(http://ahmedbesbes.com)

Ahmed BESBES (http://ahmedbesbes.com)

Aspiring data scientist

(https://fr.linkedin.com/in/ahmed-besbes-99a91661) (https://twitter.com/ahmed_besbes_?lang=fr) (https://github.com/ahmedbesbes) (https://www.facebook.com/ahmed.besbes.5

Sentiment analysis on Twitter using word2vec and keras

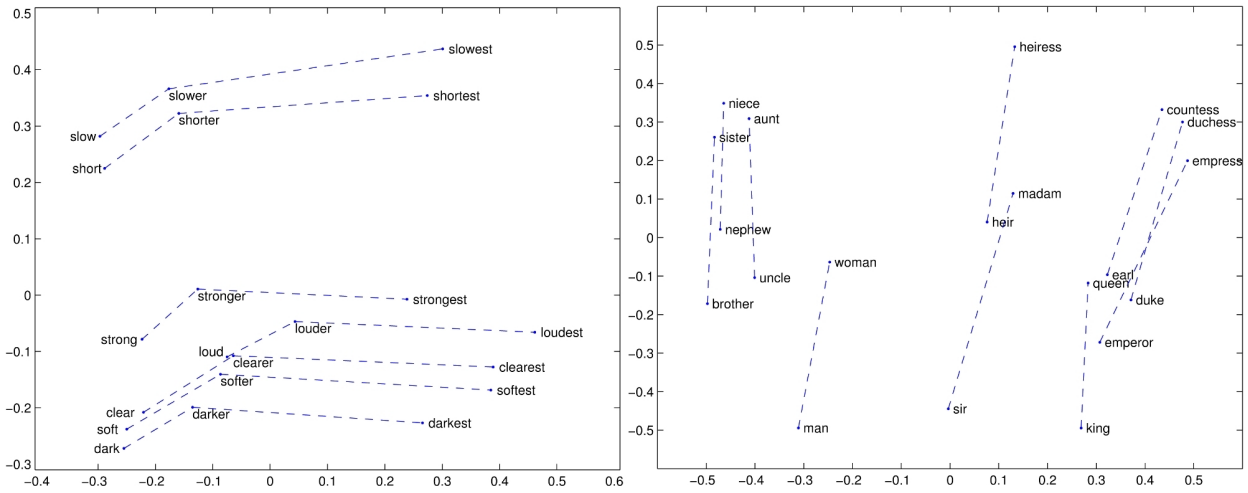
Posted on Jeu 20 avril 2017 in NLP (http://ahmedbesbes.com/category/nlp.html)

1 - Introduction

In this post I am exploring a new way of doing sentiment analysis. I'm going to use **word2vec**.

word2vec is a group of Deep Learning models developed by Google with the aim of capturing the context of words while at the same time proposing a very efficient way of preprocessing raw text data. This model takes as input a large corpus of documents like tweets or news articles and generates a vector space of typically several hundred dimensions. Each word in the corpus is being assigned a unique vector in the vector space.

The powerful concept behind word2vec is that word vectors that are close to each other in the vector space represent words that are not only of the same meaning but of the same context as well.
What I find interesting about the vector representation of words is that it automatically embeds several features that we would normally have to handcraft ourselves. Since word2vec relies on Deep Neural Nets to detect patterns, we can rely on it to detect multiple features on different levels of abstractions.
Let's look at these two charts I found in this blog (http://sebastianruder.com/secret-word2vec/). They visualize some word vectors projected on 2D space after a dimensionality reduction.



A couple of things to notice:

- On the right chart, the words of similar meaning, concept and context are grouped together. For example, niece, aunt and sister are close to each other since they describe females and family relationships. Similarly, countess, duchess and empress are grouped together because they represent female royalty. The second thing to see from this chart

is that the geometric distance between words translates a semantic relationship. For example, the vector woman - man is somewhat colinear to the vector queen - king something we would translate to *"woman is to man as queen is to king"*. This means that word2vec is able to infer different relationships between words. Something that we human do naturally.
- The chart on the left is quite similar to the one on the right except that it translates the syntaxic relationships between words. slow - slowest = short - shortest is such an example.

On a more general level, word2vec embeds non trivial semantic and syntaxic relationships between words. This results in preserving a rich context.

In this post we'll be applying the power of word2vec to build a sentiment classifier. We'll use a large dataset of 1.5 million tweets where each tweet is labeled 1 when it's positive and 0 when it's negative. The word2vec model will learn a represenation for every word in this corpus, a represenation that we'll use to transform tweets, i.e sentences, into vectors as well. Then we'll use this new represenation of tweets to train a Neural Network classifier by Keras (since we already have the labels.)

Do you see how **useful** word2vec is for this text classification problem? It provides enhanced feature engineering for raw text data (not the easiest form of data to process when building classifiers.)

Ok now let's put some word2vec in action on this dataset.

