1.Feature_Extraction

May 5, 2018

0.1 Feature extraction : to vectorize given words

- we are converting each word to a vectorized form which is a binary vector (containing only 0s and 1s)
- Each feature/column is a binary question. e.g. is first letter capitalized or not ? (Yes = 1 / No = 0)
- We are considering mainly suffix , prefix , prev and next context , some english rules of words as features
- At the end we will store this feature dataframe to a csv file , for future use : as this code takes time

```
In [1]: # importing necessary packages

        import pandas as pd
        import numpy as np
        import time
```

0.1.1 Loading training data

```
In [2]: df = pd.read_csv('./Data_for_postagging/postrain', sep = " ")
        df.head()
        df.shape

Out[2]: (211727, 3)
```

```
In [3]: # final dataframe : it ll be our data matrix

        final= pd.DataFrame(index = range(211730) , columns = ['word', 'tag', 'capitalize',
                                                               'digit_first',
                                                               's_dig__e_alpha', 'p_anti',
                                                               'p_pre', 'p_un',
                                                               'p_dis', 'p_inter', 'p_mis',
                                                               'p_non', 'p_over_under',
                                                               'p_in_im', 'p_en_em','s_able',
                                                               's_al_ial',
                                                              's_ed_ing','s_tion_ion','s_est',
                                                              's_less', 's_e_es',
                                                              's_en','s_ly','s_er','s_\'s_s\''
                                                              'prev_word'])
```

```
#prev_word represents that prev word is a, an or the
final

#df = pd.DataFrame(index=range(numRows),columns=range(numCols))

#final['word'][2] = 1
final.shape
final.head()
```

Out[3]:
```
  word  tag capitalize digit_first s_dig__e_alpha p_anti p_pre p_un p_dis  \
0  NaN  NaN        NaN         NaN            NaN    NaN   NaN  NaN   NaN
1  NaN  NaN        NaN         NaN            NaN    NaN   NaN  NaN   NaN
2  NaN  NaN        NaN         NaN            NaN    NaN   NaN  NaN   NaN
3  NaN  NaN        NaN         NaN            NaN    NaN   NaN  NaN   NaN
4  NaN  NaN        NaN         NaN            NaN    NaN   NaN  NaN   NaN

  p_inter   ...    s_ed_ing s_tion_ion s_est s_less s_e_es s_en s_ly s_er  \
0     NaN   ...         NaN        NaN   NaN    NaN    NaN  NaN  NaN  NaN
1     NaN   ...         NaN        NaN   NaN    NaN    NaN  NaN  NaN  NaN
2     NaN   ...         NaN        NaN   NaN    NaN    NaN  NaN  NaN  NaN
3     NaN   ...         NaN        NaN   NaN    NaN    NaN  NaN  NaN  NaN
4     NaN   ...         NaN        NaN   NaN    NaN    NaN  NaN  NaN  NaN

  s_'s_s' prev_word
0     NaN       NaN
1     NaN       NaN
2     NaN       NaN
3     NaN       NaN
4     NaN       NaN

[5 rows x 27 columns]
```

