Word embedding based classification

1 Logistic regression tagging with word embeddings

In this section, we will be comparing the performance of two models: the count-based baseline model and the word-vector based multinomial logistic regression model. The baseline model is a generative model and works by counting how a word is tagged in the training data(Equation 1) and then selecting the most probable one; as any count-based methods, it overlooks the problem of zero occurrence. This problem is even more obvious in the 'person_name' slot. Basically, the model failed to identify any person names and tagged them with 'O', because the name words in validation set never appear in the training data. As a result, we observed a precision and recall of 0.

$$\begin{aligned} P(tag|word) &= \frac{P(tag, word)}{P(word)} \\ &= \frac{Count(tag, word)}{Count(word)} \end{aligned} \tag{1}$$

In the logistic model, instead of simple word forms, we take account word sense, which is quantified into word vectors. On the other hand, we employed a discriminative approach which models $P(tag|word\_embedding)$ directly following:

$$P(tag = k|\vec{X_{word}}) = \frac{\exp(\vec{X}\beta_k)}{\sum_{j=1}^{K} \exp(\vec{X}\beta_j)} \tag{2}$$

where:

k = current tag label
K = total number of tag types
$\vec{X}$ = word vector
β = weight vector

Our model will learn the weight for each feature in the word vector and creates discriminative boundaries between classes (here tags), while generative models learn the joint-distribution of classes and features, which we have little interest in for the current task. Moreover, employing word embedding itself brings some benefits: 1) there will be no 0-occurrence case as long as the word has a word vector and ideally they would be tagged similarly as words of similar embeddings. 2) the features are more well-clustered so that for the model to better discriminate. Another finding we would like to put here is that the vectors from spaCy 'eng_core_web_sm'

pineline are context-sensitive tensors(Honnibal & Montani, 2017). Therefore, word senses are given by local contexts. As anticipated, the performance improved significantly; Table 1 presents the selected instances.

	True tag	Baseline_tag	LR_tag
bruggeman	B-person_name	O	B-person_name
shaida	B-person_name	O	B-person_name
murad	B-person_name	O	I-rooms
dalana	B-person_name	O	O
palomares	I-person_name	O	O
sharon	I-person_name	O	I-person_name
jacuzzi	O	O	B-person_name
jon	O	O	B-person_name

Table 1: Taggings of 'person_names' in two models

It is shown that the precision and recall increased by 0.5, however, there were two over-predicting cases ('jacuzzi' and 'jon') in 'B-person_name', which decreased the overall precision.

	Baseline			LR(word vector)			
	precision	recall	f1-score	precision	recall	f1-score	support
B-person_name	0	0	0	0.50	0.50	0.50	4
I-person_name	0	0	0	1.00	0.50	0.67	2

Table 2: Evaluation metrics scores of 'person_name'

In general, the performance remarkedly increased after shifting the models and input features. The macro averaged f1-score doubled from 0.16 to 0.38, that is, the average performance for each class doubled. The micro averaged f1-score also doubled but is not meaningful in our case since the data is quite unbalanced and it over-emphasizes performances in some labels of larger 'quantity'.

	Baseline			LR(word vector)		
	precision	recall	f1-score	precision	recall	f1-score
micro avg	0.31	0.22	0.26	0.53	0.49	0.51
macro avg	0.23	0.15	0.16	0.45	0.37	0.38
weighted avg	0.32	0.22	0.24	0.54	0.49	0.50

Table 3: Evaluation metrics scores of the whole system

2 BIO tagging for slot labeling

first tag \ second tag	B-adults	I-adults	B-rooms	I-rooms	O
B-adults	1	1	1	0	1
I-adults	1	1	1	0	1
B-rooms	1	0	1	1	1
I-rooms	1	0	1	1	1
O	1	0	1	0	1

Table 4: Excerpt of BIO tag transition constraint matrix

Table 4 above presents a small part of the BIO tag transition matrix, which shows valid occurrences of two consecutive tags: if valid then marked '1', otherwise, '0'. Restrictions are as follows: an inside tag ('I-') is not allowed to follow any tag except 1)a beginning tag ('B-') of the same slot or 2)a same inside tag. Other types of tag can follow any tag freely.

