CSCI544 HW3

Yuhang Xiao - 6913860906 - yxiao776@usc.edu

Environment requirements:

- **Python 3.11.5**
- **Numpy**
- **Pandas**
- **json**

Report

Vocabulary Creation

Utilize pandas to process the data.

- **Threshold of replacement: 3**
- **Size of vocabulary: 16920**
- **Total occurrence of '< unk >': 32537**

Model Learning

- **Number of transition parameters: 2070**
- **Number of emission parameters: 761400**

Greedy Decoding with HMM

- **Accuracy on dev set: 0.9298995203691336**

Viterbi Decoding with HMM

- **Accuracy on dev set: 0.9437116750652662**

In [16]:

```
import pandas as pd
import numpy as np
import json
```

Vocabulary Creation

In [17]:

```
data = pd.read_csv('data/train', sep='\t', names=['index', 'word', 'tag'])
data['word'] = data['word'].fillna('')
data['occurrence'] = data.groupby('word')['word'].transform('count')
threshold = 3
data['word'] = data.apply(lambda x: '< unk >' if x.occurrence < threshold else x.word, a
xis=1)
vocab = data.word.value_counts().rename_axis('word').reset_index(name='occurrence')
vocab = pd.concat([vocab[vocab.word == '< unk >'], vocab.drop(vocab[vocab.word == '< unk
>'].index)]).reset_index(drop=True)
vocab['index'] = vocab.index + 1
vocab = vocab[['word', 'index', 'occurrence']]
vocab.to_csv('vocab.txt', sep='\t', index=False, header=False)
print('Threshold of replacement:', threshold)
```

```
print('Size of vocabulary:', vocab.shape[0])
print('Total occurrence of \'< unk >\':', vocab.iloc[0]['occurrence'])
vocab.head()
```

```
Threshold of replacement: 3
Size of vocabulary: 16920
Total occurrence of '< unk >': 32537
```

Out[17]:

	word	index	occurrence
0	< unk >	1	32537
1	,	2	46476
2	the	3	39533
3	.	4	37452
4	of	5	22104

Model Learning

Index the Part-Of-Speech (POS) tags

A special < bos > tag is added here for initial probability.

In [18]:

```
tag = data.tag.value_counts().rename_axis('tag').reset_index(name='count')
# create a new tag for the beginning of a sentence
tag = pd.concat([pd.DataFrame({'tag': ['< bos >'], 'count': [0]}), tag]).reset_index(dro
p=True)
print('Number of POS tags:', tag.shape[0])
tag.head()
```

```
Number of POS tags: 46
```

Out[18]:

	tag	count
0	< bos >	0
1	NN	127534
2	IN	94758
3	NNP	87608
4	DT	78775

Create index dict for vocab and tags

get the index dict for tags and vocabulary

In [19]:

```
tag_dict = pd.Series(tag.index.values, index=tag.tag).to_dict()
vocab_dict = pd.Series(vocab.index.values, index=vocab.word).to_dict()
```

Transition Matrix

In [20]:

```
trans = np.zeros((tag.shape[0], tag.shape[0]))
for row in data.itertuples():
```

```
    if row.index == 1:
        tag.at[0, 'count'] += 1
        i = 0
        j = tag_dict[row.tag]
        trans[i, j] += 1
    else:
        i = tag_dict[data.at[row.Index - 1, 'tag']]
        j = tag_dict[row.tag]
        trans[i, j] += 1
trans /= tag['count'].to_numpy().reshape(-1, 1)
print('Transition matrix:', trans.shape[0], 'x', trans.shape[1])
```

```
Transition matrix: 46 x 46
```

Emission Matrix

In [21]:

```
emis = np.zeros((tag.shape[0], vocab.shape[0]))
for row in data.itertuples():
    i = tag_dict[row.tag]
    j = vocab_dict[row.word]
    emis[i, j] += 1
emis /= tag['count'].to_numpy().reshape(-1, 1)
print('Emission matrix:', emis.shape[0], 'x', emis.shape[1])
```

```
Emission matrix: 46 x 16920
```

Transfer Matrix to Dict

In [22]:

```
transition = dict()
for i in range(trans.shape[0]):
    for j in range(1, trans.shape[1]):
        transition['('+tag.at[i, 'tag']+','+tag.at[j, 'tag']+')'] = trans[i, j]
emission = dict()
for i in range(1, emis.shape[0]):
    for j in range(emis.shape[1]):
        emission['('+tag.at[i, 'tag']+','+vocab.at[j, 'word']+')'] = emis[i, j]
print('Number of transition parameters:', len(transition))
print('Number of emission parameters:', len(emission))
```

```
Number of transition parameters: 2070
Number of emission parameters: 761400
```

