Classifying influenza-related Tweets as influenza-aware or influenza influenza with Recurrent Neural Networks (Bidirectional GRU with Attention)

```
import os
import re
import json
import string
import logging
import multiprocessing

from utils import *
import numpy as np
import pandas as pd
from pylab import rcParams
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# Twitter preprocess, tokenizer
import twokenize
import preprocess_twitter
import preprocessor as p
p.set_options(p.OPT.URL, p.OPT.HASHTAG, p.OPT.EMOJI,
               p.OPT.SMILEY, p.OPT.RESERVED, p.OPT.MENTION)

def tokenize(row):
    punct = string.punctuation.replace("-", "")
    punct = string.punctuation.replace("'", "")
    exclude = set(string.punctuation)
    text = str(row)

    if text is not None:
        text = text.lower()
        text = ''.join(ch for ch in text if ch not in exclude)
        text = re.sub('['+punct+']', ' ', text)
        tokens = twokenize.tokenize(text)
        return tokens
    else:
        return None

# Keras utils
from keras.preprocessing.text import Tokenizer
from keras.utils.np_utils import to_categorical
from keras.preprocessing.sequence import pad_sequences

# Tensorflow
import tensorflow as tf
from tensorflow.contrib.rnn import GRUCell, LSTMCell
from tensorflow.python.ops.rnn import bidirectional_dynamic_rnn
from tensorflow.contrib.layers import fully_connected

# Convert labels to probability
cls2probs = lambda x: [1., 0] if x == 0 else [0., 1]

# Notebook related
from IPython.display import clear_output
from IPython.core.display import HTML
from IPython.display import display

logging.getLogger("tensorflow").setLevel(logging.WARNING)
pd.set_option('max_colwidth',200)
%matplotlib inline
```

Loading tweets dataset

Awareness vs Infection Tweets Classification

- **0: Influenza infection**
- **1: Influenza awareness**

```
df = pd.read_csv("AwareInfection_all.csv", sep = "\t",
                  engine="c", header=None)
# Some oddly formated lines
df = df[[0,1]]
df.dropna(inplace=True)
df[1] = df[1].astype(int)
# Shuffle df
df = df.sample(frac=1).reset_index()[[0,1]]
```

```
df.shape
```

```
(3151, 2)
```

```
df[1].value_counts()
```

```
1    1703
0    1448
Name: 1, dtype: int64
```

The Aware vs Infection dataset contains 3151 manually labelled data points. 1703 are labelled as 'Infection' while 1448 are labelled as 'Awareness'. This small difference does not imply an imbalanced class representation.

Pre-processing and Tokenizing Tweets

As recommended by Haddi, 2013 we cleaned specific Twitter sequences, such as RT (retweet), user references (@user), emoticons (":)", etc) and also URLs (http://*).

```
text = list(map(p.clean, df[0]))  # Pre-processing
text = list(map(tokenize, text))  # Pre-processing
text = [" ".join(i) for i in text]

# Maximum vocabulary size
MAX_NB_WORDS = 6000

tokenizer = Tokenizer(num_words = MAX_NB_WORDS)
tokenizer.fit_on_texts(text)
sequences = tokenizer.texts_to_sequences(text)

word_index = tokenizer.word_index
print('%s unique tokens.' % len(word_index))
```

```
5559 unique tokens.
```

```
# Define end of sequence ("eos") as 0, to match zero-padding.
eos_id = 0

# Max sequence length (nr. of words)
MAX_SEQUENCE_LENGTH = 35

# zero-padding (post-sequence)
data = pad_sequences(sequences, padding="post",
                      maxlen=MAX_SEQUENCE_LENGTH)
```

Test-train split

We considered a 15%-85% test-train split.

```
labels = list(df[1])
labels = to_categorical(np.asarray(labels))
print('Data tensor shape:', data.shape)
print('Label tensor shape:', labels.shape)

VALIDATION_SPLIT = 0.15
np.random.seed(42)
indices = df.index.tolist()
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

xx_train = data[:-nb_validation_samples]
yy_train = labels[:-nb_validation_samples]
xx_val = data[-nb_validation_samples:]
yy_val = labels[-nb_validation_samples:]

print('Training shape: ', xx_train.shape)
print('Validation shape: ', xx_val.shape)
```

```
Data tensor shape: (3151, 35)
Label tensor shape: (3151, 2)
Training shape:  (2679, 35)
Validation shape:  (472, 35)
```

