main_python_final

April 17, 2019

Loading packages

In [4]:
```
import glob
import errno
import string
import numpy as np
import csv
import pandas as pd
import time
import os
import re
import itertools
from collections import Counter as mset
```

0.0.1 Data Cleaning

In [2]:
```
#data cleanning
file_list = os.listdir("../data/ground_truth")
#create and save error terms in error_list
error_df = pd.DataFrame()
tess_error_list = list()
ground_error_list = list()

for file in file_list:
    #read ground_truth lines
    ground_truth_file_path = "../data/ground_truth_trimmed/"+str(file)
    ground_truth = open(ground_truth_file_path, "r",encoding="utf8")
    ground_truth_lines = ground_truth.read().split('\n')

    #read tesseract lines
    tesseract_file_path = "../data/tesseract/"+str(file)
    tesseract = open(tesseract_file_path, "r",encoding="utf8")
    tesseract_lines = tesseract.read().split('\n')

    #print(tesseract_lines)
    #print(ground_truth_lines)

    #define function: get words split by space
```

```
        def get_words(lines):
            lines_list = list()
            for element in lines:
                #element  = element.split(" ")
                element = [ word for word in element.split(" ") if not word.isdigit() and w
                lines_list.append(element)
            return lines_list

        #create documents for tesseract and ground truth
        tesseract_doc = get_words(tesseract_lines)
        ground_truth_doc = get_words(ground_truth_lines)

        tess_len = len(tesseract_doc)
        ground_len = len(ground_truth_doc)

        ################################################
        #######      Check for Mismatch          ######
        ################################################

        ##if tess_len != ground_len:
        ##    print(file)

        #####After Trimming 13 mismatched files, no mismatch in number of lines detected



        #create and save error terms in error_list
        #error_df = pd.DataFrame()
        #tess_error_list = list()
        #ground_error_list = list()
        for i in range(len(tesseract_doc)):
            if len(tesseract_doc[i]) == len(ground_truth_doc[i]):
                tess = tesseract_doc[i]
                ground = ground_truth_doc[i]
                for j in range(len(tess)):
                    #if tess[j] != ground[j]:
                        tess_error_list.append(tess[j])
                        ground_error_list.append(ground[j])

    error_df['Tesseract'] = tess_error_list
    error_df['Ground_Truth'] = ground_error_list
```

0.0.2 Error Detection

```
In [5]: def error_detect(word):
            #rule_1
            rule_1 = len(list(word))>20
```

```
        #rule_2
        num_punct = len([char for char in word if char in string.punctuation])
        num_alpha = len([char for char in word if char.isalpha()])
        rule_2 = num_punct>num_alpha

        #rule_3
        rule_3 = len(set([char for char in word[1:-1] if char in string.punctuation]))>=2

        #rule_4
        rule_4 = max([len(list(v)) for k,v in itertools.groupby(word)])>=3

        #rule_5
        num_upper = len([char for char in word if char.isupper()])
        num_lower = len([char for char in word if char.islower()])
        rule_5 = num_upper > num_lower and num_upper < len(word)

        #rule_6
        if word.isalpha():
            vowels = 'aeiouAEIOU'
            num_vowels = len([char for char in word if char in vowels])
            num_consonants = len(word) - num_vowels
            rule_6 = num_vowels > 8*num_consonants or num_consonants > 8*num_vowels
        else : rule_6 = False

        #rule_7
        try:
            vowels_len = max([len(w) for w in re.findall(r'[aeiou]+',word,re.IGNORECASE)])
        except ValueError:
            vowels_len = 0
        try:
            conson_len = max([len(w) for w in re.findall(r'[^aeiou]+',word,re.IGNORECASE)])
        except ValueError:
            conson_len = 0
        rule_7 = vowels_len>=4 or conson_len >=5

        #rule_8
        rule_8 = word[0].islower() and word[-1].islower() and word[1:-1].isupper()

        return rule_1 or rule_2 or rule_3 or rule_4 or rule_5 or rule_6 or rule_7 or rule_8

    #######Detect Error #######
    error_df.insert(loc =2, column="IS_error", value= error_df.Tesseract.map(error_detect))
```

In [6]:
```
def remove_punc_digit(string):
    string = re.sub(r'[^\w\s]','',string)
    string = re.sub(r'[^\D]','',string)
    return string
```

```
        error_df['Tesseract'] = error_df['Tesseract'].apply(lambda x: remove_punc_digit(x))
        error_df['Ground_Truth'] = error_df['Ground_Truth'].apply(lambda x: remove_punc_digit(x
```

