A Hybrid Approach to Mining Conditions

Fernando O. Gallego(✉) and Rafael Corchuelo(✉)

ETSI Informática, University of Seville,
Avda. Reina Mercedes, s/n, Sevilla, Spain
{fogallego,corchu}@us.es

Abstract. Text mining pursues producing valuable information from natural language text. Conditions cannot be neglected because it may easily lead to misinterpretations. There are naive proposals to mine conditions that rely on user-defined patterns, which falls short; there is only one machine-learning proposal, but it requires to provide specific-purpose dictionaries, taxonomies, and heuristics, it works on opinion sentences only, and it was evaluated very shallowly. We present a novel hybrid approach that relies on computational linguistics and deep learning; our experiments prove that it is more effective than current proposals in terms of F_1 score and does not have their drawbacks.

1 Introduction

Text mining pursues processing natural language text to produce useful information. Unfortunately, current state-of-the-art text miners do not take conditions into account, which may easily result in misinterpretations. For instance, given sentence "Let it happen and John will leave Acme", current entity-relation extractors [6,12] return fact ("John","will leave","Acme"); similarly, current opinion miners [17,19] return a negative score since "will leave" typically conveys a negative opinion. Neglecting the conditions clearly results in misinterpretations.

The simplest approach to mine conditions consists in searching for user-defined patterns [3,10], which falls short regarding recall because there are many common conditions do not fit common patterns. There is only one machine-learning approach [13], but it must be customised with several specific-purpose dictionaries, taxonomies, and heuristics, and it mines conditions regarding opinions only, not to mention that it was evaluated very shallowly.

In this paper, we present a proposal to mine conditions that hybridises computational linguistics and deep learning, without any of the previous problems. We have performed a comprehensive experimental analysis on a dataset with 3 779 000 sentences on 15 common topics in English and Spanish; our results prove that our approach is comparable to others in terms of precision [5], but improves recall enough to beat them in terms of F_1 score [22].

Supported by Opileak.com and the Spanish R&D programme (grants TIN2013-40848-R and TIN2013-40848-R). The computing facilities were provided by the Andalusian Scientific Computing Centre (CICA). We also thank Dr. Francisco Herrera for his hints on statistical analyses and sharing his software with us.

The rest of the paper is organised as follows: Sect. 2 provides an insight into the related work; Sect. 3 describes our proposal; Sect. 4 reports on our experimental analysis; finally, Sect. 5 presents our conclusions.

2 Related Work

Narayanan et al. [14] range amongst the first authors who realised the problem with conditions in the field of opinion mining. However, they did not report on a proposal to mine them.

The simplest approaches to mine conditions build on searching for user-defined patterns. Mausam et al. [10] studied the problem in the field of entity-relation extraction and suggested that conditions might be identified by locating adverbial clauses whose first word is one of the sixteen one-word condition connectives in English; unfortunately, they did not report on the effectiveness of their approach to mine conditions, only on the overall effectiveness of their proposal for entity-relation extraction. Chikersal et al. [3] proposed a similar, but simpler approach: they searched for sequences of words in between connectives "if", "unless", "until", and "in case" and the first occurrence of "then" or a comma. Unfortunately, the previous proposals are not generally appealing because handcrafting such patterns is not trivial and the results typically fall short regarding recall, as our experimental analysis confirms.

The only existing machine-learning approach was introduced by Nakayama and Fujii [13], who worked in the field of opinion mining. They devised a model that is based on features that are computed by means of a syntactic parser and a semantic analyser. The former identifies so-called "bunsetus", which are Japanese syntactic units that consists of one independent word and one or more ancillary words, as well as their inter-dependencies; the latter identifies opinion expressions, which requires to provide several specific-purpose dictionaries, taxonomies, and heuristics. They used Conditional Random Fields and Support Vector Machines to learn classifiers that make "bunsetus" that can be considered conditions apart from the others. Unfortunately, their approach was only evaluated on a small dataset with 3 155 Japanese sentences regarding hotels and the best F_1 score attained was 0.5830. As a conclusion, this proposal is not generally applicable and its effectiveness is poor for practical purposes.

