Grounding NBA Matchup Summaries

Tadashi Nomoto
National Institute of Japanese Literature
Tachikawa Tokyo 190-0014, Japan
nomoto@acm.org

Abstract

The present paper summarizes an attempt we made to meet a shared task challenge on grounding machine generated summaries of NBA matchups.[1] In the first half, we discuss methods and in the second, we report results, together with a discussion on what feature may have had an effect on the performance.

1 Introduction

What led Reiter and Thomson (2020) to launch a shared task competition in 2020 was a concern that fact-checking automatically generated texts (machine texts, or M_TEXTs) in the context of data to text generation (Wiseman et al., 2017), is hugely labor intensive, making it virtually impossible to run it at a scale. In an effort towards putting it under control, the project asks participants to find a way to do the assessment automatically, without any human intervention. The problem is set out as follows: you receive M_TEXTs, along with other external information such as box scores and human created summaries (or H_TEXTs). Your goal is to locate factual mistakes in M_TEXTs and classify them according to a pre-defined scheme of error types (‘*word,*’ ‘*number,*’ ‘*name,*’, ‘*context,*’ ‘*not checkable*’).

2 Method

The following sections detail our approach, which in essence is a multi-pronged strategy. We deploy separate mechanisms to deal with different types of error.

2.1 Detecting Word/Name Errors

We split an M_TEXT into three parts, LEAD, MIDDLE, TAIL (Figure 1), and use a separate set of rules targeting a particular part of the text, to identify errors with word or name.

2.1.1 Lead Section

For the lead section, we focus on date (day of week, DOW) and venue, in particular those located in the first 3 sentences of an M_TEXT. We compare each sentence (call it an m-sentence)[2] in the lead against names of US basketball arenas listed in Wikipedia[3] to get one most similar (based on how much they overlap) and use it as a canonical name. We locate a date expression by going through each token in an *m*-sentence and pick one that best matches a DOW name we prepared beforehand. We report a name error if there is any conflict between M_TEXT and H_TEXT in DOW or in venue. We do not work with a full sentence. Rather, we work with a *clause*, a minimal sentential unit that serves a building block of a complex sentence.[4] This is meant to ensure that we have no more than one occurrence of a venue and a date in an input we feed to the process. We call a clause contained in ‘m-sentence,’ an m-clause and that in h-sentence (see Fn. 2), an ‘h-clause.’ See Algorithm 1 for a more precise picture of what we do here. **search**(X, Y) goes over each of strings given in X to tell if it exists in Y.

2.1.2 Middle Section

In this part, we intend to determine whether a state of affairs described by a cue word holds up, by querying box-office scores. Cue words include words like ‘next,’ ‘led,’ ‘bench,’ and ‘defeated,’ which make a specific verifiable statement about players and teams. We go through each sentence, to see if it has a player name together with a cue

[1] https://github.com/ehudreiter/accuracySharedTask.git

[2] Similarly we mean by ‘h-sentence’ a sentence that occurs in H_TEXT.

[3] https://en.wikipedia.org/wiki/List_of_National_Basketball_Association_arenas

[4] We identify and isolate clauses by breaking up a sentence using a dependency tag ‘mark’ provided by spaCy (https://spacy.io/). For details on what the tag means, consult https://universaldependencies.org/docs/en/dep/mark.html.

Proceedings of the 14th International Conference on Natural Language Generation (INLG), pages 276–281,
Aberdeen, Scotland, UK, 20-24 September 2021. ©2021 Association for Computational Linguistics

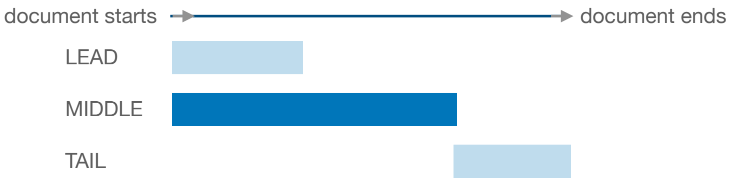

Figure 1: We apply rule based heuristics to different parts of the text to identify errors.

