13_toefl_clean

March 21, 2017

2/20/17 - smiel

1 Cleaning the TOEFL essay data.

```
In [2]: %matplotlib inline
        # run me when first starting this notebook
        import os

        import numpy as np
        import pandas as pd

        path = '/research/ella/rivendell/toefl'
```

The TOEFL data comes to us in a bunch of text files, with an index csv. We'll use these to create an essays csv.

```
In [6]: essays = pd.read_csv(os.path.join(path, 'data', 'text', 'index.csv'), encoding='ASCII')
        print(essays.columns)
```

```
Index([u'Filename', u'Prompt', u'Language', u'Score Level'], dtype='object')
```

First we should translate the langauage abbreviations to the ones we use.

```
In [7]: print(essays.groupby('Language').size())
```

```
Language
ARA    1100
DEU    1100
FRA    1100
HIN    1100
ITA    1100
JPN    1100
KOR    1100
SPA    1100
TEL    1100
TUR    1100
ZHO    1100
dtype: int64
```

Mostly correct, except for: - DEU -> GER - HIN -> IND - ZHO -> CHN

```
In [8]: lang_map = {'DEU': 'GER', 'HIN': 'IND', 'ZHO': 'CHN'}
        essays['L1'] = essays.Language.apply(lambda lang: lang_map.get(lang, lang))
```

Now we can work on mapping the scores to numbers. "Score Level" is - low - medium - high which is their transmogrification of the TOEFL/iBT scores. From the research paper about this data set:

"When collapsing the combined scores into the 3-point scale, low is for essays scoring between 1.0 and 2.0, medium is for 2.5 to 3.5, and high is for 4.0 to 5.0." We will take the middle of those ranges as the numeric score for each essay.

```
In [10]: score_map = {'low': 1.5, 'medium': 3.0, 'high': 4.5}
         essays['score'] = essays['Score Level'].apply(lambda sl: score_map[sl])
         essays['score_type'] = 'TOEFL/iBT'

         # let's use the filename (without the extension) as the essay ID
         essays['essay_id'] = essays.Filename.str.slice(stop=-4)

         # save our progress
         essays.to_csv(os.path.join(path, 'all_essays.csv'), encoding='ASCII', index=False)
```

Ok. Time to go get the essay texts.

```
In [12]: texts = []
         for filename in essays.Filename.values:
             with open(os.path.join(path, 'data', 'text', 'responses', 'original', filename), 'r
                 texts.append(fin.read().strip())

         essays['text'] = texts

         # save our progress
         essays.to_csv(os.path.join(path, 'all_essays.csv'), encoding='ASCII', index=False)
```

1.1 From Essays to Sentences

Now let's start building the sentences data frame. For unidecode to work properly, the following should print "True":

```
In [13]: import sys
         print(sys.maxunicode > 0xffff)
```

```
True
```

```
In [3]: from utilitybelt.text import get_sentences
        import copy
        from unidecode import unidecode
        import numpy as np
```

```
# load data
df_in = pd.read_csv(os.path.join(path, 'all_essays.csv'), encoding='ASCII')

# while we're at it, let's add a little more metadata
# the TOEFL is taken as a college entrance test, so let's assume the students were all 1
df_in['age'] = 17

# convert text to ascii
print('Converting to ASCII')
df_in['ascii_text'] = df_in.text.apply(lambda t: unidecode(t))

# normalize line endings
df_in.ascii_text = df_in.ascii_text.str.replace('\r\n', '\n')
df_in.ascii_text = df_in.ascii_text.str.replace('\r', '\n')

# use space instead of tab
df_in.ascii_text = df_in.ascii_text.str.replace('\t', ' ')

# now remove any non-printable ascii char
df_in.ascii_text = df_in.ascii_text.str.replace(r'[^ -~\n]', '')

# # make sure all is printable
# for i, t in enumerate(df_in.ascii_text.values):
#     for ci, c in enumerate(t):
#         if (32 <= ord(c) <= 126) or c in '\n\t':
#             continue
#         else:
#             print u"Unprintable character {} in {} at char {}:\n\n{}\n================
#                 ord(c), i, ci, t, df_in.iloc[i].clean_text
#             )
#             raise ValueError

# shush the utilitybelt sentence splitter logging
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

print('Splitting sentences')
# create records for every sentence
records = []
for i, row in df_in.iterrows():
    rec = {
        'dataset': 'TOEFL', 'prompt_id': row.Prompt, 'essay_id': row.essay_id, 'L1': row
        'score': row.score, 'score_type': 'TOEFL', 'age': row.age
    }
    prev_end = 0
    text = row.ascii_text
```

```
            si = 0
            for start, end, sentence in zip(*get_sentences(text)):
                srec = {}
                srec.update(rec)
                srec['text'] = sentence
                srec['sentence_id'] = si
                srec['trailing_whitespace'] = text[prev_end:start]
                si += 1
                prev_end = end
                records.append(srec)

            if i % 1000 == 0:
                print('{} of {}'.format(i, len(df_in)))

        print('Creating data frame')
        df_out = pd.DataFrame.from_records(records)
        df_out['uid'] = df_out[['dataset', 'essay_id', 'sentence_id']].astype(unicode).apply(lam

        print('{} sentences'.format(len(df_out)))
        print('Saving data frame')
        df_out.to_csv(os.path.join(path, 'TOEFL-11_sentences.csv'), encoding='utf8', index=False
```

```
Converting to ASCII
Splitting sentences
0 of 12100
1000 of 12100
2000 of 12100
3000 of 12100
4000 of 12100
5000 of 12100
6000 of 12100
7000 of 12100
8000 of 12100
9000 of 12100
10000 of 12100
11000 of 12100
12000 of 12100
Creating data frame
193721 sentences
Saving data frame
```

Let's do a little descriptive analysis to make sure we got what we want.

```
In [20]: df = pd.read_csv(os.path.join(path, 'TOEFL-11_sentences.csv'), encoding='utf8')
```

```
In [21]: age = df.groupby('age').size()
         print(age)
         print('{} sentences with age data'.format(pd.notnull(df.age).sum()))
```

```
age
17    193721
dtype: int64
193721 sentences with age data


In [22]: score = df.groupby('score').size()
         print(score)

score
1.5     15067
3.0    103711
4.5     74943
dtype: int64


In [23]: df.text.apply(len).describe()

Out[23]: count    193721.000000
         mean        109.563171
         std          68.349997
         min           1.000000
         25%          66.000000
         50%          96.000000
         75%         136.000000
         max        2283.000000
         Name: text, dtype: float64
```