Our conclusion is that mining conditions is a problem to which researchers are paying attention recently because it is a must for software agents to mine text properly so as to avoid misinterpretations. Unfortunately, the few existing techniques have many drawbacks that hinder their general applicability. This motivated us to work on a new approach that overcomes their weaknesses and outperforms them by means of a hybrid approach that combines computational linguistics and deep learning; our proposal only requires a stemmer, a dependency parser, and a word embedder, which are readily-available components.

3 A Hybrid Approach to Mining Conditions

In this section, we first describe the main methods of our proposal, which work in co-operation to learn a regressor that assesses the candidate conditions in a sentence before the most promising ones are returned; then, we describe some ancillary methods to generate candidate conditions, to compute their scores, to set up a regressor using a deep neural network, and to remove overlapping candidate conditions. We use sentence "If you're someone who likes cakes, then try John's." as a running example where appropriate.

3.1 Description of the Main Methods

The main methods are sketched in Fig. 1, namely: method *train*, which is used to learn a regressor that assesses candidate conditions, and method *apply*, which selects the best candidate conditions in a sentence and returns them.

Method *train* takes a dataset ds as input and returns a regressor r. The input dataset is of the form $\{(s_i, L_i)\}_{i=1}^{n}$, where each s_i denotes a sentence and each L_i denotes a set of labels that identify the conditions in that sentence ($n \geq 0$). The output regressor is a function that given a candidate condition returns a score that assesses how likely it is an actual condition. The method first initialises training set T to the empty set and then loops over dataset ds; for each sentence s and set of labels L in ds, it first computes a set of candidate conditions; then, for each condition c, it computes a score σ and stores a tuple of the form (c, σ) in training set T. When the main loop finishes, it learns a regressor from T using a deep-learning approach.

Method *apply* takes a sentence s, a regressor r, and a threshold θ as input and returns a set R of tuples of the form $\{(c_i, \sigma_i)\}_{i=1}^{m}$, where each c_i denotes a condition and σ_i its corresponding score, which must be equal or greater than the threshold ($m \geq 0$). The method first generates the candidate conditions in s, stores them in set C, and initialises the result R to an empty set; it then iterates over set C; for each candidate condition c in set C, it first computes its score by applying regressor r to it; if it is equal or greater than threshold θ, then

```
 1: method train(ds) returns r
 2:     T := ∅
 3:     for each (s, L) ∈ ds do
 4:         C := generateCandidates(s)
 5:         for each c ∈ C do
 6:             σ := computeScore(c, L)
 7:             T := T ∪ {(c, σ)}
 8:         end
 9:     end
10:     r := learnRegressor(T)
11: end
```

```
 1: method apply(s, r, θ) returns R
 2:     C := generateCandidates(s)
 3:     R := ∅
 4:     for each c ∈ C do
 5:         σ := apply r to c
 6:         if σ > θ then
 7:             R := R ∪ {(c, σ)}
 8:         end
 9:     end
10:     R := removeOverlaps(R)
11: end
```

Fig. 1. Main methods of our proposal.

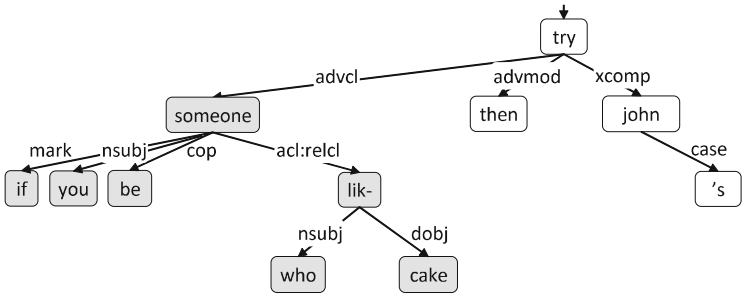

Fig. 2. Sample dependency tree.

