دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده کامپیوتر

گزارش کار تمرین شماره 3

«ابهام زدایی معنایی کلمات»

نگارش
زهرا علی میرزایی

استاد درس
دکتر ممتازی

بخش اول: تعرف مسئله و معرفی دادگان:

با استفاده از قطعه کد زیر ستون pos و context را از کل مجموعه جدا میکنیم و برای ستون pos به صفت ها کد 1، به اسم ها کد 2، و به افعال کد 3 اختصاص میدهیم:

```
[ ]  context_list=list(dataframe['context'].values)
```

```
[ ]  def assignNewLabels(label):
         if label == 'a':
             return 1
         elif label == 'n':
             return 2
         elif label == 'v':
             return 3
```

```
[ ]  dataframe['pos'] = dataframe['pos'].apply(assignNewLabels)
```

```
[ ]  pos_list=list(dataframe['pos'].values)
```

با استفاده از کد زیر علایم نگارشی را از لیست context حذف میکنیم:

```
[ ]  # define punctuation
     punctuations = '''!()-[]{};:'"\,.?@#$%^&*_~'''
     for i in range(len(context_list)):

       my_str = context_list[i]

       # To take input from the user

       # remove punctuation from the string
       no_punct = ""
       for char in my_str:
         if char not in punctuations:
             no_punct = no_punct + char

       # display the unpunctuated string
       context_list[i]=no_punct
```

با استفاده از قطعه کد زیر کلمات مبهم که درون تگ head قرار دارند را از درون متون جدا کرده و در یک لیست جدا قرار میدهیم:

```
word_disg=[]
for i in range (len(context_list)):
  text = context_list[i]
  left = '<head>'
  right = '</head>'

    # Output: 'string'
  word_disg.append(text[text.index(left)+len(left):text.index(right)])
```

با استفاده از کد زیر تگ های head را از متون حذف میکنیم و درون یک لیست جدید میریزیم:

```
import re

def cleanhtml(raw_html):
  cleanr = re.compile('<.*?>')
  cleantext = re.sub(cleanr, '', raw_html)
  return cleantext
```

```
context_list_without_html=[]
for i in range (len(context_list)):
  context_list_without_html.append(cleanhtml(context_list[i]))
```

مثال:

```
context_list_without_html[0]
```

```
'For another  many of the genes carried by plasmids  such as those specifying resistance to the antibiotics kanamycin or pe
nicillin  are flanked by special DNA which enables them to jump from plasmid to chromosome and back  or from one plasmid to
another  Again  these jumping genes  or transposons  cause chromosomal fluctuations  Moreover  the chromosomes of some
strains of E coli contain enigmatic lengths of DNA  insertion sequences  which code for no known product but can move about
the chromosome during bacterial multiplication  activating or silencing genes  They are present in plasmids  too  Finally
the chromosome is not even intrinsically stable  it mutates  '
```

با استفاده از کد زیر کلمه مبهم را درون متن پیدا میکنیم و +9 و -9 کلمه قبل و بعد آن را نیز انتخاب میکیم و درون لیست train میریزیم تا بعدا برای آموزش شبکه استفاده کنیم. نکته آن است که بعضی از کلمات کمتر از 9 کلمه قبلشان وجود دارد که برای آنها شعاع را به اندازه ی فاصله ی آنها تا ابتدای متن شعاع همسایگی را قرار میدهیم:

```
[ ]  train_list=[]
     for i in range (len(context_list_without_html)):
       m=context_list_without_html[i].split()
       b=context_list[i].split()
       z='<head>'+word_disg[i]+'</head>'
       n=b.index(z)
       if n<9:
         f=m[0:2*n+1]
       else:
         f=m[n-9:n+10]
       s=' '.join(f)
       train_list.append(s)
```

مثال:

```
[ ]  train_list[0]
```

```
'but can move about the chromosome during bacterial multiplication activating or silencing genes They are present in plasmid
s too'
```

با استفاده از کد زیر لیست کلمات مبهم را بصورت یک دست به حروف کوچک تبدیل میکنیم:

```
[ ]  for i in range(len(word_disg)):
       word_disg[i]=word_disg[i].lower()
```

