Sarcasm* Detection in News Headlines Dataset

Dana Faiez, March 2021

Motivation:

- Detecting sarcasm is challenging even for human; can a machine do as good/ better?

- Can we do better than off-the-shelf BERT classifier, using Bi/LSTM architecture?

- Detecting sarcasm in relation to voice assistants may be useful because it can:
  - be a way to receive feedback from user,
  - strengthen the personal element between voice assistants and user.

What is sarcasm?

Sarcasm involves the use of language to **mean something other than its literal meaning**, but always with the **intention to mock or criticize** someone or something.

Sarcasm can take different forms such as:

1. **Ironic** (statements or citations contrary to what one expects)
2. **Satire** (A humor that draws attention to political/social issues)
3. Self-description (Act of denigrating yourself as a means of humor)
4. Flattery (Insincere praise)

Sources: simplicable.com, litcharts.com

Challenges in detecting sarcasm:

- Context dependent
- Sometimes requires knowing
  - the speaker/author
  - time / place / current events / culture
  - relationship between the speaker and listener
  - tone and facial expression of the speaker/author which we lose in text

Challenges regarding dataset:

- Twitter / reddit forums, relying on self-annotation:
  - spelling mistakes
  - noisy due to mistakes in labels/hashtags
- News Headlines dataset, relying on authors of *TheOnion*/HuffPost*:*
  - both websites focus on news *
  - smaller dataset than twitter dataset

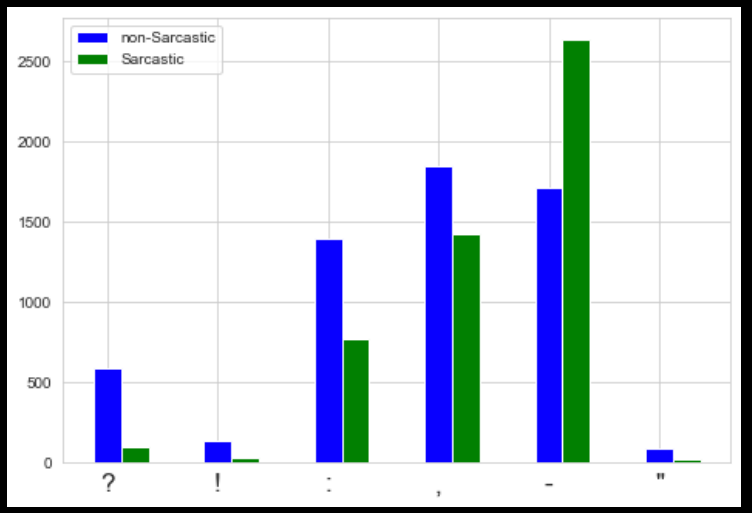
Dataset:

News Headlines dataset

collected from two news website:

- **TheOnion** (sarcastic/ironic versions of current events)
- **HuffPost** (real/non-sarcastic news)

Number of punctuations in each set:

