Task B: Named Entity Recognition with CRF on Hindi Dataset. (Total: 60 Points out of 100)

In this part, you will use a CRF to implement a named entity recognition tagger. We have implemented a CRF for you in crf.py along with some functions to build, and pad feature vectors. Your job is to add more features to learn a better tagger. Then you need to complete the traiing loop implementation.

Finally, you can checkout the code in `crf.py` -- reflect on CRFs and span tagging, and answer the discussion questions.

We will use the Hindi NER dataset at: https://github.com/cfiltnlp/HiNER

The first step would be to download the repo into your current folder of the Notebook

In [51]:

```
!git clone https://github.com/cfiltnlp/HiNER.git
```

```
fatal: destination path 'HiNER' already exists and is not an empty directory.
```

In [52]:

```
pip install -r requirements.txt
```

```
Collecting en_core_web_sm
  Using cached en_core_web_sm-3.4.0-py3-none-any.whl
Requirement already satisfied: torch in /opt/anaconda3/lib/python3.9/site-packages (from
-r requirements.txt (line 1)) (1.12.1)
Requirement already satisfied: spacy<4.0.0,>=3.0.0 in /opt/anaconda3/lib/python3.9/site-p
ackages (from -r requirements.txt (line 2)) (3.4.1)
Requirement already satisfied: spacytextblob in /opt/anaconda3/lib/python3.9/site-package
s (from -r requirements.txt (line 4)) (4.0.0)
Requirement already satisfied: sklearn in /opt/anaconda3/lib/python3.9/site-packages (fro
m -r requirements.txt (line 5)) (0.0)
Requirement already satisfied: tqdm in /opt/anaconda3/lib/python3.9/site-packages (from -
r requirements.txt (line 6)) (4.64.0)
Requirement already satisfied: pytorch-crf==0.7.2 in /opt/anaconda3/lib/python3.9/site-pa
ckages (from -r requirements.txt (line 7)) (0.7.2)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.9/site-packages (from
-r requirements.txt (line 8)) (1.21.5)
Requirement already satisfied: nltk in /opt/anaconda3/lib/python3.9/site-packages (from -
r requirements.txt (line 9)) (3.7)
Requirement already satisfied: typing-extensions in /opt/anaconda3/lib/python3.9/site-pac
kages (from torch->-r requirements.txt (line 1)) (4.1.1)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /opt/anaconda3/lib/python3.9/site-p
ackages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (2.4.4)
Requirement already satisfied: typer<0.5.0,>=0.3.0 in /opt/anaconda3/lib/python3.9/site-p
ackages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (0.4.2)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /opt/anaconda3/lib/python3.
9/site-packages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (1.0.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.10.0,>=1.7.4 in /opt/anaconda3/li
b/python3.9/site-packages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (1.9.2
)
Requirement already satisfied: pathy>=0.3.5 in /opt/anaconda3/lib/python3.9/site-packages
(from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (0.6.2)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /opt/anaconda3/lib/python3.9/si
te-packages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (3.3.0)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.9 in /opt/anaconda3/lib/python3.9
/site-packages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (3.0.10)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /opt/anaconda3/lib/python3.9/si
te-packages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (2.0.8)
Requirement already satisfied: jinja2 in /opt/anaconda3/lib/python3.9/site-packages (from
spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (2.11.3)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/python3.9/site-packages (
from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (61.2.0)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /opt/anaconda3/lib/python3.9/
```

```
site-packages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (1.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /opt/anaconda3/lib/python3.9/site
-packages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (3.0.7)
Requirement already satisfied: wasabi<1.1.0,>=0.9.1 in /opt/anaconda3/lib/python3.9/site-
packages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (0.10.1)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /opt/anaconda3/lib/python3.9/si
te-packages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (2.27.1)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /opt/anaconda3/lib/python3.9/site-p
ackages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (2.0.6)
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/lib/python3.9/site-packa
ges (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (21.3)
Requirement already satisfied: thinc<8.2.0,>=8.1.0 in /opt/anaconda3/lib/python3.9/site-p
ackages (from spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (8.1.3)
Requirement already satisfied: textblob<0.16.0,>=0.15.3 in /opt/anaconda3/lib/python3.9/s
ite-packages (from spacytextblob->-r requirements.txt (line 4)) (0.15.3)
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.9/site-packages
(from sklearn->-r requirements.txt (line 5)) (1.0.2)
Requirement already satisfied: regex>=2021.8.3 in /opt/anaconda3/lib/python3.9/site-packa
ges (from nltk->-r requirements.txt (line 9)) (2022.3.15)
Requirement already satisfied: joblib in /opt/anaconda3/lib/python3.9/site-packages (from
nltk->-r requirements.txt (line 9)) (1.1.0)
Requirement already satisfied: click in /opt/anaconda3/lib/python3.9/site-packages (from
nltk->-r requirements.txt (line 9)) (8.0.4)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/anaconda3/lib/python3.9/s
ite-packages (from packaging>=20.0->spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (3
.0.4)
Requirement already satisfied: smart-open<6.0.0,>=5.2.1 in /opt/anaconda3/lib/python3.9/s
ite-packages (from pathy>=0.3.5->spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (5.2.
1)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.9/site-packages
(from requests<3.0.0,>=2.13.0->spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/anaconda3/lib/python3.9/site
-packages (from requests<3.0.0,>=2.13.0->spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2
)) (1.26.9)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.9/site-pa
ckages (from requests<3.0.0,>=2.13.0->spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2))
(2021.10.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/anaconda3/lib/python3.9/
site-packages (from requests<3.0.0,>=2.13.0->spacy<4.0.0,>=3.0.0->-r requirements.txt (li
ne 2)) (2.0.4)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /opt/anaconda3/lib/python3.9/site-pa
ckages (from thinc<8.2.0,>=8.1.0->spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (0.7
.8)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /opt/anaconda3/lib/python3.9/s
ite-packages (from thinc<8.2.0,>=8.1.0->spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)
) (0.0.3)
Requirement already satisfied: MarkupSafe>=0.23 in /opt/anaconda3/lib/python3.9/site-pack
ages (from jinja2->spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (2.0.1)
Requirement already satisfied: scipy>=1.1.0 in /opt/anaconda3/lib/python3.9/site-packages
(from scikit-learn->sklearn->-r requirements.txt (line 5)) (1.7.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.9/site-
packages (from scikit-learn->sklearn->-r requirements.txt (line 5)) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [53]:

```python
import torch
```

In [54]:

```python
# This is so that you don't have to restart the kernel everytime you edit hmm.py

%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

First we load the data and labels. Feel free to explore them below.

Since we have provided a separate train and dev split, there is not need to split the data yourself.

Since we have provided a seperate train and dev split, there is not need to split the data yourself.

In [55]:

```
from crf import load_data, make_labels2i

train_filepath = "./HiNER/data/collapsed/train.conll"
dev_filepath = "./HiNER/data/collapsed/validation.conll"
labels_filepath = "./HiNER/data/collapsed/label_list"

train_sents, train_tag_sents = load_data(train_filepath)
dev_sents, dev_tag_sents = load_data(dev_filepath)
labels2i = make_labels2i(labels_filepath)

print("train sample", train_sents[2], train_tag_sents[2])
print()
print("labels2i", labels2i)
```

```
train sample ['रामनगर', 'इगलास', ',', 'अलीगढ़', ',', 'उत्तर', 'प्रदेश', 'स्थित', 'एक', 'गाँव', 'है।']
['B-LOCATION', 'B-LOCATION', 'O', 'B-LOCATION', 'O', 'B-LOCATION', 'I-LOCATION', 'O', 'O'
, 'O', 'O']

labels2i {'<PAD>': 0, 'B-LOCATION': 1, 'B-ORGANIZATION': 2, 'B-PERSON': 3, 'I-LOCATION':
4, 'I-ORGANIZATION': 5, 'I-PERSON': 6, 'O': 7}
```

Feature engineering. (Total 30 points)

Notice that we are learning features to some extent: we start with one unique feature for every possible word. You can refer to figure 8.15 in the textbook for some good baseline features to try.

identity of w_i, identity of neighboring words
embeddings for w_i, embeddings for neighboring words
part of speech of w_i, part of speech of neighboring words
presence of w_i in a **gazetteer**
w_i contains a particular prefix (from all prefixes of length ≤ 4)
w_i contains a particular suffix (from all suffixes of length ≤ 4)
word shape of w_i, word shape of neighboring words
short word shape of w_i, short word shape of neighboring words
gazetteer features

Figure 8.15 Typical features for a feature-based NER system.

There is no need to worry about embeddings now.

Hindi POS Tagger (10 Points)

Although this step is not entirely necessary, if you want to use the HMM pos tagger to extract feature corresponding to the pos of the word in the sentence, we need to add this into the pipeline.

You get 10 points if you use your pos_tagger to featurize the sentences

In [56]:

```
from hmm import get_hindi_dataset
import pickle
from typing import List

words, tags, observation_dict, state_dict, all_observation_ids, all_state_ids = get_hindi
_dataset()

# we need to add the id for unknown word (<unk>) in our observations vocab
UNK_TOKEN = '<unk>'
```

```python
observation_dict[UNK_TOKEN] = len(observation_dict)
print("id of the <unk> token:", observation_dict[UNK_TOKEN])

## load the pos tagger
# pos_tagger = pickle.load(open('hindi_pos_tagger.pkl', 'rb'))

def encode(sentences: List[List[str]]) -> List[List[int]]:
    """
    Using the observation_dict, convert the tokens to ids
    unknown words take the id for UNK_TOKEN
    """
    return [
        [observation_dict[t] if t in observation_dict else observation_dict[UNK_TOKEN]
            for t in sentence]
        for sentence in sentences]

def get_pos(pos_tagger, sentences) -> List[List[str]]:
    """
    The the pos tag for input sentences
    """
    sentence_ids = encode(sentences)
    decoded_pos_ids = pos_tagger.decode(sentence_ids)
    return [
        [tags[i] for i in d_ids]
        for d_ids in decoded_pos_ids
    ]
```

```
id of the <unk> token: 2186
```

Feature Engineering Functions (20 Points)

In [57]:

```python
from typing import List

# TODO: Update this function to add more features
#       You can check crf.py for how they are encoded, if interested.
def make_features(text: List[str]) -> List[List[int]]:
    """Turn a text into a feature vector.

    Args:
        text (List[str]): List of tokens.

    Returns:
        List[List[int]]: List of feature Lists.
    """
    feature_lists = []

    prev_word = '<S>'

    for i, token in enumerate(text):
        feats = []
        # We add a feature for each unigram.
        feats.append(f"word={token}")
        # TODO: Add more features here
        feats.append(f"prev_word={prev_word}")

        # We append each feature to a List for the token.
        feature_lists.append(feats)
        prev_word = token

        if i != len(text) - 1:
          next_word = text[i+1]
          feats.append(f"next_word={next_word}")
        else:
          feats.append(f"next_word=</S>")

    sent_tags = get_pos(pos_tagger, [text])[0]

    for i,tag in enumerate(sent_tags):
```

```
        feature_lists[i].append(f"pos={tag}")

    return feature_lists
```

In [58]:

```
def featurize(sents: List[List[str]]) -> List[List[List[str]]]:
    """Turn the sentences into feature Lists.

    Eg.: For an input of 1 sentence:
         [[['I','am','a','student','at','CU','Boulder']]]
        Return list of features for every token for every sentence like:
        [[
         ['word=I',   'prev_word=<S>','pos=PRON',...],
         ['word=an',  'prev_word=I'   , 'pos=VB' ,...],
         [...]
        ]]

    Args:
        sents (List[List[str]]): A List of sentences, which are Lists of tokens.

    Returns:
        List[List[List[str]]]: A List of sentences, which are Lists of feature Lists
    """
    feats = []
    print(len(sents))
    counter = 0
    for sent in sents:
        # Gets a List of Lists of feature strings
        feats.append(make_features(sent))
        counter+=1
        if counter%1000 == 0:
          print(counter)
    return feats
```

Finish the training loop. (10 Points)

See the previous homework, and fill in the missing parts of the training loop.

In [59]:

```
from crf import f1_score, predict, PAD_SYMBOL
import random
from tqdm.autonotebook import tqdm

# TODO: Implement the training loop
# HINT: Build upon what we gave you for HW2.
# See cell below for how we call this training loop.


def logistic_loss(prediction: torch.Tensor, label: torch.Tensor) -> torch.Tensor:
    log_loss = -(label * torch.log(prediction)) - ( (1 - label) * (torch.log(1 - predict
ion)) )
    return log_loss.mean()

def training_loop(
    num_epochs,
    batch_size,
    train_features,
    train_labels,
    dev_features,
    dev_labels,
    optimizer,
    model,
    labels2i,
    pad_feature_idx
):

    # TODO: Zip the train features and labels
    # TODO: Randomize them, while keeping them paired.
```

```python
    # TODO: Build batches

    samples = list(zip(train_features, train_labels))

    random.shuffle(samples)

    batches = []

    for i in range(0, len(samples), batch_size):
        batches.append(samples[i:i+batch_size])


    for i in range(num_epochs):
        losses = []
        for batch in tqdm(batches):
            # Here we get the features and labels, pad them,
            # and build a mask so that our model ignores PADs
            # We have abstracted the padding from you for simplicity,
            # but please reach out if you'd like learn more.
            features, labels = zip(*batch)
            features = pad_features(features, pad_feature_idx)
            features = torch.stack(features)

            # Pad the label sequences to all be the same size, so we
            # can form a proper matrix.
            labels = pad_labels(labels, labels2i[PAD_SYMBOL])
            labels = torch.stack(labels)
            mask = (labels != labels2i[PAD_SYMBOL])

            # TODO: Empty the dynamic computation graph
            optimizer.zero_grad()

            # TODO: Run the model. Since we use the pytorch-crf model,
            # our forward function returns the positive log-likelihood already.
            # We want the negative log-likelihood. See crf.py forward method in NERTagger

            loss = -1*model.forward(features,labels,mask)
            # TODO: Backpropogate the loss through our model
            loss.backward()

            # TODO: Update our coefficients in the direction of the gradient.
            optimizer.step()
            # TODO: Store the losses for logging
            losses.append(loss.item())

        # TODO: Log the average Loss for the epoch
        print(f"epoch {i}, loss: {sum(losses)/len(losses)}")
        # TODO: make dev predictions with the `predict()` function
        dev_predictions = predict(model, dev_features)
        # TODO: Compute F1 score on the dev set and log it.
        dev_f1 = f1_score(dev_predictions, dev_labels,labels2i['O'])
        print(f"F1 Score: {dev_f1}")

    # Return the trained model
    return model
```

Run the training loop (10 Points)

We have provided the code here, but you can try different hyperparameters and test multiple runs.

In [62]:

```python
from crf import build_features_set
from crf import make_features_dict
from crf import encode_features, encode_labels
from crf import NERTagger
from crf import pad_features,pad_labels

import numpy as np
```

```python
# Build the model and featurized data
# train_features = featurize(train_sents)
# dev_features = featurize(dev_sents)

# np.save('train_features.npy', train_features)
# np.save('dev_features.npy', dev_features)
# print("saved features")

train_features = np.load('train_features.npy',allow_pickle=True)
dev_features = np.load('dev_features.npy',allow_pickle=True)

# Get the full inventory of possible features
all_features = build_features_set(train_features)
# Hash all features to a unique int.
features_dict = make_features_dict(all_features)


# Initialize the model.
model = NERTagger(len(features_dict), len(labels2i))

encoded_train_features = encode_features(train_features, features_dict)
encoded_dev_features = encode_features(dev_features, features_dict)
encoded_train_labels = encode_labels(train_tag_sents, labels2i)
encoded_dev_labels = encode_labels(dev_tag_sents, labels2i)


# TODO: Play with hyperparameters here.
num_epochs = 30
batch_size = 256
LR=0.35
optimizer = torch.optim.SGD(model.parameters(), LR)

model = training_loop(
    num_epochs,
    batch_size,
    encoded_train_features,
    encoded_train_labels,
    encoded_dev_features,
    encoded_dev_labels,
    optimizer,
    model,
    labels2i,
    features_dict[PAD_SYMBOL]
)
```

```
Building features set!

100%|██████████████████████████████| 75827/75827 [00:00<00:00, 304340.39it/s]

Found 76547 features


epoch 0, loss: 9.558348005468195
F1 Score: tensor([0.3739])


epoch 1, loss: 6.9789704923276545
F1 Score: tensor([0.4579])


epoch 2, loss: 6.154564615050551
F1 Score: tensor([0.4937])


epoch 3, loss: 5.65030921268142
F1 Score: tensor([0.5353])


epoch 4, loss: 5.299533598350756
F1 Score: tensor([0.5635])


epoch 5, loss: 5.034214173904573
F1 Score: tensor([0.5821])
```

```
epoch 6, loss: 4.821945697715796
F1 Score: tensor([0.5998])

epoch 7, loss: 4.645695863749443
F1 Score: tensor([0.6147])

epoch 8, loss: 4.495520088407728
F1 Score: tensor([0.6287])

epoch 9, loss: 4.365076382152159
F1 Score: tensor([0.6393])

epoch 10, loss: 4.250056992476235
F1 Score: tensor([0.6513])

epoch 11, loss: 4.1474162504729195
F1 Score: tensor([0.6580])

epoch 12, loss: 4.054929321462458
F1 Score: tensor([0.6660])

epoch 13, loss: 3.970928267597751
F1 Score: tensor([0.6707])

epoch 14, loss: 3.8941276900294652
F1 Score: tensor([0.6781])

epoch 15, loss: 3.8235159634741067
F1 Score: tensor([0.6858])

epoch 16, loss: 3.758279425527913
F1 Score: tensor([0.6911])

epoch 17, loss: 3.6977534671423813
F1 Score: tensor([0.6953])

epoch 18, loss: 3.6413862376100687
F1 Score: tensor([0.6999])

epoch 19, loss: 3.588713603388982
F1 Score: tensor([0.7049])

epoch 20, loss: 3.5393417282939357
F1 Score: tensor([0.7094])

epoch 21, loss: 3.4929305391279537
F1 Score: tensor([0.7156])

epoch 22, loss: 3.449188150540747
F1 Score: tensor([0.7191])

epoch 23, loss: 3.4078581854951904
F1 Score: tensor([0.7211])

epoch 24, loss: 3.368718385696411
F1 Score: tensor([0.7258])

epoch 25, loss: 3.3315720510001134
F1 Score: tensor([0.7293])

epoch 26, loss: 3.2962460293111575
F1 Score: tensor([0.7323])
```

```
epoch 27, loss: 3.2625880249421604
F1 Score: tensor([0.7358])

epoch 28, loss: 3.2304616094839695
F1 Score: tensor([0.7382])

epoch 29, loss: 3.1997464835041702
F1 Score: tensor([0.7406])
```

Quiz (10 Points)

1. Look at the `NERTagger` class in crf.py

```
a) What does the CRF add to our model that makes it different from the sentiment cl
assifier?

b) Why is this helpful for NER?
```

Answers: a.) In sentiment classifier assignment, we used the Bag of Words(BOW) and the TF-IDF for the classification where we just consider the occurence of the words, but here we leverage Conditional Random Fields which use sequential modelling, here we emphasise on the sequence of the words occuring in the text which helps in named entity recognition and part of speech identification. We have the advantage of extracting POS tags, word distances, modelling linguistic features such as words and characters, non linguistic features such as spaces, punctuations. Here, we make the assumption that the features we use are interdependent, and we learn the pattern while taking future observations into account. b.) CRF not only exracts tokens but it also takes non-linguistic, POS tags and few other things as features. A CRF is a log-linear model that, given the full input (word) sequence X, gives a probability to an entire output (tag) sequence Y out of all possible sequences Y'. Specific features are extracted for each token in a sequence using the sentential model for feature extraction. The linear chain CRF, the version of the CRF most commonly used for language processing, and the one whose conditioning closely matches to the HMM.

2. Why computing F1 here is not straightforward?

Hint: Refer to the class in which Jim went over the evaluation metrics for NER

In [ ]:

```
The F1 measure is the harmonic mean of recall and precision where recall is the ratio of
the correctly labeled
responses to the total that should have been labelled and precision is the ratio of the n
umber of correctly labeled
responses to the total labeled.
We use the paired bootstrap test to know if there is any significant difference between
two F1 scores of 2 MT system.
In named entity recognition the entity itslef is the unit of response not the word.
Thus the example taken in the class the two entities Jane Villanueva and United Airline H
olding and the non-entoty
discussed.
Named entity tagging has a segmentation component which is not present in tasks like tex
t classification or parts of
speech tagging causes some problem with the evaluation.
For example when a system takes Kennedy as a person but not Johannes Kennady(might have
considered the airport name)
will bring up two errors,a false positive O and a false negative for I-PER, also we are
using entities as the unit of
response and the words as unit of training which implies that there is a mismatch betwee
n the training and
test conditions.
```