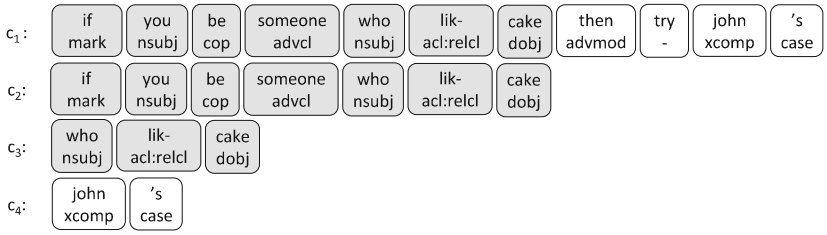

Fig. 3. Sample candidate conditions.

candidate condition c is added to the result set. When the main loop finishes, R provides a collection of candidates and scores; before returning it, we must remove the conditions that overlap others with a higher score.

3.2 Method to Generate Candidates

Our first ancillary method is *generateCandidates*, which takes a sentence as input and returns a set of candidate conditions. A naive approach would simply generate as many sub-strings as possible, but it would be very inefficient because a sentence with n words has $O(n^2)$ such sub-strings. In order to reduce the candidate space we use a dependency tree to generate them since conditions are clauses from a grammatical point of view and the non-leaf nodes of a dependency tree typically represent many such clauses.

Method *generateCandidates* first computes the dependency tree of the input sentence, then changes the words in its nodes to lowercase, and finally stems them. Now, for each non-leaf node in the dependency tree, we compute all of the sequences of tokens that originate from that node; a token is a tuple of the form (w, d), where w denotes a stem and d the dependency tag that links its corresponding node in the dependency tree to its parent, if any. Note that we do not select leaf nodes because we have not found a single example in which one word can be considered a condition. As a conclusion, a condition is modelled as a sequence of the form $\langle (w_i, d_i) \rangle_{i=1}^{n}$ where each w_i is a stem and d_i is its corresponding dependency tag ($n \geq 2$).

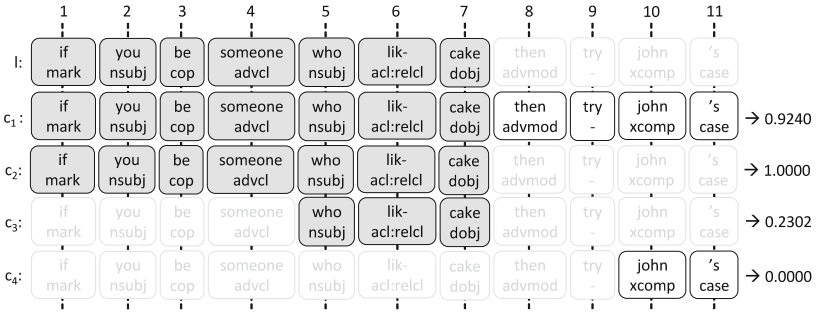

Fig. 4. Sample matchings.

Example 1. Figure 2 shows the dependency tree of our running example; the nodes that correspond to the condition are highlighted in grey. Figure 3 shows the candidates that are generated from the previous dependency tree. Candidate c_1 is generated from the root node, candidate c_2 is generated from the node with stem "someone", candidate c_3 is generated from the node with stem "lik-", and candidate c_4 is generated from the node with stem "john". Note that, as expected, the condition is confined to one of the nodes in the dependency tree.

3.3 Method to Compute Scores

Our second ancillary method is *computeScore*, which takes a candidate condition c and a set of labels L as input and returns its corresponding score. A naive approach would simply return 0.0000 if c does not exactly match any of the labels in L and 1.0000 otherwise, but that is too crisp. An approach in which a candidate gets a score in range $[0.0000, 1.0000]$ better captures the chances that it is an actual condition. We use an approach that is based on the well-known F_1 score in order to balance the precision and the recall of candidate conditions.