2 - Environment set-up and data preparation

Let's start by setting up the environment.
To have a clean installation that would not mess up my current python packages, I created a conda virtual environment named **nlp** on an Ubuntu 16.04 LTS machine. The python version is 2.7.

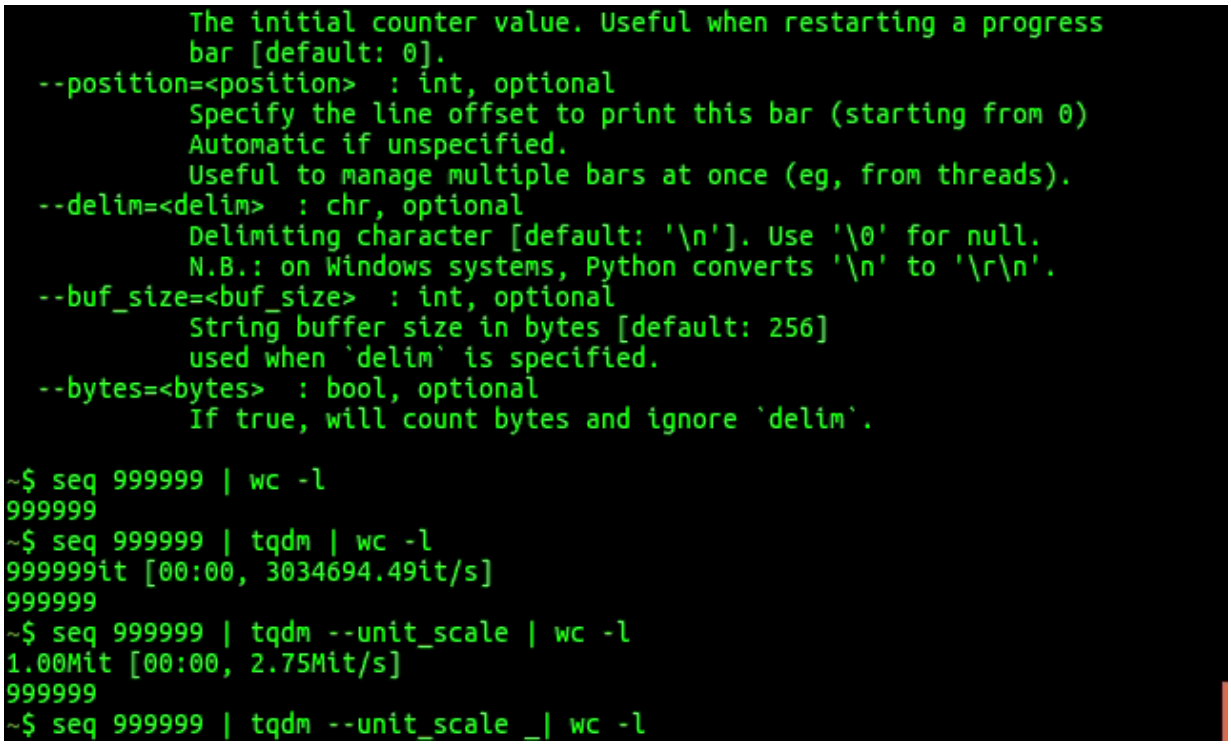
```
conda create -n nlp python=2.7 anaconda
```

Now activate the environment.

```
source activate nlp
```

Inside this virtual environment, we'll need to install these libraries:
- gensim (https://radimrehurek.com/gensim/) is a natural language processing python library. It makes text mining, cleaning and modeling very easy. Besides, it provides an implementation of the word2vec model.
- Keras (https://keras.io/) is a high-level neural networks API, written in Python and capable of running on top of either TensorFlow or Theano. We'll be using it to train our sentiment classifier. In this tutorial, it will run on top of TensorFlow.

- TensorFlow (https://www.tensorflow.org/) is an open source software library for machine learning. It's been developed by Google to meet their needs for systems capable of building and training neural networks.
- Bokeh (http://bokeh.pydata.org/en/latest/) is a Python interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of novel graphics in the style of D3.js, and to extend this capability with high-performance interactivity over very large or streaming datasets.
- tqdm (https://pypi.python.org/pypi/tqdm) is cool progress bar utility package I use to monitor dataframes creation (Yes, It integrates with pandas) and loops. Demo:



Cool, hein?

```
pip install --upgrade gensim
pip install nltk
pip install tqdm
pip install --upgrade https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-1.0.1-cp27-no
ne-linux_x86_64.whl
pip install keras
pip install bokeh
```

The environment should now be ready.
The dataset can be downloaded from this link (https://drive.google.com/uc?id=0B04GJPshIjmPRnZManQwWEdTZjg&export=download). It's a csv file that contains **1.6 million rows**. Each row has amongst other things the text of the tweet and the corresponding sentiment.

