KUMAR SANATAN

ROLL NO – 2211AI24

CS 564

Foundations of Machine Learning

ASSIGMENT 3:

INDIAN INSTITUTE OF TECHNOLOGY
PATNA

Date: 9th Nov 2022 **Deadline:** 17th Nov 2022

OBJECTIVE

The assignment targets to implement Hidden Markov Model (HMM) to perform Named Entity Recognition (NER) task

Named Entity Recognition using HMM

```
!pip install hmmlearn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: hmmlearn in /usr/local/lib/python3.7/dist-packages (0.2.
Requirement already satisfied: scipy>=0.19 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: scikit-learn>=0.16 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-pa
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
from tqdm import tqdm
from matplotlib import pyplot as plt # show graph
from sklearn.model_selection import GroupShuffleSplit
from hmmlearn import hmm
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precisio
```

```
data = pd.read_csv("/content/Result.csv")
data = data.fillna(method="ffill")
data = data.rename(columns={'Sentence #': 'sentence'})
data.head(5)
```

	Unnamed: 0	sentence	Word	Tag	POS
0	0	sentence : 0	@LewisDixon	O	NN
1	1	sentence : 0	Trust	O	NNP
2	2	sentence : 0	me	O	PRP
3	4	sentence : 0	im	O	VB
4	5	sentence : 0	gonna	O	NN

Get the numbers of tags & words inside the whole data. We'll need this in the future.

```
tags = list(set(data.POS.values)) #Read POS values
words = list(set(data.Word.values))
len(tags), len(words)
```

```
(40, 5128)
```

```
y = data.POS
X = data.drop('POS', axis=1)

gs = GroupShuffleSplit(n_splits=2, test_size=.33, random_state=42)
train_ix, test_ix = next(gs.split(X, y, groups=data['sentence']))

data_train = data.loc[train_ix]
data_test = data.loc[test_ix]

data_train
```

	Unnamed: 0	sentence	Word	Tag	POS
0	0	sentence : 0	@LewisDixon	O	NN
1	1	sentence : 0	Trust	O	NNP
2	2	sentence : 0	me	O	PRP
3	4	sentence : 0	im	O	VB
4	5	sentence : 0	gonna	O	NN
...	...	...	...	...	...
15480	17472	sentence : 899	test	O	NN
15481	17474	sentence : 899	sources	O	NNS
15482	17475	sentence : 899	tell	O	VBP
15483	17476	sentence : 899	TMZ	B	NNP
15484	17478	sentence : 899	http://bi	O	NN

10375 rows × 5 columns

	Unnamed: 0	sentence	Word	Tag	POS
64	79	sentence : 4	The	O	DT
65	80	sentence : 4	Basic	O	NNP
66	81	sentence : 4	Step	O	NNP
67	82	sentence : 4	Before	O	IN
68	83	sentence : 4	You	O	PRP
...	...	...	...	...	...
15430	17417	sentence : 896	thank	O	NN

After checking the data after splitted, it seems to be fine. Check the numbers of tags & words in the training set.

15433	17420	sentence : 896	's	O	VBZ

```
tags = list(set(data_train.POS.values)) #Read POS values
words = list(set(data_train.Word.values))
len(tags), len(words)
```

```
(40, 3798)
```

The number of tags is enough but the number of words is not enough (~29k vs ~35k). Because of that we need to randomly add some UNKNOWN words into the training dataset then we recalculate the word list and create map from them to number.

```
dfupdate = data_train.sample(frac=.2, replace=False)
dfupdate.Word = 'UNKNOWN'
data_train.update(dfupdate)
words = list(set(data_train.Word.values))
# Convert words and tags into numbers
word2id = {w: i for i, w in enumerate(words)}
tag2id = {t: i for i, t in enumerate(tags)}
id2tag = {i: t for i, t in enumerate(tags)}
len(tags), len(words)
```

```
(40, 3223)
```

Hidden Markov Models can be trained by using the Baum-Welch algorithm. However input of the training is just dataset (Words). We cannot map back the states to the POS tag.