The F_1 score is computed as $\frac{2\,tp}{(tp+fp)+(tp+fn)}$, where tp, fp, and fn denote, respectively, the number of true positives, false positives, and false negatives. Given a candidate condition c and a label l, it makes sense to interpret the tokens that they have in common as true positive tokens, the tokens in c that are not in l as false positive tokens, and the tokens in l that are not in c as false negative tokens. We also realised that the first few tokens in a condition typically provide a landmark that characterises it. Thus, we decided to measure the degree of matching between a condition and a label as follows:

$$match(c, l) = \sum_{i=1}^{|l|} \left\{ \begin{array}{ll} 1/i & \text{if } l_i \in c \\ 0 & \text{otherwise} \end{array} \right\} \tag{1}$$

Simply put: let l_i denote the i-th token in the label ($i = 1..|l|$); if l_i is in the candidate, we then add $1/i$ to the score and zero otherwise. This way, the first few tokens in the label contribute much more to the score than the remaining ones.

That is, given a candidate condition c and a label l, $match(c, l)$ is a measure of the number of true positive tokens in c.

Given the previous definition, the maximum degree of matching for a candidate condition or a label x is defined as follows:

$$match^*(x) = \sum_{i=1}^{|x|} {}^1/_i \tag{2}$$

Realise that given a candidate condition c, $match^*(c)$ is a measure of the number of true positive tokens (the tokens that belong to both the candidate condition and the label) and the false positive tokens (the tokens that belong to the candidate, but not to the label); similarly, given a label l, $match^*(l)$ is a measure of the number of true positive tokens (the tokens that belong to both the label and the candidate condition) and the number of false negative tokens (the tokens that belong to the label, but not to the candidate condition).

Our proposal to compute the score of candidate condition c with respect to the set of labels L is then as follows:

$$score(c, L) = \max_{l \in L} \frac{2\ match(c, l)}{match^*(c) + match^*(l)} \tag{3}$$

Note the similarity to the F_1 score since $match(c, l)$, $match^*(c)$, and $match^*(l)$ are measures of tp, $tp + fp$, and $tp + fn$, respectively; the difference is that we do not count the actual number of true positive, false positive, or false negative tokens, but a measure that puts an emphasis on the first few tokens and decays asymptotically.

Example 2. Figure 4 shows label l, which corresponds to the condition in our running example, and how the candidate conditions match it. Candidate condition c_1 represents the whole input sentence, which obviously contains the label, i.e., seven true positive tokens, but also four false positive tokens, which results in a score of 0.9240. Candidate c_2 matches the label perfectly, i.e., it matches seven true positive tokens and no false positive or false negative token, which results in a score of 1.0000. Candidate c_3 is a partial match with three true positive tokens and three false negative tokens, which results in a score of 0.2302. Finally, condition c_4 does not match any true positive token, but two false positive tokens and seven false negative tokens, which results in a score of 0.0000.

3.4 Model to Learn a Regressor

Our third ancillary method is *learnRegressor*, which takes a training set T as input and returns a regressor r. We implemented it using a deep-learning approach because of its natural ability to transform data into feature-based representations that help learn good regressors.

Prior to learning a regressor, the candidate conditions in the training set must be vectorised. Our labelled dataset, which is presented in Sect. 4, suggests that the length of common conditions ranges from a few tokens to a few dozens,