Let's load the python libraries and have a look at the dataset.

```
import pandas as pd # provide sql-like data manipulation tools. very handy.
pd.options.mode.chained_assignment = None
import numpy as np # high dimensional vector computing library.
from copy import deepcopy
from string import punctuation
from random import shuffle

import gensim
from gensim.models.word2vec import Word2Vec # the word2vec model gensim class
LabeledSentence = gensim.models.doc2vec.LabeledSentence # we'll talk about this down below

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

from nltk.tokenize import TweetTokenizer # a tweet tokenizer from nltk.
tokenizer = TweetTokenizer()

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

Let's define a function that loads the dataset and extracts the two columns we need:

- The sentiment: a binary (0/1) variable
- The text of the tweet: string

```
def ingest():
    data = pd.read_csv('./tweets.csv')
    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)
    print 'dataset loaded with shape', data.shape
    return data

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

Sentiment	SentimentText
1	@jonah_bailey Sorry about the loss. I have been there and it sucks. Have a great day!
0	I think I pulled a pectoral muscle. And no, I'm not kidding.
1	My room is TRASHED
1	Raining.
0	at work the stupidst job in the world LoL I can't wait until my last day YAY!

The format of the SentimenText is not useful. It needs to be tokenized and cleaned.
We will limit to 1 million tweets.
Here's my tokenizing function that splits each tweet into tokens and removes user mentions, hashtags and urls. These elements are very common in tweets but unfortunately they do not provide enough semantic information for the task. If you manage to sucessfully integrate them

in the final classification, please tell me your secret.

```
def tokenize(tweet):
    try:
        tweet = unicode(tweet.decode('utf-8').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'
```

The results of the tokenization should now be cleaned to remove lines with 'NC', resulting from a tokenization error (usually due to weird encoding.)

```
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.
    data = data[data.tokens != 'NC']
    data.reset_index(inplace=True)
    data.drop('index', inplace=True, axis=1)
    return data

data = postprocess(data)
```

The data is now tokenized and cleaned. We are ready to feed it in the word2vec model.

3 - Building the word2vec model

First, let's define a training set and a test set.

```
x_train, x_test, y_train, y_test = train_test_split(np.array(data.head(n).tokens),
                                                    np.array(data.head(n).Sentiment), test_size=0.
2)
```

Before feeding lists of tokens into the word2vec model, we must turn them into LabeledSentence objects beforehand. Here's how to do it:

```
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

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

Let's check the first element from x_train.

```
x_train[0]

Out[13]:
TaggedDocument(words=[u'thank', u'you', u'!', u'im', u'just', u'a', u'tad', u'sad', u'u', u'r',
u'off', u'the', u'market', u'tho', u'...'], tags=['TRAIN_0'])
```

Ok so each element is basically some object with two attributes: a list (of tokens) and a label. Now we are ready to build the word2vec model from **x_train** i.e. the corpus.

```
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)])
```

The code is self-explanatory.

- On the first line the model is initialized with the dimension of the vector space (we set it to 200) and min_count (a threshold for filtering words that appear less)
- On the second line the vocabulary is created.
- On the third line the model is trained i.e. its weights are updated.

Once the model is built and trained on the corpus of tweets, we can use it to convert words to vectors. Here's an example:

```
tweet_w2v['good']