Word embeddings

Word embeddings trained on Twitter data can be found in the GloVe project. These were trained on 2 billion tweets, comprising 27 billion tokens with a vocabulary length of around 1.2 million. We used the highest embedding dimension availaible, 200.

```
embeddings_index = {}

with open('glove.twitter.27B.200d.txt') as file:
    for line in file:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

print('Found %s word vectors.' % len(embeddings_index))
```

```
Found 1193514 word vectors.
```

```
# Word embeddings dimension
# GloVe Twitter 200d
EMBEDDING_DIM = 200

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
embeddings = embedding_matrix
```

Bidirectional RNN (GRU) with Attention Mechanism

Based on Ilya's TensorFlow implementation.

```
# Network Parameters
EMBED_DIM = 200
HIDDEN_SIZE = 32 # GRU
ATTENTION_SIZE = 256
```

```
NUM_CLASSES = 2

# Optimizer (Adam) parameters
LEARNING_RATE = 0.000175
EPSILON = 1e-4
BETA1 = 0.9
BETA2 = 0.99

# L2 regularization coefficient
BETA = 0

# Probability of keeping a neuron
DROPOUT = 0.5

# Batch size and epochs
BATCH_SIZE = 96
EPOCHS = 32
```

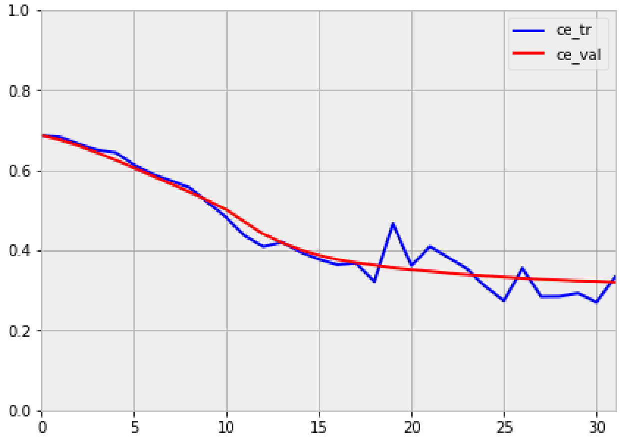
```
tf.reset_default_graph()

# Placeholders
batch_ph    = tf.placeholder(tf.int32, [None, MAX_SEQUENCE_LENGTH])
target_ph   = tf.placeholder(tf.float32, [None, NUM_CLASSES])
seq_len_ph = tf.placeholder(tf.int32, [None])
keep_prob_ph = tf.placeholder(tf.float32)
embeddings_ph = tf.placeholder(tf.float32, [len(word_index)+1, EMBED_DIM])
```

```
# Embedding layer
embeddings_ph = tf.placeholder(tf.float32, [len(word_index)+1, EMBED_DIM])
embeddings_var = tf.Variable(tf.constant(0., shape=[len(word_index)+1, EMBED_DIM]),
                                trainable=False)
init_embeddings = embeddings_var.assign(embeddings_ph)
batch_embedded = tf.nn.embedding_lookup(embeddings_var, batch_ph)

# Bi-RNN layer
outputs, _ = tf.nn.bidirectional_dynamic_rnn(GRUCell(HIDDEN_SIZE), GRUCell(HIDDEN_SIZE),
                     inputs=batch_embedded,sequence_length=seq_len_ph,
                                                  dtype=tf.float32, scope="bi_rnn1")
outputs = tf.concat(outputs, 2)


# Attention mechanism
W_omega = tf.Variable(tf.random_normal([2 * HIDDEN_SIZE, ATTENTION_SIZE], stddev=0.1))
b_omega = tf.Variable(tf.random_normal([ATTENTION_SIZE], stddev=0.1))
u_omega = tf.Variable(tf.random_normal([ATTENTION_SIZE], stddev=0.1))

v = tf.tanh(tf.matmul(tf.reshape(outputs, [-1, 2 * HIDDEN_SIZE]), W_omega) +
             tf.reshape(b_omega, [1, -1]))

vu = tf.matmul(v, tf.reshape(u_omega, [-1, 1]))
exps = tf.reshape(tf.exp(vu), [-1, MAX_SEQUENCE_LENGTH])
alphas = exps / tf.reshape(tf.reduce_sum(exps, 1), [-1, 1])

# Output of Bi-RNN reduced with attention vector
output = tf.reduce_sum(outputs * tf.reshape(alphas, [-1, MAX_SEQUENCE_LENGTH, 1]), 1)

# Dropout
drop = tf.nn.dropout(output, keep_prob_ph)

# Fully connected layer
W = tf.Variable(tf.truncated_normal([HIDDEN_SIZE * 2, NUM_CLASSES], stddev=0.1), name="W")
b = tf.Variable(tf.constant(0., shape=[NUM_CLASSES]), name="b")
y_hat = tf.nn.xw_plus_b(output, W, b, name="scores")

# Loss function
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=target_ph,
                                                                              logits=y_hat),
                                  name="cross_entropy")
l2_loss = tf.nn.l2_loss(W, name="l2_loss")
loss = cross_entropy + l2_loss * BETA

# Optimizer
optimizer = tf.train.AdamOptimizer(learning_rate=LEARNING_RATE,
                                        beta1=BETA1, beta2=BETA2,
                                        epsilon=EPSILON).minimize(loss)

# Accuracy
accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(target_ph, 1),
                                                tf.argmax(y_hat, 1)),
                                     tf.float32))
```

