CS310 Natural Language Processing Assignment 5:Dependency Parsing

12310520 芮煜涵

Task5

The result of the evaluation is shown below: left is WordPosModel, right is BaseModel.

```
Start time:  1745670843.5962582
Start time:  Sat Apr 26 20:34:03 2025
Loading WordPOSModel...
Evaluating on ../data/dev.conll
100%|██████████| 1700/1700 [00:10<00:00, 164.44it/s]
1700 sentence.
Micro Avg. Labeled Attachment Score: 0.7142465343572355
Micro Avg. Unlabeled Attachment Score: 0.7736611149895283
Macro Avg. Labeled Attachment Score: 0.7279998895681858
Macro Avg. Unlabeled Attachment Score: 0.7874118188073156

Evaluating on ../data/test.conll
100%|██████████| 2416/2416 [00:14<00:00, 166.54it/s]
4116 sentence.
Micro Avg. Labeled Attachment Score: 0.7168368401471622
Micro Avg. Unlabeled Attachment Score: 0.7760014054813774
Macro Avg. Labeled Attachment Score: 0.7287188688132584
Macro Avg. Unlabeled Attachment Score: 0.787451533272849

Time: 24.959131717681885
```

```
Start time:  1745670883.745255
Start time:  Sat Apr 26 20:34:43 2025
Loading BaseModel...
Evaluating on ../data/dev.conll
100%|██████████| 1700/1700 [00:10<00:00, 162.82it/s]
1700 sentence.
Micro Avg. Labeled Attachment Score: 0.650497295410923
Micro Avg. Unlabeled Attachment Score: 0.7345265099583718
Macro Avg. Labeled Attachment Score: 0.6532837621197343
Macro Avg. Unlabeled Attachment Score: 0.7381804363857297

Evaluating on ../data/test.conll
100%|██████████| 2416/2416 [00:14<00:00, 166.23it/s]
4116 sentence.
Micro Avg. Labeled Attachment Score: 0.6521141747332101
Micro Avg. Unlabeled Attachment Score: 0.7349766009979442
Macro Avg. Labeled Attachment Score: 0.657974826705334
Macro Avg. Unlabeled Attachment Score: 0.7410706395178712

Time: 25.089192867279053
```

Both is better than 0.7.

Task6

The complete code is in arc_eager_parse_util.py.

The revised version of class State:

```python
class State(object):
    def __init__(self, sentence=[]):
        self.stack = []
        self.buffer = []
        if sentence:
            self.buffer = list(sentence) # Arc-eager的buffer是正序
        self.deps = set()

```

```
    def shift(self):
        self.stack.append(self.buffer.pop())

    def left_arc(self, label):
        head = self.buffer[0]
        dependent = self.stack.pop()
        self.deps.add((head, dependent, label))

    def right_arc(self, label):
        head = self.stack[-1]
        dependent = self.buffer.pop(0)
        self.deps.add((head, dependent, label))
        self.stack.append(dependent)

    def reduce(self):
        self.stack.pop()

    def __repr__(self):
        return "{},{},{}".format(self.stack, self.buffer, self.deps)
```

The revised version of function get_training_instances:

```
def get_training_instances(dep_tree: DependencyTree) -> List[Tuple[State, Tuple[str, str]]]:
    deprels = dep_tree.deprels
    sorted_nodes = [k for k, v in sorted(deprels.items())]
    state = State(sorted_nodes)
    state.stack.append(0)  # 加入ROOT

    childcount = defaultdict(int)
    for ident, node in deprels.items():
        childcount[node.head] += 1

    seq = []
    while state.buffer or len(state.stack) > 1:
        if not state.stack:
            seq.append((copy.deepcopy(state), ("shift", None)))
            state.shift()
            continue

        if state.stack[-1] == 0:
            stackword = RootDummy()
        else:
            stackword = deprels[state.stack[-1]]

        if state.buffer:
            bufferword = deprels[state.buffer[0]]
        else:
            bufferword = None

        # 根据Arc-Eager标准，优先判断动作
```

```
        if bufferword and bufferword.head == stackword.id:
            seq.append((copy.deepcopy(state), ("right_arc", bufferword.deprel)))
            state.right_arc(bufferword.deprel)
            childcount[stackword.id] -= 1
        elif bufferword and stackword.head == bufferword.id:
            seq.append((copy.deepcopy(state), ("left_arc", stackword.deprel)))
            state.left_arc(stackword.deprel)
            childcount[bufferword.id] -= 1
        elif childcount[stackword.id] == 0:
            seq.append((copy.deepcopy(state), ("reduce", None)))
            state.reduce()
        else:
            seq.append((copy.deepcopy(state), ("shift", None)))
            state.shift()

    return seq
```

The example shows the difference between the two transition system:

Arc-Standard Transition System:

Step	Stack	Buffer	Action
0	[ROOT]	[The dog barked at the stranger loudly]	SHIFT
1	[ROOT, The]	[dog barked at the stranger loudly]	SHIFT
2	[ROOT, The, dog]	[barked at the stranger loudly]	RIGHT-ARC
3	[ROOT, dog]	[barked at the stranger loudly]	SHIFT
4	[ROOT, dog, barked]	[at the stranger loudly]	LEFT-ARC
5	[ROOT, barked]	[at the stranger loudly]	SHIFT
6	[ROOT, barked, at]	[the stranger loudly]	SHIFT
7	[ROOT, barked, at, the]	[stranger loudly]	SHIFT
8	[ROOT, barked, at, the, stranger]	[loudly]	LEFT-ARC
9	[ROOT, barked, at, stranger]	[loudly]	RIGHT-ARC
10	[ROOT, barked, at]	[loudly]	RIGHT-ARC
11	[ROOT, barked]	[loudly]	SHIFT
12	[ROOT, barked, loudly]	[]	RIGHT-ARC
13	[ROOT, barked]	[]	RIGHT-ARC

Arc-Eager Transition System:

Step	Stack	Buffer	Action
0	[ROOT]	[The dog barked at the stranger loudly]	SHIFT
1	[ROOT, The]	[dog barked at the stranger loudly]	LEFT-ARC
2	[ROOT]	[dog barked at the stranger loudly]	SHIFT
3	[ROOT, dog]	[barked at the stranger loudly]	LEFT-ARC
4	[ROOT]	[barked at the stranger loudly]	SHIFT
5	[ROOT, barked]	[at the stranger loudly]	RIGHT-ARC
6	[ROOT, barked, at]	[the stranger loudly]	SHIFT
7	[ROOT, barked, at, the]	[stranger loudly]	LEFT-ARC
8	[ROOT, barked, at]	[stranger loudly]	RIGHT-ARC
9	[ROOT, barked, at, stranger]	[loudly]	REDUCE
10	[ROOT, barked, at]	[loudly]	REDUCE
11	[ROOT, barked]	[loudly]	RIGHT-ARC
12	[ROOT, barked, loudly]	[]	REDUCE
13	[ROOT, barked]	[]	REDUCE
14	[ROOT]	[]	DONE