Table 1: Cue Words

single-word cues	*next, bench, reserve, starter, led, leader, leads, best, paces, pace, pacing, paced*
multi-word cues	*player of the game, team - high, high - point, double figures, double - double, triple - double*

Algorithm 1 Finding a name error in LEAD

Input: m-clause, h-clause, DOWs, Venues
Output: **True** or **False**

$H \leftarrow h$-clause ▷ String
$M \leftarrow m$-clause ▷ String
$\mathcal{D} \leftarrow$ DOWs ▷ Pre-def. List of Strings
$\mathcal{V} \leftarrow$ Venues ▷ Pre-def. List of Strings
$d_h \leftarrow$ **search**$(\mathcal{D}, H)$ ▷ returns a date in H
$d_m \leftarrow$ **search**$(\mathcal{D}, M)$ ▷ returns a date in M
$v_h \leftarrow$ **search**$(\mathcal{V}, H)$ ▷ returns a venue in H
$v_m \leftarrow$ **search**$(\mathcal{V}, M)$ ▷ returns a venue in M
if $d_h \neq d_m$ **or** $d_h \neq d_m$ **then**
  return False
else
  return True
end if

Algorithm 2 Finding a word error in MIDDLE

Input: m-clause, Box-Scores, Cue Words, Player Names
Output: **True** or **False**

$S \leftarrow m$-clause ▷ String
$\mathcal{X} \leftarrow$ **pd_load**(Box Scores) ▷ Load into Pandas
$\mathcal{C} \leftarrow$ Cue Words ▷ Pre-def. List of Strings
$\mathcal{P} \leftarrow$ Player Names ▷ Pre-def. List of Strings
for each $c \in \mathcal{C}$ **do**
  for each $p \in \mathcal{P}$ **do**
    if match(s, S) & **match**(p, S) **then**
      $T \leftarrow$ **ask_pandas**$(c, p, \mathcal{X})$ ▷ Ask Pandas if $\mathcal{X}$ supports what c says about p.
      return T
    end if
  end for
end for

word. If found, we go to the box score to decide whether it supports the statement. For example, if the text says that player A is off the bench, we know that for it to be true, the player should not be listed under starter. Or if the text states that the team is led by player A, it has to be the case that the player scored the most points. We flag the statement as correct or incorrect depending on whether it is supported by the box-office scores.

Listed in Table 1 are cue words we used, each of which indicates a particular state of affairs that can be checked with the box scores (which we have done using Pandas.)[5] We also break a sentence where possible into clauses (see Fn. 4). Algorithm 2 gives a general idea of how the process works. **match**(X, Y) is a boolean function that holds true if X is found in Y. We load the box scores into a Pandas' data frame prior to the loop operation. **ask_pandas** handles a query for the data frame, returns true if it finds a piece of data that matches the query and false if not. The code shown in Table 2, for instance, asks whether a player started off the bench.

2.1.3 Tail Section

For this part, our goal is to see if there is any error about future matchups. We gather matchup information, such as date (day of week), home name, visitor name from the last two sentences of M_TEXT and check them against a corresponding part of H_TEXT. Specific operations involved are shown in Algorithm 3. **find_matchup** looks for home name, visitor name and date in a clause given as input. It works on both m- and h-clause.