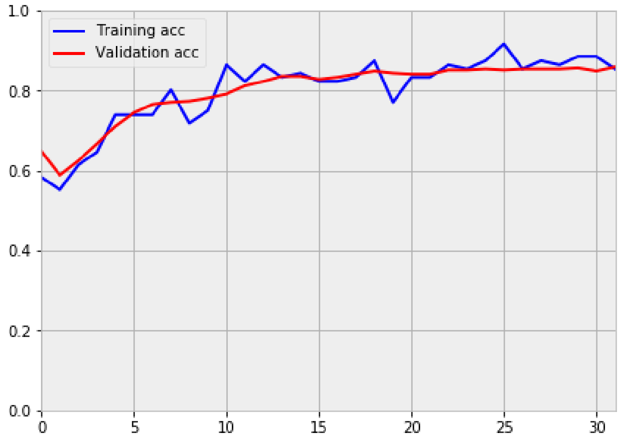
```python
# Initialize mode
train_batch_generator = batch_generator(xx_train, yy_train, BATCH_SIZE)

loss_tr_l = []
loss_val_l = []
ce_tr_l = []  # Cross-entropy
ce_val_l = []
acc_tr_l = []  # Accuracy
acc_val_l = []

with tf.Session() as sess:

    sess.run(tf.global_variables_initializer())
    sess.run(init_embeddings, feed_dict={embeddings_ph: embeddings})

    for epoch in range(EPOCHS):
        for i in range(int(xx_train.shape[0] / BATCH_SIZE)):

            x_batch, y_batch = train_batch_generator.__next__()

            # Using variable sequence length
            seq_len_tr = np.array([list(x).index(eos_id) + 1 for x in x_batch])
            sess.run(optimizer, feed_dict={batch_ph: x_batch, target_ph: y_batch,
                                           seq_len_ph: seq_len_tr, keep_prob_ph: DROPOUT})

        y_pred_tr, ce_tr, loss_tr, acc_tr = sess.run([y_hat, cross_entropy, loss, accuracy],
                                                    feed_dict={batch_ph: x_batch,
                                                               target_ph: y_batch,
                                                               seq_len_ph: seq_len_tr,
                                                               keep_prob_ph: 1.0})

        y_pred_val, ce_val, loss_val, acc_val = [], 0, 0, 0
        num_val_batches = int(xx_val.shape[0] / BATCH_SIZE)
        for i in range(num_val_batches):
            x_batch_val, y_batch_val = xx_val[i * BATCH_SIZE : (i + 1) * BATCH_SIZE],\
                                       yy_val[i * BATCH_SIZE : (i + 1) * BATCH_SIZE]
            seq_len_val = np.array([list(x).index(eos_id) + 1 for x in x_batch_val])
            y_pred_val_, ce_val_, loss_val_, acc_val_ = sess.run([y_hat,
                                                                  cross_entropy,
                                                                  loss, accuracy],
                                                        feed_dict={batch_ph: x_batch_val,
                                                                   target_ph: y_batch_val,
                                                                   seq_len_ph: seq_len_val,
                                                                   keep_prob_ph: 1.0})

            y_pred_val += list(y_pred_val_)
            ce_val += ce_val_
            loss_val += loss_val_
            acc_val += acc_val_

        y_pred_val = np.array(y_pred_val)
        ce_val /= num_val_batches
        loss_val /= num_val_batches
        acc_val /= num_val_batches

        y_pred_tr = np.array([cls2probs(cls) for cls in np.argmax(y_pred_tr, 1) - 1])
        y_pred_val = np.array([cls2probs(cls) for cls in np.argmax(y_pred_val, 1) - 1])


        loss_tr_l.append(loss_tr)
        loss_val_l.append(loss_val)
        ce_tr_l.append(ce_tr)
        ce_val_l.append(ce_val)
        acc_tr_l.append(acc_tr)
        acc_val_l.append(acc_val)



        clear_output(wait=True)
        print("epoch: {}".format(epoch))
        print("\t Train loss: {:.3f}\t ce: {:.3f}\t acc: {:.3f}".format(
            loss_tr, ce_tr, acc_tr))
        print("\t Valid loss: {:.3f}\t ce: {:.3f}\t acc: {:.3f}".format(
            loss_val, ce_val, acc_val))

        plt.figure(figsize=(7,5))
        plt.plot(ce_tr_l, color='blue', label='ce_tr')
        plt.plot(ce_val_l, color='red', label='ce_val')
        plt.xlim(0, EPOCHS - 1)
        plt.ylim(0, 1)
        plt.legend()
        plt.show()
```

```
    results = [acc_val]

    saver = tf.train.Saver()
    saver.save(sess, 'BiGRU_att_1')
```

```
epoch: 31
        Train loss: 0.334       ce: 0.334       acc: 0.854
        Valid loss: 0.320       ce: 0.320       acc: 0.859
```




```
plt.figure(figsize=(7,5))
plt.plot(acc_tr_l, color='blue', label='Training acc')
plt.plot(acc_val_l, color='red', label='Validation acc')
plt.xlim(0, EPOCHS - 1)
plt.ylim(0, 1)
plt.legend()
plt.show()
```