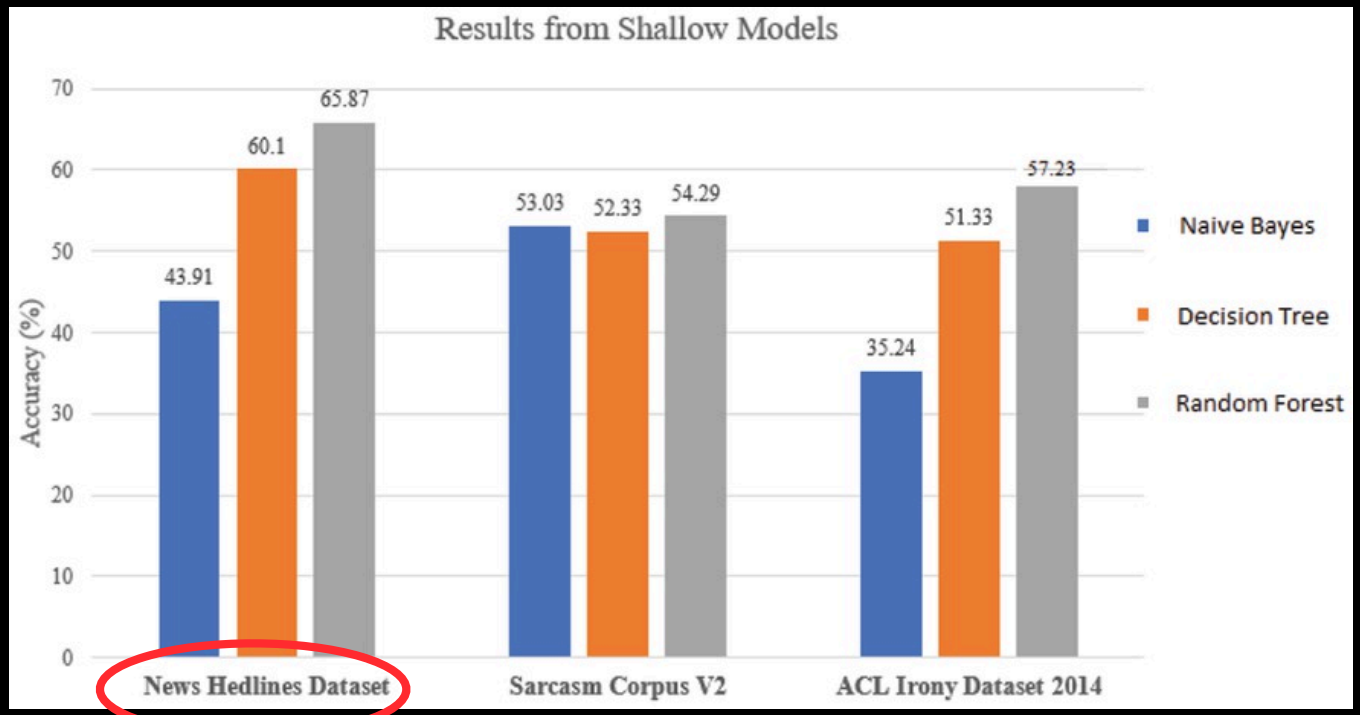


Ave number of words in a sentence in sarcastic / non-sarcastic set ≃ 9

Sarcastic	Non-sarcastic
nuclear bomb detonates during rehearsal for 'spider-man' musical.	ted cruz hits the panic button: 'we could lose both houses of congress'
whale regrets eating 290,000 plastic poker chips that fell off container ship.	donald trump heading for a series of wins in the northeast, polls say
‘right to live life in complete, stunned horror’, added to constitution	the lawsuit against black lives matter and the central meaning of the first amendment

Related work:

Results from Naive Bayes, Decision Tree, Random Forest models



Pulkit Mehndiratta, Devpriya Soni

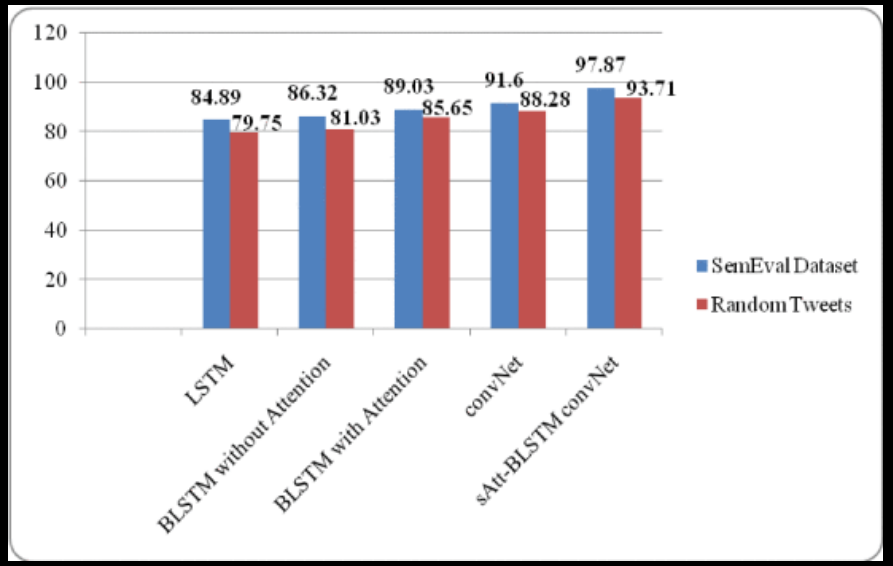
Related work:

Results from NN, RNN, CNN, Attention

Using the news headline dataset:

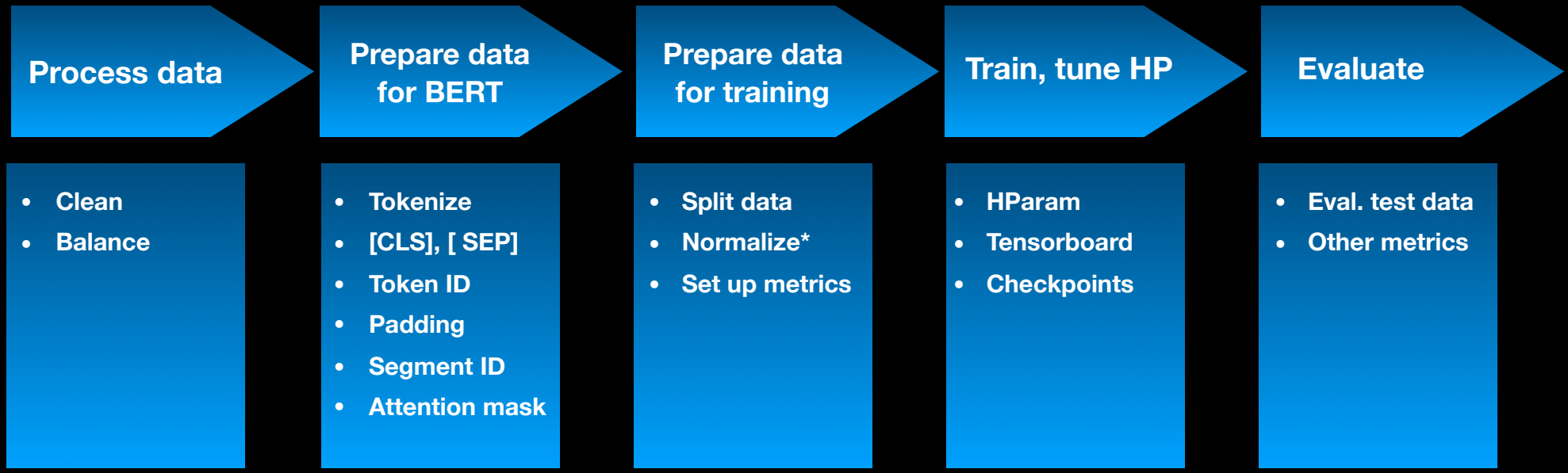
	Embedding/Model	Accuracy
Harshita Pandey	Embeddig layer + Dense layer architecture	87%
NC State ECE	Glove embedding + LSTM	82%
	BERT classifier	92%

Using the Twitter dataset:



Hoang Son et al. (2019)