Out[49]:
array([-1.04697919,  0.79274398,  0.23612066,  0.05131698,  0.0884205 ,
       -0.08569263,  1.45526719,  0.42579028, -0.34688851, -0.08249081,
        0.45873582,  2.36241221,  1.05570769, -1.99480379, -1.80352235,
       -0.15522274, -0.20937157, -0.07138395,  0.62345725,  1.50070465,
       -0.02835625, -0.08164574,  1.28233111,  1.75218308, -1.38450599,
        2.12812686,  0.8680591 ,  0.32937863, -0.72903335, -0.57105792,
        0.53118968, -0.39874691, -1.13426244, -1.43289971, -0.24573426,
        0.33088401, -0.88485849, -1.01581001, -0.62239277, -0.11345106,
        1.33353424, -0.49697381, -0.36537576,  0.76834393,  1.68711364,
       -1.03052866,  0.28044644,  0.41823646, -3.47753811,  0.13425322,
       -0.38362527,  2.05668998, -0.57867765,  1.93026328,  0.03931952,
        0.82015723,  0.11956126,  1.37940645, -0.47558281, -0.34119719,
        0.57716691, -1.48764825, -0.5627619 ,  0.55062455,  1.50399065,
        1.92141354,  0.68401223,  0.65160483, -0.0780897 ,  0.30544546,
        1.10016072, -0.78553891,  0.56758875, -0.1569887 , -0.65370613,
        1.4484303 ,  0.83104122,  1.25601828, -0.69895804,  1.50273371,
       -1.18243968, -0.11410122, -0.78283215,  0.49858055, -1.18107879,
       -0.27116439, -0.08542393, -1.55652511,  0.58338171,  1.57096207,
       -1.8447808 , -0.46724516, -0.38275295, -1.59960091,  0.84984666,
        0.04224969,  1.69833565,  1.54516482,  1.16857958, -1.36557281,
        0.71473658, -0.29552877, -1.55104351,  1.5771817 ,  0.29726478,
        1.40087354,  0.57571775,  0.16996922, -0.00522773,  0.06951592,
        1.81895649,  2.49026823,  0.93306369, -1.06985188, -2.24981689,
        1.37557173, -0.67554522,  1.10980988, -1.41456699,  2.3959322 ,
       -0.50968719,  2.00125957,  2.67398214,  0.1460651 , -1.47851145,
        0.87178862, -1.80060601, -0.53303391, -0.58677369,  0.53175789,
        1.96556151, -0.61592281, -1.42631042, -1.48672009,  1.13366914,
       -0.56834996,  1.63328636, -2.1681726 ,  2.34943199, -2.60359526,
        0.79754263,  0.35759249, -1.43931878,  1.48475873, -0.58964026,
       -0.01640507,  1.23911965,  0.7186386 ,  1.19023228,  0.25102213,
        0.30787438,  0.89110869, -0.21623117, -0.38933367, -0.36786464,
        0.25948352, -0.06296721, -0.46233487, -0.32560897, -1.05537581,
       -1.02237189,  1.73827136,  1.31880462,  0.82397497, -0.98476279,
       -0.67370725,  0.55092269,  2.07977676,  0.06780072,  2.09787917,
       -0.87865865, -0.09497593, -0.51874167,  2.30745101,  0.5561167 ,
        1.92726147, -0.27955261, -1.48783088,  0.59695238, -0.2615312 ,
        0.50918317, -0.53439486,  1.48705184, -1.00359023,  2.24436331,
        0.8697992 ,  1.04027569, -0.17122032,  0.26632035, -0.34332612,
       -0.40858006,  1.0153631 , -2.03206563, -0.18333849,  1.24701536,
       -1.37875533,  1.95987856,  0.2176083 ,  1.66856909,  1.58423829], dtype=float32)
```

You can check: this is a 200-dimension vector. Of course, we can only get the vectors of the words of the corpus.
Let's try something else. We spoke earlier about semantic relationships. Well, the Word2Vec gensim implementation provides a cool method named most_similar. Given a word, this method returns the top n similar ones. This is an interesting feature. Let's try it on some words:

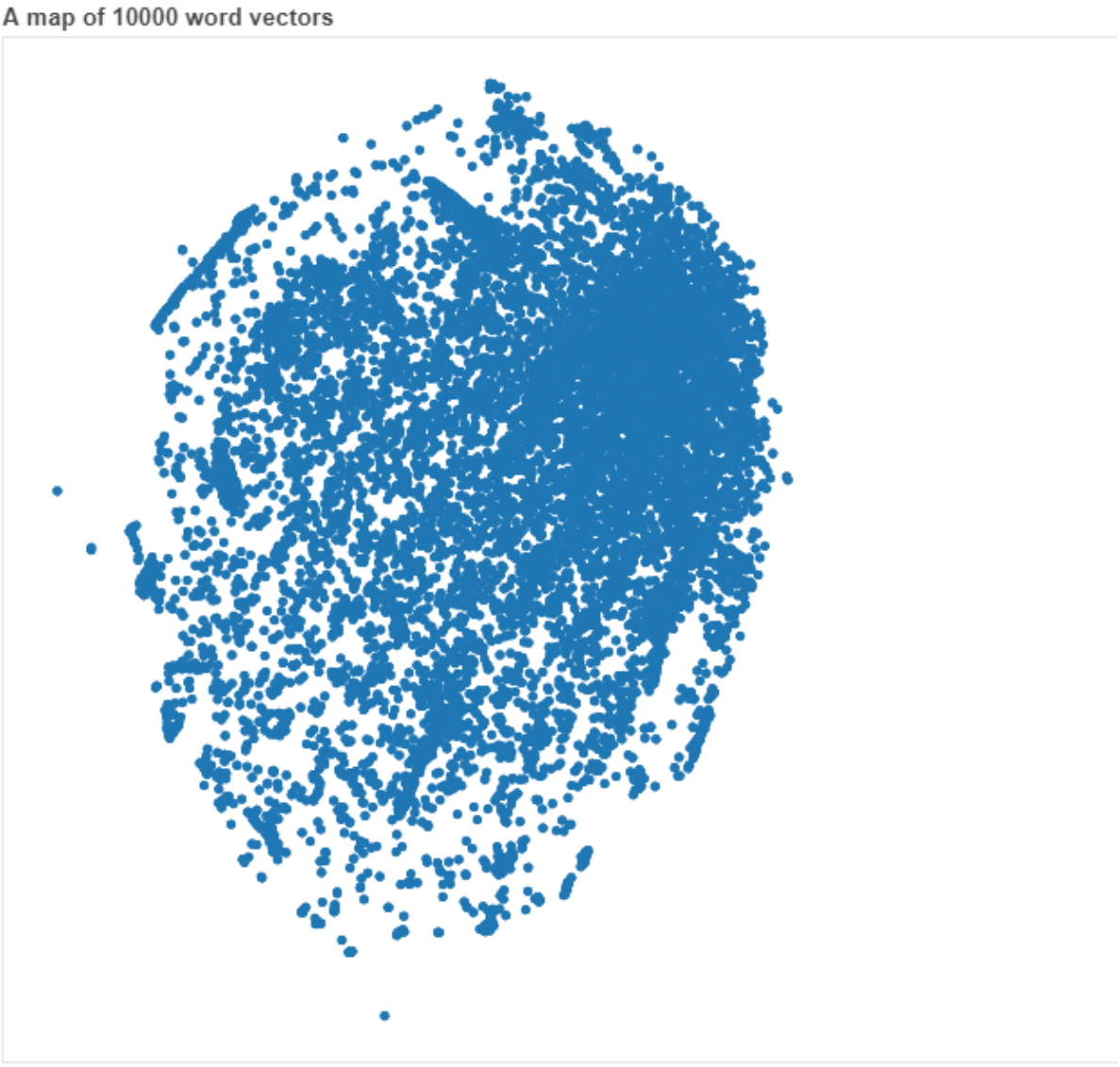
```
tweet_w2v.most_similar('good')
Out[52]:
[(u'goood', 0.7355118989944458),
 (u'great', 0.7164269685745239),
 (u'rough', 0.656904935836792),
 (u'gd', 0.6395257711410522),
 (u'goooood', 0.6351571083068848),
 (u'tough', 0.6336284875869751),
 (u'fantastic', 0.6223267316818237),
 (u'terrible', 0.6179217100143433),
 (u'gooood', 0.6099461317062378),
 (u'gud', 0.6096700429916382)]