0.1.2 Extracting feautures

In [ ]:
```
for i in df.index.values:

    j += 1

   # if(j > 10):
   #     break

    if df.word[i][0].isupper():
        final['capitalize'][i] = 1
    else:
        final['capitalize'][i] = 0

    if df.word[i][0].isdigit():
```

```
        final['digit_first'][i] = 1
    else:
        final['digit_first'][i] = 0

    if df.word[i][0].isdigit() and df.word[i][-1].isalpha():
        final['s_dig__e_alpha'][i] = 1
    else:
        final['s_dig__e_alpha'][i] = 0

    if df.word[i].startswith('anti'):
        final['p_anti'][i] = 1
    else:
        final['p_anti'][i] = 0

    if df.word[i].startswith('pre'):
        final['p_pre'][i] = 1
    else:
        final['p_pre'][i] = 0

    if df.word[i].startswith('un'):
        final['p_un'][i] = 1
    else:
        final['p_un'][i] = 0
    if df.word[i].startswith('dis'):
        final['p_dis'][i] = 1
    else:
        final['p_dis'][i] = 1
    if df.word[i].startswith('inter'):
        final['p_inter'][i] = 1
    else:
        final['p_inter'][i] = 0
    if df.word[i].startswith('mis'):
        final['p_mis'][i] = 1
    else:
        final['p_mis'][i] = 0
    if df.word[i].startswith('non'):
        final['p_non'][i] = 1
    else:
        final['p_non'][i] = 0

    if df.word[i].startswith('over') or df.word[i].startswith('under'):
        final['p_over_under'][i] = 1
    else:
        final['p_over_under'][i] = 1

    if df.word[i].startswith('in') or df.word[i].startswith('im'):
        final['p_in_im'][i] = 1
    else:
```

```
        final['p_in_im'][i] = 0
    if df.word[i].startswith('en') or df.word[i].startswith('em'):
        final['p_en_em'][i] = 1
    else:
        final['p_en_em'][i] = 0

    if df.word[i].endswith('able'):
        final['s_able'][i] = 1
    else:
        final['s_able'][i] = 0

    if df.word[i].endswith('al') or df.word[i].endswith('ial'):
        final['s_al_ial'][i] = 1
    else:
        final['s_al_ial'][i] = 0

    if df.word[i].endswith('ed') or df.word[i].endswith('ing'):
        final['s_ed_ing'][i] = 1
    else:
        final['s_ed_ing'][i] = 0

    if df.word[i].endswith('tion') or df.word[i].endswith('ion'):
        final['s_tion_ion'][i] = 1
    else:
        final['s_tion_ion'][i] = 0
    if df.word[i].endswith('est'):
        final['s_est'][i] = 1
    else:
        final['s_est'][i] = 0
    if df.word[i].endswith('less'):
        final['s_less'][i] = 1
    else:
        final['s_less'][i] = 0
    if df.word[i].endswith('e') or df.word[i].endswith('es'):
        final['s_e_es'][i] = 1
    else :
        final['s_e_es'][i] = 0
    if df.word[i].endswith('en'):
        final['s_en'][i] = 1
    else:
        final['s_en'][i] = 0
    if df.word[i].endswith('ly'):
        final['s_ly'][i]=1
    else:
        final['s_ly'][i]=0
    if df.word[i].endswith('er'):
        final['s_er'][i] = 1
    else:
```

```
            final['s_er'][i] = 0
        if df.word[i].endswith('\'s') or  df.word[i].endswith('s\''):
            final['s_\'s_s\''][i] = 1
        else:
            final['s_\'s_s\''][i] = 0

        if i >= 1:
            if df.word[i-1] == 'a' or df.word[i-1] == 'an' or df.word[i-1] == 'the':
                final['prev_word'][i] = 1
            else:
                final['prev_word'][i] = 0
        else:
            final['prev_word'][i] = 0

        final['word'][i]= df.word[i]
        final['tag'][i]= df['pos_tag'][i]

        #if(j % 20000 == 0):
        #    print(j)

    # append tks 2 mins for first 20000 and then time increases exponentially
    # fixed size array tks 10 mins for every 20k rows

    # NEVER EVER use append (means arraylist) when u knw array size in advance
    # performance of array is way way better than arraylist
```

0.1.3 Storing featurized dataframe to disk : for future use

```
In [4]: # storing the featurized df to disk into csv format
        final.to_csv('./final_df_by_arr.csv',sep = ',' , index = False , encoding = 'utf-8')

        # reading from disk
        checkdf = pd.read_csv('./final_df.csv' , sep = ',' )
        print(checkdf.shape)
        checkdf.head()
```

```
(211727, 27)
```

```
Out[4]:          word  tag  capitalize  digit_first  s_dig__e_alpha  p_anti  p_pre  \
        0  Confidence   NN           1            0               0       0      0
        1          in   IN           0            0               0       0      0
        2         the   DT           0            0               0       0      0
        3       pound   NN           0            0               0       0      0
        4          is  VBZ           0            0               0       0      0

           p_un  p_dis  p_inter    ...      s_ed_ing  s_tion_ion  s_est  s_less  \
        0     0      0        0    ...             0           0      0       0
```

```
1      0       0         0     ...             0           0      0       0
2      0       0         0     ...             0           0      0       0
3      0       0         0     ...             0           0      0       0
4      0       0         0     ...             0           0      0       0

   s_e_es  s_en  s_ly  s_er  s_'s_s'  prev_word
0       1     0     0     0        0          0
1       0     0     0     0        0          0
2       1     0     0     0        0          0
3       0     0     0     0        0          1
4       0     0     0     0        0          0

[5 rows x 27 columns]
```

```
In [ ]: checkdf['tag'].value_counts()
```

