23736522],
 [0.100387305, 0.13197719, 0.24514884, 0.23232232, 0.29016432],
 [0.09079904, 0.11937169, 0.22173429, 0.21013263, 0.35796234]]
```

```
# For cv vectors
cv_vecs = []
for i in range(len(X_cv)):
    top_cv_topics = lda_model.get_document_topics(corpus1[i], minimum_probability=0.0)
    topic_cv_vec = [top_cv_topics[i][1] for i in range(5)]
    cv_vecs.append(topic_cv_vec)
# Printing top five test vectors
cv_vecs[:5]
```

```
[[0.09080016, 0.16712835, 0.17397994, 0.30563962, 0.2624519],
 [0.23157655, 0.07987076, 0.23264326, 0.25131276, 0.20459664],
 [0.132223, 0.1595133, 0.16605367, 0.33729813, 0.2049119],
 [0.06035246, 0.14152181, 0.17819482, 0.3855083, 0.23442256],
 [0.1206431, 0.13538024, 0.18808436, 0.24763924, 0.30825308]]
```

```
# For test vectors
te_vecs = []

for i in range(len(X_te)):
    top_te_topics = lda_model.get_document_topics(corpus11[i], minimum_probability=0.0)
    topic_te_vec = [top_te_topics[i][1] for i in range(5)]
    te_vecs.append(topic_te_vec)
# Printing top five test vectors
te_vecs[:5]
```

```
     [[0.1436283, 0.28519964, 0.16165379, 0.14947933, 0.26003894],
```

```
# For actual test vectors
test_vecs = []

for i in range(len(test)):
    top_test_topics = lda_model.get_document_topics(corpus_test[i], minimum_probability=0.
    topic_test_vec = [top_test_topics[i][1] for i in range(5)]
    test_vecs.append(topic_test_vec)


# Create the new df with 5 topics
train_features = pd.DataFrame(train_vecs,columns=['Topic-1','Topic-2','Topic-3','Topic-4',
cv_features = pd.DataFrame(cv_vecs,columns=['Topic-1','Topic-2','Topic-3','Topic-4','Topic
te_features = pd.DataFrame(te_vecs,columns=['Topic-1','Topic-2','Topic-3','Topic-4','Topic
test_features = pd.DataFrame(test_vecs,columns=['Topic-1','Topic-2','Topic-3','Topic-4','T
```

Count number of positive and negative words in each comm

- https://gist.github.com/mkulakowski2/4289441
- https://gist.github.com/mkulakowski2/4289437

```
#I have created 2 files one for  positive words taken from https://gist.github.com/mkulako
```

```
#load pos,neg words
pos_path='/content/drive/My Drive/jigsaw/pos.txt'
neg_path='/content/drive/My Drive/jigsaw/neg.txt'

def load_file(path):
  fp=open(path,'r')
  all_lines=fp.readlines()
  words_list=[]
  for line in all_lines:
    words_list.append(line.strip())
  fp.close()
  return words_list

pos_words=load_file(pos_path)
neg_words=load_file(neg_path)

#count  number of positive and negative words in each comment
def pos_word_count(comment):
  count=0
  for word in comment.split():
    if word in pos_words:
      count=count+1
  return count

def neg_word_count(comment):
  count=0
  for word in comment.split():
    if word in neg_words:
      count=count+1
```

```
    return count


# pos_word_count
count=[]
for i in X_train['clean_text'].values:
  count.append(pos_word_count(str(i)))
train_features['pos_word_count']=count

count=[]
for i in X_cv['clean_text'].values:
  count.append(pos_word_count(str(i)))
cv_features['pos_word_count']=count

count=[]
for i in X_te['clean_text'].values:
  count.append(pos_word_count(str(i)))
te_features['pos_word_count']=count

count=[]
for i in test['clean_text'].values:
  count.append(pos_word_count(str(i)))
test_features['pos_word_count']=count

# neg_word_count
count=[]
for i in X_train['clean_text'].values:
  count.append(neg_word_count(str(i)))
train_features['neg_word_count']=count

count=[]
for i in X_cv['clean_text'].values:
  count.append(neg_word_count(str(i)))
cv_features['neg_word_count']=count

count=[]
for i in X_te['clean_text'].values:
  count.append(neg_word_count(str(i)))
te_features['neg_word_count']=count

count=[]
for i in test['clean_text'].values:
  count.append(pos_word_count(str(i)))
test_features['neg_word_count']=count
```

Find the sentiment of each comment

```
#https://www.pluralsight.com/guides/natural-language-processing-extracting-sentiment-from-
%%time
sentiment_count=[]
for i in X_train['clean_text'].values:
  sentiment_count.append(TextBlob(i).sentiment[0])
```

```
train_features['sentiment']=sentiment_count

sentiment_count=[]
for i in X_cv['clean_text'].values:
  sentiment_count.append(TextBlob(i).sentiment[0])
cv_features['sentiment']=sentiment_count

sentiment_count=[]
for i in X_te['clean_text'].values:
  sentiment_count.append(TextBlob(i).sentiment[0])
te_features['sentiment']=sentiment_count

sentiment_count=[]
for i in test['clean_text'].values:
  sentiment_count.append(TextBlob(i).sentiment[0])
test_features['sentiment']=sentiment_count
```