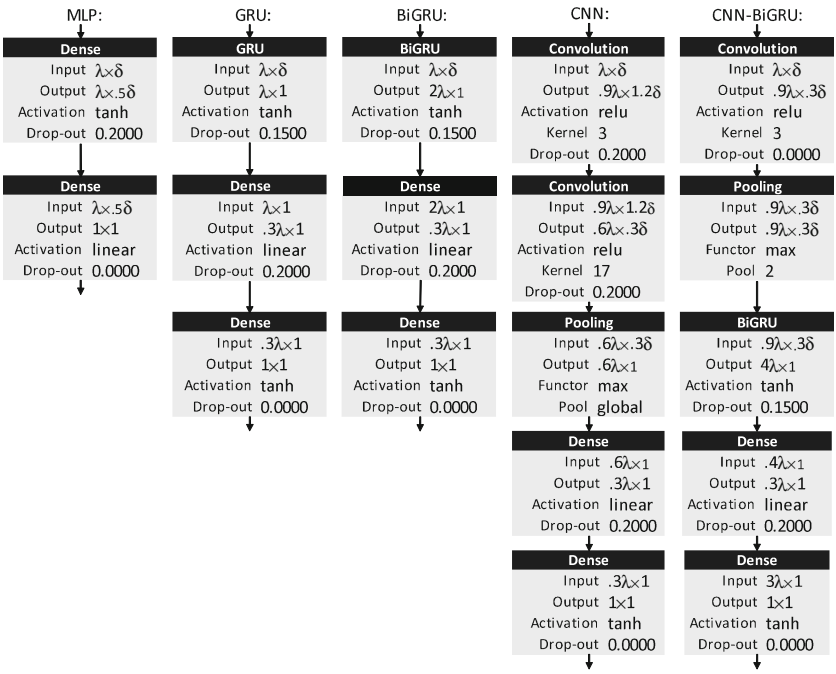

Fig. 5. Neural networks for learning regressors.

so it makes sense to represent them as large-enough fixed-size sequences. Given a candidate condition of the form $\langle (w_i, d_i) \rangle_{i=1}^{n}$, we transform it into a sequence of the form $\langle \overline{w_i} \oplus \overline{d_i} \rangle_{i=1}^{\lambda}$, where $\overline{w_i}$ denotes the vectorisation of stem w, $\overline{d}$ denotes the vectorisation of dependency tag d, $\overline{w} \oplus \overline{d}$ the vector that results from catenating the previous ones, and λ denotes the size of the longest possible condition; note that padding tokens need to be added if the original condition is shorter than λ.

Given a stem w, we compute its vectorisation $\overline{w}$ by using word embedding [11], which can unsupervisedly produce vectors that preserve some semantic relationships amongst the original stems; to reduce the stem space, we replaced numbers, email addresses, URLs, and stems whose frequency is equal or smaller than five by class words "NUMBER", "EMAIL", "URL", and "UNK", respectively. Given dependency tag d, we compute its vectorisation $\overline{d}$ by means of one-hot encoding [23], which vectorises a finite set of tags using binary features. Note that the vectorisation of a condition can then be interpreted as a matrix with λ rows and δ columns, where δ denotes the dimensionality of the word embedding vectorisation plus the dimensionality of the one-hot vectorisation.

Our baseline architecture was a multi-layer perceptron (MLP) with two dense layers. We devised a dozen more architectures, but the best ones were based on the following components: gated recurrent units (GRU), bi-directional gated recurrent units (BiGRU), convolutional neural networks (CNN), and a hybrid

approach that combines convolutional neural networks and bi-directional gated recurrent units (CNN-BiGRU).