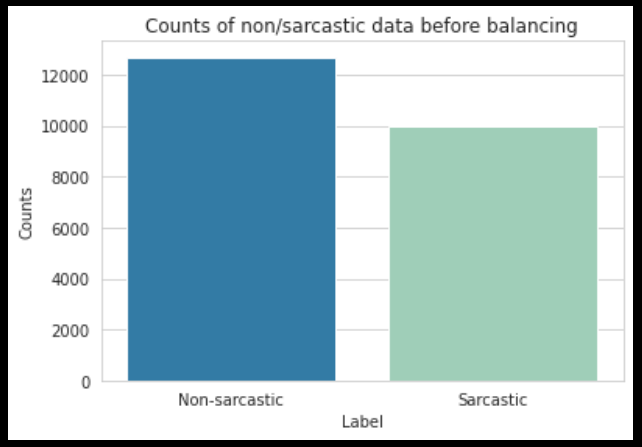
Process:

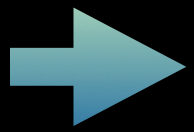

Process data
Prepare data for BERT
Prepare data for training
Train, tune HP
Evaluate
• Clean
• Balance
• Tokenize
• [CLS], [SEP]
• Token ID
• Padding
• Segment ID
• Attention mask
• Split data
• Normalize*
• Set up metrics
• HParam
• Tensorboard
• Checkpoints
• Eval. test data
• Other metrics

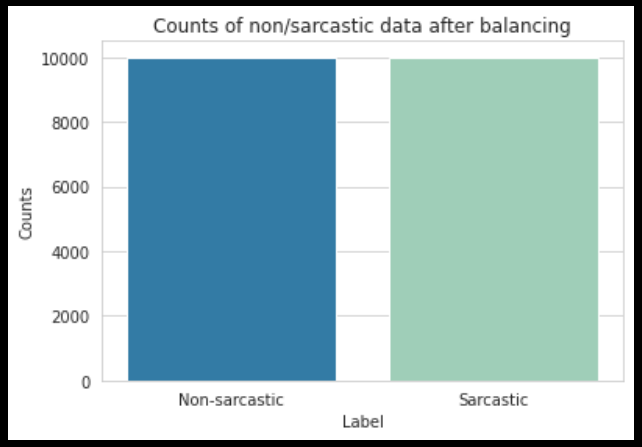
Here are the main parts of the code

```
# Getting and cleaning the data
start_time = time.time()
n =  22692

df = load_data(n)

df = balanced_data(df)

df = shuffle_data(df)

sentences, labels = get_sentences_labels(df)
```

20k total sentences after balancing the dataset







```
# Tokenizing
input_ids = tokenizer_inptID(sentences)
MAX_LEN = get_max_len(input_ids)
print("MAX_LEN:",MAX_LEN)
```

MAX_LEN = 57 tokens ; used later for padding

```
# Input ID, Segment ID, and attention mask
partition_size = 5000
input_ids_pad = []; start = 0
for i in range(1, 5):
  m = partition_size*i
  input_ids_pad.extend(list(padding(input_ids[start:m], MAX_LEN)))
  start = m
input_ids = np.array(input_ids_pad)

attention_masks, segments_ids = Create_segmentsids_attentionmasks(input_ids)

# Get the embeddings
partition_size = 5000
res = [] ; start = 0
input_ids_list = list(input_ids)

for i in range(1,5):
  m = partition_size*i
  print("start")
  temp = get_embedding(input_ids_list[start:m], segments_ids[start:m], attention_masks[start:m])
  print("Done i:", i)
  res.append(temp)
  start = m

encoded_layers =  torch.cat(res,dim=0)
```

```
labels = np.array(labels)


#normalizing data
def normalize(norm_option, embedding):
  input = []
  if norm_option == 'min_max_scaler': #[0,1]
    min_max_scaler = preprocessing.MinMaxScaler()
    embedding = np.array([min_max_scaler.fit_transform(sent) for sent in embedding])

  elif norm_option == 'max_abs_scaler': #[-1,1]  #
    max_abs_scaler = preprocessing.MaxAbsScaler()
    embedding = np.array([max_abs_scaler.fit_transform(sent) for sent in embedding])

  elif norm_option == 'Unit_norm':
    embedding = np.array([preprocessing.normalize(sent, norm='l2') for sent in embedding])
  return embedding
```

```
start_time = time.time()
# metric
metric = [keras.metrics.BinaryAccuracy(name='accuracy'), keras.metrics.AUC(name='auc')]

####### Hyper param tuning #######

#Clear any logs from previous runs
!rm -rf ./logs/*

HP_NUM_UNITS = hp.HParam('num_units', hp.Discrete([40,70]))
HP_DROPOUT = hp.HParam('dropout', hp.Discrete([0.70]))
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['AdaDelta']))
HP_L_RATE = hp.HParam('learning_rate', hp.Discrete([1.0]))
HP_DROPOUT_lstm = hp.HParam('dropout_lsm', hp.Discrete([0.15]))

hp_dict = {'hidden_units' : HP_NUM_UNITS, 'dropout': HP_DROPOUT, 'optimizer': HP_OPTIMIZER,
'lr':HP_L_RATE,'dropout_lstm': HP_DROPOUT_lstm}

batch_size = 32
epochs = 10
```