tweet_w2v.most_similar('bar')
Out[53]:
[(u'pub', 0.7254607677459717),
 (u'restaurant', 0.7147054076194763),
 (u'cafe', 0.7105239629745483),
 (u'table', 0.6781781911849976),
 (u'ranch', 0.6559066772460938),
 (u'club', 0.6470779180526733),
 (u'panera', 0.6464691162109375),
 (u'bakery', 0.6429882049560547),
 (u'grill', 0.6425997018814087),
 (u'gate', 0.6346235275268555)]

tweet_w2v.most_similar('facebook')
Out[54]:
[(u'fb', 0.8862842321395874),
 (u'myspace', 0.8414138555526733),
 (u'bebo', 0.7763116359710693),
 (u'yahoo', 0.7672140598297119),
 (u'msn', 0.7638905048370361),
 (u'twitter', 0.7276350259780884),
 (u'tumblr', 0.7209618091583252),
 (u'flickr', 0.712773323059082),
 (u'skype', 0.7116719484329224),
 (u'aim', 0.7065393924713135)]

tweet_w2v.most_similar('iphone')
Out[55]:
[(u'itouch', 0.7907721996307373),
 (u'blackberry', 0.7342787981033325),
 (u'firmware', 0.7048080563545227),
 (u'jailbreak', 0.7042940855026245),
 (u'mac', 0.7014051675796509),
 (u'3gs', 0.697465717792511),
 (u'pc', 0.6917887330055237),
 (u'upgrade', 0.6857078075408936),
 (u'mms', 0.6838993430137634),
 (u'3.0', 0.6824861764907837)]
```

How **awesome** is that?

For a given word, we get similar surrounding words of same context. Basically these words have a probability to be closer to that given word in most of the tweets.
It's interesting to see that our model gets facebook, twitter, skype together and bar, restaurant and cafe together as well. This could be useful for building a knowledge graph. Any thoughts about that?

How about visualizing these word vectors? We first have to reduce their dimension to 2 using t-SNE. Then, using an interactive visualization tool such as Bokeh, we can map them directly on 2D plane and interact with them.
Here's the script, and the bokeh chart below.

```
# importing bokeh library for interactive dataviz
import bokeh.plotting as bp
from bokeh.models import HoverTool, BoxSelectTool
from bokeh.plotting import figure, show, output_notebook

# defining the chart
output_notebook()
plot_tfidf = bp.figure(plot_width=700, plot_height=600, title="A map of 10000 word vectors",
    tools="pan,wheel_zoom,box_zoom,reset,hover,previewsave",
    x_axis_type=None, y_axis_type=None, min_border=1)

# getting a list of word vectors. limit to 10000. each is of 200 dimensions
word_vectors = [tweet_w2v[w] for w in tweet_w2v.wv.vocab.keys()[:5000]]

# dimensionality reduction. converting the vectors to 2d vectors
from sklearn.manifold import TSNE
tsne_model = TSNE(n_components=2, verbose=1, random_state=0)
tsne_w2v = tsne_model.fit_transform(word_vectors)

# putting everything in a dataframe
tsne_df = pd.DataFrame(tsne_w2v, columns=['x', 'y'])
tsne_df['words'] = tweet_w2v.wv.vocab.keys()[:5000]