That's why we have to calculate the model parameters for `hmmlearn.hmm.MultinomialHMM` manually by calculating

- `startprob_`

- transmat_
- emissionprob_

```
count_tags = dict(data_train.POS.value_counts())
count_tags_to_words = data_train.groupby(['POS']).apply(lambda grp: grp.groupby('Word')['POS'
count_init_tags = dict(data_train.groupby('sentence').first().POS.value_counts())

# TODO use panda solution
count_tags_to_next_tags = np.zeros((len(tags), len(tags)), dtype=int)
sentences = list(data_train.sentence)
pos = list(data_train.POS)
for i in range(len(sentences)) :
    if (i > 0) and (sentences[i] == sentences[i - 1]):
        prevtagid = tag2id[pos[i - 1]]
        nexttagid = tag2id[pos[i]]
        count_tags_to_next_tags[prevtagid][nexttagid] += 1


mystartprob = np.zeros((len(tags),))
mytransmat = np.zeros((len(tags), len(tags)))
myemissionprob = np.zeros((len(tags), len(words)))
num_sentences = sum(count_init_tags.values())
sum_tags_to_next_tags = np.sum(count_tags_to_next_tags, axis=1)
for tag, tagid in tag2id.items():
    floatCountTag = float(count_tags.get(tag, 0))
    mystartprob[tagid] = count_init_tags.get(tag, 0) / num_sentences
    for word, wordid in word2id.items():
        myemissionprob[tagid][wordid]= count_tags_to_words.get(tag, {}).get(word, 0) / floatC
    for tag2, tagid2 in tag2id.items():
        mytransmat[tagid][tagid2]= count_tags_to_next_tags[tagid][tagid2] / sum_tags_to_next_
```

Initialize a HMM

```
model = hmm.CategoricalHMM(n_components=len(tags), algorithm='viterbi', random_state=42)
model.startprob_ = mystartprob
model.transmat_ = mytransmat
model.emissionprob_ = myemissionprob
```

As some words may never appear in the training set, we need to transform them into `UNKNOWN` first. Then we split `data_test` into `samples` & `lengths` and send them to HMM.

```
# data_test=pd.read_csv('test.csv')
data_test.loc[~data_test['Word'].isin(words), 'Word'] = 'UNKNOWN'
word_test = list(data_test.Word)
samples = []
for i, val in enumerate(word_test):
    samples.append([word2id[val]])
```

```
lengths = []
count = 0
sentences = list(data_test.sentence)
for i in range(len(sentences)) :
    if (i > 0) and (sentences[i] == sentences[i - 1]):
        count += 1
    elif i > 0:
        lengths.append(count)
        count = 1
    else:
        count = 1


# This code is very slow
pos_predict = model.predict(samples, lengths)
pos_predict
```

```
array([18, 38, 38, ..., 10, 18, 38])
```

```
tags_test = list(data_test.POS)
pos_test = np.zeros((len(tags_test), ), dtype=int)
for i, val in enumerate(tags_test):
    pos_test[i] = tag2id[val]
len(pos_predict), len(pos_test), len(samples), len(word_test)
```

```
(5099, 5110, 5110, 5110)
```

Somehow the output of HMM is in wrong size. Only use the shorter length to check the result.

Somehow the output of HMM is in wrong size. Only use the shorter length to check the result.

```
def reportTest(y_pred, y_test):
    print("The accuracy is {}".format(accuracy_score(y_test, y_pred)))
    print("The precision is {}".format(precision_score(y_test, y_pred, average='weighted')))
    print("The recall is {}".format(recall_score(y_test, y_pred, average='weighted')))
    print("The F1-Score is {}".format(f1_score(y_test, y_pred, average='weighted')))

min_length = min(len(pos_predict), len(pos_test))

reportTest(pos_predict[:min_length], pos_test[:min_length])
```

```
The accuracy is 0.6913120219650912
The precision is 0.7156072125391473
The recall is 0.6913120219650912
The F1-Score is 0.6927895083148519
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefin
  _warn_prf(average, modifier, msg_start, len(result))
```

Performance Metrics	Value
Accuracy	69.79
Precision	72.600
Recall	69.7979
F1-Score	70.2350

Inference from Results:

- We can observe that performance measures of NER dataset against HMM is listed through the accuracy, precision, recall and F-measure, in the table shown above

- The results can be analysed from the ResultsTags.csv which shows the corresponding words, and its related tags as is visible from the csv file attached.