In [7]:
```
# Error ground truth pairs
error_df = error_df[error_df.IS_error==True]
error_df = error_df.drop(columns="IS_error")
error_df.Tesseract.replace('', np.nan, inplace=True)
error_df.Ground_Truth.replace('', np.nan, inplace=True)
error_df = error_df.dropna()
error_df = error_df.reset_index(drop = True)
error_df.to_csv("../output/Error_df_rules_based.csv")
```

Creating corpus

In [8]:
```
truth = []
path = '../data/ground_truth/*.txt'
files = glob.glob(path)
for name in files:
    try:
        with open(name, encoding='utf8') as f:
            for line in f:
                out = line.translate(str.maketrans('', '', string.punctuation))
                out = ''.join([i for i in out if not i.isdigit()])
                out = out.lower().split()
                truth.extend(out)
    except IOError as exc:
        if exc.errno != errno.EISDIR:
            raise
np.savetxt('../output/truth_corpus.dat', truth, fmt='%s', encoding='utf8')

tess = []
path = '../data/tesseract/*.txt'
files = glob.glob(path)
for name in files:
    try:
        with open(name, encoding='utf8') as f:
            for line in f:
                out = line.translate(str.maketrans('', '', string.punctuation))
                out = ''.join([i for i in out if not i.isdigit()])
                out = out.lower().split()
                tess.extend(out)
    except IOError as exc:
        if exc.errno != errno.EISDIR:
            raise
np.savetxt('../output/tess_corpus.dat', tess, fmt='%s', encoding='utf8')
```

Reading data for feature extraction

```
In [9]: Error = []
        Truth = []
        pair = []
        with open('../output/Error_df_rules_based.csv', encoding='utf8') as f:
            csv_reader = csv.reader(f, delimiter=',')
            for row in csv_reader:
                err = row[1].lower()
                trt = row[2].lower()
                if err != trt and [err, trt] not in pair:
                    Error.append(err)
                    Truth.append(trt)
                    pair.append([err, trt])

        Error = Error[2:]
        Truth = Truth[2:]
```

Feature extraction

```
In [10]: import sys
         sys.path.append("../lib")
         from feature_scoring import n_gram
         from feature_scoring import candidate_search
         from feature_scoring import LED_score
         from feature_scoring import SS_score
         from feature_scoring import LP_score
         from feature_scoring import ECP_score
         from feature_scoring import RCP_score
```

```
In [ ]: W_error=['Typo']
        W_truth=['Truth']
        W_cand = ['Candidate']
        Label = ['Label']
        LED = ['led_score']
        SS = ['ss_score']
        LP = ['lp_score']
        ECP = ['ECP_score']

        n = 3 # n_gram
        for i in range(len(Error)):
            w_e = Error[i]
            w_c = Truth[i]
            cand_list = candidate_search(truth, w_e)
            print('word ',i+1,', error: ', w_e, ', truth: ', w_c)

        #    gram_list = n_gram(w_e, tess, n)
            LP_freq = []
        #    ECP_freq = []
            for s in cand_list:
```

```
            lp_freq = LP_score(s, truth)
            LP_freq.append(lp_freq)
        #         ecp_freq = ECP_score(gram_list, s, truth, n)
        #        ECP_freq.append(ecp_freq)

        for j in range(len(cand_list)):
            s = cand_list[j]
            led = LED_score(w_e, s)
            ss = SS_score(w_e, s, N=3)
            lp = LP_score(s, truth)/max(LP_freq)
        #         if max(ECP_freq)==0: ecp=0
        #         else: ecp = ECP_score(gram_list, s, truth, n)/max(ECP_freq)
        #         rcp = RCP_score(w_e, s, tess, truth)
            label = int(s == w_c)
        #         print('candidate:', s, '\tscores =', '{:03.2f}'.format(led),', {:03.2f}'.form
            W_error.append(w_e)
            W_truth.append(w_c)
            W_cand.append(s)
            Label.append(label)
            LED.append(led)
            SS.append(ss)
            LP.append(lp)
            #ECP.append(ecp)
```

```
In [12]: np.savetxt('../output/feature.csv', [p for p in zip(W_error, W_truth, W_cand, LED, SS
```

