VINCENT MICHAEL SUTANTO - 2111434
vincent.michael_sutanto.vg6@is.naist.jp

Q. Traditional sequence labelling and parsing techniques were applied in various fields of NLP, now also they can be useful with deep learning. In this assignment, we will learn traditional techniques through part-of-speech tagging and dependency parsing. Your missions are:

- Create your own model with handcraft features to generate the correct trees of the dataset. It can be: Eisner, Shift-Reduce, Maximum Spanning Tree(MST), etc (DO NOT USE DEEP LEARNING method).

- Write your short report: Write a short explanation about your program and show us your code. Also, show your accuracy on the test set.

Training file: https://github.com/neubig/nlptutorial/blob/master/data/mstparser-en-train.dep

Test file: https://github.com/neubig/nlptutorial/blob/master/data/mstparser-en-test.dep

Abstract

Dependency parsing is a task in natural language processing that aims to analyze the structure of a sentence based on the relationship of each word in it. In this task, ***Shift-Reduce Algorithm*** *with* ***Support Vector Machine*** *as its oracle is used to do dependency parsing. The final result produced an* ***accuracy score of 64.6%.***

**please see the notebook for the code, because I think it would be too long to explain all the code in this report*

Data Preprocessing and Analysis

The data provided is 200 sentences for train data and test data. The data is presented in the form of a table containing the dependencies between each sentence and other sentences and the part-of-speech (POS) tag of each word.

	index	word	pos	parent	type
0	1	in	IN	43	PP
1	2	an	DT	5	DEP
2	3	oct.	NNP	5	DEP
3	4	19	CD	5	DEP
4	5	review	NN	1	NP
...	...	...	...	...	...
5275	4	,	,	7	DEP
5276	5	"	"	7	DEP
5277	6	he	PRP	7	NP-SBJ
5278	7	says	VBZ	0	ROOT
5279	8	.	.	7	DEP

Distribution of train data

I process the data so that each sentence has its own dataframe. This will make it easier to extract features from the gold tree. The function used to do this is *dataFramePerSentence()*, where this function accepts 2 arguments, namely the train / test data file and the number of sentences you want to form a dataframe. Here is the code snippet from the *dataFramePerSentence()* function and its output.

```
def dataFramePerSentence(dframe, numbSentence):
  tempList = []
  startPoint = 0

  for i in range(numbSentence):
    temp = dframe[startPoint:].copy()
    temp.reset_index(inplace = True)
    temp['level_0'] = temp['level_0'] - temp['level_0'][0] + 1
    temp = temp[temp['level_0'] == temp['index']]
    startPoint = startPoint + len(temp)

    temp.drop(['index','level_0'], axis=1, inplace = True)

    temp.parent = temp.parent - 1
    temp['parentKey'] = temp.parent

    for index, row in temp.iterrows():
      if row.parent is -1:
        temp['parent'][index] = 'ROOT'
      elif not isinstance(temp['parent'][index], str):
        key = temp['parent'][index]
        temp['parent'][index] = temp['word'][key]

    temp['wordKey'] = temp.index
    tempList.append(temp)

  return tempList
```

dataFramePerSentence()

	word	pos	parent	type	parentKey	wordKey
0	ms.	NNP	haag	DEP	1	0
1	haag	NNP	plays	NP-SBJ	2	1
2	plays	VBZ	ROOT	ROOT	-1	2
3	elianti	NNP	plays	NP-OBJ	2	3
4	.	.	plays	DEP	2	4

Output ofdataFramePerSentence()

To generate features from each dataframe, a function *generateFeatures()* is formed. This function accepts 2 arguments, namely a list of dataframes and a boolean to determine whether to print the result or not. This function is made up of 2 big blocks: The first block will parse each dataset and the second block will generate the resulting features from the parsing.

This function generates features in the form of the top 2 items on the stack, position 0 in the buffer, POS tags for the top 2 items on the stack and position 0 in the buffer, the length of the temporary stack, and the length of the temporary buffer. I added a temporary stack length and a temporary buffer length to give awareness to the model (e.g. when the buffer is running low / depleted then the left or right step takes priority over the shift step).