We included the transition constraints and predicted the optimal tag sequence through scoring. Equation 3 below illustrates how our model predicts the tag sequence $Y = y_1, y_2, ..., y_n$ given the token sequence $X = x_1, x_2, ..., x_n$:

$$\begin{aligned} Y^* &= \underset{Y}{\mathbf{argmax}}\, p(Y|X) \\ &= \underset{Y}{\mathbf{argmax}} \prod_i p(y_i|y_{i-1}, x_i) \\ &= \underset{Y}{\mathbf{argmax}} \sum_i \log p(y_i|y_{i-1}, x_i) \\ &= \underset{Y}{\mathbf{argmax}} \sum_i \log \frac{\exp(t(y_i, y_{i-1}) + e(y_i, x_i))}{Z(y_{i-1}, x_i)} \\ &= \underset{Y}{\mathbf{argmax}} \sum_i (t(y_i, y_{i-1}) + e(y_i, x_i)) \end{aligned} \tag{3}$$

where:

Y^* = the optimal tag sequence
Y = $y_1, y_2, ..., y_n$; tag sequence
X = $x_1, x_2, ..., x_n$; token sequence

Different from the baseline logistic regression model, we reckon that the probability of the tag y_i at each time step should depend not only on the current token/word x_i but further on the previous tag y_{i-1} to obey the BIO tag constraints. Based on the assumption, We can break down the sequence probability into the product of probabilities at each time step, or into the sum of their logarithmic forms. We also assume that x_i and y_{i-1} are independent. x_i is used to estimate the score of each tag, while y_{i-1} is employed to check whether y_i is valid. So, our model computes the $p(y_i|y_{i-1}, x_i)$ through two score functions. $t(y_i, y_{i-1})$ provides the transition score from y_{i-1} to y_i, while $e(y_i, x_i)$ computes the emission score, evaluating how good y_i is. Because of the normalization term Z is identical to different y_i, we can omit it under the $\mathbf{argmax}$ operation. Eventually, we get the formula that endeavors to find the sequence with the highest global score(for both t and e), a potential optimal solution.

In consistency with HMM, we would name the score functions as transition and emission, but in practice, $e(y_i, x_i)$ directly uses the results from the logistic regression prediction, which gives the probability of y_i given x_i (a discriminative process), instead of a score of emission from y_i to x_i (a generative process). Besides, the $t(y_i, y_{i-1})$ only outputs two possible values: zero if y_i is allowed to follow y_{i-1} based on the constraints, otherwise, negative infinity to exclude invalid sequence strictly.

To accelerate the inference, we employ a Viterbi-like dynamic programming method:

Algorithm 1 Viterbi_Score (token sequence length T, tag number N)

create a path score matrix $score[N, T]$ and back-pointer matrix $backpointer[N, T]$
for each tag s from 1 to N **do**
  $score[s, 1] \leftarrow 0$
  $backpointer[s, 1] \leftarrow 0$
end for
for each timestep i from 2 to T **do**
  for each tag s from 1 to N **do**
    $score[s, i] \leftarrow \max_{s'=1}^{N} s[s', i-1] + t(s, s') + e(s, o_i)$
    $backpointer[s, i] \leftarrow \text{argmax}_{s'=1}^{N} s[s', i-1] + t(s, s') + e(s, o_i)$
  end for
end for
$bestpathpointer \leftarrow \text{argmax}_{s=1}^{N} score[s, T]$
$bestpath \leftarrow$ the path starting at bestpathpointer, that follows $backpointer[]$ to tags back in time **return** $bestpath$

At each time-step, this algorithm computes the best score so far to each tag, and records the previous tag it follows, which provides the score. After iteration, it selects the final tag with the highest score, which indicates the highest global score, and a series of backtraces, following

which the scores are selected. The result turns out to be the optimal tag sequence predicted by our model.