2.Softmax_Training

May 5, 2018

0.1 Training softmax model on train data : For standalone probabilities

- This notebook will add some more feautres we have considered later : such as length and adding prev and next context means : feature of prev word and next word ll be appended to current word feature
- Then we ll at last run softmax model for multiclass classfication and train it on training dataset

```
In [1]: # importing necessary packages
        import pandas as pd
        import numpy as np
```

0.1.1 loading featurized data matrix that we have already stored in disk (final_df.csv)

```
In [2]: df = pd.read_csv('./final_df.csv')
        df.head()

Out[2]:          word  tag  capitalize  digit_first  s_dig__e_alpha  p_anti  p_pre  \
        0  Confidence   NN           1            0               0       0      0
        1          in   IN           0            0               0       0      0
        2         the   DT           0            0               0       0      0
        3       pound   NN           0            0               0       0      0
        4          is  VBZ           0            0               0       0      0

           p_un  p_dis  p_inter   ...      s_ed_ing  s_tion_ion  s_est  s_less  \
        0     0      0        0   ...             0           0      0       0
        1     0      0        0   ...             0           0      0       0
        2     0      0        0   ...             0           0      0       0
        3     0      0        0   ...             0           0      0       0
        4     0      0        0   ...             0           0      0       0

           s_e_es  s_en  s_ly  s_er  s_'s_s'  prev_word
        0       1     0     0     0        0          0
        1       0     0     0     0        0          0
        2       1     0     0     0        0          0
        3       0     0     0     0        0          1
        4       0     0     0     0        0          0

        [5 rows x 27 columns]
```

0.1.2 Adding some more features : length , hyphen , all capital etc.

In [3]:
```
length = []
for s in df['word']:
    length.append(len(s))

length_2 = np.asarray(length).reshape(-1,1)
df['length'] = length_2
df.head()
```

Out[3]:
```
         word  tag  capitalize  digit_first  s_dig__e_alpha  p_anti  p_pre  \
0  Confidence   NN           1            0               0       0      0
1          in   IN           0            0               0       0      0
2         the   DT           0            0               0       0      0
3       pound   NN           0            0               0       0      0
4          is  VBZ           0            0               0       0      0

   p_un  p_dis  p_inter  ...   s_tion_ion  s_est  s_less  s_e_es  s_en  \
0     0      0        0  ...            0      0       0       1     0
1     0      0        0  ...            0      0       0       0     0
2     0      0        0  ...            0      0       0       1     0
3     0      0        0  ...            0      0       0       0     0
4     0      0        0  ...            0      0       0       0     0

   s_ly  s_er  s_'s_s'  prev_word  length
0     0     0        0          0      10
1     0     0        0          0       2
2     0     0        0          0       3
3     0     0        0          1       5
4     0     0        0          0       2

[5 rows x 28 columns]
```

In [48]:
```
hyphen = []
all_cap = []
sp_char = []
cap_dot = []
number = []
st_alpha_dot = []
e_day = []
for i in df['word']:
    if('-' in i):
        hyphen.append(1)
    else:
        hyphen.append(0)
    if(i.isupper()):
        all_cap.append(1)
    else:
        all_cap.append(0)
```

```
        if(('$' in i or ',' in i or '%' in i or '!' in i or '\'' in i or '"' in i) \
           and len(i) == 1):
            sp_char.append(1)
        else:
            sp_char.append(0)
        if(i[0].isupper() and i.endswith('.')):
            cap_dot.append(1)
        else:
            cap_dot.append(0)
        if(i[0].isdigit() and i[-1].isdigit() and ('.' in i or ',' in i)):
            number.append(1)
        else:
            number.append(0)
        if(i[0].isalpha() and '.' in i):
            st_alpha_dot.append(1)
        else:
            st_alpha_dot.append(0)
        if(i.endswith('day')):
            e_day.append(1)
        else:
            e_day.append(0)
    #print('\'')
```

```
In [50]: df['hyphen'] = hyphen
         df['all_cap'] = all_cap
         df['sp_char'] = sp_char
         df['cap_dot'] = cap_dot
         df['number'] = number
         df['st_alpha_dot'] = st_alpha_dot
         df['e_day'] = e_day
```