با استفاده از کد زیر لیست آموزش برای کلمات مبهم و 3 کلمه قبل و بعد آن را برای آموزش مدل Bert جدا میکنیم:

```
[ ]  rep_list_plus3=[]
     for i in range(len(result2)):
       sum=0
       vector=0
       for j in range(len(result2[i][0])):
         z = dic.get(result2[i][0][j])
         if z is not None:
           n=dic[result2[i][0][j]]
         else:
           n=1
         vector=vector+n*result2[i][1][j]
         sum=sum+n
       rep=vector/sum
       rep_list_plus3.append(rep)
```

بخش دوم: انتخاب ویژگی:

الف) استفاده از مدل از پیش آموزش داده شده BERT و استخراج بازنمایی تنها برای کلمه مبهم موجود در متن:

با استفاده از کد زیر لیست آموزشی شامل متون 19 کلمه ای (کلمه مبهم + 9 کلمه اطراف آن) را به مدل BERT میدهیم:

```
[ ]  from bert_embedding import BertEmbedding

     bert_embedding = BertEmbedding(model='bert_12_768_12',max_seq_length=30)
     result = bert_embedding(train_list)
```

```
Vocab file is not found. Downloading.
Downloading /root/.mxnet/models/book_corpus_wiki_en_uncased-a6607397.zip from htt
Downloading /root/.mxnet/models/bert_12_768_12_book_corpus_wiki_en_uncased-75cc78
```

با استفاده از کد زیر بردار بدست آمده برای کلمات مبهم را درون لیست جدید ذخیره میکنیم و یک آرایه 2 بعدی 5987*768 از آنها میسازیم:

```
rep_disg_word=[]
for i in range(len(result)):
  #print(i)
  k=result[i][0].index(word_disg[i])
  rep_disg_word.append(result[i][1][k])
```

```
[ ]  import numpy as np
     w=[  list(rep_disg_word[i]) for i in range(len(rep_disg_word)) ]
     matrix_of_rep = np.array(w)
```

الگوریتم PCA را بر روی آرایه 2 بعدی اعمال میکنیم تا ابعاد بردار ها به 300 کاهش یابند:

```
[ ]  from sklearn.decomposition import PCA
     import pandas as pd
     pca = PCA(n_components=300)
     principalComponents = pca.fit_transform(matrix_of_rep)
     principalDf = pd.DataFrame(data = principalComponents)
```

ب)استفاده از مدل از پیش آموزش داده شده BERT و استخراج بازنمایی کلمات از مدل BERT برای کلمه هدف و سایر کلمات موجود در بافت کلمه هدف به شعاع 3 ± کلمه و سپس استفاده از میانگین وزندار بازنمایی کلمات موجود در بافت کلمه مبهم با استفاده از IDF-TF هر یک از کلمات:

مدل TF-idf کلمات را بصورت یک دیکشنری برای کلمات متن ها اعمال کرده و عدد TF-idf هر کلمه را به عنوان value آن کلمه ذخیره خواهیم کرد:

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
# our corpus
data = context_list_without_html
cv = CountVectorizer()

# convert text data into term-frequency matrix
data = cv.fit_transform(data)

tfidf_transformer = TfidfTransformer()

# convert term-frequency matrix into tf-idf
tfidf_matrix = tfidf_transformer.fit_transform(data)

# create dictionary to find a tfidf word each word
word2tfidf = dict(zip(cv.get_feature_names(), tfidf_transformer.idf_))
dic={}
for word, score in word2tfidf.items():
  dic[word] = score
```

مدل BERT را بر روی لیست متون 7 کلمه ای (کلمه مبهم + 3 کلمه قبل و بعد آن) آموزش میدهیم:

```
[ ] from bert_embedding import BertEmbedding

    bert_embedding = BertEmbedding(model='bert_12_768_12',max_seq_length=30)
    result2 = bert_embedding(train_list2)
```

لیست بازنمایی کلمات بدست آمده را با استفاده از میانگین گیری وزن دار بر روی هر 7 کلمه بصورت زیر بدست می آوریم:

```
[ ]  rep_list_plus3=[]
     for i in range(len(result2)):
       sum=0
       vector=0
       for j in range(len(result2[i][0])):
         z = dic.get(result2[i][0][j])
         if z is not None:
           n=dic[result2[i][0][j]]
         else:
           n=1
         vector=vector+n*result2[i][1][j]
         sum=sum+n
       rep=vector/sum
       rep_list_plus3.append(rep)
```

همانند بخش قبل ابتدا یک ماتریس 5987*768 بعدی از بازنمایی ها میسازیم و سپس الگوریتم PCA را روی آن اعمال میکنیم:

```
[ ]  import numpy as np
     w=[  list(rep_list_plus3[i]) for i in range(len(rep_list_plus3)) ]
     matrix_of_rep_plus3 = np.array(w)
```

```
[ ]  from sklearn.decomposition import PCA
     import pandas as pd
     pca = PCA(n_components=300)
     principalComponents = pca.fit_transform(matrix_of_rep_plus3)
     principalDf_plus3 = pd.DataFrame(data = principalComponents)
```

ج) تکرار بند (ب) ولی با در نظر گرفتن تمام کلمات موجود در بافت کلمه مبهم:

تکرار تمامی مراحل قبلی با این تفاوت که برای آموزش از مجموعه دادگان 19 کلمه ای استفاده میشود و در نهایت برای باز نمایی هر کلمه از میانگین گیری وزندار با استفاده از TF-idf مربوط به هر کلمه استفاده میکنیم:

```
[ ]  from bert_embedding import BertEmbedding

     bert_embedding = BertEmbedding(model='bert_12_768_12',max_seq_length=30)
     result3 = bert_embedding(train_list3)
```

```
rep_list_plus9=[]
for i in range(len(result3)):
  sum=0
  vector=0
  for j in range(len(result3[i][0])):
    z = dic.get(result3[i][0][j])
    if z is not None:
      n=dic[result3[i][0][j]]
    else:
      n=1
    vector=vector+n*result3[i][1][j]
    sum=sum+n
  rep=vector/sum
  rep_list_plus9.append(rep)
```

```
import numpy as np
w=[  list(rep_list_plus9[i]) for i in range(len(rep_list_plus9)) ]
matrix_of_rep_plus9 = np.array(w)
```

اعمال PCA :

```
[ ]  from sklearn.decomposition import PCA
     import pandas as pd
     pca = PCA(n_components=300)
     principalComponents = pca.fit_transform(matrix_of_rep_plus9)
     principalDf_plus9 = pd.DataFrame(data = principalComponents)
```

د) استفاده از بازنمایی از پیش آموزش داده شده Word2Vec برای کلمه هدف و سایر کلمات موجود در بافت کلمه هدف به شعاع 3 ± کلمه و سپس استفاده از میانگین وزندار بازنمایی کلمات موجود در بافت کلمه مبهم با استفاده از TF-IDFهر یک از کلمات:

لیست 7 کلمه ای را به مدل Word2Vec داده و بر روی بردار های بدست آمده برای هر نمونه میانگین وزن دار TF-idf گرفته و درون یک لیست جدید بردار ها را ذخیره میکنیم:

```
from gensim.models import Word2Vec
# define training data
sentences = l
# train model
model = Word2Vec(train_list_word2vec,size=300, min_count=1)
```

```
WARNING:gensim.models.base_any2vec:under 10 jobs per worker: consider s
```

```
rep_list_word2vec=[]
for i in range(len(train_list_word2vec)):
  sum=0
  vector=0
  for j in range(len(train_list_word2vec[i])):
    z = dic.get(train_list_word2vec[i][j])
    if z is not None:
      n=dic[train_list_word2vec[i][j]]
    else:
      n=1
    vector=vector+n*model[train_list_word2vec[i][j]]
    sum=sum+n
  rep=vector/sum
  rep_list_word2vec.append(rep)
```

**بردار های حاصل از ابتدا 300 بعدی در نظر گرفته شده اند و نیازی به الگوریتم PCA برای کاهش ابعاد نیست.

ه) تکرار بند (د) ولی با در نظر گرفتن تمام کلمات موجود در بافت کلمه مبهم:

تکرار مراحل قبل با این تفاوت که برای آموزش مدل Word2Vec از مجموعه دادگان 19 کلمه ای استفاده کردیم:

```
[ ]  from gensim.models import Word2Vec
     # train model
     model = Word2Vec(train_list_word2vec9,size=300, min_count=1)
```

```
[ ]  rep_list_word2vec9=[]
     for i in range(len(train_list_word2vec9)):
       sum=0
       vector=0
       for j in range(len(train_list_word2vec9[i])):
         z = dic.get(train_list_word2vec9[i][j])
         if z is not None:
           n=dic[train_list_word2vec9[i][j]]
         else:
           n=1
         vector=vector+n*model[train_list_word2vec9[i][j]]
         sum=sum+n
       rep=vector/sum
       rep_list_word2vec9.append(rep)
```

بخش سوم: دسته بندی:

الف) برای دسته بندی از روشهای بازنمایی مطرح شده در بخش قبل و دسته بندهای Regression Logistic و Forest Random استفاده کنید. نتایج آزمایش های ذکرشده را با معیارهای Accuracy و Measure-F ارزیابی کنید .معیارهای ارزیابی را برای اسم ها، فعل ها و صفت ها به صورت مجزا، میانگین نتایج را گزارش کنید:

مانند بخش قبل داده های تست را نیز به شبکه دادیم و ابعاد بردار ها را با استفاده از PCA کاهش دادیم سپس هر کدام از حالات را به هر 2 شبکه اعمال کرده و نتایج زیر را بدست آوردیم(کلاس اول معرف صفت ها کلاس دوم اسم ها و کلاس سوم افعال هستند):

1)رندوم فارست + بردار بازنمایی فقط برای خود کلمه مبهم:

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.55	0.57	0.56	470
class verb	0.64	0.66	0.65	567
accuracy			0.60	1077
macro avg	0.40	0.41	0.40	1077
weighted avg	0.58	0.60	0.59	1077

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classi

2)لاجستیک رگرسیون + بردار بازنمایی فقط برای خود کلمه مبهم:

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.57	0.63	0.60	470
class verb	0.68	0.66	0.67	567
accuracy			0.62	1077
macro avg	0.42	0.43	0.42	1077
weighted avg	0.61	0.62	0.61	1077

3) رندوم فارست + بازنمایی +-3 کلمه:

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.43	0.34	0.38	470
class verb	0.53	0.66	0.59	567
accuracy			0.50	1077
macro avg	0.32	0.34	0.33	1077
weighted avg	0.47	0.50	0.48	1077

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_cl

4) لاجستیک رگرسیون + بازنمایی +-3 کلمه:

	precision	recall	f1-score	support
class adj	0.04	0.03	0.03	40
class noun	0.42	0.44	0.43	470
class verb	0.51	0.50	0.50	567
accuracy			0.46	1077
macro avg	0.32	0.32	0.32	1077
weighted avg	0.45	0.46	0.45	1077

5) رندوم فارست + بازنمایی +-9 کلمه:

```
print(classification_report(pos_list_test, y_pred, target_r
```

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.41	0.39	0.40	470
class verb	0.52	0.56	0.54	567
accuracy			0.47	1077
macro avg	0.31	0.32	0.31	1077
weighted avg	0.45	0.47	0.46	1077

6) لاجستیک رگرسیون + بازنمایی +- 9 کلمه :

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.41	0.39	0.40	470
class verb	0.52	0.56	0.54	567
accuracy			0.47	1077
macro avg	0.31	0.32	0.31	1077
weighted avg	0.45	0.47	0.46	1077

7) رندوم فارست + word2vec +- 3 کلمه:

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.44	0.99	0.61	470
class verb	0.67	0.01	0.02	567
accuracy			0.44	1077
macro avg	0.37	0.33	0.21	1077
weighted avg	0.54	0.44	0.28	1077

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_cla

8) لاجستیک رگرسیون +word2vec +- 3 کلمه:

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.00	0.00	0.00	470
class verb	0.53	1.00	0.69	567
accuracy			0.53	1077
macro avg	0.18	0.33	0.23	1077
weighted avg	0.28	0.53	0.36	1077

9)رندوم فارست + word2vec +- 9 کلمه :

