[redacted]

March 1, 2024

hmmlearn

[ ]:
```
from IPython.display import clear_output
```

[ ]:
```
!pip install sklearn-crfsuite
!pip install tabulate
!pip install nervaluate
clear_output()
```

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from tabulate import tabulate
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import itertools
from nervaluate import Evaluator
from tqdm.auto import tqdm
import re

tqdm.pandas()
```

[ ]:
```
tag_name_pattern = re.compile(r"(B|I)-")

def map_seq(seq, dictionary):
    return [dictionary[elem] for elem in seq]

def plot_confusion_matricies(y_true, y_pred, class_labels):
    cm = confusion_matrix(y_true, y_pred)

    precision_cm = cm / np.sum(cm, axis=0)
    recall_cm = cm / np.sum(cm, axis=1)

    precision_cm = pd.DataFrame(np.flip(precision_cm, axis=0),␣
 ↪columns=class_labels)
```

```
    recall_cm = pd.DataFrame(np.flip(recall_cm, axis=0), columns=class_labels)

    precision_cm = precision_cm.set_index(np.flip(class_labels)).fillna(0)
    recall_cm = recall_cm.set_index(np.flip(class_labels)).fillna(0)

    _, axs = plt.subplots(ncols=2, figsize=(22,8))
    axs[0].set_title("Precision per class")
    axs[1].set_title("Recall per class")

    sns.heatmap(precision_cm, annot=True, fmt=".2f", robust=True, cbar=False,␣
 ↪ax=axs[0]);
    sns.heatmap(recall_cm, annot=True, fmt=".2f", robust=True, cbar=False,␣
 ↪ax=axs[1]);

def plot_eval_result(results):
    keys = list(results.keys())
    #_, axs = plt.subplots(nrows=len(keys), ncols=2, figsize=(20,12))
    normalized_stats = ["f1", "recall", "precision"]
    absolute_stats = ["correct", "incorrect", "partial", "missed", "spurious",␣
 ↪"possible", "actual"]
    for ind, key in enumerate(keys):
        fig, axs = plt.subplots(ncols=2, figsize=(20,3))
        fig.suptitle(key)
        stats = results[key]
        temp_df = pd.DataFrame(stats, index=[0], columns=stats.keys())
        #temp_df[absolute_stats] = temp_df[absolute_stats] /␣
 ↪temp_df[absolute_stats].sum(axis=1).iloc[0]
        sns.barplot(data=temp_df[normalized_stats], palette="crest", ax=axs[0])
        sns.barplot(data=temp_df[absolute_stats], palette="crest", ax=axs[1])
```

[ ]:
```
data1 = pd.read_csv("ner.csv", encoding = "ISO-8859-1", index_col=0,␣
 ↪error_bad_lines=False)
data2 = pd.read_csv("ner_dataset.csv", encoding="latin1")
```

```
<ipython-input-5-baa0f446c6a5>:1: FutureWarning: The error_bad_lines argument
has been deprecated and will be removed in a future version. Use on_bad_lines in
the future.


  data1 = pd.read_csv("ner.csv", encoding = "ISO-8859-1", index_col=0,
error_bad_lines=False)
Skipping line 281837: expected 25 fields, saw 34
```

[ ]:
```
chrs = ["\x85", "\x94"]
lengths = data2["Word"].apply(lambda x: len(x.split()))
data2 = data2[lengths == 1]
```

```
data2 = data2[data2["Word"].apply(lambda x: x not in chrs)]
data2 = data2.reset_index(drop=True)



first_words_inds = data2[data2["Sentence #"].notna()].index.to_list()
for i in tqdm(range(len(first_words_inds)-1)):
    data2["Sentence #"].loc[range(first_words_inds[i], first_words_inds[i+1])]␣
 ↪= i
data2["Sentence #"].loc[range(first_words_inds[-1],data2["Sentence #"].
 ↪shape[0])] = len(first_words_inds) - 1
data2 = data2.astype({"Sentence #": int, "Word": "string", "POS": "string",␣
 ↪"Tag": "string"})



texts_df = data2[["Sentence #", "Word", "POS", "Tag"]].groupby(by="Sentence #").
 ↪aggregate(lambda x: " ".join(x))
texts_df.columns = ["text", "pos seq", "tag seq"]
```

```
0%|          | 0/47958 [00:00<?, ?it/s]
```

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn_crfsuite.metrics import flat_classification_report
```