```
CPU times: user 19min 6s, sys: 2.87 s, total: 19min 9s
Wall time: 19min 9s
```

Features calculated during the EDA

```
from tqdm import tqdm
#comment_word_count
count=[]
for i in tqdm(X_train['clean_text'].values):
  count.append(len(i.split()))
train_features['comment_word_count']=count

count=[]
for i in tqdm(X_cv['clean_text'].values):
  count.append(len(i.split()))
cv_features['comment_word_count']=count

count=[]
for i in tqdm(X_te['clean_text'].values):
  count.append(len(i.split()))
te_features['comment_word_count']=count

count=[]
for i in tqdm(test['clean_text'].values):
  count.append(len(i.split()))
test_features['comment_word_count']=count

#comment_char_count

count=[]
for i in tqdm(X_train['clean_text'].values):
  count.append(len(i))
train_features['comment_char_count']=count

count=[]
```

```
for i in tqdm(X_cv['clean_text'].values):
  count.append(len(i))
cv_features['comment_char_count']=count

count=[]
for i in tqdm(X_te['clean_text'].values):
  count.append(len(i))
te_features['comment_char_count']=count

count=[]
for i in tqdm(test['clean_text'].values):
  count.append(len(i))
test_features['comment_char_count']=count
```

```
100%|██████████| 1461947/1461947 [00:03<00:00, 379699.89it/s]
100%|██████████| 162439/162439 [00:00<00:00, 351337.65it/s]
100%|██████████| 180488/180488 [00:00<00:00, 354918.62it/s]
100%|██████████| 97320/97320 [00:00<00:00, 390665.21it/s]
100%|██████████| 1461947/1461947 [00:00<00:00, 1607047.00it/s]
100%|██████████| 162439/162439 [00:00<00:00, 1488215.71it/s]
100%|██████████| 180488/180488 [00:00<00:00, 1514697.43it/s]
100%|██████████| 97320/97320 [00:00<00:00, 1503715.79it/s]
```

```
test_features
```

	Topic-1	Topic-2	Topic-3	Topic-4	Topic-5	pos_word_count	neg_word_count
0	0.149603	0.176198	0.133320	0.316583	0.224297	1	1
1	0.195654	0.175510	0.132552	0.220671	0.275613	0	0
2	0.105486	0.196813	0.275211	0.191042	0.231447	3	3
3	0.183384	0.153363	0.218330	0.211242	0.233680	5	5
4	0.178390	0.231265	0.128876	0.252196	0.209274	2	2
...	...	...	...	...	...	...	...
97315	0.095354	0.141435	0.132553	0.304900	0.325758	0	0
97316	0.136550	0.126796	0.222251	0.236765	0.277638	2	2
97317	0.212566	0.177888	0.139512	0.281822	0.188212	1	1
97318	0.189127	0.242099	0.098241	0.195365	0.275169	3	3
97319	0.232680	0.114726	0.201673	0.277116	0.173805	0	0

97320 rows × 10 columns

Encoding Numerical features

```
numerical_train_1=train_features['Topic-1'].values.reshape(-1, 1)
numerical_train_2=train_features['Topic-2'].values.reshape(-1, 1)
numerical_train_3=train_features['Topic-3'].values.reshape(-1, 1)
```

```
numerical_train_4=train_features['Topic-4'].values.reshape(-1, 1)
numerical_train_5=train_features['Topic-5'].values.reshape(-1, 1)
numerical_train_6=train_features['pos_word_count'].values.reshape(-1, 1)
numerical_train_7=train_features['neg_word_count'].values.reshape(-1, 1)
numerical_train_8=train_features['sentiment'].values.reshape(-1, 1)
numerical_train_9=train_features['comment_word_count'].values.reshape(-1, 1)
numerical_train_10=train_features['comment_char_count'].values.reshape(-1, 1)

numerical_cv_1=cv_features['Topic-1'].values.reshape(-1, 1)
numerical_cv_2=cv_features['Topic-2'].values.reshape(-1, 1)
numerical_cv_3=cv_features['Topic-3'].values.reshape(-1, 1)
numerical_cv_4=cv_features['Topic-4'].values.reshape(-1, 1)
numerical_cv_5=cv_features['Topic-5'].values.reshape(-1, 1)
numerical_cv_6=cv_features['pos_word_count'].values.reshape(-1, 1)
numerical_cv_7=cv_features['neg_word_count'].values.reshape(-1, 1)
numerical_cv_8=cv_features['sentiment'].values.reshape(-1, 1)
numerical_cv_9=cv_features['comment_word_count'].values.reshape(-1, 1)
numerical_cv_10=cv_features['comment_char_count'].values.reshape(-1, 1)