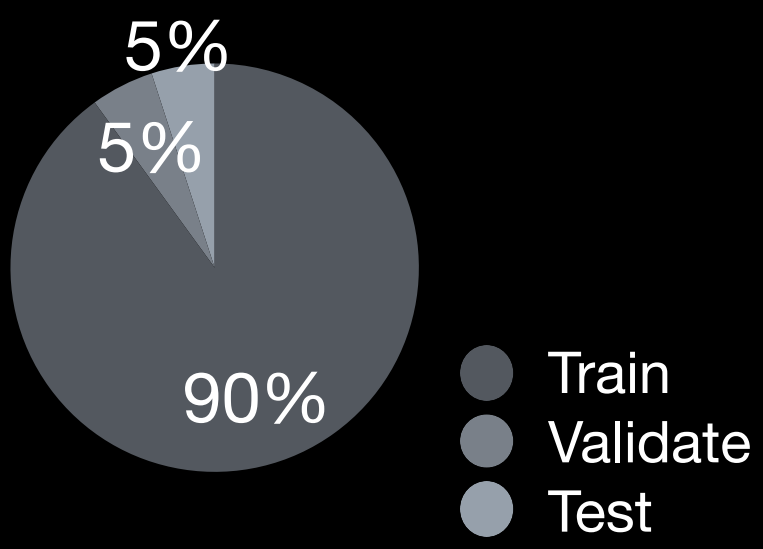
```
def run(logdir, hparams, xtrain, ytrain, xval, yval, xtest, ytest, path_session, best_weights, eval, arch_name):
  with tf.summary.create_file_writer(logdir).as_default():
    hp.hparams(hparams)  # record the values used in this trial
    model = build_model(MAX_LEN, batch_size, epochs, metric, hparams, arch_name)
    dot_img_file = '/content/model_1.png'
    tf.keras.utils.plot_model(model, to_file = dot_img_file, show_shapes=True)

    if eval==False:
      out_model = fit_model(model, xtrain, ytrain, xval, yval, batch_size, epochs, path_session, logdir, hparams)
      valacc = (out_model.history['val_accuracy'])
      valauc = (out_model.history['val_auc'])
      print('val_acc:', valacc[-1], 'val_auc:', valauc[-1])
      tf.summary.scalar('accuracy', valacc[-1], step = epochs)

    else:
      model.load_weights(best_weights)
      loss_test, acc_test, auc_test =  model.evaluate(xtest, ytest, batch_size, verbose=1)
      print("loss_test:", loss_test, "acc_test:", acc_test, "auc_test:", auc_test)

      pred = model.predict(xtest)
      print("len of test dataset:", len(pred))
      for thresh in np.linspace(0.3, 0.7, num=5):
        pred_temp = [int(prob > thresh) for prob in pred]

        print("#### Threshold: ####" , thresh)
        (tn, fp, fn, tp) = confusion(pred_temp, ytest)
        print("False predictions of non-sarcasm:", fn)
        print("False predictions of sarcasm:", fp)
        print("True predictions of non-sarcasm:", tn)
        print("True predictions of sarcasm:", tp)

        print("Precision:", get_precision(pred_temp, ytest))
        print("Recall:", get_recall(pred_temp, ytest))
        print("F1:", get_F1(pred_temp, ytest))
        print("Accuracy:", get_accuracy(pred_temp, ytest))
        print('\n')
    return model
```

```
def run_tensorboard(hp_dict, best_weights, eval, arch_name, norm, norm_options):
  session_num = 0
  for num_units in hp_dict['hidden_units'].domain.values:
    for dropout_rate in hp_dict['dropout'].domain.values:
        for opt in hp_dict['optimizer'].domain.values:
          for learning_rate in hp_dict['lr'].domain.values:
            for dropout_rate_lstm in hp_dict['dropout_lstm'].domain.values:
                hparams = {
                    HP_NUM_UNITS : num_units,
                    HP_DROPOUT : dropout_rate,
                    HP_OPTIMIZER : opt,
                    HP_L_RATE : learning_rate,
                    HP_DROPOUT_lstm : dropout_rate_lstm,

                }
                # Preperaing the data for training
                embedding = encoded_layers.numpy()

                if norm == True:
                  for norm_option in norm_options:
                    print("_____norm_option_____:", norm_option)
                    embedding = normalize(norm_options, embedding)
                    x_train, y_train, x_val, y_val, x_test, y_test = split(embedding, labels)
                    run_name = "run-%d" % session_num
                    print('---- Starting trial ----: %s' % run_name)
                    print({h.name: hparams[h] for h in hparams})
                    path_session = "file_" + str(session_num) + ".hdf5" # Best weights are saved here for each set of
hyperparameters
```

(continue next slide)

```
                    model = run('logs/hptune_2/' + run_name, hparams, x_train, y_train, x_val, y_val, x_test, y_test,
path_session, best_weights, eval, arch_name)
                     session_num += 1
                     print("\n")
                else:
                  print("_____norm_option_____:" + " not normalizing")
                  x_train, y_train, x_val, y_val, x_test, y_test = split(embedding, labels)
                  run_name = "run-%d" % session_num
                  print('---- Starting trial ----: %s' % run_name)
                  print({h.name: hparams[h] for h in hparams})
                  path_session = "file_" + str(session_num) + ".hdf5" # Best weights are saved here for each set of
hyperparameters
                  model = run('logs/hptune_2/' + run_name, hparams, x_train, y_train, x_val, y_val, x_test, y_test,
path_session, best_weights, eval, arch_name)
                  session_num += 1
                  print("\n")

  return model
#######  Tune Hyperparameter on Validation Dataset and Save the Best Weights #######
#norm_options = ['min_max_scaler', 'max_abs_scaler','Unit_norm']
model = run_tensorboard(hp_dict, best_weights=None, eval = False, arch_name = 'bilstm', norm = False, norm_options =
['Unit_norm']) #'attention', 'bilstm', 'lstm'
print("time to train/evaluate:\n")
print("%s" % (time.time() - start_time))
```