Part A: Parts of Speech Tagging using Hidden Markov Model and Viterbi Algorithm on Hindi Dataset (Total: 40 Points out of 100)

```
pip install -r requirements.txt

Collecting en_core_web_sm
  Using cached en_core_web_sm-3.4.0-py3-none-any.whl
Requirement already satisfied: torch in
/opt/anaconda3/lib/python3.9/site-packages (from -r requirements.txt
(line 1)) (1.12.1)
Requirement already satisfied: spacy<4.0.0,>=3.0.0 in
/opt/anaconda3/lib/python3.9/site-packages (from -r requirements.txt
(line 2)) (3.4.1)
Requirement already satisfied: spacytextblob in
/opt/anaconda3/lib/python3.9/site-packages (from -r requirements.txt
(line 4)) (4.0.0)
Requirement already satisfied: sklearn in
/opt/anaconda3/lib/python3.9/site-packages (from -r requirements.txt
(line 5)) (0.0)
Requirement already satisfied: tqdm in
/opt/anaconda3/lib/python3.9/site-packages (from -r requirements.txt
(line 6)) (4.64.0)
Requirement already satisfied: pytorch-crf==0.7.2 in
/opt/anaconda3/lib/python3.9/site-packages (from -r requirements.txt
(line 7)) (0.7.2)
Requirement already satisfied: numpy in
/opt/anaconda3/lib/python3.9/site-packages (from -r requirements.txt
(line 8)) (1.21.5)
Requirement already satisfied: nltk in
/opt/anaconda3/lib/python3.9/site-packages (from -r requirements.txt
(line 9)) (3.7)
Requirement already satisfied: typing-extensions in
/opt/anaconda3/lib/python3.9/site-packages (from torch->-r
requirements.txt (line 1)) (4.1.1)
Requirement already satisfied: typer<0.5.0,>=0.3.0 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (0.4.2)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (3.3.0)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (2.0.6)
Requirement already satisfied: jinja2 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (2.11.3)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
```

```
>-r requirements.txt (line 2)) (2.27.1)
Requirement already satisfied: setuptools in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (61.2.0)
Requirement already satisfied: packaging>=20.0 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (21.3)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.9 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (3.0.10)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (1.0.3)
Requirement already satisfied: pathy>=0.3.5 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (0.6.2)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (3.0.7)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (1.0.8)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (2.0.8)
Requirement already satisfied: wasabi<1.1.0,>=0.9.1 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (0.10.1)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.10.0,>=1.7.4
in /opt/anaconda3/lib/python3.9/site-packages (from
spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (1.9.2)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (2.4.4)
Requirement already satisfied: thinc<8.2.0,>=8.1.0 in
/opt/anaconda3/lib/python3.9/site-packages (from spacy<4.0.0,>=3.0.0-
>-r requirements.txt (line 2)) (8.1.3)
Requirement already satisfied: textblob<0.16.0,>=0.15.3 in
/opt/anaconda3/lib/python3.9/site-packages (from spacytextblob->-r
requirements.txt (line 4)) (0.15.3)
Requirement already satisfied: scikit-learn in
/opt/anaconda3/lib/python3.9/site-packages (from sklearn->-r
requirements.txt (line 5)) (1.0.2)
Requirement already satisfied: click in
/opt/anaconda3/lib/python3.9/site-packages (from nltk->-r
requirements.txt (line 9)) (8.0.4)
Requirement already satisfied: joblib in
/opt/anaconda3/lib/python3.9/site-packages (from nltk->-r
requirements.txt (line 9)) (1.1.0)
Requirement already satisfied: regex>=2021.8.3 in
```

```
/opt/anaconda3/lib/python3.9/site-packages (from nltk->-r
requirements.txt (line 9)) (2022.3.15)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/opt/anaconda3/lib/python3.9/site-packages (from packaging>=20.0-
>spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (3.0.4)
Requirement already satisfied: smart-open<6.0.0,>=5.2.1 in
/opt/anaconda3/lib/python3.9/site-packages (from pathy>=0.3.5-
>spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (5.2.1)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/anaconda3/lib/python3.9/site-packages (from
requests<3.0.0,>=2.13.0->spacy<4.0.0,>=3.0.0->-r requirements.txt
(line 2)) (2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/opt/anaconda3/lib/python3.9/site-packages (from
requests<3.0.0,>=2.13.0->spacy<4.0.0,>=3.0.0->-r requirements.txt
(line 2)) (1.26.9)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/opt/anaconda3/lib/python3.9/site-packages (from
requests<3.0.0,>=2.13.0->spacy<4.0.0,>=3.0.0->-r requirements.txt
(line 2)) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
/opt/anaconda3/lib/python3.9/site-packages (from
requests<3.0.0,>=2.13.0->spacy<4.0.0,>=3.0.0->-r requirements.txt
(line 2)) (3.3)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in
/opt/anaconda3/lib/python3.9/site-packages (from thinc<8.2.0,>=8.1.0-
>spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (0.0.3)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in
/opt/anaconda3/lib/python3.9/site-packages (from thinc<8.2.0,>=8.1.0-
>spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (0.7.8)
Requirement already satisfied: MarkupSafe>=0.23 in
/opt/anaconda3/lib/python3.9/site-packages (from jinja2-
>spacy<4.0.0,>=3.0.0->-r requirements.txt (line 2)) (2.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/anaconda3/lib/python3.9/site-packages (from scikit-learn-
>sklearn->-r requirements.txt (line 5)) (2.2.0)
Requirement already satisfied: scipy>=1.1.0 in
/opt/anaconda3/lib/python3.9/site-packages (from scikit-learn-
>sklearn->-r requirements.txt (line 5)) (1.7.3)
Note: you may need to restart the kernel to use updated packages.
```

For this assignment, we will implement the Viterbi Decoder using the Forward Algorithm of Hidden Markov Model as explained in class.

Then, we will create an HMM-based PoS Tagger for Hindi language using the annotated Tagset in nltk.indian

You need to first implement the missing code in hmm.py, then run the cells here to get the points

```
from tqdm.autonotebook import tqdm
```

```python
# This is so that you don't have to restart the kernel everytime you edit hmm.py

%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

```python
from hmm import *
```

1st-Order Hidden Markov Model Class:

The hidden markov model class would have the following attributes:

```
1. initial state log-probs vector (pi)
2. state transition log-prob matrix (A)
3. observation log-prob matrix (B)
```

The following methods:

```
1. fit method to count the probabilitis of the training set
2. path probability
3. viterbi decoding algorithm
```

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
$\quad viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
$\quad backpointer[\mathrm{s},1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
$\quad$ **for** each state s **from** 1 **to** N **do**
$\quad\quad viterbi[\mathrm{s,t}] \leftarrow \max_{s'=1}^{N} \; viterbi[s',t-1] \, * \, a_{s',s} \, * \, b_s(o_t)$
$\quad\quad backpointer[\mathrm{s,t}] \leftarrow \operatorname{argmax}_{s'=1}^{N} \; viterbi[s',t-1] \, * \, a_{s',s} \, * \, b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} \; viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} \; viterbi[s,T]$; termination step
bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Figure A.9 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

Task 1: Testing the HMM (20 Points)

```python
### DO NOT EDIT ###

# 5 points for the fit test case
# 15 points for the decode test case
```

```
# run the funtion that tests the HMM with synthetic parameters!
run_tests()
```

```
Testing the fit function of the HMM
All Test Cases Passed!
Testing the decode function of the HMM
All Test Cases Passed!
Yay! You have a working HMM. Now try creating a pos-tagger using this
class.
```

Task 2: PoS Tagging on Hindi Tagset (20 Points)

For this assignment, we will use the Hindi Tagged Dataset available with nltk.indian

Helper methods to load the dataset is provided in hmm.py

Please go through the functions and explore the dataset

Report the Accuracy for the Dev and Test Sets. You should get something between 65-85%

```
words, tags, observation_dict, state_dict, all_observation_ids,
all_state_ids = get_hindi_dataset()

# we need to add the id for unknown word (<unk>) in our observations
vocab
UNK_TOKEN = '<unk>'

observation_dict[UNK_TOKEN] = len(observation_dict)
print("id of the <unk> token:", observation_dict[UNK_TOKEN])
```

```
id of the <unk> token: 2186
```

```
print("No. of unique words in the corpus:", len(observation_dict))
print("No. of tags in the corpus", len(state_dict))
```

```
No. of unique words in the corpus: 2187
No. of tags in the corpus 26
```

```
# Split the dataset into train, validation and development sets

import random
random.seed(42)
from sklearn.model_selection import train_test_split

data_indices = list(range(len(all_observation_ids)))

train_indices, dev_indices = train_test_split(data_indices,
test_size=0.2, random_state=42)

dev_indices, test_indices = train_test_split(dev_indices,
test_size=0.5, random_state=42)
```

```
print(len(train_indices), len(dev_indices), len(test_indices))


def get_state_obs(state_ids, obs_ids, indices):
    return [state_ids[i] for i in indices], [obs_ids[i] for i in
indices]


train_state_ids, train_observation_ids = get_state_obs(all_state_ids,
all_observation_ids, train_indices)
dev_state_ids, dev_observation_ids = get_state_obs(all_state_ids,
all_observation_ids, dev_indices)
test_state_ids, test_observation_ids = get_state_obs(all_state_ids,
all_observation_ids, test_indices)
```

```
432 54 54
```

```
def add_unk_id(observation_ids, unk_id, ratio=0.05):
    """
    make 1% of observations unknown
    """
    for obs in observation_ids:
        for i in range(len(obs)):
            if random.random() < ratio:
                obs[i] = unk_id

add_unk_id(train_observation_ids, observation_dict[UNK_TOKEN])
add_unk_id(dev_observation_ids, observation_dict[UNK_TOKEN])
add_unk_id(test_observation_ids, observation_dict[UNK_TOKEN])

pos_tagger = HMM(len(state_dict), len(observation_dict))
pos_tagger.fit(train_state_ids, train_observation_ids)

assert np.round(np.exp(pos_tagger.pi).sum()) == 1
assert np.round(np.exp(pos_tagger.A).sum()) == len(state_dict)
assert np.round(np.exp(pos_tagger.B).sum()) == len(state_dict)

print('All Test Cases Passed!')
```

```
All Test Cases Passed!
```

```
def accuracy(my_pos_tagger, observation_ids, true_labels):
    tag_predictions = my_pos_tagger.decode(observation_ids)
    tag_predictions = np.array([t for ts in tag_predictions for t in
ts])
    true_labels_flat = np.array([t for ts in true_labels for t in ts])
    acc = np.sum(tag_predictions ==
true_labels_flat)/len(tag_predictions)
    return acc
```

```
print('dev accuracy:', accuracy(pos_tagger, dev_observation_ids,
dev_state_ids))
```

```
dev accuracy: 0.8127659574468085
```

```
print('test accuracy:', accuracy(pos_tagger, test_observation_ids,
test_state_ids))
```

```
test accuracy: 0.7987012987012987
```

```
# Fit a pos_tagger on the entire dataset.
import pickle

full_state_ids = train_state_ids + dev_state_ids + test_state_ids
full_observation_ids = train_observation_ids + dev_observation_ids +
test_state_ids

hindi_pos_tagger = HMM(len(state_dict), len(observation_dict))
hindi_pos_tagger.fit(full_state_ids, full_observation_ids)

pickle.dump(hindi_pos_tagger, open('hindi_pos_tagger.pkl', 'wb'))
```

```
### Finally we will use the hindi_pos_tagger as a pre-processing step
for our NER tagger
```

Assignment Title

Programming Assignment (40 points)

The programming assignement will be an implementation of the task described in the assignment

We will make sure you have enough scaffolding to build the code upon where you would only have to implement the interesting parts of the code

Evaluation

The evaluation of the assignment will be done through test scripts that you would need to pass to get the points.

Written Assignment (60 Points)

Written assignment tests the understanding of the student for the assignment's task. We have split the writing into sections. You will need to write 1-2 paragraphs describing the sections. Please be concise.

In your own words, describe what the task is (20 points)

Describe the task, how is it useful and an example.

```
1.) The first task was to perform parts of speech tagging using HMM
and implement the Viterbi decoder using the
Viterbi forward algorithm, we were provided with the initial state
log-probs vector (pi), the transition state log-prob
matrix (A) and the observation log-prob matrix (B) and the task was to
implement the fit methos to count the
probabilities of the training set and the viterbi decoding which was
implemented using back pointer methods.

The work was to create an HMM-based POS tagger for Hindi language
using the nltk.indian tagset.
Once we fit the dataset in its place we had to run the test cases for
that and see if it went right.
Then we had to decode the model with all the predictions made by the
model.
Once, the decoding part is done we just had to check on the accuracy
of the dev data and the test data and they were
found to be:
    dev accuracy: 0.8127659574468085
    test accuracy: 0.7987012987012987
Since, both the accuracies are close to 80 percent, our model has
performed well.
And, once these are determined we generated a pickle file which
```

```
contained the tags which is used in the named enity tagger.


2.) Using CRF we had to perform named entity recognition tagging on
the given dataset.
    Here, we're attempting to glean information from the text based on
the sentence's semantic context.
    Since we are using sequential modelling we are considering whether
a word which like name also can be used
    as a place name or not like Johannes Kennedy international
Airport, here Johannes Kennedy is not a name.
    Once the feature lists are created, We then design a mask to hide
the pads and pad the features with the labels.
    By padding the label sequences, we create a suitable matrix, and
then, using the CRF modelling,we
    execute the forward function to determine the loss.
```

Describe your method for the task (10 points)

Important details about the implementation. Feature engineering, parameter choice etc.

```
I have use the DP algorithm which creates a matrix with each rows and
columns as observation and states.
Each matrix cell represents the likelihood that an HMM will be in a
particular state after
seeing a certain amount of observations.
here we are using various components to compute the probability and
the bestpaths,
    The previous Viterbi path probability from the previous time step.
    Transition probability from previous state to current state.

On a Hindi data set, we employ named entity recognition with a
conditional random field.
By improving the feature set that the model uses to accurately
forecast the tags.
we add additional features in order to employ CRF.
To establish context, we provide features that offer successive POS
tags for each word.
Features like prev word, next word, suffix, etc. have been introduced
to improve the F1 score.
```

Experiment Results (10 points)

Typically a table summarizing all the different experiment results for various parameter choices

```
# Experiment with epoch=30, batch size=256, LR=0.35
| loss     | 3.1997464835041702 |
| F1 Score | [0.7406] |
```

```
# Experiment with epoch=30, batch size=16, LR=0.1
| loss     | 2.1001549219280355 |
| F1 Score | [0.8227] |


# Experiment with epoch=25, batch size=16, LR=0.2
| loss     | 1.8283142801704286 |
| F1 Score | [0.8366] |
```

```
  Input In [2]
    | loss     | 3.1997464835041702 |
    ^
SyntaxError: invalid syntax
```

Discussion (20 points)

Key takeaway from the assignment. Why is the method good? shortcomings? how would you improve? Additional thoughts?

```
From the previous assignment where we performed sentiment analysis we
just leveraged BOW and TF-IDF which focuses more
on the occurences of the words in the dictionary, but here using CRF
we are emphasising more on the sequence of the words
occuring in the text and extarct many differnet features for the
model.

Since there can be unknown words we can use forward backward
algorithm.
The HMM viterbi model uses a lot of compute time because it compare
the
max probability with all the transition and emissio probabilities.
Eventhough CRF is hard to train having high computational complexity
it
gives more accurate results.

Improve:
    I just added more features to the model as discussed in the class
with JIM.
    Features:
        Previous word
        word distance
        Last 3 characters of the word (like ing, ter, est which
describes about a noun or a happening)
        Last 2 characters of the word (like er, st which describes
about peformance of a noun)
        Lowercase of the word
        POS tags
    Discussed this with Jim post class and after converting the
```

sentences to feature lists, used the Adam
    optimizer(similiar to SGD from sentiment analysis), **and** ignoring
varios errors like false positive O **and** false
    negative tag errors like(I-PER) **and** achieved a F1 score of 86.5%.

We could also use LSTM-CRF **in** RNN **for** higher F1 scores.

Learned about many ways to program a model **and** get a hands on
experience
to train them **and** get results.