numerical_te_1=te_features['Topic-1'].values.reshape(-1, 1)
numerical_te_2=te_features['Topic-2'].values.reshape(-1, 1)
numerical_te_3=te_features['Topic-3'].values.reshape(-1, 1)
numerical_te_4=te_features['Topic-4'].values.reshape(-1, 1)
numerical_te_5=te_features['Topic-5'].values.reshape(-1, 1)
numerical_te_6=te_features['pos_word_count'].values.reshape(-1, 1)
numerical_te_7=te_features['neg_word_count'].values.reshape(-1, 1)
numerical_te_8=te_features['sentiment'].values.reshape(-1, 1)
numerical_te_9=te_features['comment_word_count'].values.reshape(-1, 1)
numerical_te_10=te_features['comment_char_count'].values.reshape(-1, 1)

numerical_test_1=test_features['Topic-1'].values.reshape(-1, 1)
numerical_test_2=test_features['Topic-2'].values.reshape(-1, 1)
numerical_test_3=test_features['Topic-3'].values.reshape(-1, 1)
numerical_test_4=test_features['Topic-4'].values.reshape(-1, 1)
numerical_test_5=test_features['Topic-5'].values.reshape(-1, 1)
numerical_test_6=test_features['pos_word_count'].values.reshape(-1, 1)
numerical_test_7=test_features['neg_word_count'].values.reshape(-1, 1)
numerical_test_8=test_features['sentiment'].values.reshape(-1, 1)
numerical_test_9=test_features['comment_word_count'].values.reshape(-1, 1)
numerical_test_10=test_features['comment_char_count'].values.reshape(-1, 1)


num_tr=np.concatenate((numerical_train_1,numerical_train_2,numerical_train_3,numerical_tra
num_cv=np.concatenate((numerical_cv_1,numerical_cv_2,numerical_cv_3,numerical_cv_4,numeric
num_te=np.concatenate((numerical_te_1,numerical_te_2,numerical_te_3,numerical_te_4,numeric
num_test=np.concatenate((numerical_test_1,numerical_test_2,numerical_test_3,numerical_test

numerical=StandardScaler()
numerical_train=numerical.fit_transform(num_tr)
numerical_cv=numerical.transform(num_cv)
numerical_te=numerical.transform(num_te)
numerical_test=numerical.transform(num_test)
```

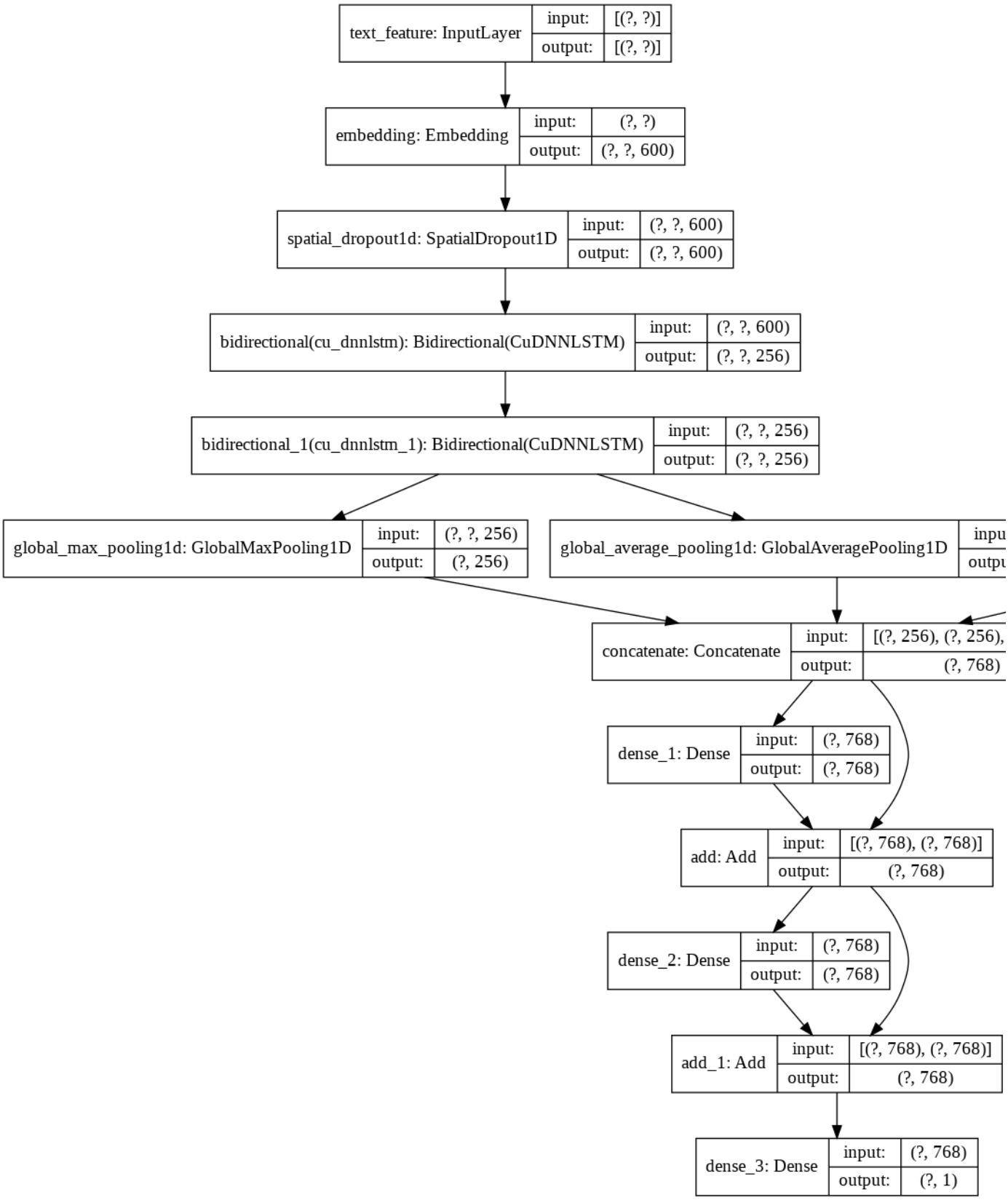
Deep Learning Model Data Preparation

