Sentiment Analysis using Convolutional Neural Network (CNN)

In [ ]:

```
"""Author: Siddhant Shrivastava
<siddhant.shrivastava23@gmail.com>\nSentiment Analysis using word2vec, NN.
\
NLP Project; Siddhant Shrivastava, Aditya Srivastava, Pranav Nair; Monsoon
2017, IIIT-Hyderabad"""
print(__doc__)
```

Reference: Sentiment analysis on Twitter using word2vec and keras by Ahmed Besbes

Import Modules

In [ ]:

```
import pandas as pd
pd.options.mode.chained_assignment = None
import numpy as np
from copy import deepcopy
from string import punctuation
from random import shuffle

import gensim
from gensim.models.word2vec import Word2Vec # the word2vec model gensim cla
ss
LabeledSentence = gensim.models.doc2vec.LabeledSentence

from tqdm import tqdm
tqdm.pandas(desc="progress-bar")

from nltk.tokenize import TweetTokenizer
tokenizer = TweetTokenizer()

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
```

Define Functions

Function to load the dataset and extract the columns we need

In [ ]:

```
def ingest():
    """Load dataset, extract the sentiment and tweet's text columns"""
    data = pd.read_csv('../../dataset/cnn_dataset/tweets.csv', encoding="IS
O-8859-1")
    # data.drop(['ItemID', 'Date', 'Blank', 'SentimentSource'], axis=1, inp
lace=True)
    data.drop(['ItemID', 'SentimentSource'], axis=1, inplace=True)
    data = data[data.Sentiment.isnull() == False]
    data['Sentiment'] = data['Sentiment'].map(int)
    data = data[data['SentimentText'].isnull() == False]
    data.reset_index(inplace=True)
    data.drop('index', axis=1, inplace=True)
    data['Sentiment'] = data['Sentiment'].map({4:1, 0:0})
    print('dataset loaded with shape: ' + str(data.shape))
    return data
```

Tokenizing function

Splits each tweet into tokens and removes user mentions, hashtags and urls as they do not provide enough semantic information for the task

In [ ]:

```
def tokenize(tweet):
    try:
        # tweet = unicode(tweet.decode('utf-8').lower())
        # tweet = unicode(tweet.decode('latin-1').lower())
        tweet = tweet.lower()
        tokens = tokenizer.tokenize(tweet)
        # tokens = filter(lambda t: not t.startswith('@'), tokens)
        # tokens = filter(lambda t: not t.startswith('#'), tokens)
        # tokens = filter(lambda t: not t.startswith('http'), tokens)
        return tokens
    except:
        return 'NC'
```

Process tokenized data

Tokenization results should now be cleaned to remove lines with 'NC', resulting from a tokenization error

In [ ]:

```
def postprocess(data, n=1000000):
    data = data.head(n)
    data['tokens'] = data['SentimentText'].progress_map(tokenize)  ##
progress_map is a variant of the map function plus a progress bar. Handy
to monitor DataFrame creations.
    print("Tokenization done")
    print(data.head(5))
    # print(data.tokens.value_counts())
    data = data[data.tokens != 'NC']
    data.reset_index(inplace=True)
    data.drop('index', inplace=True, axis=1)
    return data
```

Function to turn tokens to LabeledSentence objects before feeding to the word2vec model

In [ ]:

```
def labelizeTweets(tweets, label_type):
    labelized = []
    for i,v in tqdm(enumerate(tweets)):
        label = '%s_%s'%(label_type,i)
        labelized.append(LabeledSentence(v, [label]))
    return labelized
```

Function to create averaged tweet vector

In [ ]:

```
def buildWordVector(tokens, size):
    """Given a list of tweet tokens, creates an averaged tweet vector."""
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in tokens:
        try:
            vec += tweet_w2v[word].reshape((1, size)) * tfidf[word]
            count += 1.
        except KeyError: # handling the case where the token is not
                         # in the corpus. useful for testing.
            continue
    if count != 0:
        vec /= count
    return vec
```

Load and Process Data

In [ ]:

```
data = ingest()
data.head(5)
```

In [ ]:

```
data.Sentiment.value_counts()
# {'0': "negative sentiment", '1': "positive sentiment"}
```

Tokenize and clean data

In [ ]:

```
data = postprocess(data)
```

We are considering 1,000,000 (1 million) records.

In [ ]:

```
data.shape
```

In [ ]:

```
n = 1000000
```

Build the word2vec model

Define the training and test dataset

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(np.array(data.head(n).t
okens),
                                                    np.array(data.head(n).Se
timent), test_size=0.2)
```

Turn tokens into LabeledSentence Object

Before feeding lists of tokens into the word2vec model, we must turn them into LabeledSentence objects beforehand.

In [ ]:

```
x_train.shape
```

In [ ]:

```
x_train = labelizeTweets(x_train, 'TRAIN')
x_test = labelizeTweets(x_test, 'TEST')
```

In [ ]:

```
print(x_train[0])
```

Build the word2vec model from x_train i.e. the corpus.

Set the number of dimensions of the vector space

In [ ]:

```
n_dim = 200
```

In [ ]:

```
tweet_w2v = Word2Vec(size=n_dim, min_count=10)
tweet_w2v.build_vocab([x.words for x in tqdm(x_train)])
tweet_w2v.train([x.words for x in tqdm(x_train)],total_examples=tweet_w2v.c
orpus_count, epochs=tweet_w2v.iter)
```

Check semantic realatioship set by word2vec

In [ ]:

```
tweet_w2v.most_similar('good')
```

Build the Sentiment Classifier

Build the tf-idf matrix

to compute the tf-idf score which is a weighted average where each weight gives the importance of the word with respect to the corpus.

In [ ]:

```
print('building tf-idf matrix ...')
vectorizer = TfidfVectorizer(analyzer=lambda x: x, min_df=10)
matrix = vectorizer.fit_transform([x.words for x in x_train])
tfidf = dict(zip(vectorizer.get_feature_names(), vectorizer.idf_))
print('vocab size : %s' % (len(tfidf)))
```

Convert x_train and x_test to a list of vectors

Also scale each column to have zero mean and unit standard deviation.

In [ ]:

```
from sklearn.preprocessing import scale
train_vecs_w2v = np.concatenate([buildWordVector(z, n_dim) for z in tqdm(map(lambda x: x.words, x_train))])
train_vecs_w2v = scale(train_vecs_w2v)

test_vecs_w2v = np.concatenate([buildWordVector(z, n_dim) for z in tqdm(map(lambda x: x.words, x_test))])
test_vecs_w2v = scale(test_vecs_w2v)
```

Convolutional Neural Network

Import Modules

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, Convolution1D, Flatten, Dropout
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
```

Pad Sequence to the same length

In [ ]:

```
max_review_length = 1600
```

Using embeding from Keras

In [ ]:

```
embedding_vecor_length = 300
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))
```

Convolutional model (3x conv, flatten, 2x dense)

In [ ]:

```
model.add(Convolution1D(64, 3, border_mode='same'))
model.add(Convolution1D(32, 3, border_mode='same'))
model.add(Convolution1D(16, 3, border_mode='same'))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(180,activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(1,activation='sigmoid'))

model.fit(train_vecs_w2v, y_train, nb_epoch=9, batch_size=128, verbose=2)
```

Evaluate on test set

In [ ]:

```
score = model.evaluate(test_vecs_w2v, y_test, batch_size=128, verbose=2)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Accuracy: 81.46% (~81.5%)

Save neural net

Save and Load Keras Deep Learning models

In [ ]:

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```