After 31 epochs we attained ~86% of validation accuracy (85% training accuracy), with similiar cross-entropy levels (around 0.32-0.33) in both sets.

Classification of influenza-related tweets

The trained RNN is now used to classify a set of around 800 thousand English influenza-related tweets extracted from the SNAP Twitter dataset.

```
def return_preds(text):

    all_preds = []
    top_words = []

    # Pre-processing
    text = list(map(p.clean, text))
    text = list(map(tokenize, text))
    text = [" ".join(i) for i in text]
    input_seqs = tokenizer.texts_to_sequences(text)
```

```
    input_seqs = pad_sequences(input_seqs, maxlen=MAX_SEQUENCE_LENGTH, padding='post')

    # Get predictions and top words
    with tf.Session() as sess:
        new_saver = tf.train.import_meta_graph('BiGRU_att_1.meta')
        new_saver.restore(sess, tf.train.latest_checkpoint('./'))
        for i in input_seqs:
            x_batch = [i]
            seq_len = np.array([MAX_SEQUENCE_LENGTH])
            y_pred, alphas_pred = sess.run([y_hat, alphas],feed_dict={batch_ph: x_batch,
                                                                      seq_len_ph: seq_len,
                                                                      keep_prob_ph: 1.0})
            all_preds.append(y_pred)
            top_words.append(i[np.argmax(alphas_pred)])

    # Convert softmax output to class, convert word index to word
    all_preds = [np.argmax(i) for i in all_preds]
    top_words = [list(word_index.keys())[list(word_index.values()).index(i)]
                 if i > 0 else 'None' for i in top_words]

    return all_preds, top_words
```

```
# Load english flu-related tweets (from SNAP7 Twitter dataset)
topred = pd.read_csv("/home/user/Thesis/FluTweets2009_EN.csv", sep="\t", header=None)
```

```
%%time
preds, top = return_preds(topred[1].tolist())
```

```
CPU times: user 5h 52min 17s, sys: 1h 19min 41s, total: 7h 11min 58s
Wall time: 1h 34min 42s
```

```
topred['preds'] = preds
topred['topword'] = top
```

```
topred.preds.value_counts()
```

```
1    606163
0    212135
Name: preds, dtype: int64
```