```
EMBEDDING_FILES = [
    '/content/drive/My Drive/jigsaw/crawl-300d-2M.gensim',
    '/content/drive/My Drive/jigsaw/glove.840B.300d.gensim'
]
NUM_MODELS = 2
BATCH_SIZE = 512
LSTM_UNITS = 128
DENSE_HIDDEN_UNITS = 4 * LSTM_UNITS
EPOCHS = 4
MAX_LEN = 220
IDENTITY_COLUMNS = ['male', 'female', 'homosexual_gay_or_lesbian', 'christian', 'jewish',

TEXT_COLUMN = 'comment_text'
TARGET_COLUMN='target'
CHARS_TO_REMOVE = '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n“”’\'∞θ÷α•à−β∅³π‘₹´º£€\×™√²—'

x_train = X_train[TEXT_COLUMN].astype(str)
y_train = X_train[TARGET_COLUMN].values

x_cv = X_cv[TEXT_COLUMN].astype(str)
y_cv = X_cv[TARGET_COLUMN].values

x_te = X_te[TEXT_COLUMN].astype(str)
x_test = test[TEXT_COLUMN].astype(str)

for column in IDENTITY_COLUMNS + [TARGET_COLUMN]:
    X_train[column] = np.where(X_train[column] >= 0.5, True, False)
    X_cv[column] = np.where(X_cv[column] >= 0.5, True, False)
    X_te[column] = np.where(X_te[column] >= 0.5, True, False)


tokenizer = text.Tokenizer(filters=CHARS_TO_REMOVE, lower=False)
tokenizer.fit_on_texts(list(x_train))

x_train = tokenizer.texts_to_sequences(x_train)
x_cv = tokenizer.texts_to_sequences(x_cv)
x_te = tokenizer.texts_to_sequences(x_te)
x_test = tokenizer.texts_to_sequences(x_test)

x_train = sequence.pad_sequences(x_train, maxlen=MAX_LEN)
x_cv = sequence.pad_sequences(x_cv, maxlen=MAX_LEN)
x_te = sequence.pad_sequences(x_te, maxlen=MAX_LEN)
x_test = sequence.pad_sequences(x_test, maxlen=MAX_LEN)


def build_matrix(word_index, path):
    ''' this function prepares embedding matrix'''
    embedding_index = KeyedVectors.load(path, mmap='r')
    embedding_matrix = np.zeros((len(word_index) + 1, 300))
    for word, i in word_index.items():
        for candidate in [word, word.lower()]:
            if candidate in embedding_index:
                embedding_matrix[i] = embedding_index[candidate]
                break
```

```
                break
    return embedding_matrix

sample_weights = np.ones(len(x_train), dtype=np.float32)
sample_weights += X_train[IDENTITY_COLUMNS].sum(axis=1)
sample_weights += X_train[TARGET_COLUMN] * (~X_train[IDENTITY_COLUMNS]).sum(axis=1)
sample_weights += (~X_train[TARGET_COLUMN]) * X_train[IDENTITY_COLUMNS].sum(axis=1) * 5
sample_weights /= sample_weights.mean()

embedding_matrix = np.concatenate([build_matrix(tokenizer.word_index, f) for f in EMBEDDIN
```

```
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:253: UserWarning:

This function is deprecated, use smart_open.open instead. See the migration notes for
```

Deep Learning Model 2 : Dropout 0.2 + text feature + Additio




```
def build_model(embedding_matrix):
    words = Input(shape=(None,),name="text_feature")
    x = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)(wo
    x = SpatialDropout1D(0.2)(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)