Table 5 and Table 6 below display the evaluation results of both BIO tags and slot labeling on the validation data:

BIO tag	metrics			
	precision	recall	f1-score	support
B-adults	0.60	0.67	0.63	9
B-date	0.50	0.50	0.50	8
B-date_from	0.58	0.75	0.65	20
B-date_period	0.78	0.78	0.78	9
B-date_to	0.62	0.73	0.67	11
B-kids	0.62	0.83	0.71	6
B-number	0.00	0.00	0.00	3
B-people	0.33	0.75	0.46	4
B-person_name	0.60	0.75	0.67	4
B-rooms	0.33	0.33	0.33	9
B-time	0.62	0.83	0.71	6
B-time_from	0.00	0.00	0.00	1
B-time_period	1.00	0.33	0.50	3
I-adults	1.00	0.25	0.40	8
I-date	0.00	0.00	0.00	5
I-date_from	0.58	0.47	0.52	15
I-date_period	1.00	0.56	0.71	9
I-date_to	0.00	0.00	0.00	5
I-kids	0.00	0.00	0.00	0
I-number	0.00	0.00	0.00	0
I-people	0.00	0.00	0.00	0
I-person_name	1.00	0.50	0.67	2
I-rooms	0.67	0.17	0.27	12
I-time	0.56	0.45	0.50	11
I-time_from	0.00	0.00	0.00	3
I-time_period	0.00	0.00	0.00	3
micro avg	0.59	0.49	0.54	166
macro avg	0.44	0.37	0.37	166
weighted avg	0.57	0.49	0.49	166

Table 5: BIO tagging evaluation result

		metrics			
		precision	recall	f1-score	support
span	adults	0.30	0.33	0.32	9
	date	0.38	0.38	0.38	8
	date_from	0.42	0.55	0.48	20
	date_period	0.56	0.56	0.56	9
	date_to	0.54	0.64	0.58	11
	kids	0.62	0.83	0.71	6
	number	0.00	0.00	0.00	3
	people	0.33	0.75	0.46	4
	person_name	0.60	0.75	0.67	4
	rooms	0.22	0.22	0.22	9
	time	0.38	0.50	0.43	6
	time_from	0.00	0.00	0.00	1
	time_period	1.00	0.33	0.50	3
	micro avg	0.43	0.49	0.46	93
	macro avg	0.41	0.45	0.41	93
	weighted avg	0.43	0.49	0.45	93

Table 6: Slot labeling evaluation result

The BIO tag results show the performance of the model on each tag, and we claim that both precision and recall are useful in analysing the mistakes. A low precision is caused by over-prediction, which indicates the lack of features to restrict the prediction to be more precise, while a low recall is an indication of insufficient information to recognise a specific tag. Nontheless, f1-scores contribute less since it averages the precision and recall, 'concealing' the specific causes of errors detected by the aforementioned two metrics. Furthermore, because of transition constraints, the performance on the inside tag heavily relies on the prediction of the beginning tag of that span. In other words, the bad performance on beginning tags can be the cause of errors on inside tags. Thus, the results on beginning tags can provide more information in most error analysis cases.

On the other hand, f1-score on span labeling is the most helpful when assessing the model globally, because we do not care much about the cause of errors, but to what extent the model fails. It combines the two evaluations, which detect different types of mistake, into one score, thus more convenient for assessing and visualising. Nonetheless, in a real dialogue system, we argue that precision outweighs recall slightly. Because the system can simply prompt the user for necessary information, say, when some slots are missing (occurs frequently with low recall), whereas revision is needed if collecting wrong slot information (occurs frequently with low precision), which tends to be more complicated. Therefore it is better to conduct a f_β-measure, where $\beta < 1$, to favor precision more.

3 Error Analysis and Feature Engineering

By inspecting the evaluation of the present model, we noticed there was an outstanding malperformance in the label 'rooms' (Table 7), where both precision and recall were only 0.22. Inside, 'I-rooms' and 'B-rooms' had unsatisfactory recalls of 0.33 and 0.17 respectively, whereas precision was high in the former(0.67), still low in the latter(0.33). However, the great precision of 'I-rooms' came at the expense of low recall, which may have been a consequence of under-predicting 'B-rooms'. 'B-rooms' eventually emerged as the crux of the issue.