GRUs are a kind of recurrent neural network (RNN) [8] and BiGRUs are a kind of bi-directional recurrent neural networks (BiRNN) [20]. In both RNNs and BiRNNs the connections between units form a directed cycle, which allows to apply them to sequences of varying length; the difference is that RNNs cannot take future elements in a sequence into account whereas BiRNNs can. Unfortunately, both RNNs and BiRNNs suffer from the so-called exploding/vanishing gradient problems [16], which is overcome by using gated recurrent units (GRUs) [4] or bi-directional gated recurrent units (BiGRUs) [15] that help control the amount of data that is passed on to the next epoch or forgotten. The CNN network [18] includes two convolutional layers and a pooling layer. The former consists of several filtering units that take a small region of the input data as input and applies a non-linear function to it; the latter consists of pooling units that apply a merging method to the results of the previous layer. Our proposal is to use a convolutional layer with a large number of filters in order to create a wide range of first-level features, but a smaller number of filters in the second convolutional layer to obtain a more specific range of second-level features that combine de first ones. Finally, the pooling layer combines the previous deep features using a global maximum function as the global pooling strategy since our experiments prove that it performs better than others. The CNN-BiGRU network uses a convolutional layer with a number of filters similar to the input length, and then applies a local pooling that captures the most relevant features only. We then apply a BiGRU layer that takes the dependencies between tokens into account, from both the beginning to the end of the sentences and vice versa.

Figure 5 summarises the previous architectures. The boxes represent the layers and provide information about the corresponding parameters, namely: in all cases, the input and output dimensions in terms of λ and δ (rounding to the closest natural number is assumed); in all cases, but pooling layers, the activation function and the drop-out ratio; in the case of convolutions, the kernel size; and, in the case of pooling layers, the functor and the pool used. The parameters were computed using the Stochastic Gradient Descent method [9] with batch size equal to 32. In order to prevent over-fitting as much as possible, we used some drop-out regularisations [21] and early stopping [2] when the loss did not improve enough after 10 epochs. We used the Mean Squared Error as the loss function since it is very common in regression problems [1]. We did not apply a decay momentum because we observed that the loss always converges smoothly, even if it needs more epochs for some networks than for others.

3.5 Criteria to Remove Overlaps

The fourth ancillary method is *removeOverlaps*, which takes a set of tuples of the form (c, σ) as input, where c denotes a candidate and σ its corresponding score, and filters some of them out.

It is the simplest method in our proposal. Basically, it works as follows: it iterates over the set of input tuples and removes those whose conditions overlap

a condition with a higher score. In other words, given the input set of tuples R, it computes the following subset:

$$\{(c,\sigma) \mid (c,\sigma) \in R \wedge \nexists (c',\sigma') : (c',\sigma') \in R \wedge c' \cap c \neq \langle\rangle \wedge \sigma' > \sigma\} \quad (4)$$

Example 3. Assume that the threshold to select the best candidates is set to $\theta = 0.5000$. In our running example, method *apply* would return candidate conditions c_1 and c_2 since they are the only whose scores exceed the threshold, cf. Fig. 4. Note that both candidate conditions overlap, so the one with the lowest score is filtered out. In this case, method *apply* would then return condition c_2 only, which, indeed, represents the condition in our running example.

4 Experimental Analysis

Computing Facility: We run our experiments on a virtual computer that was equipped with one Intel Xeon E5-2690 core at 2.60 GHz, 2 GiB of RAM, and an Nvidia Tesla K10 GPU accelerator with 2 GK-104 GPUs at 745 MHz with 3.5 GiB of RAM each; the operating system was CentOS Linux 7.3.

Prototype Implementation:[1] We implemented our proposal with Python 3.5.4 and the following components: Snowball 1.2.1 to stemmise words, the Stanford NLP Core Library 3.8.0 to generate dependency trees, Gensim 2.3.0 to compute word embedders using a Word2Vec implementation, and Keras 2.0.8 with Theano 1.0.0 to learn the regressors.

Evaluation Dataset:[2] We used a dataset with 3 779 000 sentences in English and Spanish that were randomly gathered from the Web between April 2017 and May 2017. The sentences were classified into 15 topics according to their sources, namely: adults, baby care, beauty, books, cameras, computers, films, headsets, hotels, music, ovens, pets, phones, TV sets, and video games. None of the conditions that we found in this dataset was smaller than two tokens or longer than 50 tokens, so we set those limits to vectorise candidate conditions.