[5] https://pandas.pydata.org/

Table 2: A code in Pandas

```
data_frame.loc[['START_POSITION'],[player_name]].values.flatten()[0]
```

Algorithm 3 Finding a name error in TAIL

Input: m-clause, h-clause, DOWs, Team Names
Output: **True** or **False**

$H \leftarrow h$-clause ▷ String
$M \leftarrow m$-clause ▷ String
$\mathcal{D} \leftarrow$ DOWs ▷ Pre-def. List of Strings
$\mathcal{N} \leftarrow$ Team Names ▷ Pre-def. List of Strings
$m_a, m_b, m_c =$ **find_matchup**$(M, \mathcal{N}, \mathcal{D})$
▷ m_a, m_b, m_c represent home name, visitor name, date found in m-clause, respectively
$h_a, h_b, h_c =$ **find_matchup**$(H, \mathcal{N}, \mathcal{D})$
▷ h_a, h_b, h_c represent home name, visitor name, date found in h-clause, respectively.
if $m_a \neq h_a$ **and** $m_b \neq h_b$ **and** $m_c \neq h_c$ **then**
  return Not_Checkable
end if
if $m_a \neq h_a$ **or** $m_b \neq h_b$ **or** $m_c \neq h_c$ **then**
  return False
else
  return True
end if

In case the search is successful with m-clause but not with h-clause (meaning that none of the targets was found in h-caluse), we stop, reporting that they are unverifiable or uncheckable. Otherwise, we look for a discrepancy between triplets in m- and h-clause, and report an error if any is found.

We collectively call a set of rules we brought together for detecting word/name mistakes, 'WED,' hereafter.

2.2 Detecting Number Errors

2.2.1 Building Training Data

In detecting number errors, we essentially rely on data_utils.py[6] (UTL, hereafter) which extracts from the Rotowire dataset, what we call 'relation quadruples' (relQs), each of which contains information on who scored what points in what category.[7] Having relQs at hand is a useful first step towards error detection as they can tell us where to look for potential errors. For example, given a sentence "*Marco and Spencer came off the bench to combine for 31 points, eight rebounds and 10 assists as well.*", UTL would output relQs like those shown in Table 4. OFFSET indicates where the relevant number starts in the sentence.

We recognize however two problems with UTL: (1) it allows a number to get associated with more than one relation; (2) it could fail to assign any relation at all. Our plan is to avoid these annoyances by bootstrapping UTL with a neural model to predict a correct relation given a player name, a number and a context, i.e. a sentence, in which they occur.

In a move in this direction, we transform relQs into source-label pairs of the form shown in Table 3. The process involves acquiring an m-sentence where a relQ comes from, replacing a player name with '@' and a target number (one for which we are trying to find a relation) with '#,' with all other numbers reduced to '⟨NUMBER⟩.'

In addition, we made sure that each relQ we use for training is supported by the box-office scores, that is, evidence exists in the box scores that demonstrates the veracity of the relQ. This means that we accept relQs in Table 4 as training data only if there are records in the box scores showing that Macro had 31 points, 8 rebounds, and 10 assists. If not, they are all discarded. Also dismissed are relQs where a number occurs ahead of a player name (Table 5).

Moreover, in case a number gets assigned to more than one relQ, the preference is given to one that is consistent with a word that immediately follows that number (shots, rebounds, assists). For example, if we have a sentence '*Macro led the team with a spectacular output of 31 points.*' for which UTL may give ('Marco', OFFSET_0, '31', 'PTS') and ('Marco', OFFSET_0, '31', 'AST'), we will take the first relQ and drop the second, as it contradicts what the sentence says about how the number came about (it is not about how many assists he made).

[6] https://github.com/harvardnlp/data2text/

[7] UTL works by locating a player name and a number in a sentence and searching box office scores for records that match the name and the number. It returns all the matches, together with relevant categories, e.g. points, rebounds, assists, steals, blocks, threes, field-goal percentage, free-throw percentage, etc. If the search fails, it returns a relQ with a category named 'NONE.' Throughout the paper, we refer to categories as *relations*, following Wiseman et al. (2017).

Table 3: Source Label Pairs. '@' is a proxy for a person name and '#' that for a numeral of interest.