0.1 Parameter Tunning

Retreive Data

```
In [12]: from sklearn.model_selection import GridSearchCV
         from sklearn.ensemble import AdaBoostRegressor
```

```
In [13]: feature_output = pd.read_csv('../output/feature.csv', delimiter = ',')
```

```
In [14]: feature_output.head(20)
```

Out[14]:

	Typo	Truth	Candidate	led_score	ss_score	lp_score	Label
0	willlam	william	will	0.25	1.636364	1.000000	0
1	willlam	william	willful	0.25	1.607143	0.001712	0
2	willlam	william	william	0.75	2.142857	0.039954	1
3	willlam	william	williams	0.50	1.866667	0.003995	0
4	willlam	william	willing	0.25	1.285714	0.006849	0
5	willlam	william	wills	0.25	1.500000	0.000571	0
6	nvolvng	involving	cooling	0.25	0.857143	0.011494	0
7	nvolvng	involving	evolve	0.25	0.961538	0.045977	0
8	nvolvng	involving	evolved	0.25	0.892857	0.034483	0
9	nvolvng	involving	evolves	0.25	0.892857	0.011494	0
10	nvolvng	involving	evolving	0.50	1.633333	0.057471	0

11	nvolvng	involving	involvad	0.25	0.833333	0.011494	0
12	nvolvng	involving	involve	0.25	0.892857	0.126437	0
13	nvolvng	involving	involved	0.25	0.833333	1.000000	0
14	nvolvng	involving	involves	0.25	0.833333	0.160920	0
15	nvolvng	involving	involveŋ	0.25	0.833333	0.011494	0
16	nvolvng	involving	involving	0.50	1.656250	0.505747	1
17	nvolvng	involving	lnvolva	0.25	0.892857	0.011494	0
18	nvolvng	involving	noting	0.25	0.807692	0.057471	0
19	nvolvng	involving	solving	0.25	1.035714	0.022989	0

In [15]:
```
X = feature_output[feature_output.columns[0:6]]
y = feature_output["Label"]
```

Train & Test split

In [16]:
```
from sklearn.model_selection import GroupShuffleSplit
group = pd.Categorical(feature_output["Typo"])
train_inds, test_inds = next(GroupShuffleSplit(random_state=42).split(X, y, group))
X_train, X_test, y_train, y_test = X.iloc[list(train_inds)], X.iloc[list(test_inds)],
train_words = X_train[X_train.columns[0:3]]
test_words = X_test[X_test.columns[0:3]]
X_train = X_train[X_train.columns[3:6]]
X_test = X_test[X_test.columns[3:6]]
```

In [17]:
```
print('X_train shape:',X_train.shape,'\n','y_train shape:',y_train.shape)
```

```
X_train shape: (1238133, 3)
 y_train shape: (1238133,)
```

In [18]:
```
print('X_test shape:',X_test.shape,'\n','y_test shape:',y_test.shape)
```

```
X_test shape: (165302, 3)
 y_test shape: (165302,)
```

0.2 Ada Boost

In [19]:
```
model = AdaBoostRegressor()
```

0.3 Parameters Tunning

In [20]:
```
parameters = {
 'n_estimators': [50, 100],
 'learning_rate' : [0.01,0.05,0.1,0.3,1],
 'loss' : ['linear', 'square', 'exponential']
 }
```

```
In [21]: start = time.time()
         ada_grid_search = GridSearchCV(model,parameters,cv = 3,n_jobs =-1)
         ada_grid_search_fit = ada_grid_search.fit(X_train, y_train)
         end = time.time()
         print('Time:',end - start)

Time: 2520.1292009353638


In [22]: ada_grid_search_fit.best_params_

Out[22]: {'learning_rate': 0.01, 'loss': 'exponential', 'n_estimators': 50}

In [57]: ada_grid_search_fit.best_estimator_

Out[57]: AdaBoostRegressor(base_estimator=None, learning_rate=0.01, loss='exponential',
                   n_estimators=50, random_state=None)

In [42]: cvres = ada_grid_search.cv_results_

In [70]: for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
             print(mean_score, params)

0.40374915824108776 {'learning_rate': 0.01, 'loss': 'linear', 'n_estimators': 50}
0.3680232694914383 {'learning_rate': 0.01, 'loss': 'linear', 'n_estimators': 100}
0.4023762621878514 {'learning_rate': 0.01, 'loss': 'square', 'n_estimators': 50}
0.34098242730811906 {'learning_rate': 0.01, 'loss': 'square', 'n_estimators': 100}
0.4083847633391772 {'learning_rate': 0.01, 'loss': 'exponential', 'n_estimators': 50}
0.39643609293192866 {'learning_rate': 0.01, 'loss': 'exponential', 'n_estimators': 100}
0.15414374962757887 {'learning_rate': 0.05, 'loss': 'linear', 'n_estimators': 50}
-0.23349487386235154 {'learning_rate': 0.05, 'loss': 'linear', 'n_estimators': 100}
-0.10284196383632471 {'learning_rate': 0.05, 'loss': 'square', 'n_estimators': 50}
-0.9978515475452031 {'learning_rate': 0.05, 'loss': 'square', 'n_estimators': 100}
0.27868712935106005 {'learning_rate': 0.05, 'loss': 'exponential', 'n_estimators': 50}
-0.05210748438201115 {'learning_rate': 0.05, 'loss': 'exponential', 'n_estimators': 100}
-0.18220429066384375 {'learning_rate': 0.1, 'loss': 'linear', 'n_estimators': 50}
-0.3433878616741871 {'learning_rate': 0.1, 'loss': 'linear', 'n_estimators': 100}
-0.9308229281848934 {'learning_rate': 0.1, 'loss': 'square', 'n_estimators': 50}
-1.0131045582770672 {'learning_rate': 0.1, 'loss': 'square', 'n_estimators': 100}
-0.06313284282744469 {'learning_rate': 0.1, 'loss': 'exponential', 'n_estimators': 50}
-0.7592209661142575 {'learning_rate': 0.1, 'loss': 'exponential', 'n_estimators': 100}
-0.31279929981489774 {'learning_rate': 0.3, 'loss': 'linear', 'n_estimators': 50}
-0.37670143401303086 {'learning_rate': 0.3, 'loss': 'linear', 'n_estimators': 100}
-1.0867650054545688 {'learning_rate': 0.3, 'loss': 'square', 'n_estimators': 50}
-0.976816075191342 {'learning_rate': 0.3, 'loss': 'square', 'n_estimators': 100}
-1.0901767341119117 {'learning_rate': 0.3, 'loss': 'exponential', 'n_estimators': 50}
-1.5741717968173896 {'learning_rate': 0.3, 'loss': 'exponential', 'n_estimators': 100}
0.37608463662658714 {'learning_rate': 1, 'loss': 'linear', 'n_estimators': 50}
0.38336523445017 {'learning_rate': 1, 'loss': 'linear', 'n_estimators': 100}
```

```
0.38065389767817553 {'learning_rate': 1, 'loss': 'square', 'n_estimators': 50}
0.38312320725422755 {'learning_rate': 1, 'loss': 'square', 'n_estimators': 100}
-0.574025360829864 {'learning_rate': 1, 'loss': 'exponential', 'n_estimators': 50}
-0.4591920761561897 {'learning_rate': 1, 'loss': 'exponential', 'n_estimators': 100}
```

0.4 Prediction

```
In [24]: start = time.time()
         regessor = AdaBoostRegressor(base_estimator=None, learning_rate=0.01, loss='exponentia
                  n_estimators=50, random_state=None)
         regessor_fit = regessor.fit(X_train, y_train)
         end = time.time()
         print('Time:',end - start)
```

```
Time: 45.84798288345337
```

```
In [25]: result = regessor.predict(X_test)
```

```
In [26]: predicted_confidence = pd.DataFrame({"predicted_confidence": result})
         test_typo =  pd.DataFrame({"typo": np.array(test_words['Typo'])})
         test_truth = pd.DataFrame({"truth": np.array(test_words['Truth'])})
         test_candidate = pd.DataFrame({"candidate": np.array(test_words['Candidate'])})
         label = pd.DataFrame({"label": np.array(y_test)})
```

```
In [ ]: unsorted_test_final_output = pd.concat([test_typo, test_truth, test_candidate, predicte
```

0.5 Model Evaluation

```
In [28]: #define files path
         tess_dir = "../data/tesseract/"
         ground_dir = "../data/ground_truth_trimmed/"
         file_name = os.listdir(tess_dir)
```

```
In [29]: #Avreage Number of Candidates
         feature_output.shape[0]/feature_output[feature_output.Label==1].shape[0]
```

```
Out[29]: 373.4526343799894
```

```
In [30]: #Top_n candidates
         candidate_10 = unsorted_test_final_output.groupby("typo").apply(lambda x: x.nlargest(
         candidate_5 = unsorted_test_final_output.groupby("typo").apply(lambda x: x.nlargest(5
         candidate_3 = unsorted_test_final_output.groupby("typo").apply(lambda x: x.nlargest(3
         candidate_1 = unsorted_test_final_output.groupby("typo").apply(lambda x: x.nlargest(1
```

```
In [31]: candidate_10.loc['acrsv']
```

Out[31]:

	typo	truth	candidate	predicted_confidence	label
141602	acrsv	acrs	acrs	0.395005	1
141598	acrsv	acrs	acr	0.077035	0
141601	acrsv	acrs	across	0.077035	0
141599	acrsv	acrs	acra	0.030645	0
141600	acrsv	acrs	acre	0.030645	0
141603	acrsv	acrs	acs	0.023050	0
141604	acrsv	acrs	acsh	0.000898	0
141608	acrsv	acrs	activ	0.000898	0
141611	acrsv	acrs	acts	0.000898	0
141679	acrsv	acrs	cars	0.000898	0

In [32]:
```
#Top_n candidates wordwise precision
total_typo = sum(y_test==1)
top_10= "{0:.2%}".format(candidate_10[candidate_10.label==1].shape[0]/total_typo)
top_5= "{0:.2%}".format(candidate_5[candidate_5.label==1].shape[0]/total_typo)
top_3= "{0:.2%}".format(candidate_3[candidate_3.label==1].shape[0]/total_typo)
top_1= "{0:.2%}".format(candidate_1[candidate_1.label==1].shape[0]/total_typo)

top = pd.DataFrame({"top": np.array([1,3,5,10])})
precision = pd.DataFrame({"precision": np.array([top_1, top_3, top_5, top_10])})
pd.concat([top, precision], axis=1)
```

Out[32]:

	top	precision
0	1	71.55%
1	3	86.74%
2	5	88.95%
3	10	91.85%

0.6 Measurement

In [33]:
```
pred = regessor.predict(X[X.columns[3:6]])
```

In [34]:
```
predicted_confidence = pd.DataFrame({"predicted_confidence": pred})
test_typo =  pd.DataFrame({"typo": X['Typo']})
test_truth = pd.DataFrame({"truth": X['Truth']})
test_candidate = pd.DataFrame({"candidate": X['Candidate']})
label = pd.DataFrame({"label": y})
unsorted_test_final_output = pd.concat([test_typo, test_truth, test_candidate, predict
```

In [35]:
```
candidate_1 = unsorted_test_final_output.groupby("typo").apply(lambda x: x.nlargest(1
cand = candidate_1.candidate.values
typo = candidate_1.typo.values
cand_dict = dict(zip(typo, cand))
```

In [36]:
```
new_tess = []
for s in tess:
    if s in cand_dict:
        new_tess.append(cand_dict[s])
```

```
            else:
                new_tess.append(s)
```

In [37]:
```
char_tess = " ".join(tess)
char_new_tess = " ".join(new_tess)
char_truth = " ".join(truth)
```

In [38]:
```
def insect(sa,sb):
    S_a = set(sa)
    n=0
    for s in S_a:
        n += min(sa.count(s), sb.count(s))
    return n
```

In [39]:
```
old_word = insect(tess, truth)
new_word = insect(new_tess, truth)
old_char = insect(char_tess, char_truth)
new_char = insect(char_new_tess, char_truth)
```

In [40]:
```
recall_word_tess = old_word/len(truth)
precision_word_tess = old_word/len(tess)
recall_word_post = new_word/len(truth)
precision_word_post = new_word/len(new_tess)

recall_char_tess = old_char/len(char_truth)
precision_char_tess = old_char/len(char_tess)
recall_char_post = new_char/len(char_truth)
precision_char_post = new_char/len(char_new_tess)

Measure = pd.DataFrame({"Measure": np.array(['word_wise_recall','word_wise_precision'
Tesseract = pd.DataFrame({"Tesseract": np.array([recall_word_tess, precision_word_tess
PostProcessing = pd.DataFrame({"Post": np.array([recall_word_post, precision_word_post
pd.concat([Measure, Tesseract, PostProcessing], axis=1)
```

Out[40]:

	Measure	Tesseract	Post
0	word_wise_recall	0.645514	0.737668
1	word_wise_precision	0.651473	0.744477
2	character_wise_recall	0.921204	0.941688
3	character_wise_precision	0.950425	0.969093