	LR (word vector)			LR(word vector + head)			
	precision	recall	f1-score	precision	recall	f1-score	support
B-rooms	0.33	0.33	0.33	0.58	0.78	0.67	9
I-rooms	0.67	0.17	0.27	0.92	0.92	0.92	12
rooms (overall)	0.22	0.22	0.22	0.58	0.78	0.67	9

Table 7: Metrics scores before and after adding head information

Another observation we made was that 'B-rooms' was mainly occupied by determiners or other modifiers such as numbers or adjectives. These modifiers pose the issue that they can be tagged differently in accordance to what entity (i.e., rooms, kids, adults and etc.) they are modifying. As an illustration, 'one' is tagged 'B-rooms' when qualifying 'room', 'B-adults' when qualifying 'person' and 'O' when the modificand is unconcerned. We argue that modifiers' inability to identify the type of modificand in the current model will lead to an ambiguity in tagging. One effective way to address that is to involve syntactic dependencies, which connect the modifiers to their heads, that is, the modificands. In the case of 'room', we observed that components in this label usually connect to a head that has a lemma in the form of 'room'(see Figure 1, 'an' 'en' 'suite'). Therefore, we made our model to inspect the lemma of each token's head when tagging(i.e., whether it is 'room' or not).

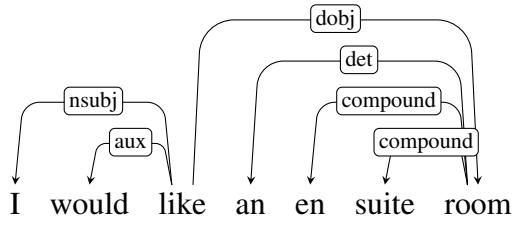

Figure 1: Dependency tree for a 'room' slot instance

As shown in Table 7, the performance improved remarkably, especially for 'I-rooms', where only one instance failed. The recall of 'B-rooms' doubled because determiners now 'know' what type of entity they are modifying, and thus those unidentified were identified. For example, 'an' in 'an('O') en suite room' was correctly tagged as 'B-rooms'. On the other hand, the outstanding performance in 'I-rooms', we assume, has a two-fold reason: 1) more 'B-rooms' were correctly recalled and as a consequence, the probability of assigning 'I-rooms' to next word increased due to tagging scheme constraints. 2) head's lemma information helped.

Nevertheless, we only saw a slight improvement in the overall performance(see Table 8). Since every label is of equal importance and we do not want tags of a large number have too much of an influence, we focus on the macro averaged f1-score, which increased only by 3 percent. Therefore, we assume, the improvement we made did have effects on single label but not too much on the whole system.

	micro avg f1-score	macro avg f1-score	weighted avg f1-score
LR (word vector)	0.46	0.41	0.45
LR(word vector + head)	0.51	0.44	0.50

Table 8: Evaluation of the overall system performance

Adding a dependency feature did help to some extent, but some accompanying problems are lurking around. The improved system over-predicted 'B-rooms' in the determiner group (i.e., a, an, the and etc.), which lowered the precision. Another issue is that we only considered the lemma 'room' as it is the major case, but it can be, say, 'bedroom' as well. Apart from that, the annotation itself can be erroneous and may bring more idiosyncrasies. For instance, 'one family room' is tagged as ['B-rooms', 'O', 'O'], while 'A family room' is tagged as ['B-rooms', 'I-rooms', 'I-rooms']. To make a reasonable judgment on annotations, we need the help of contexts because some statements are only meaningful in the context of the earlier speech (Casanueva et al., 2022, p. 2006).