[ ]:
```
class HMMTaggerTemplate():
    def __init__(self, states, observations):
        # add 'Unk' to handle unkown tokens
        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)}


    def fit(self, train_data):
```

```
        self.emission_matrix += 1 # smoothing
        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.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)
        #return self


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

        delta = [tmp[:]]  # Compute initial state probabilities
        for i in range(self.states_num):
          delta[0][i] = self.init_prob[0,i] * self.emission_matrix[i,␣
↪obs_sequence_indices[0]]

        phi = [tmp[:]]

        for obs in obs_sequence_indices[1:]:  # For all observations except the␣
↪inital one
          delta_t = tmp[:]
          phi_t = tmp[:]
          for j in range(self.states_num):   # Following formula 33 in␣
↪Rabiner'89
            tdelta = tmp[:]
            tphimax = -1.0
```

```
          for i in range(self.states_num):
            tphi_tmp = delta[-1][i] * self.transition_matrix[i,j]
            if (tphi_tmp > tphimax):
              tphimax = tphi_tmp
              phi_t[j] = i
            tdelta[i] = tphi_tmp * 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]  # Last state with maximum probability

      phi.reverse()  # Because we start from the end of the sequence
      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
```

[ ]:
```
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)
```

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

[ ]:
```
states = data2["Tag"].unique().to_numpy()
observ = data2["Word"].unique().to_numpy()
hmm = HMMTaggerTemplate(states, observ)
```

[ ]:
```
hmm.fit(ziped_train)
```

[ ]:
```
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))
```

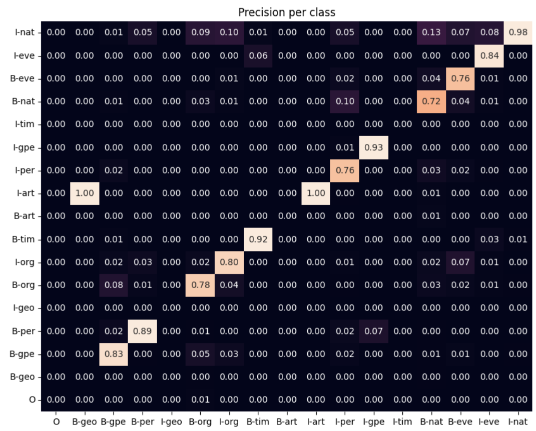
[ ]:
```
print(classification_report(y_test_flat, y_pred_flat))
```

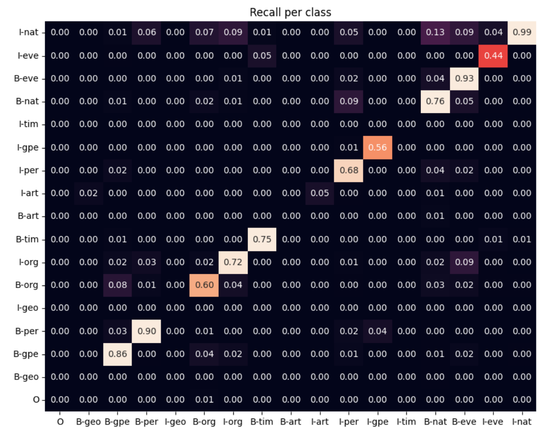
```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

       B-art       0.00      0.00      0.00        63
       B-eve       0.00      0.00      0.00        48
       B-geo       0.83      0.86      0.85      5777
       B-gpe       0.89      0.90      0.90      2349
       B-nat       0.00      0.00      0.00        40
       B-org       0.78      0.60      0.68      2976
       B-per       0.80      0.72      0.76      2562
       B-tim       0.92      0.75      0.83      3065
       I-art       0.00      0.00      0.00        45
       I-eve       1.00      0.05      0.09        42
       I-geo       0.76      0.68      0.72      1102
       I-gpe       0.93      0.56      0.70        25
       I-nat       0.00      0.00      0.00         9
       I-org       0.72      0.76      0.74      2484
       I-per       0.76      0.93      0.83      2589
       I-tim       0.84      0.44      0.58       974
           O       0.98      0.99      0.99    132448

    accuracy                           0.96    156598
   macro avg       0.60      0.49      0.51    156598
weighted avg       0.95      0.96      0.95    156598

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:
```python
plot_confusion_matricies(y_test_flat, y_pred_flat, states)
```

```
<ipython-input-4-f37e1a220c3b>:9: RuntimeWarning: invalid value encountered in
divide
  precision_cm = cm / np.sum(cm, axis=0)
```





[ ]:
```python
ent_types = list(set([tag_name_pattern.sub("", tag) for tag in states]))
```

[ ]:
```python
evaluator = Evaluator(y_test, y_pred, tags=ent_types, loader="list")
```

[ ]:
```python
results, results_by_tag = evaluator.evaluate()
```