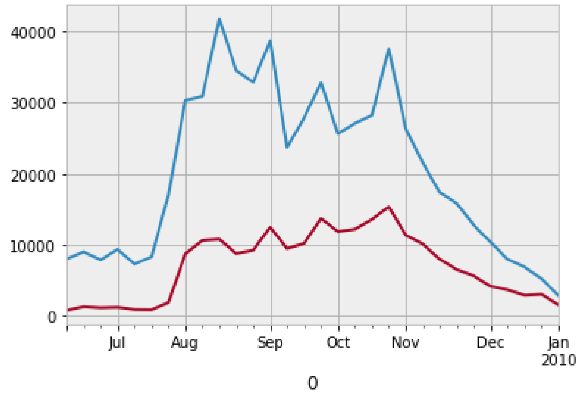
~3/4 of the tweets were classified as awareness, ~1/4 classified as infection.

```
topred.index = pd.to_datetime(topred[0], infer_datetime_format=True)
topred.drop(0, axis=1, inplace=True)
```

We show the absolute count of classified tweets overtime.

```
ax = topred[topred['preds']==1]['05-2009':'01-2010'].groupby(pd.TimeGrouper('W')).size().plot()
topred[topred['preds']==0]['05-2009':'01-2010'].groupby(pd.TimeGrouper('W')).size().plot(ax = ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f041f39cb38>
```




```
prop = pd.DataFrame(topred[topred['preds']==1]['05-2009':'01-2010'].groupby(pd.TimeGrouper('W')).size())
prop.columns = ['Awareness']
prop = prop.join(pd.DataFrame(topred[topred['preds']==0]['05-2009':'01-
2010'].groupby(pd.TimeGrouper('W')).size()))
prop.columns = ['Awareness', 'Infection']
```

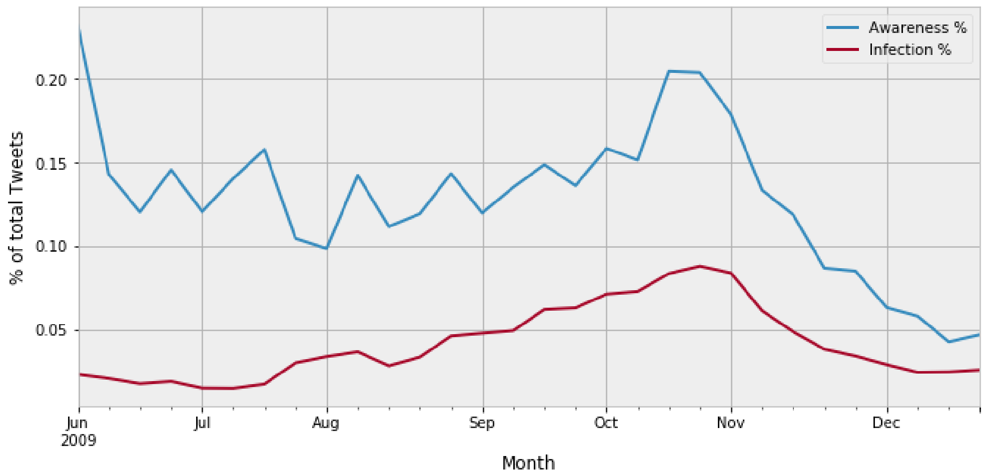
```
weekly = pd.read_csv("/home/user/Thesis/dailyTweets.csv", sep="\t", header=None)
weekly.index = pd.to_datetime(weekly[0], infer_datetime_format=True)
weekly = weekly.groupby(pd.TimeGrouper('W')).sum()
```

```
prop = prop.join(weekly)
prop.index.name = "Month"
prop['Awareness %'] = prop['Awareness'] / prop[1] * 100
prop['Infection %'] = prop['Infection'] / prop[1] * 100
prop[['Awareness %', 'Infection %']].to_csv("/home/user/Thesis/AwInf_prop.csv", sep="\t")
```

```
monthly_prop = prop[['Awareness', 'Infection', 1]].groupby(pd.TimeGrouper('M')).sum()
monthly_prop['Awareness %'] = monthly_prop['Awareness'] / monthly_prop[1] * 100
monthly_prop['Infection %'] = monthly_prop['Infection'] / monthly_prop[1] * 100
monthly_prop.to_csv("/home/user/Thesis/Twitter_AwInf_monthly.csv", sep="\t")
```