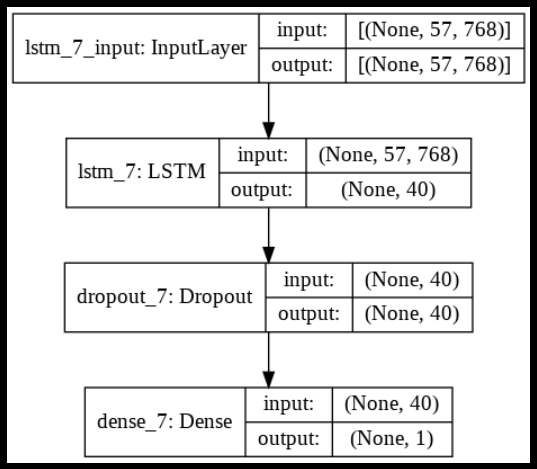
x_train	(18000, 47, 768)
x_validation	(1000, 47, 768)
x_test	(1000, 47, 768)



```
#######  Evaluating on Test Dataset  #######
HP_NUM_UNITS = hp.HParam('num_units', hp.Discrete([40]))
HP_DROPOUT = hp.HParam('dropout', hp.Discrete([0.7]))
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['AdaDelta']))
HP_L_RATE = hp.HParam('learning_rate', hp.Discrete([1.0]))
HP_DROPOUT_lstm = hp.HParam('dropout_lsm', hp.Discrete([0.15]))

hp_dict_opt = {'hidden_units' : HP_NUM_UNITS, 'dropout': HP_DROPOUT, 'optimizer': HP_OPTIMIZER,
'lr':HP_L_RATE,'dropout_lstm': HP_DROPOUT_lstm}

best_weights = None ## add the file with best weights here
best_weights='/content/file_0.hdf5'


#model = run_tensorboard(hp_dict_opt, x_train, y_train, x_test, y_test, best_weights, eval = True, arch_name =
‘bilstm')
model = run_tensorboard(hp_dict, best_weights=best_weights, eval = True, arch_name = 'bilstm', norm = False,
norm_options = ['Unit_norm'])
```

Results:

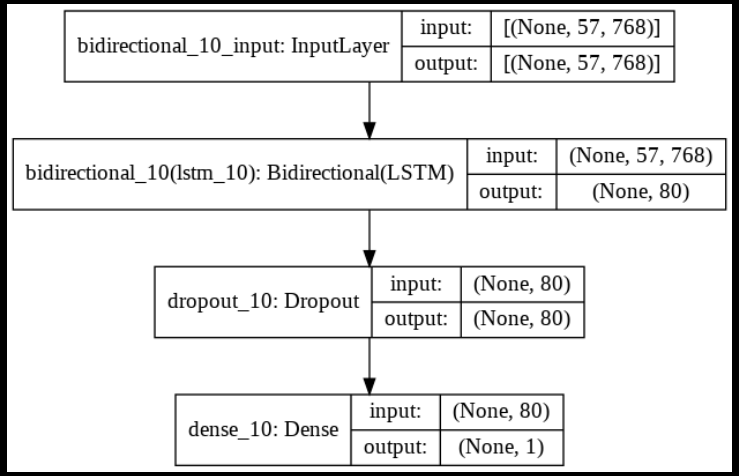
LSTM:



	Training data	Validation	Test data
Loss	**0.25**	**0.26**	**0.28**
Accuracy	90.1%	89.3%	**88.7%**
Precision			88.7%
Recall			89.2%
F1			88.9%

0.8 min / epoch on CPU

BiLSTM:

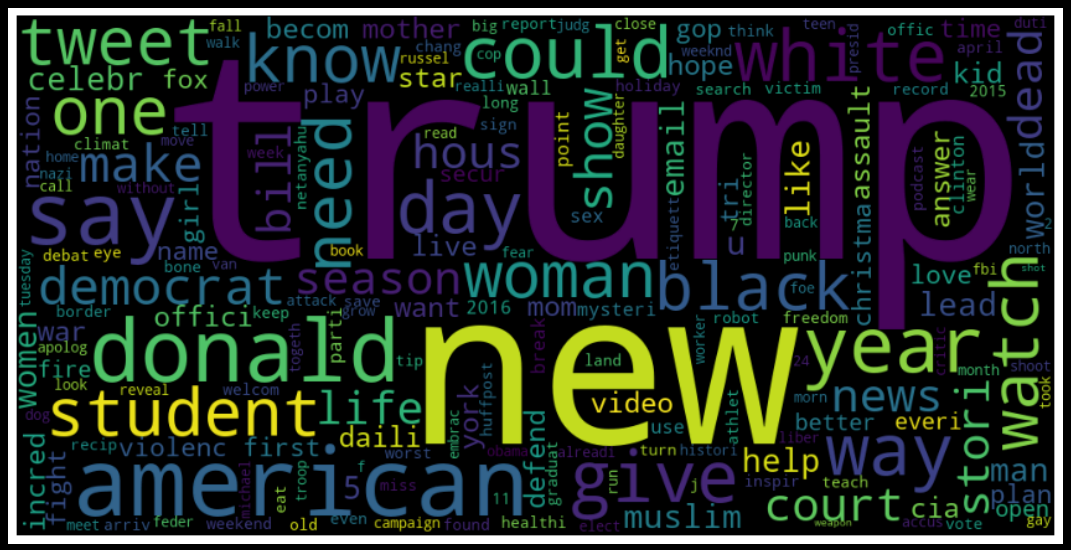


	Training data	Validation	Test data
Loss	**0.16**	**0.25**	**0.30**
Accuracy	93.8%	90.2%	**89.5%**
Precision			92.3%
Recall			86.8%
F1			89.4%

