Raaghav_94_NLP5_NER

February 25, 2024

1 Named Entity Recognition

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from tqdm.auto import tqdm
from sklearn.metrics import classification_report, confusion_matrix
import itertools
import re
```

1.1 Hidden Markov Model

[ ]:
```
data = pd.read_csv("ner_dataset.csv", encoding="latin1")
texts_df = data[["Sentence #", "Word", "POS", "Tag"]].groupby(by="Sentence #").
 ↪aggregate(lambda x: " ".join(x))
texts_df.columns = ["text", "pos seq", "tag seq"]

texts_df.head()
```

[ ]:

Sentence #	text	pos seq	tag seq
Sentence: 1	Thousands	NNS	O
Sentence: 10	Iranian	JJ	B-gpe
Sentence: 100	Helicopter	NN	O
Sentence: 1000	They	PRP	O
Sentence: 10000	U.N.	NNP	B-geo

[ ]:
```
texts = texts_df["text"].apply(lambda x: x.split())
tags = texts_df["tag seq"].apply(lambda x: x.split())
X_train, X_test, y_train, y_test = train_test_split(texts.to_numpy(), tags.
 ↪to_numpy(), test_size=0.15, random_state=42)
```

[ ]:
```
zipped_train = []
for pair in np.stack((X_train, y_train), axis=1):
    zipped_train.append(np.stack((pair[1], pair[0]), axis=1))
```

[ ]:
```
states = data["Tag"].unique()
observ = data["Word"].unique()
print(states)
```

```
['O' 'B-geo' 'B-gpe' 'B-per' 'I-geo' 'B-org' 'I-org' 'B-tim' 'B-art'
 'I-art' 'I-per' 'I-gpe' 'I-tim' 'B-nat' 'B-eve' 'I-eve' 'I-nat']
```

[ ]:
```
class HMMTagger():
    def __init__(self, states, observations):
        self.states = states
        self.observations = [*observations, 'UNK']
        self.states_num = len(self.states)
        self.observations_num = len(self.observations)

        self.init_prob = np.zeros(shape=(1, self.states_num))
        self.transition_matrix = np.zeros(shape=(self.states_num, self.
 ↪states_num))
        self.emission_matrix = np.zeros(shape=(self.states_num, self.
 ↪observations_num))

        self.states_to_idx = {state:idx for idx, state in enumerate(self.
 ↪states)}
        self.observations_to_idx = {obs:idx for idx, obs in enumerate(self.
 ↪observations)}

        self.test = np.zeros(shape=(self.states_num, self.states_num))

    def fit(self, train_data):
        self.emission_matrix += 1
        c_final = np.zeros(shape=(1, self.states_num))

        for example in train_data:
            first_state_ind = self.states_to_idx[example[0][0]]
            last_state_ind = self.states_to_idx[example[-1][0]]
            last_obs_ind = self.observations_to_idx[example[-1][1]]

            self.init_prob[0, first_state_ind] += 1
            c_final[0, last_state_ind] += 1

            for ind in range(len(example) - 1):
                curr_state_ind = self.states_to_idx[example[ind][0]]
                curr_obs_ind = self.observations_to_idx[example[ind][1]]
                next_state_ind = self.states_to_idx[example[ind+1][0]]

                self.transition_matrix[next_state_ind, curr_state_ind] += 1
                self.test = self.transition_matrix
                self.emission_matrix[curr_state_ind, curr_obs_ind] += 1
```