    numerical_feats = Input(shape=(10,),name="numerical_features")
    numerical_featss = Dense(256,activation="relu",kernel_initializer="he_normal")(numeric
```

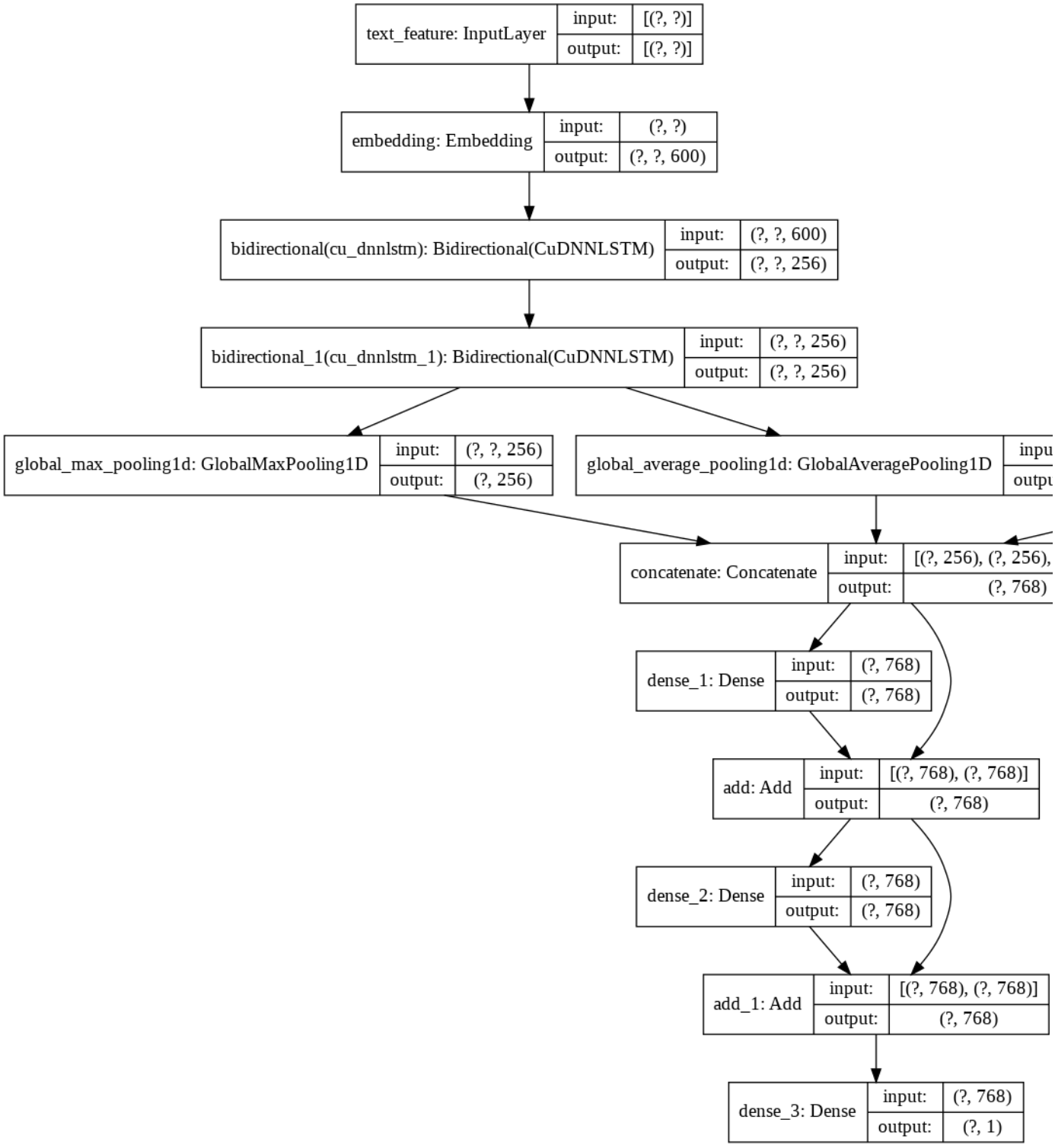
```
    hidden = concatenate([GlobalMaxPooling1D()(x),GlobalAveragePooling1D()(x),numerical_fe
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    result = Dense(1, activation='sigmoid')(hidden)

    model = Model(inputs=[words,numerical_feats], outputs=[result])#, aux_result])
    model.compile(loss='binary_crossentropy', optimizer='adam')
    plot_model(model, to_file='/content/drive/My Drive/jigsaw/Model2.png', show_shapes=Tru

    return model


#all epochs to fit once
from datetime import datetime, timedelta

checkpoint = tensorflow.keras.callbacks.ModelCheckpoint('/content/drive/My Drive/jigsaw/Mo
log_dir="/content/drive/My Drive/jigsaw/Model22/logs/fit/" + datetime.now().strftime("%Y%m
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, w
model = build_model(embedding_matrix)
model.fit([x_train,numerical_train], y_train,batch_size=BATCH_SIZE,epochs=5,verbose=2,vali
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoa
2020-05-31 12:09:21,855 : WARNING : `write_grads` will be ignored in TensorFlow 2.0 f
Epoch 1/5

Epoch 00001: val_loss improved from inf to 0.25208, saving model to /content/drive/My
2856/2856 - 915s - loss: 0.4270 - val_loss: 0.2521
Epoch 2/5

Epoch 00002: val_loss improved from 0.25208 to 0.24228, saving model to /content/driv
2856/2856 - 818s - loss: 0.4096 - val_loss: 0.2423
Epoch 3/5

Epoch 00003: val_loss did not improve from 0.24228
2856/2856 - 816s - loss: 0.4045 - val_loss: 0.2423
Epoch 4/5

Epoch 00004: val_loss improved from 0.24228 to 0.23914, saving model to /content/driv
2856/2856 - 826s - loss: 0.4002 - val_loss: 0.2391
Epoch 5/5

Epoch 00005: val_loss improved from 0.23914 to 0.23825, saving model to /content/driv
2856/2856 - 821s - loss: 0.3964 - val_loss: 0.2382
<tensorflow.python.keras.callbacks.History at 0x7f75ee0ca128>
```

```
%tensorboard --logdir='/content/drive/My Drive/jigsaw/Model22/logs/fit'


# load the model
model1=tensorflow.keras.models.load_model('/content/drive/My Drive/jigsaw/Model22.hdf5')
MODEL_NAME = 'with_DO1'
X_te[MODEL_NAME] =model1.predict([x_te,numerical_te], batch_size=2048).flatten()
bias_metrics_df = compute_bias_metrics_for_model(X_te, IDENTITY_COLUMNS, MODEL_NAME, TARGE
get_final_metric(bias_metrics_df, calculate_overall_auc(X_te, MODEL_NAME))
```

0.932494270491741

Deep Learning Model 3: Without Dropout + text feature + Add




```
def build_model(embedding_matrix):
    words = Input(shape=(None,),name="text_feature")
    x = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)(wo
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)

    numerical_feats = Input(shape=(10,),name="numerical_features")
    numerical_featss = Dense(256,activation="relu",kernel_initializer="he_normal")(numeric
```

```
    hidden = concatenate([GlobalMaxPooling1D()(x),GlobalAveragePooling1D()(x),numerical_fe
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    result = Dense(1, activation='sigmoid')(hidden)

    model = Model(inputs=[words,numerical_feats], outputs=result)
    model.compile(loss='binary_crossentropy', optimizer='adam')
    plot_model(model, to_file='/content/drive/My Drive/jigsaw/model33.png', show_shapes=Tr
    return model


#all epochs to fit once
from datetime import datetime, timedelta

checkpoint = tensorflow.keras.callbacks.ModelCheckpoint('/content/drive/My Drive/jigsaw/Mo
log_dir="/content/drive/My Drive/jigsaw/Model33/logs/fit/" + datetime.now().strftime("%Y%m
tensorboard_callback = tensorflow.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_f
model = build_model(embedding_matrix)
model.fit([x_train,numerical_train], y_train,batch_size=BATCH_SIZE,epochs=15,verbose=2,val
```

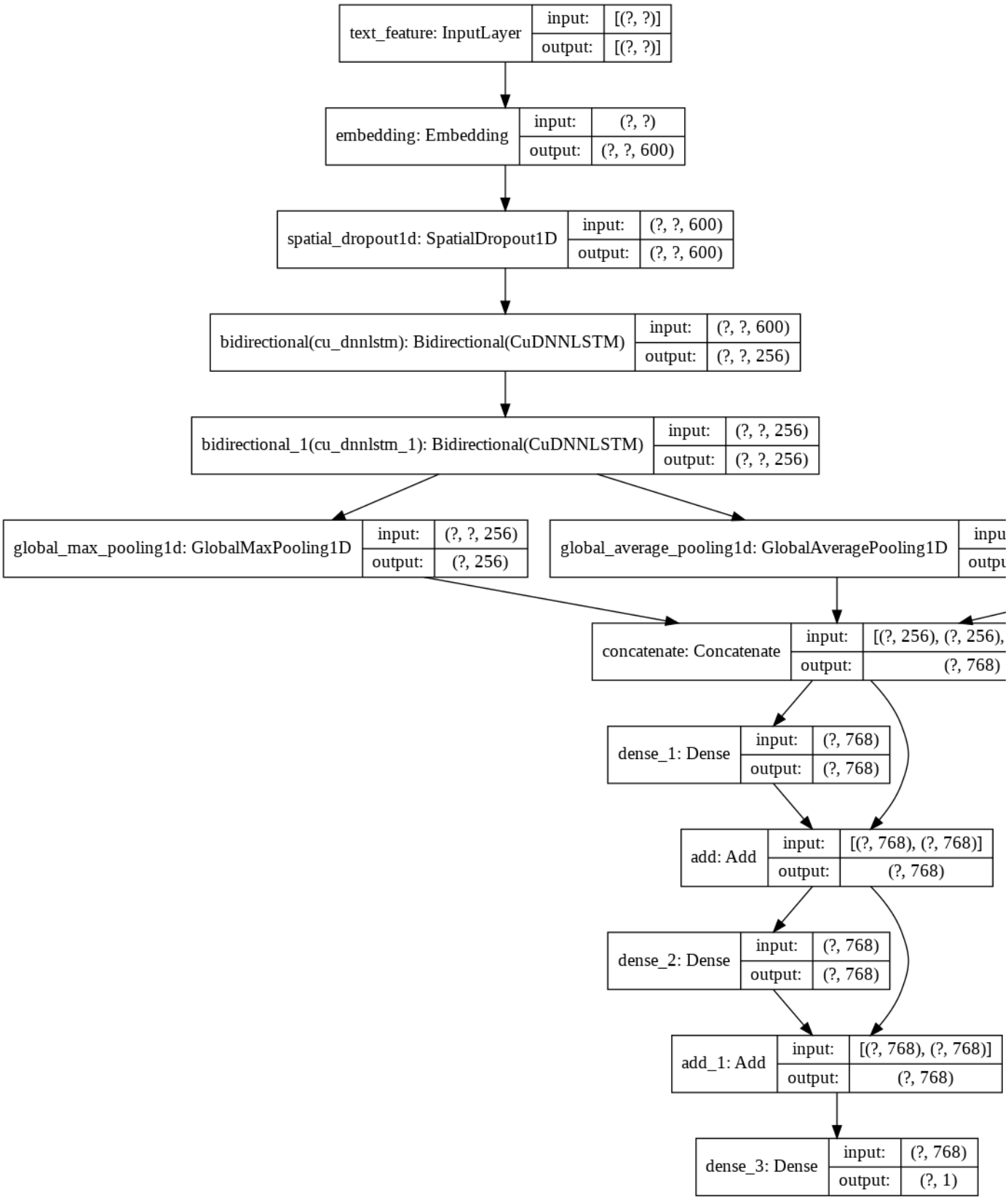
```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoa
2020-06-01 05:37:32,131 : WARNING : `write_grads` will be ignored in TensorFlow 2.0 f
Epoch 1/15