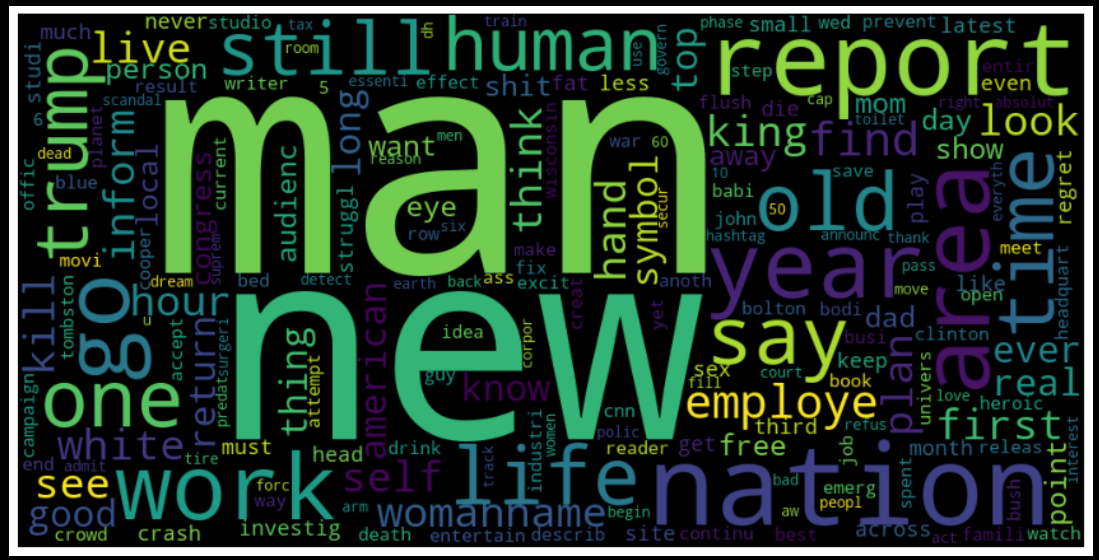
1.3 min / epoch on CPU

Most important words
(using frequency of words in each dataset, excluding stop-words)

non-sarcastic dataset:

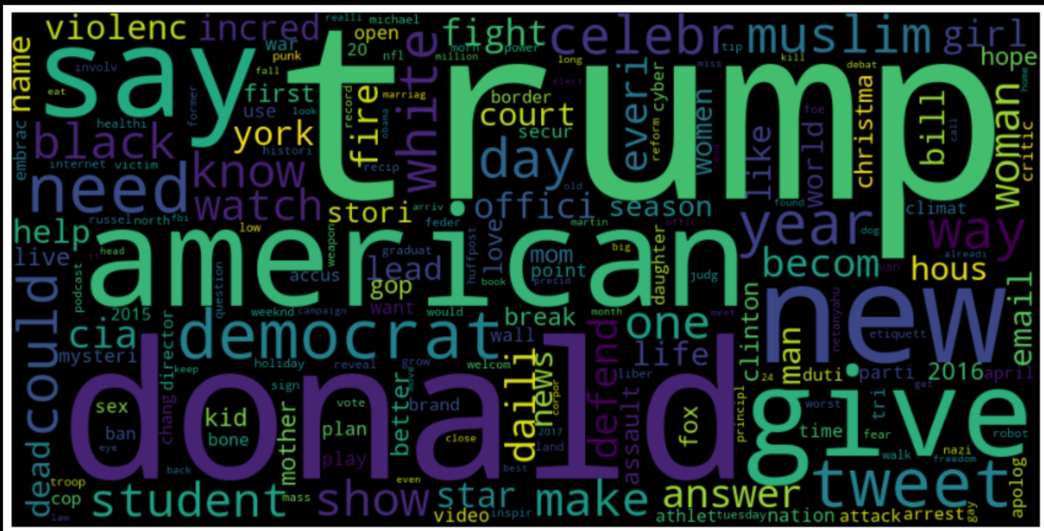


sarcastic dataset:

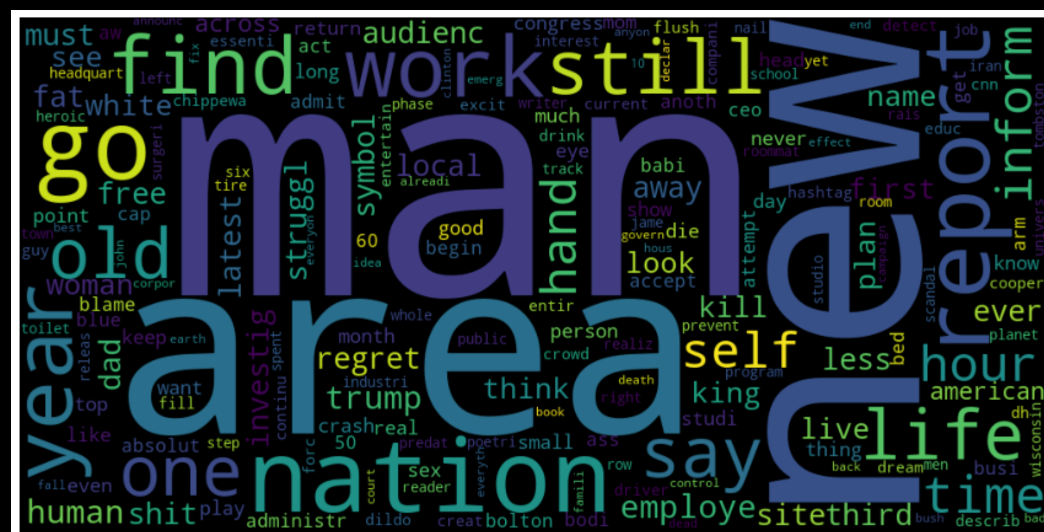


Most important words
(using TfidfVectorizer)

non-sarcastic dataset:



sarcastic dataset:



Functions used in the main code

```
def load_data(n):
  def clean_text(headline):
            headline = re.sub(r"[,.;@_#?!\"&$:]+", ' ', headline)
            headline = re.sub(r"\s+", ' ', headline)
            return headline

  srcsm_json = requests.get('https://storage.googleapis.com/laurencemoroney-blog.appspot.com/sarcasm.json')
  sentences_clean = []
  labels = []
  i=0
  for item in srcsm_json.json():
      if i<n:
        sentences_clean.append(clean_text(item['headline']))
        labels.append(item['is_sarcastic'])
        i+=1

  df = pd.DataFrame({'text' : sentences_clean[:], 'label':labels[:]})
  # label 0: not-sarcastic; label 1: sarcastic
  return df
```

```
####  News Dataset  ####

def balanced_data(df):
  # Make data balanced:

  ## Get the number of sarcastic vs non-sarcastic rows
  def get_diff_classes(df):
    count_list = list(df['label'].value_counts())
    return count_list[0]-count_list[1]

  diff = get_diff_classes(df)


  def plot_data_count(df, title):
    f, ax = plt.subplots(1)
    sns.set_style('whitegrid')
    sns.countplot(x='label',data=df, palette='YlGnBu_r', ax=ax)
    ax.set_xticklabels(["Non-sarcastic","Sarcastic"])
    ax.set(title = title, xlabel='Label', ylabel='Counts')

  if diff !=0:
    plot_data_count(df, 'Counts of non/sarcastic data before balancing')
    print('Before balancing the data, (non-sarcasic - sarcasic) sentences :', get_diff_classes(df))
    if diff>0: datatype = 0 # remove from non-sarcasic class
    else: datatype = 1 # remove from sarcasic class
    np.random.seed(0)
    drop_indices = np.random.choice(df[df['label'] == datatype].index, abs(diff), replace=False)
    df = df.drop(drop_indices)
    print('After balancing the data, (non-sarcasic - sarcasic) sentences :', get_diff_classes(df))
    plot_data_count(df, 'Counts of non/sarcastic data after balancing')

  print("df.info():\n",df.info())
  print("There are", len(df), "total sentences.\n")
  return df
```

```python
####  Shuffling Dataset  ####
def shuffle_data(df):
  df = df.reindex(np.random.permutation(df.index))
  print("Top 10 sentences and labels:\n", df.head(10))
  return df
```

```python
def get_sentences_labels(df):
  sentences = df['text'].tolist()
  labels = df['label'].tolist()
  return sentences, labels
```

```
# bert-base-uncased : 12-layer, 768-hidden, 12-heads, 110M parameters. Trained on lower-cased English text.
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
```

```
def tokenizer_inptID(sentences):

  # Tokenize all of the sentences and map the tokens to thier word IDs.
  input_ids = []

  for sent in sentences:
      # `encode` will:
      #   (1) Tokenize the sentence.
      #   (2) Prepend the `[CLS]` token to the start (with ID 101).
      #   (3) Append the `[SEP]` token to the end (with ID 102).
      #   (4) Map tokens to their IDs.
      encoded_sent = tokenizer.encode(
                          sent,
                          add_special_tokens = True,
                     )

      input_ids.append(encoded_sent)

  return input_ids
```

Example text = "this is troublesome"

tokenized : [[CLS], 'this', 'is', 'troubles', '##ome', [SEP]]

input_ids : [[101, 2023, 2003, 13460, 8462, 102]]

```
# Get the length of the longest encoded sentence
def get_max_len(input_ids):
  MAX_LEN = max([len(sen) for sen in input_ids])
  return MAX_LEN
```

```
def padding(input_ids, MAX_LEN):

  # Pad our input tokens with value 0.
  # "post" indicates that we want to pad and truncate at the end of the sequence, as opposed to the beginning.
  input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long",
                            value=0, truncating="post", padding="post")


  return input_ids
```

tokenized : [[CLS], 'this', 'is', 'troubles', '##ome', [SEP]] ; e.g. MAX_LEN = 7

input_ids after padding : [[101 2023 2003 13460 8462 102 0]]

```
def Create_segmentsids_attentionmasks(input_ids):
  attention_masks = []
  segments_ids = []
  # For each sentence...
  for id_sent in input_ids:

      # Create the attention mask:
      ## If a token ID is 0, then it's padding, then set the mask to 0 otherwise it's a real token, set the mask to 1.
      att_mask = [int(token_id > 0) for token_id in id_sent]
      # Create the segmentation embeddings:
      segm_id = [0 for token_id in id_sent]

      attention_masks.append(att_mask)
      segments_ids.append(segm_id)


  return attention_masks, segments_ids
```

segments_ids: [[0, 0, 0, 0, 0, 0, 0]]