SOURCE	LABEL
@ and Spencer came off the bench to combine for # points , ⟨NUMBER⟩ rebounds and ⟨NUMBER⟩ assists as well .	PTS
@ and Spencer came off the bench to combine for ⟨NUMBER⟩ points , # rebounds and ⟨NUMBER⟩ assists as well .	REB
@ and Spencer came off the bench to combine for ⟨NUMBER⟩ points , ⟨NUMBER⟩ rebounds and # assists as well .	AST

Table 4: Relation Quadruples, each composed of player name, location, number (points), and label (i.e. category in which points are earned).

('Marco', OFFSET_0, '31', 'PTS')
('Marco', OFFSET_1, '8', 'REB')
('Marco', OFFSET_2, '10', 'AST')

Table 5: Player name has to appear ahead of number. 'w' represents an arbitrary word.

ALLOW	DISALLOW
w w @ w w # w w w	w w w w w # w @ w
@ w w w w # w w w	w w w w w # @ w w
w @ w w w # w w w	w w w w w # w w @

2.2.2 Model

The training data are fed into an LSTM-based Sequence to Label classifier (bidirectional, batch-normalized with the RELU non-linearity):

$$o = \mathrm{softmax}(r(\ell_2(r(\ell_1(m(\mathbf{W})) \tag{1}$$

$\mathbf{W}$ is an input (a sequence of words that represents a sentence (see Table 3)) where each token is replaced by a word embedding from GloVe,[8] $r(\cdot)$ denotes the RELU activation, $\ell(\cdot)$ a fully connected layer and $m(\cdot)$ a bidirectional LSTM, all of which were built with PyTorch.[9]

After processing the test set in the same way as we did with the training set, we run the model (Eqn. 1), making a prediction about the relation for each relQ instance we find in the text. We label a relQ instance as wrong if it is predicted to have a relation inconsistent with one given by UTL.[10] We refer to the model described here as 'NED.'

3 Resolving Coreference

Given the way UTL works, it is important that we make explicit what a referring expression points to, in order for UTL to successfully build a relQ. To this end, we make use of NeuralCoref 4.0,[11] which operates as an add-on functionality for spaCy.[12] Resolving coreferences with NeuralCoref (NC) results in every referring expression (r-expression, hereafter) in a text being replaced with a corresponding root entity (i.e. its canonical name). This can be troublesome though, because it may disrupt the way in which words are originally laid out, which we need to retain in order to report results conformant to the shared task format policy (which asks to report errors by indicating where they are in the original position). In response, we pursued an approach where we represented a text with a linked list structure in which each word is represented as a node which contains information on what node it is preceded by and what it is followed by, in addition to where it occurs relative to others.[13] For each r-expression NC found, we replaced a token string held by a relevant node with its antecedent while keeping other information (occurrence site, forward/backward connections) in tact (Fig. 2). Furthermore, we restricted an r-expression subject to replacement, to be among 'their,' 'they,' 'he,' 'his,' 'its,' 'it,' and 'him.'

4 Setup and Results

The training data that NED used are sourced from part of the Rotowire corpus (Wiseman et al., 2017),[14] called 'train.json,' which contains 3,398 matchup results each with a summary manually

[8] https://nlp.stanford.edu/projects/glove/
[9] https://pytorch.org/
[10] For instance, we take the following situation as mistake.
UTL ouptut: ('Marco', 0, '31', 'NONE')
Prediction: ('Marco', 0, '31', 'PTS')
[11] https://github.com/huggingface/neuralcoref.git
[12] https://spacy.io/
[13] A node is a structure schematically defined as:
node := ⟨word-token, preceded-by, followed-by, position⟩
[14] https://github.com/harvardnlp/boxscore-data.git