```
print(classification_report(pos_list_test, y_pred, target
```

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.48	0.12	0.19	470
class verb	0.53	0.90	0.67	567
accuracy			0.53	1077
macro avg	0.34	0.34	0.29	1077
weighted avg	0.49	0.53	0.44	1077

10) لاجستیک رگرسیون+ word2vec +-9 کلمه:

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.00	0.00	0.00	470
class verb	0.53	1.00	0.69	567
accuracy			0.53	1077
macro avg	0.18	0.33	0.23	1077
weighted avg	0.28	0.53	0.36	1077

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_cl

ب) با توجه به ۵ بازنمایی مختلف و دو دسته بند مختلف، 10 حالت را در این تمرین آزمایش نموده اید. از میان این 10 حالت 3 نتیجه برتر را انتخاب کنید و با کمک این سه مدل یک مدل گروهی بسازید. برای پیدا کردن مفهوم مناسب برای هر کلمه مبهم بین نتایج 3 دسته بند رای اکثریت بگیرید. مانند قسمت قبل معیارهای ارزیابی Accuracyو Measure-Fرا برای هر POS به صورت جداگانه گزارش کنید:

با بررسی دقت حالت ها بهترین دقت ها بترتیب بصورت زیر بدست آمده است:

- حالت الف بخش 2 در مدل لاجستیک رگرسیون
- حالت الف بخش دوم در مدل رندوم فارست
- حالت د بخش دوم در مدل لاجستیک رگرسیون

از آنجایی که برای استفاده از کتابخانه آماده ی sklearn نیاز بود تا ورودی هر 3 مدل یکسان باشد با استفاده از قطعه کد زیر برنامه ای نوشتم تا با توجه به اکثریت هر لیبل بتواند یک لیبل انتخاب کند:

```
def frequent(list_no1):
    count = 0
    no = list_no1[0]
    #for loop
    for i in  list_no1:
        current_freq = list_no1.count(i
        if (current_freq > count):
            count = current_freq
            num = i

    return num

y_pred_total=[]
for i in range(len(y_pred1)):
  l=[]
  l.append(y_pred1[i])
  l.append(y_pred2[i])
  l.append(y_pred3[i])
  max=frequent(l)
  y_pred_total.append(max)
```

دقت کلاس بندی با این روش بصورت زیر بدست آمد:

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.62	0.52	0.56	470
class verb	0.65	0.77	0.70	567
accuracy			0.63	1077
macro avg	0.42	0.43	0.42	1077
weighted avg	0.61	0.63	0.62	1077

ج) با استفاده از 3 تا بهترین دستهبند انتخاب شده در بند (ب) یک مدل گروهی دیگر بسازید:

لیبل های 3 حالت برتر را به یک آرایه 2 بعدی تبدیل کرده و برای تحویل به مدل آن را transpose کردم:

```
new_train_set= np.array([y_pred1,y_pred2,y_pred3]).transpose()
```

```
new_train_set=pd.DataFrame(new_train_set)
```

```
new_train_set
```

	0	1	2
0	3	3	3
1	3	3	3
2	3	3	3
3	3	3	3
4	3	2	3
...	...	...	...
1072	3	3	3
1073	3	2	3
1074	3	3	3
1075	3	3	3
1076	2	2	3

1077 rows × 3 columns

داده ها را به یک مدل رندوم فارست اعمال کردم و نتایج زیر بدست آمد:

	precision	recall	f1-score	support
class adj	0.00	0.00	0.00	40
class noun	0.62	0.52	0.56	470
class verb	0.65	0.77	0.70	567
accuracy			0.63	1077
macro avg	0.42	0.43	0.42	1077
weighted avg	0.61	0.63	0.62	1077

د) در این قسمت میخواهیم برای ابهام زدایی معنایی کلمات از شبکه های عصبی استفاده کنیم. ابتدا بازنمایی توکن[cls] را از مدل BERT استخراج کنید و به یک شبکه عصبی پیشخور دهید (برای استفاده از مدل BERT به ابتدای هر متن ورودی، توکن [cls]اضافه میشود. بازنمایی که از مدل BERT برای این توکن در هر متن استخراج میشود، به عنوان بازنمایی برای کل متن در نظر و از آن در وظیفه دستهبندی متون کمک گرفته میشود.) مانند قسمت های قبل معیارهای ارزیابی Accuracy و Measure-Fرا برای هر POS به صورت جداگانه گزارش کنید:

برای این قسمت از کتابخانه ی دیگری به نام transformers استفاده کردم و بعد از مرتب سازی داده ها همانند بخش 1 آن را به یک شبکه Bert اعمال کردم:

اضافه کردن توکن های CLS و SEP به داده های 19 کلمه ای درون train ست:

```
from transformers import BertTokenizer, BertModel
train_data['sent'] = train_data['sent'].apply(lambda x : "[CLS] " + x + " [SEP]" )
train_data['tokens'] = train_data['sent'].apply(tokenizer.tokenize)
train_data['indx'] = train_data['tokens'].apply(tokenizer.convert_tokens_to_ids)
train_data['segments_ids'] = train_data['tokens'].apply(lambda x : [1] * len(x))
test_data['sent'] = test_data['sent'].apply(lambda x : "[CLS] " + x + " [SEP]" )
test_data['tokens'] = test_data['sent'].apply(tokenizer.tokenize)
test_data['indx'] = test_data['tokens'].apply(tokenizer.convert_tokens_to_ids)
test_data['segments_ids'] = test_data['tokens'].apply(lambda x : [1] * len(x))
```

اعمال داده های آماده شده به شبکه ی جدید:

```
model = BertModel.from_pretrained('bert-base-uncased', output_hidden_states = True )
```

ذخیره سازی بردار های بدست آمده:

```
train_data['embedding'] = None
for i in range(len(train_data)):
  with torch.no_grad():
    outputs = model(train_data['tokens_tensor'][i],train_data['segments_tensors'][i])
  train_data['embedding'][i] = outputs[2]

test_data['embedding'] = None
for i in range(len(test_data)):
  with torch.no_grad():
    outputs = model(test_data['tokens_tensor'][i],test_data['segments_tensors'][i])
  test_data['embedding'][i] = outputs[2]
```

با استفاده از یک MLP باز نمایی های بدست آمده را کلاس بندی کردیم:

```
from sklearn.neural_network import MLPClassifier
acc = []
f1 = []
for word in word_list:
  x = np.vstack(dataset[word]['cls'].values)
  y = dataset[word]['label'].values
  nn = MLPClassifier(hidden_layer_sizes=(256,) ,activation='relu' , solver='adam' )
  nn.fit(x,y)
  xx = np.vstack(test_dataset[word]['cls'].values)
  yy = test_dataset[word]['label'].values
  yp = nn.predict(xx)
  acc.append(accuracy_score(yy,yp))
  f1.append(f1_score(yy,yp , average='weighted'))

print (f'Accuracy => {100 * np.mean(acc):0.2f} %' )
print (f'F1-Measure => {np.mean(f1):0.2f}' )
```

در انتها دقت های زیر را بدست آوردیم:

```
Accuracy => verb: 57.65 %  noun: 60.83 %  adjective: 33.71 %
F1-Measure => verb: 0.53  noun: 0.57   adjective: 0.29
```

نتیجه گیری:

با توجه به نتایج بدست آمده در بررسی هر 10 حالت ذکر شده در گزارش مشخص شد که آموزش یک مدل logestic regression روی بازنمایی بدست آمده تنها برای از کلمه ی مبهم به تنهایی بهترین دقت را در پی خواهد داشت اما در ادامه با استفاده از یک مدل گروهی توانستیم آن را نیز بهبود بدهیم ، در ادامه با استفاده از بازنمایی توکن[CLS] و آموزش یک شبکه عصبی پیش خور نتایج مناسبی برای رفع ابهام از کلمات مبهم بدست آمد با بررسی نتایج برای هر گروه از pos ها مشخص شد طبقه بندها برای رفع ابهام از افعال و اسم ها دقت بالا تری داشته است اما در مواجه با صفت از رفع ابهام آنها عاجز بوده است این امر میتواند ریشه در تعداد بسیار کم صفت ها در دادگان آموزشی نسبت به دو گروه فعلی و اسمی داشته باشد. بهره وری از یک شبکه عصبی پیشخور توانست ما را تا حدودی در تشخیص صفت ها نیز یاری دهد.