Epoch 00001: val_loss improved from inf to 0.23934, saving model to /content/drive/My
2856/2856 - 776s - loss: 0.4260 - val_loss: 0.2393
Epoch 2/15

Epoch 00002: val_loss did not improve from 0.23934
2856/2856 - 767s - loss: 0.4070 - val_loss: 0.2395
Epoch 3/15

Epoch 00003: val_loss improved from 0.23934 to 0.23773, saving model to /content/driv
2856/2856 - 772s - loss: 0.4009 - val_loss: 0.2377
Epoch 4/15

Epoch 00004: val_loss improved from 0.23773 to 0.23526, saving model to /content/driv
2856/2856 - 773s - loss: 0.3952 - val_loss: 0.2353
Epoch 5/15

Epoch 00005: val_loss did not improve from 0.23526
2856/2856 - 768s - loss: 0.3895 - val_loss: 0.2416
Epoch 6/15

Epoch 00006: val_loss did not improve from 0.23526
```

```
%tensorboard --logdir='/content/drive/My Drive/jigsaw/Model33/logs/fit'
```

```
# load the model
model1=tensorflow.keras.models.load_model('/content/drive/My Drive/jigsaw/Model33.hdf5')
MODEL_NAME = 'withoutDP'
X_te[MODEL_NAME] =model1.predict([x_te,numerical_te], batch_size=2048).flatten()
bias_metrics_df = compute_bias_metrics_for_model(X_te, IDENTITY_COLUMNS, MODEL_NAME, TARGE
get_final_metric(bias_metrics_df, calculate_overall_auc(X_te, MODEL_NAME))
```

```
0.930495739941519

Epoch 10/15
```

Deep Learning Model 4: With 0.5 Dropout + text feature + Ad




```
def build_model(embedding_matrix):
    words = Input(shape=(None,),name="text_feature")
    x = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)(wo
    x = SpatialDropout1D(0.5)(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)

    numerical_feats = Input(shape=(10,),name="numerical_features")
    numerical_featss = Dense(256,activation="relu",kernel_initializer="he_normal")(numeric
```

```
    hidden = concatenate([GlobalMaxPooling1D()(x),GlobalAveragePooling1D()(x),numerical_fe
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    hidden = add([hidden, Dense(768, activation='relu')(hidden)])
    result = Dense(1, activation='sigmoid')(hidden)

    model = Model(inputs=[words,numerical_feats], outputs=result)
    model.compile(loss='binary_crossentropy', optimizer='adam')
    plot_model(model, to_file='/content/drive/My Drive/jigsaw/model44.png', show_shapes=Tr

    return model


from datetime import datetime, timedelta

checkpoint = tensorflow.keras.callbacks.ModelCheckpoint('/content/drive/My Drive/jigsaw/Mo
log_dir="/content/drive/My Drive/jigsaw/Model44/logs/fit/" + datetime.now().strftime("%Y%m
tensorboard_callback = tensorflow.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_f
model = build_model(embedding_matrix)
model.fit([x_train,numerical_train], y_train,batch_size=BATCH_SIZE,epochs=5,verbose=2,vali
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoa
2020-06-01 08:59:33,564 : WARNING : `write_grads` will be ignored in TensorFlow 2.0 f
Epoch 1/5

Epoch 00001: val_loss improved from inf to 0.25312, saving model to /content/drive/My
2856/2856 - 793s - loss: 0.4400 - val_loss: 0.2531
Epoch 2/5

Epoch 00002: val_loss improved from 0.25312 to 0.25297, saving model to /content/driv
2856/2856 - 794s - loss: 0.4153 - val_loss: 0.2530
Epoch 3/5

Epoch 00003: val_loss did not improve from 0.25297
2856/2856 - 788s - loss: 0.4106 - val_loss: 0.2542
Epoch 4/5