Baselines: We used the proposals by Mausam et al. [10] and Chikersal et al. [3] as baselines. The proposal by Nakayama and Fujii [13] was not considered because it is not clear if it can be customised to deal with languages other than Japanese and its best F_1 was 0.5830; neither could we find an implementation.

Performance Measures: We measured the standard performance measures, namely: precision, recall, and the F_1 score. Regarding the baselines, we computed the measures from our dataset since there is no machine-learning involved; regarding our proposals, we computed the measures using 5-fold cross-validation. We computed the measures independently for each of our approaches and set threshold θ to 0.2500, 0.5000, and 0.7500.

[1] Available at https://github.com/FernanOrtega/HAIS18.
[2] Available at https://www.kaggle.com/fogallego/reviews-with-conditions.

Lang	Proposal	θ = 0.2500			θ = 0.5000			θ = 0.7500		
		P	R	F_1	P	R	F_1	P	R	F_1
en	MB	0.6270	0.6144	0.6206	0.6270	0.6144	0.6206	0.6270	0.6144	0.6206
	CB	0.7979	0.4642	0.5870	0.7979	0.4642	0.5870	0.7979	0.4642	0.5870
	Averages	0.7125	0.5393	0.6038	0.7125	0.5393	0.6038	0.7125	0.5393	0.6038
	MLP	0.4741	0.7799	0.5897	0.5612	0.5271	0.5436	0.5739	0.4582	0.5096
	GRU	0.9999	0.4421	0.6131	0.9999	0.4421	0.6131	0.9999	0.4421	0.6131
	BiGRU	0.5448	0.5262	0.5353	0.8999	0.4421	0.5929	0.9999	0.4421	0.6131
	CNN	0.5908	0.7546	0.6628	0.6211	0.6278	0.6244	0.6571	0.5432	0.5948
	CNN-BiGRU	0.5586	0.8052	0.6596	0.6318	0.6529	0.6422	0.7327	0.4914	0.5883
	Averages	0.6336	0.6616	0.6121	0.7428	0.5384	0.6033	0.7927	0.4754	0.5838
es	MB	0.6699	0.5285	0.5909	0.6699	0.5285	0.5909	0.6699	0.5285	0.5909
	CB	0.7953	0.4399	0.5665	0.7953	0.4399	0.5665	0.7953	0.4399	0.5665
	Averages	0.7326	0.4842	0.5787	0.7326	0.4842	0.5787	0.7326	0.4842	0.5787
	MLP	0.4232	0.8295	0.5604	0.5382	0.5678	0.5526	0.5771	0.4465	0.5034
	GRU	0.5246	0.7483	0.6168	0.7089	0.4304	0.5356	0.9999	0.4153	0.5869
	BiGRU	0.5321	0.7451	0.6209	0.6335	0.4692	0.5391	0.9999	0.4153	0.5869
	CNN	0.5997	0.7519	0.6672	0.6606	0.6521	0.6563	0.7065	0.5467	0.6164
	CNN-BiGRU	0.5227	0.8221	0.6390	0.6195	0.6968	0.6559	0.6843	0.5369	0.6017
	Averages	0.5205	0.7794	0.6209	0.6321	0.5633	0.5879	0.7935	0.4721	0.5790

Fig. 6. Experimental results in comparison with baselines.

θ = 0.2500

Proposal	Ranking	Comparison	z	p-value
CNN	1.0000	CNN x CNN	-	-
CNN-BiGRU	2.0000	CNN x CNN-BiGRU	1.4142	0.1573
BiGRU	3.5000	CNN x BiGRU	3.5355	0.0008
MLP	4.1000	CNN x MLP	4.3841	0.0000
GRU	4.4000	CNN x GRU	4.8083	0.0000

(a)