Proportion of classified tweets over time.

```
ax = prop[['Awareness %','Infection %' ]].plot(figsize=(11,5))
ax.set_ylabel("% of total Tweets")
fig = ax.get_figure()
fig.savefig('/home/user/Thesis/Twitter_AWvsINF.pdf', format='pdf')
```




```
prop.index = prop.index.shift(freq='W', n=-1)
```

The top words (with the most weight) for classification can now be extracted.

```
topred.groupby("preds")['topword'].value_counts()[0].head(15).tail(-1)
```

```
topword
got         21341
sick        21321
have        13697
getting      6381
up           6143
feel         4370
feeling      4363
from         4348
been         3712
down         3562
had          3166
fever        2890
caught       2750
swine        2487
Name: topword, dtype: int64
```

```
topred.groupby("preds")['topword'].value_counts()[1].head(15).tail(-1)
```

```
topword
vaccine        71022
swine          42119
shot           35655
shots          16682
from           13814
deaths          9800
vaccination     8864
cases           8447
health          7587
influenza       5566
up              5441
have            4773
out             4043
vaccines        3965
Name: topword, dtype: int64
```

```
import scipy.stats as stats
```

```
topred[(topred['topword']!='flu') & (topred['preds']==0)]['topword']['2009-06':'2009-12'].groupby(
    pd.TimeGrouper('M')).apply(
    lambda x: x.value_counts().head(2))
```

```
2009-06-30  got      284
            sick     270
2009-07-31  got     1061
            sick     724
2009-08-31  got     4462
            sick    3764
2009-09-30  sick    5441
            got     5328
2009-10-31  sick    6498
            got     5487
2009-11-30  sick    3270
            got     3222
2009-12-31  got     1497
            sick    1354
Name: topword, dtype: int64
```

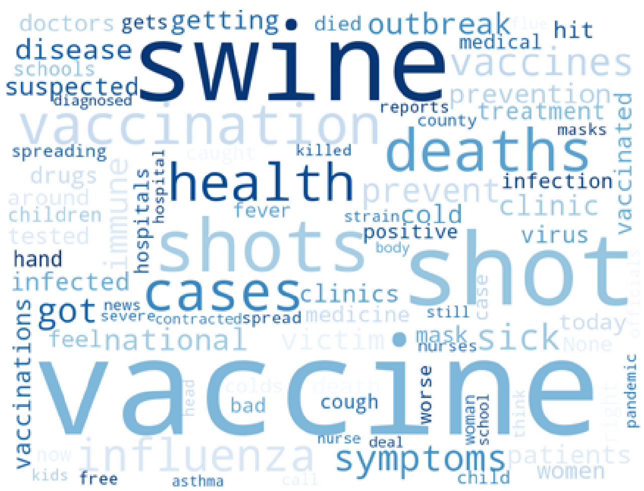
```
topred[(topred['topword']!='flu') & (topred['preds']==1)]['topword']['2009-06':'2009-12'].groupby(
    pd.TimeGrouper('M')).apply(
    lambda x: x.value_counts().head(2))
```

```
2009-06-30  swine       2425
            cases       1499
2009-07-31  vaccine     6051
            swine       5306
2009-08-31  swine      14115
            vaccine    12005
2009-09-30  vaccine    14371
            shot        8577
2009-10-31  vaccine    21296
            shot       14389
2009-11-30  vaccine    11324
            shot        7248
2009-12-31  vaccine     4911
            shot        2684
Name: topword, dtype: int64
```