The following is the result of using the *generateFeatures()* function. Because the code is too long, please see the available notebooks.

```
==================
◉RIGHT:  5  -->  7
stack:  ['ROOT', 3, 4, 5]
buffer:  [8]
==================
◉RIGHT:  4  -->  5
stack:  ['ROOT', 3, 4]
buffer:  [8]
==================
◉RIGHT:  3  -->  4
stack:  ['ROOT', 3]
buffer:  [8]
==================
◉shift outer
stack:  ['ROOT', 3, 8]
buffer:  []
==================
◉RIGHT:  3  -->  8
stack:  ['ROOT', 3]
buffer:  []
==================
```

	stack0	stack1	buffer0	posstack0	posstack1	posbuffer0	lenstack	lenbuffer	target
0	ROOT	the	centers	NONE	DT	NNS	2	8	shift
1	the	centers	normally	DT	NNS	RB	3	7	left
2	ROOT	centers	normally	NONE	NNS	RB	2	7	shift
3	centers	normally	are	NNS	RB	VBP	3	6	shift
4	normally	are	closed	RB	VBP	VBN	4	5	left
5	centers	are	closed	NNS	VBP	VBN	3	5	left
6	ROOT	are	closed	NONE	VBP	VBN	2	5	shift

Output generateFeatures()

Next I did the undersampling technique to the features. This is because the majority class of the formed dataset is the 'shift' class, which is why I don't want the model to overfit this class. The undersampling process is carried out using the Imbalance-Learning library. The following is the amount of data before and after the undersampling process is carried out.

```
shift    5279  2453
left     2627  2453
right    2453  2453
```

Left: before; right: after

Oracle: Support Vector Machine

Support Vector Machine is a traditional machine learning algorithm that separates each class by projecting each data to a higher dimension. Hyperplane with soft-margin is then used to divide the projected data into 2 different classes. In this task, I used SVM (ThunderSVM, a GPU-enabled SVM library) because when I use the perceptron I built from scratch it takes a very long time to train.

The kernel I'm using is the Radial-Basis Function (RBF) kernel. Therefore, I had to do hyperparameter tuning C and gamma. The C values used are 1, 2, 4, 8, 10 and the gamma values used are 0.01, 0.05, 0.1, 0.5. Each combination of C and gamma was trained with 5-Fold cross validation, so that the combination that produces the highest accuracy value would be the main candidate for the oracle. The best parameter combination are C = 10 and gamma = 0.01.

C	Gamma	Accuracy (%)
1	0.01	77.78
1	0.05	79.44
1	0.1	79.25
1	0.5	70.8
2	0.01	80.32
2	0.05	81.7
2	0.1	80.88
2	0.5	71.69
4	0.01	82.88
4	0.05	82.42
4	0.1	80.87
4	0.5	71.69
8	0.01	84.5
8	0.05	82.4
8	0.1	80.91
8	0.5	71.69
10	**0.01**	**84.56**
10	0.05	82.29
10	0.1	80.90
10	0.5	71.69

Parameter tuning for SVM

Shift-Reduce Algorithm

Shift-Reduce algorithm is one of the algorithms that can be used to perform dependency parsing. This algorithm utilizes stacks and buffers to form relations for each word. There are 3 main steps involved in this algorithm:

1. Shift: pop the frontmost item from the buffer and push it into the stack (must be executed when there is only 1 item in stack).
2. Left: the top of the stack is a child of the frontmost buffer. Pop the top of the stack and push the leading item of the buffer onto the stack.
3. Right: the top of the stack is the parent of the leading buffer. Pop the leading item from the buffer.

To determine which step to choose, machine learning algorithms are used, in this case I use the Support Vector Machine.

In this task, I created a *ShiftReduce* class to handle this algorithm. This class accepts a list of test data dataframes, machine learning models, and encoders (because categorical data must be converted into numeric form in order to be processed by machine learning algorithms). please look at the notebook because the code is too long to be attached here.

Results

The task I did yielded an accuracy of **64.6%**. This is quite surprising for me considering that the best 5-fold cross-validation produces an accuracy rate of up to 84.56%. But maybe this is a natural thing considering the use of oracle in shift-reduce is similar to a chaos system, where a few changes in the previous steps can produce big changes in the future steps.

```
sr = ShiftReduce()
dfPredicted = sr.process(dfTestList, svm, featureEncoder)
```

```
arrPred = np.array([])
arrActual = np.array([])

for i in range(len(dfPredicted)):
  arrPred = np.concatenate((arrPred, np.array(dfPredicted[i].parentKey)))
  arrActual = np.concatenate((arrActual, np.array(dfTestList[i].parentKey)))
```

```
correct = (arrActual == arrPred)
correct.sum() / correct.size
```

```
0.6460444061220091
```

Prediction result