4 Final Test Reporting and Reflection

		precision	recall	f1-score	support
span	adults	0.50	0.45	0.48	11
	date	0.09	0.11	0.10	9
	date_from	0.48	0.58	0.52	19
	date_period	0.57	0.80	0.67	5
	date_to	0.67	0.57	0.62	14
	kids	0.50	0.43	0.46	7
	number	0.33	0.50	0.40	2
	people	0.71	1.00	0.83	5
	person_name	0.25	0.20	0.22	5
	rooms	0.50	0.38	0.43	8
	time	0.25	0.20	0.22	10
	time_from	0.00	0.00	0.00	1
	time_period	0.00	0.00	0.00	2
	micro avg	0.44	0.45	0.44	98
	macro avg	0.37	0.40	0.38	98
	weighted avg	0.44	0.45	0.44	98

Table 9: Slot labeling evaluation result of LR_base on test set

		precision	recall	f1-score	support
span	adults	0.50	0.45	0.48	11
	date	0.09	0.11	0.10	9
	date_from	0.48	0.58	0.52	19
	date_period	0.57	0.80	0.67	5
	date_to	0.67	0.57	0.62	14
	kids	0.50	0.43	0.46	7
	number	**0.50**	0.50	**0.50**	2
	people	**0.83**	1.00	**0.91**	5
	person_name	0.25	0.20	0.22	5
	rooms	*0.33*	0.38	*0.35*	8
	time	**0.38**	**0.30**	**0.33**	10
	time_from	0.00	0.00	0.00	1
	time_period	0.00	0.00	0.00	2
	micro avg	**0.45**	**0.46**	**0.45**	98
	macro avg	**0.39**	**0.41**	**0.40**	98
	weighted avg	**0.45**	**0.46**	**0.45**	98

Table 10: Slot labeling evaluation result of LR_our on test set (numbers in bold show an improvement, otherwise italicised)

In this section, we evaluate the performance of two models on the test set. One is the basic logistic regression model (LR_base), which considers only the current word vector, and the other is our improved one (LR_our), which further checks whether the lemma of the current word's head is 'room' so that to help distinguish whether the token should be categorised as part of the span 'rooms'.

Table 9 and Table 10 show the slot labeling evaluation result of LR_base and LR_our respectively. We anticipated a performance improvement on span 'rooms', but the test result is beyond our expectation: the precision of the span 'rooms' dropped while the recall remained the same, so also a slight decrease on the f1-score.

	LR_base			LR_our			
	precision	recall	f1-score	precision	recall	f1-score	support
B-rooms	0.67	0.50	0.57	**0.78**	**0.88**	**0.82**	8
I-rooms	0.75	0.33	0.46	0.33	0.22	0.27	9
rooms (span)	0.50	0.38	0.43	0.33	0.38	0.35	8

Table 11: Metrics result of LR_base and LR_our on tags and span of 'rooms'

We further investigated the origins of this outcome. As shown in Table 11, the performance in tag 'B-rooms' is evidently improved, while the metrics results on tag 'I-rooms' were worse, especially the precision. These indicate that the LR_our is better at recognising the occurrence of the span 'rooms', but getting worse at labeling it precisely. After checking the cases, we found that it often failed at the end of the span (i.e., judging whether the word 'room' should be included in the span; see Figure 2). In the meantime, we found that the annotations themselves are ambiguous. For example, in case 1, 2 and 4, the word 'room' is not counted as a part of 'rooms' span, but it is included in case 3. Casanueva et al. (2022) pointed out that despite the fact that single-turn NLU uses less data and is simpler to model, some user utterances require context from a prior system utterance in order to make sense, so as the slots. The precise scope of a span could depend on the context utterances, which makes it hard to predict the range of span precisely given just one utterance. Moreover, in these cases, our model successfully extracts the users' requirements about the room. Whether the word 'room' is included does not affect parsing extensively.