Wordclouds of most most decisive words for each classification

```python
aw_text = " ".join([i for i in topred[topred['preds']==1]['topword'].tolist() if i != "flu"])
inf_text = " ".join([i for i in topred[topred['preds']==0]['topword'].tolist() if i != "flu"])
```

```python
from wordcloud import WordCloud
wordcloud_aw = WordCloud(random_state=42, min_font_size=9, width=1024, height=768,
                         background_color='white', margin=1,
                         colormap='Blues_r',
                         max_words=90,
                         normalize_plurals=False,
                         collocations=False,).generate(aw_text)
plt.figure(figsize=(9,6))
ax = plt.imshow(wordcloud_aw, interpolation="spline16")
plt.axis("off")
plt.show()
fig = ax.get_figure()
fig.savefig('/home/user/Thesis/Awareness_WordCloud.pdf', format='pdf')
```




```python
wordcloud_inf = WordCloud(random_state=42, margin=1, width=1024, height=768,
                         background_color='white', min_font_size=9,
                         colormap='Reds_r',
                         max_words=90,
                         normalize_plurals=False,
                         collocations=False).generate(inf_text)
plt.figure(figsize=(9,6))
ax = plt.imshow(wordcloud_inf, interpolation="spline16")
plt.axis("off")
plt.show()
fig = ax.get_figure()
fig.savefig('/home/user/Thesis/Infection_WordCloud.pdf', format='pdf')
```

Visualization of attention weights

```python
def return_attention_vis(text):

    # Pre-processing
    text = list(map(p.clean, text))
    text = list(map(tokenize, text))
    text = [" ".join(i) for i in text]
    input_seqs = tokenizer.texts_to_sequences(text)
    input_seqs = pad_sequences(input_seqs, maxlen=MAX_SEQUENCE_LENGTH, padding='post')

    # Get weights
    with tf.Session() as sess:
        new_saver = tf.train.import_meta_graph('BiGRU_att_1.meta')
        new_saver.restore(sess, tf.train.latest_checkpoint('./'))
        x_batch_test = input_seqs[0:1]
        seq_len_test = np.array([MAX_SEQUENCE_LENGTH])
        y_pred_test, alphas_test = sess.run([y_hat, alphas],
                                            feed_dict={batch_ph: x_batch_test,
                                                       seq_len_ph: seq_len_test,
                                                       keep_prob_ph: 1.0})

    # Display weights
    html = '<font size="5">'
    fmt = ' <span style="background-color: #{0:x}{0:x}ff">{1}</span>'
    text = text[0]
    min_max_scaler = MinMaxScaler(feature_range=(0,1))
    for word, coef in zip(text.split(),
                          min_max_scaler.fit_transform(alphas_test[0][:len(text.split())].reshape(-1,
1))):
        c = int(256*((1.-0.1)*(1.-coef)+0.1))
        html += fmt.format(c, word)
    display(HTML(html))
    print("Y_pred:", y_pred_test)
```

```python
# This chunk is only relevant for Jupyter Notebook aesthetics.
from pylab import rcParams
import json
from IPython.core.display import HTML
s = json.load( open("/home/user/Thesis/bmh_matplotlibrc.json") )
rcParams.update(s)
def css_styling():
    styles = open("/home/user/Thesis/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

```
/home/user/anaconda3/lib/python3.6/site-packages/matplotlib/__init__.py:913: UserWarning:
axes.color_cycle is deprecated and replaced with axes.prop_cycle; please use the latter.
  warnings.warn(self.msg_depr % (key, alt_key))
```