0.1.3 converting dataframe to Data Matrix 'X' and target column vector 'y'

```
In [51]: data = np.array(df)

         x = data[:,2:]
         x.shape
         y = df['tag']
         y.shape
         x.shape
```

```
Out[51]: (211727, 33)
```

0.1.4 Adding/Appending prev and next context : appending feature vect

```
In [52]: no_cols = x.shape[1] * 5
         features = np.zeros((211730,no_cols))
```

```
b = no_cols / 5
b = int(b)
```

In [53]:
```
# taking prev and next words as feature vectors : appending the stored
# feature vectors of the same

for i in range(x.shape[0]):
    if(i==0):
        features[i][b * 2: b* 3] = x[i]
        features[i][b* 3:b* 4] = x[i+1]
        features[i][b* 4:b* 5] = x[i+2]

    elif(i == 1):
        features[i][b* 1:b* 2] = x[i-1]
        features[i][b* 3:b* 4] = x[i+1]
        features[i][b* 2:b* 3] = x[i]
        features[i][b* 4:b* 5] = x[i+2]

    elif(i == x.shape[0]-1):
        features[i][0:b* 1] = x[i-2]
        features[i][b* 1:b* 2] = x[i-1]
        features[i][b* 2:b* 3] = x[i]

    elif(i == x.shape[0]-2):
        features[i][b* 3:b* 4] = x[i+1]
        features[i][0:b* 1] = x[i-2]
        features[i][b* 1:b* 2] = x[i-1]
        features[i][b* 2:b* 3] = x[i]

    else:
        features[i][0:b* 1] = x[i-2]
        features[i][b* 1:b* 2] = x[i-1]
        features[i][b* 2:b* 3] = x[i]
        features[i][b* 3:b* 4] = x[i+1]
        features[i][b* 4:b* 5] = x[i+2]
```

In [54]:
```
features[features.shape[0]]
```

Out[54]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 0., 0., 0., 0., 0., 6., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 3., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 2., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 6., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 4., 0., 0., 0., 0., 0., 0., 0.])
```

In [56]:
```
feature_vect = features[:-3]
feature_vect.shape
```

Out[56]: (211727, 165)

0.1.5 Importing packages for running softmax model

In [61]:
```
from sklearn import cross_validation
from sklearn.model_selection import train_test_split
from sklearn.cross_validation import cross_val_score
import pickle
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
```

0.1.6 Using label encoder to transform target label column to numerical encoding

In [62]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(y)
Y = le.transform(y)
```

0.1.7 Storing this mapping of tags to numerical numbers : we ll use this later in viterbi algorithm

In [63]:
```
pair = dict()
for i in range(Y.shape[0]):
    pair[y[i]] = Y[i]

pickle.dump(pair, open('pair_unigrams.pkl', 'wb'))
```

0.1.8 Training softmax model

In [64]:
```
softmax_2 = LogisticRegression( multi_class = 'multinomial', solver = 'sag')
softmax_2.fit(feature_vect,Y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning
  "the coef_ did not converge", ConvergenceWarning)
```

Out[64]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='multinomial',
          n_jobs=1, penalty='l2', random_state=None, solver='sag',
          tol=0.0001, verbose=0, warm_start=False)
```

In [68]:
```
# y_pred = softmax_2.predict(X_test)
# accuracy_score(y_pred,y_test)
```

0.1.9 Storing Trained softmax model

```
In [65]: pickle.dump(softmax_2, open('softmax.pkl', 'wb'))
```

3.Softmax_C^2_classes

May 5, 2018

0.1 Training softmax model on train data : For 'C^2' number of classes

- This notebook is doing the same work as previous one (2.softmax_training) with one difference that in this we ll train softmax model on c^2 number of classes

```
In [ ]: # importing necessary packages
        import pandas as pd
        import numpy as np
```

0.1.1 loading feateurzied data matrix from disk

```
In [ ]: df = pd.read_csv('./final_df.csv')
        df.head()
```

0.1.2 adding more features

```
In [ ]: length = []
        for s in df['word']:
            length.append(len(s))

        length_2 = np.asarray(length).reshape(-1,1)
        df['length'] = length_2
        df.head()
```