Epoch 00004: val_loss improved from 0.25297 to 0.25020, saving model to /content/driv
2856/2856 - 793s - loss: 0.4076 - val_loss: 0.2502
Epoch 5/5

Epoch 00005: val_loss improved from 0.25020 to 0.24852, saving model to /content/driv
2856/2856 - 794s - loss: 0.4053 - val_loss: 0.2485
<tensorflow.python.keras.callbacks.History at 0x7f3099134438>
```

```
%tensorboard --logdir='/content/drive/My Drive/jigsaw/Model44/logs/fit'


# load the model
model1=tensorflow.keras.models.load_model('/content/drive/My Drive/jigsaw/Model44.hdf5')
MODEL_NAME = 'DP'
X_te[MODEL_NAME] =model1.predict([x_te,numerical_te], batch_size=2048).flatten()
bias_metrics_df = compute_bias_metrics_for_model(X_te, IDENTITY_COLUMNS, MODEL_NAME, TARGE
get_final_metric(bias_metrics_df, calculate_overall_auc(X_te, MODEL_NAME))
```

```
0.9317927713596027
```

From above trained models Model 2(Dropout 0.2 + text feature + Additional Fe lets load that trained model and predict on test data.

```
test = pd.read_csv('/content/drive/My Drive/test.csv')
x_test=pickle.load(open('/content/drive/My Drive/jigsaw/xtest','rb'))
numerical_test=pickle.load(open('/content/drive/My Drive/jigsaw/numericaltest','rb'))


# load the model
model1=tensorflow.keras.models.load_model('/content/drive/My Drive/jigsaw/Model22.hdf5')
predictions=model1.predict([x_test,numerical_test], batch_size=2048).flatten()

submission = pd.DataFrame.from_dict({
    'id': test.id,
    'prediction': predictions
})
submission.to_csv('/content/drive/My Drive/jigsaw/submission.csv', index=False)
```

On kaggle kernel we got a score of 0.93112

Submission and Description	Private Score	Public Score
kernel1a8ed46047 (version 2/2) a few seconds ago by Priyankaad From "kernel1a8ed46047" Script	0.93112	0.00000

Results

```
from prettytable import PrettyTable
import pandas as pd

x = PrettyTable()
x.field_names = ["Model", "ROC-AUC Score", "Custom  Metric Score"]
x.add_row(["Logistic Regression", 0.95,  0.8904733235626956])
x.add_row(["Naive Bayes\t", 0.88, 0.837811746713348])
x.add_row(["SVM\t",      0.95,   0.8814130929571368])
x.add_row(["Deep Learning", " - \t",  0.9319064728629746])
print(x)
```

```
+--------------------+--------------+----------------------+
```

After adding additional features

```
|     Naive Bayes          |      0.88     |  0.837811746713348   |
```

```
x = PrettyTable()
x.field_names = ["Model\t\t\t\t\t", "\tCustom  Metric Score"]
x.add_row(["Dropout 0.2 + text feature + Additional Features\t\t", 0.932494270491741])
x.add_row(["Without Dropout + text feature + Additional Features\t", 0.930495739941519])
x.add_row(["With 0.5 Dropout + text feature + Additional Features\t", 0.9317927713596027])
print(x)
```

```
+-----------------------------------------------------+----------------------+
|                       Model
+-----------------------------------------------------+----------------------+
|   Dropout 0.2 + text feature + Additional Features               |   0.932494270491
| Without Dropout + text feature + Additional Features    |   0.930495739941519   |
| With 0.5 Dropout + text feature + Additional Features  |   0.9317927713596027  |
+-----------------------------------------------------+----------------------+
```

Conclusion

1) As we have seen machine learning models performed very well when we use roc-auc metric but given compartively low score.
2) We have tried three machine learning models out of them Logistic regression has given higher s
3) To improve performance on custom metric we have trained **five** deep learning models out of wh **+ Additional Features**"which has given **0.9324 on custom metric** ,we can see that this score is mu models. 4)**After adding those additional features ,we have improved our model from 0.93190647**

Step by Step Procedure to solve this case study

1) **Business Problem**: First go through business problem, understand problem statement,define bu understand data fields.
2)**Map the real-world problem to a Machine Learning Problem**: understand what type of Machine metric.
3)**Work on Exploratory Data Analysis**: like loading data,understanding its toxic and non-toxic featu and non-toxic words by plotting wordcloud, perform text preprocessing in which replace links with which contains characters and numbers together,demojize i.e. convert emoji's into words,remove p because having stopwords doesnt help in text classification.
4)**Feature Engineering**: we have added 'comment_word_count' ,'comment_char_count' new feature by having distribution plot,boxplot,violinplot,kernel density estimate plot and then we visualize usir
5)**Machine Learning Models**

- **Train and Cv Split**: we do 80:20 split.
- **Make Data Model Ready**: encoding numerical, text features

- **Apply ML models**:Tune Hyperparameters of ML models,plot ROC curve and confusion matri after that use custom metric to see how it will score on kaggle board.

6)**Give a try to Deep Learning Model**:build a model ,evauate using custom metric.

7)**Summarize Reults**