```
            self.emission_matrix[last_state_ind, last_obs_ind] += 1
        self.init_prob = self.init_prob / np.sum(self.init_prob)
        self.transition_matrix = (self.transition_matrix / (np.sum(self.
↪transition_matrix, axis=0))).T
        self.emission_matrix = self.emission_matrix / np.sum(self.
↪emission_matrix,axis=1).reshape(-1, 1)

    def __viterbi(self, obs_sequence_indices):
        temp = [0] * self.states_num
        delta = [temp[:]]

        for i in range(self.states_num):
            delta[0][i] = self.init_prob[0, i] * self.emission_matrix[i,␣
↪obs_sequence_indices[0]]

        phi = [temp[:]]

        for obs in obs_sequence_indices[1:]:
            delta_t = temp[:]
            phi_t = temp[:]

            for j in range(self.states_num):
                tdelta = temp[:]
                tphimax = -1.0
                for i in range(self.states_num):
                    tphi_temp = delta[-1][i] * self.transition_matrix[i, j]
                    if (tphi_temp > tphimax):
                        tphimax = tphi_temp
                        phi_t[j] = i
                    tdelta[i] = tphi_temp * self.emission_matrix[j, obs]
                delta_t[j] = max(tdelta)
            delta.append(delta_t)
            phi.append(phi_t)

        tmax = -1.0
        for i in range(self.states_num):
            if(delta[-1][i] > tmax):
                tmax = delta[-1][i]
                state_seq = [i]

        phi.reverse()
        for tphi in phi[:-1]:
            state_seq.append(tphi[state_seq[-1]])
        return reversed(state_seq)

    def predict(self, obser_seq):
```

```
        result = []

        for seq in tqdm(obser_seq):
            obser_inds_seq = [self.observations_to_idx[token] for token in seq]
            state_ind_seq = list(self.__viterbi(obser_inds_seq))
            state_seq = [self.states[state_ind] for state_ind in state_ind_seq]
            result.append(state_seq)
        return result
```

[ ]:
```
hmm = HMMTagger(states, observ)
hmm.fit(zipped_train)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_26152\1647509052.py:40: RuntimeWarning:
invalid value encountered in divide
  self.transition_matrix = (self.transition_matrix /
(np.sum(self.transition_matrix, axis=0))).T
```

[ ]:
```
y_pred = hmm.predict(X_test)
```

```
  0%|          | 0/7194 [00:00<?, ?it/s]
```

[ ]:
```
y_test_flat = list(itertools.chain.from_iterable(y_test))
y_pred_flat = list(itertools.chain.from_iterable(y_pred))
```

[ ]:
```
print(f"Sequence: {[i[0] for i in X_test[-10:]]}")
print(f"Original Tags: {[i[0] for i in y_test[-10:]]}")
print(f"Predicted Tags: {[i[0] for i in y_pred[-10:]]}")
```

```
Sequence: ['Indian', 'The', 'Since', 'Iran', 'Several', 'Mr.', 'For', 'Jupiter',
'North', 'Insurgents']
Original Tags: ['B-gpe', 'O', 'O', 'B-geo', 'O', 'B-per', 'O', 'B-per', 'B-geo',
'O']
Predicted Tags: ['B-gpe', 'O', 'O', 'B-geo', 'O', 'B-per', 'O', 'O', 'B-geo',
'O']
```

[ ]:
```
print(classification_report(y_test_flat, y_pred_flat, labels=states,␣
↪zero_division=False))
```

```
              precision    recall  f1-score   support

           O       0.89      1.00      0.94      5136
       B-geo       0.78      0.75      0.76       522
       B-gpe       0.97      0.82      0.89       428
       B-per       0.97      0.58      0.72       595
       I-geo       0.00      0.00      0.00         0
       B-org       0.94      0.31      0.47       419
       I-org       0.00      0.00      0.00         0
       B-tim       1.00      0.60      0.75        85
       B-art       0.00      0.00      0.00         6
```

I-art	0.00	0.00	0.00	0
I-per	0.00	0.00	0.00	0
I-gpe	0.00	0.00	0.00	0
I-tim	0.00	0.00	0.00	0
B-nat	0.00	0.00	0.00	1
B-eve	0.00	0.00	0.00	2
I-eve	0.00	0.00	0.00	0
I-nat	0.00	0.00	0.00	0
micro avg	0.89	0.89	0.89	7194
macro avg	0.33	0.24	0.27	7194
weighted avg	0.89	0.89	0.87	7194