```
In [ ]: hyphen = []
        all_cap = []
        sp_char = []
        cap_dot = []
        number = []
        st_alpha_dot = []
        e_day = []
        for i in df['word']:
            if('-' in i):
                hyphen.append(1)
            else:
                hyphen.append(0)
            if(i.isupper()):
                all_cap.append(1)
            else:
```

```
            all_cap.append(0)
        if(('$' in i or ',' in i or '%' in i or '!' in i or '\'' in i or '"' in i) \
           and len(i) == 1):
            sp_char.append(1)
        else:
            sp_char.append(0)
        if(i[0].isupper() and i.endswith('.')):
            cap_dot.append(1)
        else:
            cap_dot.append(0)
        if(i[0].isdigit() and i[-1].isdigit() and ('.' in i or ',' in i)):
            number.append(1)
        else:
            number.append(0)
        if(i[0].isalpha() and '.' in i):
            st_alpha_dot.append(1)
        else:
            st_alpha_dot.append(0)
        if(i.endswith('day')):
            e_day.append(1)
        else:
            e_day.append(0)
    #print('\'')
```

```
In [ ]: df['hyphen'] = hyphen
        df['all_cap'] = all_cap
        df['sp_char'] = sp_char
        df['cap_dot'] = cap_dot
        df['number'] = number
        df['st_alpha_dot'] = st_alpha_dot
        df['e_day'] = e_day
```

```
In [ ]: tags = df.tag.unique()
        len(tags)
```

0.1.3 creating c^2 number of classes : possible classes from train data

```
In [ ]: bigram = []
        for i in df.index.values:
            if( i == 0):
                s = 'DT'
                tag = " ".join([s, df.tag[i]])
                bigram.append(tag)
            else:
                tag = " ".join([df.tag[i-1], df.tag[i]])
                bigram.append(tag)
```

```
In [ ]: tags = np.asarray(bigram)
```

```
In [ ]: df['tags'] = bigram
```

```
In [ ]: df.head()
```

0.1.4 Data Matrix X and target coln y

```
In [ ]: data = np.array(df)

        x = data[:,2:-1]
        x.shape
        y = df['tags']
        y.shape
        x.shape
```

0.1.5 Adding context in feautres (prev and next)

```
In [ ]: no_cols = x.shape[1] * 5
        features = np.zeros((211730,no_cols))

        b = no_cols / 5
        b = int(b)
```

```
In [ ]: # taking prev and next words as feature vectors : appending
        # the stored feature vectors of the same

        for i in range(x.shape[0]):
            if(i==0):
                features[i][b * 2: b* 3] = x[i]
                features[i][b* 3:b* 4] = x[i+1]
                features[i][b* 4:b* 5] = x[i+2]

            elif(i == 1):
                features[i][b* 1:b* 2] = x[i-1]
                features[i][b* 3:b* 4] = x[i+1]
                features[i][b* 2:b* 3] = x[i]
                features[i][b* 4:b* 5] = x[i+2]

            elif(i == x.shape[0]-1):
                features[i][0:b* 1] = x[i-2]
                features[i][b* 1:b* 2] = x[i-1]
                features[i][b* 2:b* 3] = x[i]

            elif(i == x.shape[0]-2):
                features[i][b* 3:b* 4] = x[i+1]
                features[i][0:b* 1] = x[i-2]
                features[i][b* 1:b* 2] = x[i-1]
                features[i][b* 2:b* 3] = x[i]

            else:
```

```
                features[i][0:b* 1] = x[i-2]
                features[i][b* 1:b* 2] = x[i-1]
                features[i][b* 2:b* 3] = x[i]
                features[i][b* 3:b* 4] = x[i+1]
                features[i][b* 4:b* 5] = x[i+2]
```

```
In [ ]: features[features.shape[0]]
```

```
In [ ]: feature_vect = features[:-3]
        feature_vect.shape
```

```
In [ ]: counts = dict(df['tags'].value_counts())
```

```
In [ ]: for i in counts:
            if(counts[i] < 8):
                print(i)
```

0.1.6 importing packages for running softmax on c^2 no of classes

```
In [ ]: from sklearn import cross_validation
        from sklearn.model_selection import train_test_split
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import confusion_matrix
        from sklearn.linear_model import LogisticRegression
        import pickle
```

```
In [ ]: from sklearn import preprocessing
        le = preprocessing.LabelEncoder()
        le.fit(y)
        Y = le.transform(y)
```