# plotting. the corresponding word appears when you hover on the data point.
plot_tfidf.scatter(x='x', y='y', source=tsne_df)
hover = plot_tfidf.select(dict(type=HoverTool))
hover.tooltips={"word": "@words"}
show(plot_tfidf)
```



Zoom in, zoom out, place the cursor wherever you want and navigate in the graph. When clicking on a point, you can see the corresponding word. Convince yourself that grouped data points correspond to words of similar context.

4 - Building a sentiment classifier

Let's now get to the sentiment classification part. As for now, we have a word2vec model that converts each word from the corpus into a high dimensional vector. This seems to work fine according to the similarity tests and the bokeh chart.
In order to classify tweets, we have to turn them into vectors as well. How could we do this? Well, this task is almost done. Since we know the vector representation of each word composing a tweet, we have to "combine" these vectors together and get a new one that represents the tweet as a whole.
A first approach consists in averaging the word vectors together. But a slightly better solution I found was to compute a weighted average where each weight gives the importance of the word with respect to the corpus. Such a weight could the tf-idf score. To learn more about tf-

idf, you can look at my previous article (http://ahmedbesbes.com/how-to-mine-newsfeed-data-and-extract-interactive-insights-in-python.html).
Let's start by building a tf-idf matrix.

```
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 :', len(tfidf)
```

Now let's define a function that, given a list of tweet tokens, creates an averaged tweet vector.

```
def buildWordVector(tokens, size):
    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
```

Now we convert x_train and and x_test into list of vectors using this function. We also scale each column to have zero mean and unit standard deviation.

```
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_t
est))])
test_vecs_w2v = scale(test_vecs_w2v)
```

We should now be ready to feed these vectors into a neural network classifier. In fact, using Keras is very easy to define layers and activation functions.
Here is a basic 2-layer architecture.

```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=200))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(train_vecs_w2v, y_train, epochs=9, batch_size=32, verbose=2)
```

Now that the model is trained, let's evaluate it on the test set:

```
score = model.evaluate(test_vecs_w2v, y_test, batch_size=128, verbose=2)
print score[1]