attention_masks: [[1, 1, 1, 1, 1, 1, 0]]

```
def get_embedding(input_ids, segments_ids, attention_masks):
  # Convert the list of IDs to a tensor of IDs
  #[torch.LongTensor] is an int64 data type value.
  #[torch.Tensor] is a float32 data type value.
  model = BertModel.from_pretrained('bert-base-uncased', output_hidden_states=False)
  input_ids_tensor = torch.tensor(input_ids)
  segments_tensors = torch.tensor(segments_ids)
  attention_tensors = torch.tensor(attention_masks)


  # Set the device to GPU (cuda) if available, otherwise stick with CPU
  device = 'cuda' if torch.cuda.is_available() else 'cpu'
  model = model.to(device)
  input_ids = input_ids_tensor.to(device)
  segments_tensors = segments_tensors.to(device)
  attention_tensors = attention_tensors.to(device)
  # Put the model in "evaluation" mode, meaning feed-forward operation.
  model.eval()

  with torch.no_grad():
      outputs = model(input_ids, token_type_ids=segments_tensors, attention_mask=attention_tensors)
      # Transformers models always output tuples.
      # The first element is the hidden state of the last layer of the Bert model
      encoded_layers = outputs[0]


  return encoded_layers
```

tokenized : [[CLS], 'this', 'is', 'troubles', '##ome', [SEP]]

input_ids after padding : [[101 2023 2003 13460 8462 102 0]]

output embedding is of torch.Size ([1, 7, 768])

```
def split(embedding, labels):
  rand_state = 0

  x_train, x_test, y_train, y_test = train_test_split(embedding, labels, test_size=0.1, random_state=rand_state,
stratify=labels)
  x_test, x_val, y_test, y_val     = train_test_split(x_test, y_test, test_size=0.5, random_state=rand_state,
stratify=y_test)

  return x_train, y_train, x_val, y_val, x_test, y_test
```

```
def build_model(MAX_LEN, batch_size, epochs, metric, hparams, arch_name):

  if arch_name == 'lstm':

    model = Sequential()
    model.add(LSTM(hparams[HP_NUM_UNITS], input_shape=(MAX_LEN,768),return_sequences = False,
activation='tanh',recurrent_dropout = hparams[HP_DROPOUT_lstm]))
    #model.add(LSTM(hparams[HP_NUM_UNITS], return_sequences = False, activation='tanh',recurrent_dropout
=hparams[HP_DROPOUT_lstm]))
    model.add(Dropout(hparams[HP_DROPOUT]))
    #model.add(Dense(hparams[HP_NUM_UNITS], activation='tanh'))
    #model.add(Dropout(hparams[HP_DROPOUT]))
    model.add(Dense(1, activation="sigmoid"))

  elif arch_name == 'bilstm':
    model = Sequential()
    model.add(Bidirectional(LSTM(hparams[HP_NUM_UNITS], return_sequences=False, activation='tanh',
    recurrent_dropout = hparams[HP_DROPOUT_lstm]), input_shape=(MAX_LEN,768)))
    #model.add(Bidirectional(LSTM(hparams[HP_NUM_UNITS],return_sequences = False, activation='tanh'
    ,recurrent_dropout=0.15)))
    model.add(Dropout(hparams[HP_DROPOUT]))
    #model.add(Dense(50, activation='tanh'))
    #model.add(Dropout(hparams[HP_DROPOUT]))
    model.add(Dense(1, activation="sigmoid"))
```

(continue next slide)

```
else: raise ValueError("unexpected architecture name: %r" % (arch_name,))

  loss = keras.losses.BinaryCrossentropy(from_logits=False)
  optimizer_name = hparams[HP_OPTIMIZER]
  learning_rate = hparams[HP_L_RATE]


  if optimizer_name == "adam":
      optimizer = tf.keras.optimizers.Adam(learning_rate = learning_rate)
  elif optimizer_name == "AdaDelta":
      optimizer = tf.keras.optimizers.Adadelta(learning_rate = learning_rate, rho=0.9, epsilon=1e-06, name="Adadelta")
  else:
      raise ValueError("unexpected optimizer name: %r" % (optimizer_name,))

  model.compile(loss = loss, optimizer = optimizer, metrics = metric)
  model.summary()


  return model
```

```python
def fit_model(model, x_train, y_train, x_val, y_val, batch_size, epochs, path, logdir, hparams):

  callbacks = [#EarlyStopping(monitor = 'val_accuracy', mode='max', verbose=1, patience=2),
               #ModelCheckpoint(path, monitor = 'val_accuracy', mode='max', verbose=1, save_best_only=True),
               TensorBoard(log_dir=logdir, histogram_freq=1),
               hp.KerasCallback(logdir, hparams)
               ]

  # The returned "out_model" object holds a record of the loss values and metric values during training:
  out_model = model.fit(x_train, y_train, validation_data = (x_val, y_val), batch_size = batch_size,
                        epochs = epochs, shuffle=True, callbacks=callbacks, verbose=1)

  return out_model
```

```python
def evaluate(model, x_test, y_test, batch_sizem):

  print("evaluate on test dataset:\n")
  loss_test, acc_test, auc_test =  model.evaluate(x_test, y_test, batch_size, verbose=1)

  return loss_test, acc_test, auc_test
```

```python
def confusion(preds, labels):
  tn, fp, fn, tp = confusion_matrix(labels, preds).ravel()
  return tn, fp, fn, tp

def get_precision(preds, labels):
  tn, fp, fn, tp = confusion(preds, labels)
  return tp / (tp + fp)

def get_recall(preds, labels):
  tn, fp, fn, tp = confusion(preds, labels)
  return tp / (tp + fn)

def get_F1(preds, labels):
  precision = get_precision(preds, labels)
  recall = get_recall(preds, labels)
  F1= 2*(precision*recall)/(precision+recall)
  return F1

def get_accuracy(preds, labels):
  tn, fp, fn, tp = confusion(preds, labels)
  accuracy = (tn+tp)/(fp+fn+tn+tp)
  return accuracy
```