```
In [ ]: np.unique(Y).shape
```

```
In [ ]: softmax_2 = LogisticRegression( multi_class = 'multinomial', solver = 'sag')
        softmax_2.fit(feature_vect,Y)
```

0.1.7 storing trained softmax model

```
In [ ]: pickle.dump(softmax_2, open('softmax_dual.pkl', 'wb'))
```

0.1.8 storing c^2 no of classes mapping : we ll use this in viterbi algorithm

```
In [ ]: pair = dict()
        for i in range(Y.shape[0]):
            pair[y[i]] = Y[i]
```

```
In [ ]: len(pair)
```

```
In [ ]: pickle.dump(pair, open('pair_bigrams.pkl', 'wb'))
```

```
In [ ]: softmax_2.predict_proba(feature_vect[3].reshape(1,-1))
```

```
In [ ]: pair
```

4.Viterbi_Algorithm

May 5, 2018

0.1 Applying viterbi Algorithm : to consider tag sequence information : DP approach

- in this notebook we ll apply viterbi algorithm on test data , and conclude with accuracy on test data

```
In [ ]: # importing necessary pacakges
        import pandas as pd
        import numpy as np
        import pickle
```

0.1.1 loading test featurized data matrix to work on

```
In [ ]: df = pd.read_csv('./final_df_test.csv')
        df.head()
```

0.1.2 adding some more feautres

```
In [ ]: length = []
        for s in df['word']:
            length.append(len(s))

        length_2 = np.asarray(length).reshape(-1,1)
        df['length'] = length_2
```

```
In [ ]: hyphen = []
        all_cap = []
        sp_char = []
        cap_dot = []
        number = []
        st_alpha_dot = []
        e_day = []
        for i in df['word']:
            if('-' in i):
                hyphen.append(1)
            else:
                hyphen.append(0)
            if(i.isupper()):
                all_cap.append(1)
```

```
            else:
                all_cap.append(0)
            if(('$' in i or ',' in i or '%' in i or '!' in i or '\'' in i or '"' in i) \
               and len(i) == 1):
                sp_char.append(1)
            else:
                sp_char.append(0)
            if(i[0].isupper() and i.endswith('.')):
                cap_dot.append(1)
            else:
                cap_dot.append(0)
            if(i[0].isdigit() and i[-1].isdigit() and ('.' in i or ',' in i)):
                number.append(1)
            else:
                number.append(0)
            if(i[0].isalpha() and '.' in i):
                st_alpha_dot.append(1)
            else:
                st_alpha_dot.append(0)
            if(i.endswith('day')):
                e_day.append(1)
            else:
                e_day.append(0)
        #print('\'')
```

```
In [ ]: df['hyphen'] = hyphen
        df['all_cap'] = all_cap
        df['sp_char'] = sp_char
        df['cap_dot'] = cap_dot
        df['number'] = number
        df['st_alpha_dot'] = st_alpha_dot
        df['e_day'] = e_day
```

```
In [ ]: tags = df.tag.unique()
        len(tags)
```

0.1.3 creating a datamatrix 'X' and target label coln vector 'y'

```
In [ ]: data = np.array(df)

        x = data[:,2:]
        x.shape
        y = df['tag']
        y.shape
```

0.1.4 adding prev and next context as features

```
In [ ]: no_cols = x.shape[1] * 5
        features = np.zeros((47380,no_cols))
```

```
b = no_cols / 5
b = int(b)
```

In [ ]:
```
# taking prev and next words as feature vectors : appending the stored feature vectors

for i in range(x.shape[0]):
    if(i==0):
        features[i][b * 2: b* 3] = x[i]
        features[i][b* 3:b* 4] = x[i+1]
        features[i][b* 4:b* 5] = x[i+2]

    elif(i == 1):
        features[i][b* 1:b* 2] = x[i-1]
        features[i][b* 3:b* 4] = x[i+1]
        features[i][b* 2:b* 3] = x[i]
        features[i][b* 4:b* 5] = x[i+2]

    elif(i == x.shape[0]-1):
        features[i][0:b* 1] = x[i-2]
        features[i][b* 1:b* 2] = x[i-1]
        features[i][b* 2:b* 3] = x[i]