Out[91]: 0.78984528240986307
```

Almost **80%** accuracy. This is not bad. We could eventually tune more parameters in the word2vec model and the neural network classifer to reach a higher precision score. Please tell me if you managed to do so.

5 - Conclusion

In this post we explored different tools to perform sentiment analysis: We built a tweet sentiment classifier using word2vec and Keras.
The combination of these two tools resulted in a 79% classification model accuracy.
This Keras model can be saved and used on other tweet data, like streaming data extracted through the tweepy (http://docs.tweepy.org/en/v3.5.0/) API. It could be interesting to wrap this model around a web app with some D3.js visualization dashboard too.

Regarding the improvement of this classifier, we can investigate the doc2vec model that extracts vectors out of sentences and paragraphs. I have first tried this model but I got a lower accuracy score of 69%. So please tell me if you can get better.

I hope this tutorial was a good introductory start to word embedding. Since I'm still learning my way through this awesome topic I'm open to suggestion or any recommendation.

NLP (http://ahmedbesbes.com/tag/nlp.html) word2vec (http://ahmedbesbes.com/tag/word2vec.html)
doc2vec (http://ahmedbesbes.com/tag/doc2vec.html) deep learning (http://ahmedbesbes.com/tag/deep-learning.html)
keras (http://ahmedbesbes.com/tag/keras.html) neural network (http://ahmedbesbes.com/tag/neural-network.html)
Twitter (http://ahmedbesbes.com/tag/twitter.html)

Related posts:

- Welcome ! (http://ahmedbesbes.com/welcome.html)

Now If I want to predict model with new tweets, I guess I should use predict method? But before that do we have to convert tweets to vectors(numpy array)?

4 ^ v • Reply • Share ›

Ahmed BESBES Mod → enqush • 22 days ago

Yes that's correct. You'll need to reassemble all the pieces of code and preprocess your testing data much like we did on the training data.

^ v • Reply • Share ›

Christopher Pepe • 2 months ago

The comments helped fill in the gaps from the post.

1. Sentiment is 0 and 4 in the test data (I couldn't best 50% accuracy until I changed it to 0 and 1 in ingest(): data['Sentiment'] = data['Sentiment'].map({4:1, 0:0}))
2. tweets.csv is training.1600000.processed.noemoticon.csv renamed with column headings added
3. tweetw2v.train needs more args (see comments - update line to: tweet_w2v.train([x.words for x in tqdm(x_train)],total_examples=tweet_w2v.corpus_count, epochs=tweet_w2v.iter))
4. Define n and n_dim somewhere: n=1000000; n_dim = 200

I think that was everything. That got me to 85% accuracy and it performs fairly well against new tweets in the limited testing that I've done. Pretty neat - thanks for sharing this Ahmed.

2 ^ v • Reply • Share ›

Ahmed BESBES Mod → Christopher Pepe • 22 days ago

Hello Christopher ,
I really appreciate your efforts of synthesizing all the comments, Thank you !
I'll reinclude enverything in the post soon.

^ v • Reply • Share ›

Gnanaguru Sattanathan • 5 months ago

Very useful post !

But how did you arrive at tweets.csv ? The dataset link you provided doesnt have this file !

2 ^ v • Reply • Share ›

Ahmed BESBES Mod → Gnanaguru Sattanathan • 22 days ago

I just renamed it

^ v • Reply • Share ›

Aleksey Bondarev • a month ago

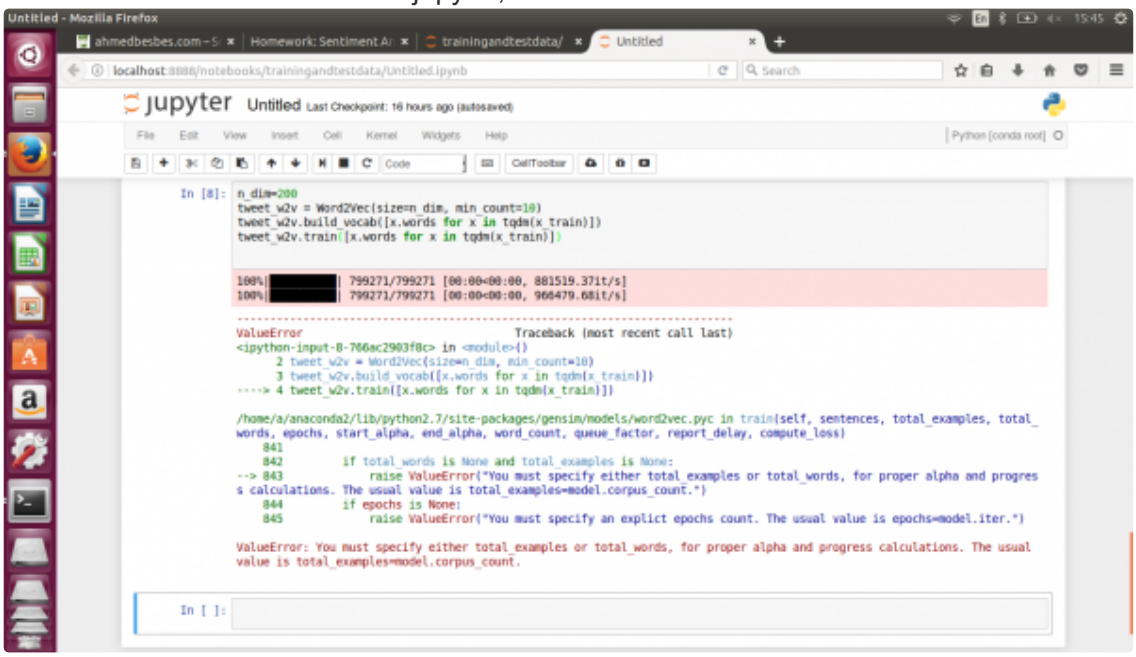
Hi Ahmed,

Thank you very much :)

1 ^ v • Reply • Share ›

mahmood naame • 3 months ago

hi ahmed
i have error for run this codes in jupyter,



Please guide me

1 • Reply • Share ›

Thami_bng ➚ mahmood naame • 2 months ago

Use this instead :
tweet_w2v.train([x.words for x in tqdm(x_train)],total_examples=tweet_w2v.corpus_count, epochs=tweet_w2v.iter)

• Reply • Share ›

adrienR • 5 months ago

Great post and nice results!

1 • Reply • Share ›

Asch Harwood • 6 months ago

Dear Admed,

Thanks for this. Just curious about the data. The filenames that you provided are
testdata.manual.2009.06.14.csv
training.1600000.processed.noemoticon.csv

Neither of them have any column names. The test data has a column that contains the numbers, 0, 2, or 4. The train set only seems to have 0 or 4. Is this supposed to be the sentiment score? I asked because you reference a binary 0/1 score.

Thanks!
Asch

1 • Reply • Share ›

Ahmed BESBES Mod ➚ Asch Harwood • 6 months ago

Hello Asch,

The csv file doesn't have column names, so the numbers 0,2,4 ... are the indexes set by default for each column.
I'm keeping columns of index 0 and 4 because they describe the label and the text of the tweet. (there are other columns in the original dataset but i'm not keeping them)

No it's not supposed to be a sentiment score. it's a binary label (0 or 1) that says whether the tweet is positive or negative. You can however use you classifier to turn your probability scores for the targets into sentiment scores.

Ahmed
1 • Reply • Share ›

adamwulf ➚ Ahmed BESBES • 2 months ago
I've re-uploaded the data with the column names. I also wanted to make sure I still had access to it in case the google drive link goes down eventually. It's available at https://github.com/adamwulf...
1 • Reply • Share ›

Hamada Zahera ➚ Ahmed BESBES • a month ago
Hello Ahmed,

Could you please refer more details about dataset ? how do you collect it or which source ?

Thanks in advance,
Hamada
• Reply • Share ›

なしエックスレイ ➚ Hamada Zahera • 7 days ago
Hi Hamada,
it's a common known corpus called sentiment140
there're several more out there.

http://keenformatics.blogsp...

You can also try to build your own, as described in this paper
https://web.stanford.edu/~j...
• Reply • Share ›

Lauren ➚ Ahmed BESBES • 5 months ago
Hi Ahmed,
Thanks for the explanation. However, in the training set, there are no binary score per tweet? Where can we obtain that information? Thanks again for a great post!
• Reply • Share ›

Andre Bloemenkamp ➚ Lauren • 3 months ago
Looks like the first column is ether a '"0"' negative sentiment and

'"4"' positive.

1 • Reply • Share ›

Ahmed BESBES Mod → Andre Bloemenkamp • 22 days ago

Yes , exactly.

• Reply • Share ›

William Briggs • 3 months ago

Hi. I don't understand the method buildWordVector. Don't we want each token in the tweet to be a 200 element array? Rather than the whole tweet itself being a 200 element array of the sum of each word vector value.

1 • Reply • Share ›

Ahmed BESBES Mod → William Briggs • 22 days ago

What we first did was coming up with a vector representation for each token in a high dimensional space (200).
Now we are interested in classifying a tweet as a whole. since a tweet is a list of tokens, one way of representing it could be two average (with a weighting scheme) the word vectors of each one of its tokens.

• Reply • Share ›

なしエックスレイ • 8 days ago

Hi Ahmed,

I've just noticed that you learn your word2vec model on x_train only. Why do wee need to differ in Training and Test for a word2vec model? Wouldn't it be also possible to create word2vec embeddings on the whole 1.6 million tweets? Just wondering because the word embeddings don't have any knowlege about our sentiment labels, so the eval process should be still fair.

• Reply • Share ›

Ahmed BESBES Mod → なしエックスレイ • 8 days ago

Actually you're right, I checked this tutorial (on doc2vec sentiment analysis)
https://github.com/linanqiu...
and it seems like you can train the word2vec on the whole dataset.
I'm not 100% sure there won't be an information leakage though.

1 • Reply • Share ›

なしエックスレイ → Ahmed BESBES • 7 days ago

Thanks for your answer. There is a discussion on reddit, which mentions a number of other points that need to be taken into account.
https://www.reddit.com/r/Ma...

• Reply • Share ›

James Martin • a month ago

Hi,
Great post, certainly the most accessible writeup of Keras and word2vec that I have seen. I am using this to cut my teeth not just on Keras but also Python so forgive me if I

am missing something obvious.

I have made the changes suggested by Christopher Pepe, The main issue I am facing is an error when trying to train the model and getting:

'object of type 'filter' has no len()'.

This to me implies that I am passing at least one nonsensical token. Tracing the data through I can list the contents of a token within the tokenize function ie

print(tokens) returns <filter object="" at="" 0x00000160d1f277f0="">

print(list(tokens)) ['I', 'looooooooovvvvvveee', 'my', 'Kindle', '2', '.', 'Not' etc……

However when I add

"data['token_list'] = list(data['tokens'])"

see more

• Reply • Share ›

なしエックスレイ ➚ James Martin • 7 days ago

Hi James,
that's the code I ran with python2 thanks to Ahmed's blog.

https://ghostbin.com/paste/...

• Reply • Share ›

なしエックスレイ • 2 months ago

Thanks for that great Post. It was really nice to read and has given an great overview about that topic. Keep up that good work! :)

• Reply • Share ›

Ahmed BESBES Mod ➚ なしエックスレイ • 22 days ago

Thanks!

• Reply • Share ›

Haydn • 6 months ago

Really great post Ahmed! I've been doing some similar stuff in R, but trying to learn Python so this is very useful. I think methodology like this has a lot of potential applications for market intelligence.

• Reply • Share ›

Ahmed BESBES Mod ➚ Haydn • 22 days ago

Happy to help!

• Reply • Share ›

Oscar Lastres • 6 months ago

Hello Ahmed,

In the 1.6 million tweets file, am I right to assume 0 and 4 represent the sentiment

column?

I managed to redefine how you tokenize the tweets and got a 79.71% accuracy.

• Reply • Share ›

ALSO ON **AHMED BESBES**

Welcome to my blog !

1 comment • a year ago

Avatar **Aziz Fourati** — I love also this great guy !!!!!Wonderful article boss!!

Welcome to my new blog !

5 comments • a year ago

Avatar **Tony Stark** — Hi Ahmed, I also want to learn machine learning, I am following many youtube videos and blogs for past 2 …

How to score 0.8134 in Titanic Kaggle Challenge

42 comments • a year ago

Avatar **Cecilia Lee** — Amazing! Thanks for your great notebook!

How to mine newsfeed data and extract interactive insights in Python

32 comments • 7 months ago

Avatar **Ahmed BESBES** — if you're using python 2, could you try this: def tokenizer(text): try: # modification I made: text = …