Save the JSON

In [23]:

```
with open('hmm.json', 'w') as f:
    json.dump({'transition': transition, 'emission': emission}, f)
```

Read Dev and Test data

In [24]:

```
dev_data = pd.read_csv('data/dev', sep='\t', names=['index', 'word', 'tag'])
dev_data['word'] = dev_data['word'].fillna('')
test_data = pd.read_csv('data/test', sep='\t', names=['index', 'word'])
test_data['word'] = test_data['word'].fillna('')
```

Greedy Decoding with HMM

Greedy decoding function

In [25]:

```
def greedy_decode(data, tag, tag_dict, vocab_dict, trans, emis):
    pred_tag = []
    for line in data.itertuples():
        if line.index == 1:
            states = trans[0, 1:]
        else:
            states = trans[tag_dict[pred_tag[-1]], 1:]
        words = emis[1:, vocab_dict[line.word] if line.word in vocab_dict else vocab_dic
t['< unk >']]
        tag_idx = np.argmax(states * words)+1
        pred_tag.append(tag.at[tag_idx, 'tag'])
    return pred_tag
```

dev data evaluation

In [26]:

```
dev_pred = greedy_decode(dev_data, tag, tag_dict, vocab_dict, trans, emis)
acc = 0
total = len(dev_pred)
assert total == dev_data.shape[0], 'Number of predictions does not match number of words
'
for line in dev_data.itertuples():
    if line.tag == dev_pred[line.Index]:
        acc += 1
print('Accuracy on dev set:', acc/total)
```

```
Accuracy on dev set: 0.9298995203691336
```

store test data prediction results

In [27]:

```
test_pred = greedy_decode(test_data, tag, tag_dict, vocab_dict, trans, emis)
assert len(test_pred) == test_data.shape[0], 'Number of predictions does not match numbe
r of words'
with open('greedy.out', 'w') as f:
    for line in test_data.itertuples():
        if line.Index == test_data.shape[0]:
            f.write(str(line.index)+'\t'+line.word+'\t'+test_pred[line.Index])
        else:
            f.write(str(line.index)+'\t'+line.word+'\t'+test_pred[line.Index]+'\n')
```

Viterbi Decoding with HMM

Viterbi Decoding Function

In [28]:

```
def viterbi_decode(data, tag, vocab_dict, trans, emis):
    def forward(seq):
        dp = np.zeros((len(seq), tag.shape[0]-1))
        parent = np.zeros((len(seq), tag.shape[0]-1), dtype=int)
        dp[0] = trans[0, 1:] * emis[1:, vocab_dict[seq[0]] if seq[0] in vocab_dict else
vocab_dict['< unk >']]
        for i in range(1, len(seq)):
            arr = dp[[i-1]].T * trans[1:, 1:] * emis[1:, vocab_dict[seq[i]] if seq[i] i
n vocab_dict else vocab_dict['< unk >']]
            parent[i] = np.argmax(arr, axis=0)
            dp[i] = arr[parent[i], np.arange(dp.shape[1])]
        return dp, parent
    def backward(dp, parent):
        pred_tag = []
```

```
        last_state = np.argmax(dp[-1])
        pred_tag.append(tag.at[last_state+1, 'tag'])
        for i in range(dp.shape[0]-2,-1,-1):
            last_state = parent[i+1, last_state]
            pred_tag.append(tag.at[last_state+1, 'tag'])
        return pred_tag[::-1]
    seq = []
    pred = []
    for line in data.itertuples():
        if line.index == 1:
            if len(seq) > 0:
                dp, parent = forward(seq)
                pred_tag = backward(dp, parent)
                pred.extend(pred_tag)
            seq = [line.word]
        else:
            seq.append(line.word)
    if len(seq) > 0: pred.extend(backward(*forward(seq)))
    return pred
```

dev data evaluation

In [29]:

```
dev_pred = viterbi_decode(dev_data, tag, vocab_dict, trans, emis)
acc = 0
total = len(dev_pred)
assert total == dev_data.shape[0], 'Number of predictions does not match number of words
'
for line in dev_data.itertuples():
    if line.tag == dev_pred[line.Index]:
        acc += 1
print('Accuracy on dev set:', acc/total)
```

```
Accuracy on dev set: 0.9437116750652662
```

store test data prediction

In [30]:

```
test_pred = viterbi_decode(test_data, tag, vocab_dict, trans, emis)
assert len(test_pred) == test_data.shape[0], 'Number of predictions does not match numbe
r of words'
with open('viterbi.out', 'w') as f:
    for line in test_data.itertuples():
        if line.Index == test_data.shape[0]:
            f.write(str(line.index)+'\t'+line.word+'\t'+test_pred[line.Index])
        else:
            f.write(str(line.index)+'\t'+line.word+'\t'+test_pred[line.Index]+'\n')
```