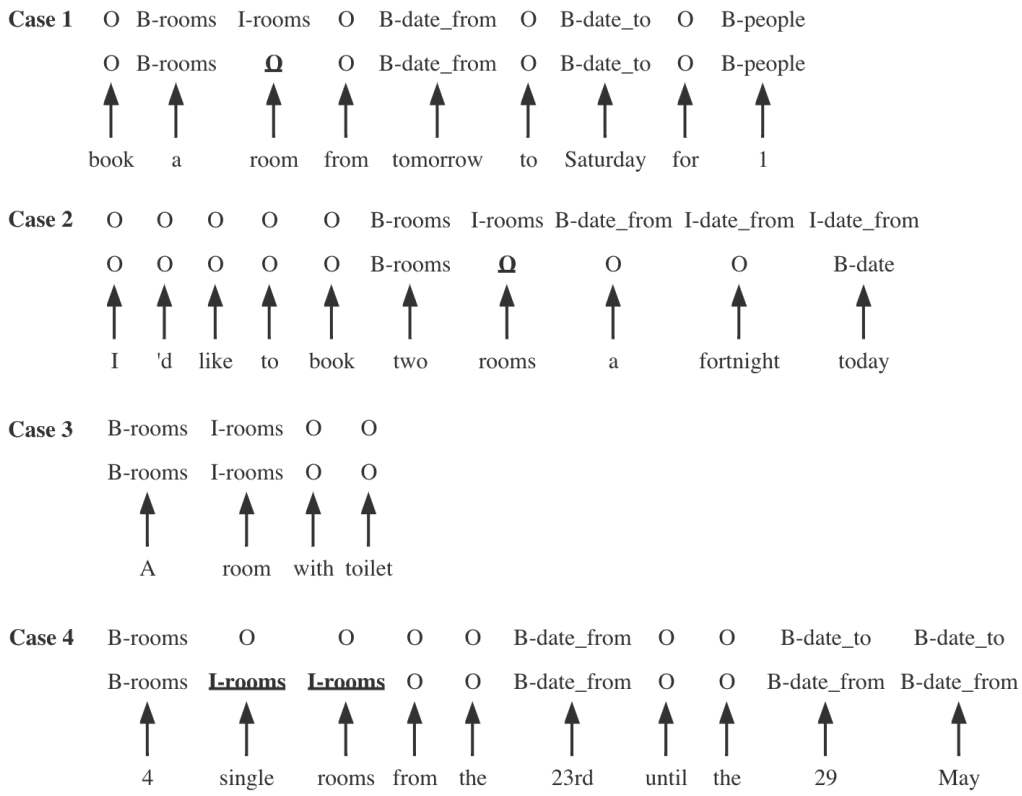

Figure 2: Test examples on LR_our. The first tag sequence is the ground truth; the second row is predicted result; the last row is the token sequence. Errors occur in span 'rooms' are highlighted with bold and underlined text.

In addition, scores on other spans such as 'number', 'people' and 'time' increased. These spans can be easily mixed with 'rooms', because they always start with a determiner or another modifier (e.g., number, adjective). It indicates that our modification on the model improves the ability to distinguish these spans, which eventually improves the overall performance.

5 Improvements

Casanueva et al. (2022, p. 2002) in their description of NLP dataset mentioned that there are generic slots (i.e., time, date) and domain-specific slots (i.e., rooms, kids), for example, 'rooms' are in the domain of 'hotels'. This domain information is coded in intent label, which is the first step in a dialogue system. This means that there are differences in the distribution of slots with different intents. Thus the intent information will help better label slots, especially domain-specific ones.

There are various ways to incorporate the intents into the labeling task. A dialogue system always pre-defines the necessary slot sets with specific intents. For example, when receiving a utterance with intent 'booking', the system need to collect 'rooms', 'people', 'date_from', etc. We can directly use these strict mapping rules in labeling to narrow the search space. However, in reality, these slots are always provided in turns, rather than a single utterance, and most utterances in the process convey general intents like 'affirm', 'change' etc. Moreover, in the provided dataset, an utterance can bring multiple intents. These facts make it inappropriate to explicitly define the slot set. Therefore, we argue that it could be a better way to encode the intent information as the features during labeling.