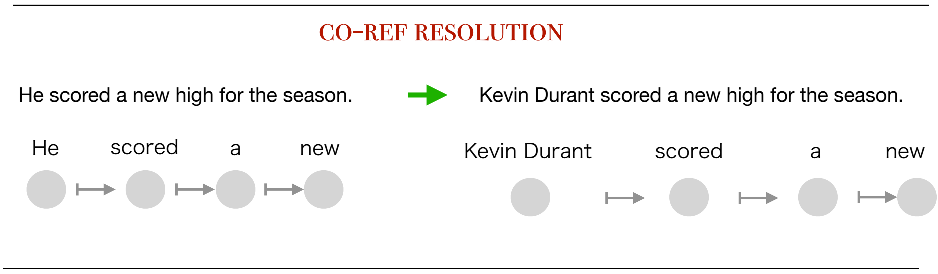

Figure 2: Doing coreference resolution without disrupting the token position

Table 6: Comparison of Per-Type Performance: Coref. vs. Non-Coref. (Shared Task 2021 Train)

	Coref.				Non-Coref.			
	Mistake		Token		Mistake		Token	
	recall	precision	recall	precision	recall	precision	recall	precision
name	0.356	0.958	0.259	0.958	0.356	0.958	0.259	0.958
number	0.781	0.561	0.764	0.561	0.641	0.476	0.628	0.476
word	0.398	0.364	0.311	0.394	0.398	0.364	0.311	0.394
context	0.000	-	0.000	-	0.000	-	0.000	-
not checkable	0.000	-	0.000	-	0.000	-	0.000	-
macro summary	0.549	0.556	0.410	0.553	0.496	0.511	0.375	0.514

Table 7: Results on Shared Task 2021 Test

	Mistake		Token	
model	recall	precision	recall	precision
WED/NED	0.523	0.494	0.349	0.505

created, providing 116,579 source-label pairs in total. The official test set, carved out of the Rotowire corpus, contained machine generated 30 matchup reports and associated box records. We trained NED for 50 epochs on the extracted pairs, achieving the classification accuracy of 0.461 on the test set (over 34 labels). We ran WED on summaries included in the test set (M_TEXTs). It also made use of human summaries provided as part of the Rotowire data (H_TEXTs). Results on Accuracy Shared Task 2021 Test (30 M_TEXTs) are shown in Table 7. The figures are in macro precision and recall. Table 6 gives a per-type comparison of coreference enabled (CrE) versus disabled (CrD) approach, on the train part of 2021 Shared Task (60 M_TEXTs), which reveals a clear advantage of CrE over CrD in the number category (highlighted in blue and red).[15]

[15]It implies that NeuralCoref, by making NED more accurate in predicting relations in relQs, may have pushed higher the system's performance in detecting number errors. We were not able to find any meaningful difference between CrE and CrD on the 2021 Test.

5 Conclusion

This paper gave an overview of the approach we took to meet the shared task challenge for 2021, which is essentially a combination of hand crafted rules (WED) and machine learning (NED): it relied on rule based heuristics to identify errors in name and word while bringing in a neural model to locate number errors. The rule based part consisted of 'translating' a cue expression into a procedure to query box scores for its veracity, while the machine learning part was driven by a neural model, whose predictions allowed us to detect inconsistencies in number related statements in M_TEXTs.[16]

[16]Response to Organizers' Question (which is about how we would cope with unknown player names, venues and teams). Bringing up to date the database we use for WED, of players, teams and venues through Wikipedia and sources devoted to NBA conferences (e.g. box-office scores), could lessen a possible negative impact due to the lack of exposure to data not available at the time of training. If that does not work, we may go to some off-the-shelf NE tool such as one by spaCy, though we expect it may hurt name error detection (due to its reduced accuracy), and to a lesser degree word error detection as it does not involve the recognition of venue names (player and team names can easily be picked by looking at box-office scores which we assume are available all the time). Cue expressions we used were fairly generic. It is highly unlikely that they become less effective on summaries beyond the training data, though we recognize the need to expand the list.

References

Ehud Reiter and Craig Thomson. 2020. Shared task on evaluating accuracy. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 227–231, Dublin, Ireland. Association for Computational Linguistics.

Sam Wiseman, Stuart M. Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation.