Writing Like Shakespeare

Classifying Word Count Vectors

This notebook is available at: https://github.com/pzirnhel/WritingLikeShakespeare.git

What problem gets solved

- Problem : get a machine to tell if if a French guy putting together 20 words in English, could be confused with Shakespeare.
- More generally, this project is an attempt to tell if what someone is writing comes from a particular author or not.
- Applications could include detecting plagiarism for example.

The data

- Two texts, 5 pages long, about 5 kB each.
- Text 1: from “The Tragedy of Coriolanus”, written by Shakespeare. From a compilation of all Shakespeare’s complete work by Liam Larsen. The first piece of the compilation. Available on kaggle.com
- Text 2: from a would be novel “The Multiple Lives of Octavio” written by me, a French guy, in English.

How to use the data

- “Bag of Words” approach: each text is considered a bag of words, that is word order is not taken into consideration
- The technique is attributed to Zellig Harris in 1954 according to Wikipedia.
- Each text’s author prefers to use some words rather than others when writing. Therefore, by drawing 20 random words from a text, identifying what words it contains and how often each word is repeated, we can hope to identify the author.
- So we need first to know what unique words are used in the two texts: the vocabulary, and how many they are: the vocabulary size.

The Tokenizer

- A tokenizer is the program that gets you the vocabulary and its size, when run on a text.
- I used the one from the keras.preprocessing.text package.

```
from keras.preprocessing.text import Tokenizer
```

- It is a program that parses the text, identifies unique strings of characters between two separators. It also counts how many times each such unique string, each “token” appears in the text. It assigns a number to each token, giving the number 0 to the token appearing in the text the most, followed by 1 for second most common etc, until all words are accounted for.

```
tokenizer = Tokenizer()

data = open('tmp/CoriolanusAndOctavioShortNoPunctuationNoApostrophe.txt').read()

corpus = data.lower().split("\n")


tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
print(total_words)
```

```
734
```

The vocabulary

- word_to_index (dictionary), gives the idx given a word
- idx_to_word is derived from it and gives the word given idx
- Words are ranked by frequency of occurrence, most first.

```
all_words=""
for idx in range(len(idx_to_word)):
    all_words=all_words+idx_to_word[idx]+" "
print(all_words)
```

```
the to you it of a and citizen we in first for is with what not was that i he us all on this our one have be but s are they so
would if as menenius she his or more did name units him these your belly speak know at an could do when image list representati
on t second them well say can must other tell like only then about each hear me let ll against why who will by answer long octa
vio her find want sutsbencuns ai input layer good poor were their give which even o now strong sir most up there off body look
hornada just today how patel geoff professor line examples any resolved rather than people done patricians think too very hath
mother cannot way need my where go had shall already care may make daily d see made from such chapter ocean perhaps left anothe
r last take going put two b get right use network given much its training goes before proceed further famish caius marcius no t
alking away citizens wholesome revenge gods has country content proud unto though soft please help here come honest rest work h
and matter arms friends yourselves dearth state ne er yet rich restrain love accused tale heard little time members thus common
kind smile came small found basket baby discovered police lead easy morning instructor breeze straight got run down maybe does
introduction familiar audience algorithm without dots between black white numbers call vector value neural feature turns lines
trying features batch suppose saying sees portion die chief enemy kill corn own price verdict word accounted authority surfeits
relieve yield superfluity while might guess relieved humanely dear leanness afflicts object misery inventory particularise abun
dance sufferance gain pikes ere become rakes hunger bread thirst especially dog commonalty consider services report fort pays h
imself being nay maliciously famously end conscienced men partly till altitude virtue nature account vice covetous barren accus
ations faults surplus tire repetition shouts side city risen stay prating capitol comes worthy agrippa always loved enough coun
```

Framing the problem

- We can think of a vector of 734 features as a histogram with 734 columns
- Each sample is a particular histogram: a vector of word counts of 734 dimensions, when drawing 20 words from one of the two texts. It will tend to resemble the histogram of one text or the other.
- Finding to which text the vector belongs is a classification problem solved by using a classifier.

Framing the problem: complications

- It is not a priori obvious that the two texts would have distributions sufficiently different, let alone be linearly separable: simple logistic regression or support vector machines may or may not work.
- So, I decided to train a keras neural network, because it is easy to add layers if the data is not linearly separable and multilayered networks can deal with non-linearly separable data.

```
import tensorflow as tf
import keras
```

- The simplest neural network implements logistic regression

The way to training and test data

Getting from two texts to training and test data required 8 steps:

- 1. Finding the last line of the first text inside the two text file

```
for current_line_nb in range(190,201): #201 is the first line of the Octavio file
    current_line=lines[current_line_nb].lower().split(" ")
    print(len(current_line))
    for idx in range(len(current_line)):
        if current_line[idx] is not '':
            print(word_index[current_line[idx]])
            print(current_line[idx])
```

- 2. Tokenizing (e. g. converting words to their numeric code) all words in each text as to obtain two lists: “coriolanus tokenized” and “octavio_tokenized”

```
coriolanus_tokenized=[]
for current_line_nb in range(201): #201 is the first line of the Octavio file
    current_line=lines[current_line_nb].lower().split(" ")
    #print(len(current_line))
    for idx in range(len(current_line)):
        if current_line[idx] is not '':
            coriolanus_tokenized.append(word_index[current_line[idx]])
```

```
print(coriolanus_tokenized)
```

```
[11, 8, 168, 9, 169, 118, 170, 73, 74, 49, 22, 49, 49, 11, 8, 3, 31, 22, 119, 1
1, 3, 50, 172, 173, 13, 261, 262, 2, 1, 122, 22, 9, 50, 59, 9, 50, 59, 11, 8, 7
266, 13, 59, 6, 267, 22, 174, 41, 175, 23, 59, 75, 4, 28, 123, 176, 176, 60, 8,
1, 124, 92, 15, 270, 271, 23, 34, 272, 21, 35, 32, 34, 273, 21, 29, 1, 274, 275
```

```
octavio_tokenized=[]
for current_line_nb in range(201,283): #201 is the first line of the
    current_line=lines[current_line_nb].lower().split(" ")
    #print(len(current_line))
    for idx in range(len(current_line)):
        if current_line[idx] is not '':
            octavio_tokenized.append(word_index[current_line[idx]])
```

```
print(octavio_tokenized)
```

```
[1, 492, 493, 5, 84, 109, 148, 26, 84, 30, 494, 84, 17, 495, 10, 496,
2, 502, 45, 20, 17, 220, 10, 6, 221, 23, 6, 503, 51, 504, 505, 506, 1
13, 1, 514, 149, 150, 39, 129, 515, 85, 222, 34, 28, 223, 81, 6, 516,
2, 38, 136, 110, 151, 1, 221, 14, 6, 523, 23, 4, 84, 109, 15, 6, 524,
```

The way to training and test data

Getting from two texts to training and test data required 8 steps:

- 3. Shuffling the lists, extracting 1000 token from each list.

```
coriolanus_shuffled=np.array(coriolanus_tokenized)
np.random.shuffle(coriolanus_shuffled)
octavio_shuffled=np.array(octavio_tokenized)
np.random.shuffle(octavio_shuffled)
```

- 4. Getting 50 samples of 20 words for each set of 1000 tokens

```
input_dim=20
n_samples=2000//input_dim
X_coriolanus_idx=np.reshape(coriolanus_shuffled[:1000],[n_samples//2,input_dim])
X_octavio_idx=np.reshape(octavio_shuffled[:1000],[n_samples//2,input_dim])
print(n_samples)
```

The way to training and test data

Getting from two texts to training and test data required 8 steps:

- 5. Converting the samples into vectors of word counts.

```
freq_words=7
last_words=total_words-freq_words
X_coriolanus=np.zeros([n_samples//2,last_words])
X_octavio=np.zeros([n_samples//2,last_words])
for sample in range(n_samples//2):
    for j in range(input_dim):
        word_idx = X_coriolanus_idx[sample,j]-freq_words
        if word_idx>=0:
            X_coriolanus[sample,word_idx]+=1.0
        word_idx = X_octavio_idx[sample,j]-freq_words
        if word_idx>=0:
            X_octavio[sample,word_idx]+=1.0
```

- 6. Making labels: 50 ones and 50 zeros

```
y_base=np.reshape(np.concatenate([np.ones([1,n_samples//2]),np.zeros([1,n_samples//2])],axis=1),[n_samples])
```

The way to training and test data

Getting from two texts to training and test data required 8 steps:

- 7. Joining the two sets of 50 samples, shuffling them the same way the labels get shuffled

```
ordering=np.arange(n_samples)
np.random.shuffle(ordering)
print(ordering.dtype)
```

```
X=np.concatenate([X_coriolanus,X_octavio],axis=0)[ordering,:]
y=y_base[ordering]
print(X.shape,y_base.shape)
```

- 8. Dividing the 100 samples into 90 for training 10 for testing.

```
X_train,X_test=X[:(9*n_samples)//10,:],X[(9*n_samples)//10:n_samples,:]
y_train,y_test=y[:(9*n_samples)//10],y[(9*n_samples)//10:n_samples]
```

The keras network detour

- I first implemented linear regression using a network, because it was easy to add layers if the data was not linearly separable.
- I got bad results, because the net had a too strong bias: I could not get the accuracy up. It could have been because the data was not linearly separable, but I believed it was because frequent words counts created confusion.
- So I got rid of the 7 first, most frequent words in the samples

```
freq_words=7
last_words=total_words-freq_words
X_coriolanus=np.zeros([n_samples//2,last_words])
X_octavio=np.zeros([n_samples//2,last_words])
for sample in range(n_samples//2):
    for j in range(input_dim):
        word_idx = X_coriolanus_idx[sample,j]-freq_words
        if word_idx>=0:
            X_coriolanus[sample,word_idx]+=1.0
        word_idx = X_octavio_idx[sample,j]-freq_words
        if word_idx>=0:
            X_octavio[sample,word_idx]+=1.0
```

Going back to sklearn

- I trained the net, got 100% on the testing set.

```
history=model.fit(X_train, y_train, batch_size=10, epochs=50,validation_split=1.0/9.0,shuffle=True)
model.evaluate(X_test, y_test)
```

```
Epoch 49/50
80/80 [==============================] - 0s 374us/step - loss: 0.1878 - acc: 1.0000 - val_loss: 0.3890 - val_acc: 0.9000
Epoch 50/50
80/80 [==============================] - 0s 361us/step - loss: 0.1840 - acc: 1.0000 - val_loss: 0.3870 - val_acc: 0.9000
10/10 [==============================] - 0s 199us/step

[0.4022069573402405, 1.0]
```

- So I trained an sklearn logistic regression model: got the same result

```
LogReg = LogisticRegression(class_weight='balanced',solver='liblinear').fit(X_train,y_train.ravel())
```

```
y_pred = LogReg.predict(X_test)
print("Accuracy:")
print(np.mean(np.float32(np.equal(y_pred,y_test))))
```

```
Accuracy:
1.0
```

Conclusion

- Using a “Bag of Words” method with logistic regression makes it possible to see if a French guy putting together 20 words could be confused with Shakespeare or not.
- This is a low bar objective though
- What would have happened for example if 20 consecutive words had been extracted from another of Shakespeare’s play? It is at least possible that the program would have concluded that Shakespeare does not write like Shakespeare.
- Or worse: that he wrote like me. I leave this for you to check.