There are total 40 intent labels in the dataset, and we encoded them as a 40-dim vector with binary element value, '1' for existence of the corresponding intent, '0' otherwise. Considering there are utterances that have no intent, which indicates they are general, we further added one dimension to store no-intent information. The 41-dim vector was concatenated with the feature vector constructed in the previous sections to contribute the label prediction. The evaluation results of the previous model (LR_our) and the intent-embedded model (LR_intent) are shown in Table 12.

Based on the results, the model did improve the performance on some domain-specific slots like 'rooms', but most improvement were made on general slots like 'date', 'time' etc. This is because the dataset is domain-specific, in which samples are all in the 'hotel' circumstances. The consideration of intent information will eventually improve those slots which have relatively strong correlation with intents. Consequently, the intent information helps improve the overall performance on these slots.

	LR_our			LR_intent			
	precision	recall	f1-score	precision	recall	f1-score	support
adults	0.50	0.45	0.48	0.44	0.36	0.40	11
date	0.09	0.11	0.10	0.25	0.11	**0.15**	9
date_from	0.48	0.58	0.52	0.44	0.79	**0.57**	19
date_period	0.57	0.80	0.67	0.50	0.80	0.62	5
date_to	0.67	0.57	0.62	0.62	0.36	0.45	14
kids	0.50	0.43	0.46	0.40	0.29	0.33	7
number	0.50	0.50	0.50	0.50	0.50	0.50	2
people	0.83	1.00	0.91	0.71	1.00	0.83	5
person_name	0.25	0.20	0.22	0.33	0.20	**0.25**	5
rooms	0.33	0.38	0.35	0.40	0.50	**0.44**	8
time	0.38	0.30	0.33	0.89	0.80	**0.84**	10
time_from	0.00	0.00	0.00	0.00	0.00	0.00	1
time_period	0.00	0.00	0.00	0.00	0.00	0.00	2
micro avg	0.45	0.46	0.45	0.50	0.51	**0.51**	98
macro avg	0.39	0.41	0.40	0.39	0.41	0.39	98
weighted avg	0.45	0.46	0.45	0.49	0.51	**0.48**	98

Table 12: Slot labeling evaluation result of LR_our and LR_intent on test set (numbers in bold show improvement on f1-score)

Appendix A: code s1

```
def train_tag_logreg(data):
    """
    Train a logistic regression model for p(<tag> | <word embedding>).

    Args:
        data: data in NLU++ format imported from JSON, with spacy annotations.
    Returns:
        Callable: a function that returns a (sorted) distribution over tags, given a word.
    """
    token_count = 0
    train_X = []
    train_y = []
    bio_tags = set(['O'])
    for sample in data:
        for token in sample['annotated_text']:
            bio_tags.add(token._.bio_slot_label)

    bio_tags = sorted(bio_tags)
    bio_tag_map = {tag: idx for idx, tag in enumerate(bio_tags)}

    for sample in data:
        for token in sample['annotated_text']:
            train_X.append(token.vector)
            train_y.append(bio_tag_map[token._.bio_slot_label])
            token_count += 1

    print(f'> Training logistic regression model on {token_count} tokens')
    model = sklearn.linear_model.LogisticRegression(multi_class='multinomial', solver='newton-cg').fit(train_X, train_y)
    print('> Finished training logistic regression')
    return lambda token: _predict_tag_logreg(token, model, bio_tags)
```