    elif(i == x.shape[0]-2):
        features[i][b* 3:b* 4] = x[i+1]
        features[i][0:b* 1] = x[i-2]
        features[i][b* 1:b* 2] = x[i-1]
        features[i][b* 2:b* 3] = x[i]

    else:
        features[i][0:b* 1] = x[i-2]
        features[i][b* 1:b* 2] = x[i-1]
        features[i][b* 2:b* 3] = x[i]
        features[i][b* 3:b* 4] = x[i+1]
        features[i][b* 4:b* 5] = x[i+2]
```

In [ ]:
```
feature_vect = features[:-3]
feature_vect.shape
```

0.1.5 label encoding

In [ ]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(y)
Y = le.transform(y)
```

0.1.6 Loading trained softmax model for both c and c^2 no of classes , and also loading mapping of class label to numerical encoding

```
In [ ]: uni_pairs = pickle.load(open('./pair_unigrams.pkl', 'rb'))
        bi_pairs = pickle.load(open('./pair_bigrams.pkl', 'rb'))
        soft_uni = pickle.load(open('./softmax.pkl', 'rb'))
        soft_bi = pickle.load(open('./softmax_dual.pkl', 'rb'))
```

0.1.7 creating list of sentences on which we ll apply viterbi algo : dynamic programming approach

```
In [ ]: # creating list of sentences
        list_sent = []
        l = []
        for word in df.word:
            l.append(word)
            if(word == '.'):
                list_sent.append(l)
                l = []
```

```
In [ ]: sent = list(df['word'][1712:1742])
```

```
In [ ]: feature_vect[45:45]
```

```
In [ ]: uni_pairs
        res = dict((v,k) for k,v in uni_pairs.items())
```

0.2 Viterbi Alogrithm Implementation

```
In [ ]: feat_vect_cnt = 0

        total_pred_cnt = 0
        total_actual_cnt = 0

        for sent in list_sent:

            table = np.zeros((44,len(sent)))
            tindex = np.zeros((44,len(sent)), dtype=int)

            # taking standalone probs of first word of a sent
            x = soft_uni.predict_proba(feature_vect[feat_vect_cnt].reshape(1,-1))

            #intialization : storing standalone probs of w1 in column 1
            for i in range(x.shape[1]):
                table[i][0] = x[0][i]

            tindex[:,0:1] = -1
            #print(table[:,0:1].shape)
```

```
# viterbi algorithm : dp logic
word = 1     # starting with 2nd coln : word2
while(word < len(sent)):
    for tag in range(0,44):
        t1name = res[tag]
        prob2 = soft_bi.predict_proba(feature_vect[word + feat_vect_cnt].reshape(1
        mulmax = 0
        index = -1
        #print(prob2[0])
        for t2 in range(0,44):      # loop for first tag
            tag_pair = " ".join([res[t2],t1name])
                #print(tag_pair)
            if(tag_pair in bi_pairs):           # if pair is thr then use prob els
                sec_term = prob2[0][bi_pairs[tag_pair]]
                #print(" a " ,sec_term)
                f_term = table[t2][word-1]
                #print(f_term)                    # pr(ti/w1)
                mul_term = f_term * sec_term
                if(mul_term > mulmax):
                    mulmax = mul_term
                    index = t2
        table[tag][word] = mulmax        # storing max prob out of 44 pssble probs
        tindex[tag][word] = index       # storing index in tindex

    word += 1



print('start of sent : ',total_actual_cnt)


# storing tag sequences tht maximizes the probability in reverse order
j = tindex.shape[1] - 1
#np.max(table[:, -1:])
k = np.argmax(table[:, -1:])
tag_index = []
tag_index.append(k)
while j > 0:
    k = tindex[k][j]
    j -= 1
    tag_index.append(k)

# actual ordering
actual = []
for i in reversed(tag_index):
    actual.append(res[i])
total_actual_cnt += len(actual)
```

```
    # calculating accuracy
    c = 0
    j = feat_vect_cnt
    for i in range(len(actual)):
        if df.tag[j] == actual[i]:
            c += 1
        j += 1
    total_pred_cnt += c

    # increasing feature_vector count to starting of the next sent
    feat_vect_cnt += len(sent)

    #print(c/len(actual))

accuracy = (total_pred_cnt/total_actual_cnt)*100
print("accuracy : " , accuracy)
```

Accuracy : 67.4441184541022