Appendix B: code s2

```
def predict_bio_tags(tag_predictor, data):
    '''
    Predict tags respecting BIO tagging constraints, implementing viterbi to find the optimal tag sequence.

    Args:
        tag_predictor: function that predicts a tag, given a spacy token
        data: data in NLU++ format imported from JSON, with spacy annotations

    Returns:
        List(List(Str)): a list (one for each sample) of a list of tags (one for each word in the sample)

    '''

    # compute the transition score
    def transition_score(pre_tag, cur_tag):
        if cur_tag[0] == 'I' and \
                (pre_tag[0] not in ['B', 'I'] or
                 pre_tag[2:] != cur_tag[2:]):
            return float("-inf")
        return 0.0

    predictions = []
    for sample in data:
        # viterbi start state (tag, optimal score, optimal previous tag)
        viterbi = [[('S', 0.0, None)]]

        # viterbi forward
        for token in sample['annotated_text']:
            predict_result = tag_predictor(token)
            emission_score = [result[1] for result in predict_result]

            cur_state = []
            # iterate on each predicted tag
            for cur_idx, result in enumerate(predict_result):
                best_pre = 0
                best_score = float("-inf")
                # iterate to find the best transition for each tag
```

```
                for pre_idx, pre_state in enumerate(viterbi[-1]):
                    total_score = pre_state[1] +
                    ↪  transition_score(pre_state[0], result[0])
                    if total_score > best_score:
                        best_score = total_score
                        best_pre = pre_idx
                cur_state.append((result[0], best_score +
                ↪  emission_score[cur_idx], best_pre))
            viterbi.append(cur_state)

        # viterbi backtrace
        prediction = []
        final_scores = [state[1] for state in viterbi[-1]]
        sel_tag_idx = final_scores.index(max(final_scores))
        state_idx = len(viterbi) - 1
        while state_idx > 0:
            prediction.insert(0, viterbi[state_idx][sel_tag_idx][0])
            sel_tag_idx = viterbi[state_idx][sel_tag_idx][2]
            state_idx -= 1
        predictions.append(prediction)

    return predictions
```

Appendix C: code s3

```
def _predict_tag_lincrf(tokens, position, model, tags):
    """
    Predict tag according to logistic regression on word embedding and pre-defined features
    Args:
        data: data in NLU++ format impored from JSON, with spacy annotations.
    Returns:
        (List(Tuple(str,float))): distribution over tags, sorted from most to least probable
    """

    # compute features
    features = OrderedDict()
    features['head_lemma_is_room'] = np.array([float(tokens[position].head.lemma_ == 'room')])

    final_input = tokens[position].vector
    for feature in features.values():
        final_input = np.concatenate((final_input, feature))
    log_probs = model.predict_log_proba([final_input])[0]
    distribution = [tag_logprob_pair for tag_logprob_pair in zip(tags, log_probs)]
    sorted_distribution = sorted(distribution, key=lambda tag_logprob_pair: -tag_logprob_pair[1])
    return sorted_distribution


def train_tag_lincrf(data):
    """
    Train a linear CRF model to compute the probability of current tag.
    Args:
        data: data in NLU++ format imported from JSON, with spacy annotations.
    Returns:
        Callable: a function that returns (sorted) scores over tags, given a word sequence and current position.
    """
    token_count = 0
    train_X = []
```

```
    train_y = []
    bio_tags = set(['O'])
    for sample in data:
        for token in sample['annotated_text']:
            bio_tags.add(token._.bio_slot_label)

    bio_tags = sorted(bio_tags)
    bio_tag_map = {tag: idx for idx, tag in enumerate(bio_tags)}

    for sample in data:
        for idx, token in enumerate(sample['annotated_text']):
            # compute features
            features = OrderedDict()
            features['head_lemma_is_room'] =
            ↪  np.array([float(token.head.lemma_ == 'room')])

            final_input = token.vector
            for feature in features.values():
                final_input = np.concatenate((final_input, feature))
            train_X.append(final_input)
            train_y.append(bio_tag_map[token._.bio_slot_label])
            token_count += 1

    print(f'> Training logistic regression model on {token_count} tokens')
    model =
    ↪  sklearn.linear_model.LogisticRegression(multi_class='multinomial',
    ↪  solver='newton-cg').fit(train_X, train_y)
    print('> Finished training logistic regression')
    return lambda tokens, position: _predict_tag_lincrf(tokens, position,
    ↪  model, bio_tags)


```

Bibliography

Casanueva, I., Vulić, I., Spithourakis, G., & Paweł Budzianowski. (2022). Nlu++: A multi-label, slot-rich, generalisable dataset for natural language understanding in task-oriented dialogue [Data available at https://www.youtube.com/watch?v=dQw4w9WgXcQ]. *TODO*. https://www.youtube.com/watch?v=dQw4w9WgXcQ

Honnibal, M., & Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing* [To appear].