θ = 0.5000

Proposal	Ranking	Comparison	z	p-value
CNN-BiGRU	1.4000	CNN-BiGRU x CNN-BiGRU	-	-
CNN	1.6000	CNN-BiGRU x CNN	0.2828	0.7773
MLP	3.1000	CNN-BiGRU x MLP	2.4042	0.0324
BiGRU	4.2000	CNN-BiGRU x BiGRU	3.9598	0.0002
GRU	4.7000	CNN-BiGRU x GRU	4.6669	0.0000

(b)

θ = 0.7500

Proposal	Ranking	Comparison	z	p-value
CNN	1.3000	CNN x CNN	-	-
CNN-BiGRU	1.7000	CNN x CNN-BiGRU	0.5657	0.5716
MLP	3.0000	CNN x MLP	2.4042	0.0324
GRU	4.5000	CNN x GRU	4.5255	0.0000
BiGRU	4.5000	CNN x BiGRU	4.5255	0.0000

(c)

Proposal	Ranking	Comparison	z	p-value
$CNN_{0.25}$	1.4000	$CNN_{0.25}$ x $CNN_{0.25}$	-	-
$CNN\text{-}BiGRU_{0.50}$	1.8000	$CNN_{0.25}$ x $CNN\text{-}BiGRU_{0.50}$	0.5657	0.5716
MB	3.4000	$CNN_{0.25}$ xMB	2.8284	0.0094
$CNN_{0.75}$	3.7000	$CNN_{0.25}$ x$CNN_{0.75}$	3.2527	0.0034
CB	4.7000	$CNN_{0.25}$ x CB	4.6669	0.0000

(d)

Fig. 7. Statistical analysis based on Hommel's test.

Experimental Results: The experimental results are presented in Fig. 6. MB and CB refer to Mausam et al.'s and Chikersal et al.'s baselines, respectively. The greyed cells highlight the approaches that beat the best baseline.

The precision of the baselines is relatively good taking into account that they are naive approaches to the problem that rely on handcrafted user-defined patterns; Mausam et al.'s proposal achieves a recall that is similar to its precision, but Chikersal et al.'s proposal falls short regarding recall. Our approaches do not generally beat the baselines regarding precision, but attain results that are almost similar. It is regarding recall that most of our approaches beat the baselines since they are able to learn patterns that are more involved; thanks to our deep learning approach, the input sentences are projected onto a rich feature space that can capture many patterns that an expert cannot easily spot. Note that the improvement regarding recall is enough for the F_1 score to improve the

baselines. Regarding the value of threshold θ, note that increasing it increases the average precision of our approaches, but decreases their average recall.

To make a decision regarding which of the approaches performs the best, we used a stratified strategy that builds on Hommel's test [7]. In Figs. 7a, b and c, we report on the results of the statistical analysis regarding our proposal; our goal was to select the best ones for each of the values of threshold θ. The previous figures show the experimental rank of each approach, and then the comparisons between the best one and the others; for every comparison, we show the value of the z statistic and its corresponding adjusted p-value. Note that the experimental results do not provide any evidences that the best-ranked approach is different from the second one since the adjusted p-value is greater than the significance level; however, there is enough evidence to prove that it is different from the remaining ones since the adjusted p-value is smaller than the significance level. Our conclusion is that CNN is the best approach when $\theta = 0.2500$, CNN-BiGRU is the best approach when $\theta = 0.5000$, and CNN is again the best approach when $\theta = 0.7500$. In Fig. 7d, we present the results of comparing the previous best approaches and the baselines (we denote the corresponding value of θ using subindices). According to Hommel's test, CNN with $\theta = 0.2500$ is similar to CNN-BiGRU with $\theta = 0.5000$, but they are better than both baselines and CNN with $\theta = 0.7500$.

Our experimental analysis confirms that our best alternatives are CNN with $\theta = 0.2500$ or CNN-BiGRU with $\theta = 0.5000$, that they are similar to the other proposals in terms of precision, but improve recall enough to beat them in terms of F_1 score. In summary, it confirms that our approach is very promising.

5 Conclusions

We have presented a novel proposal to mine conditions. It relies on a hybrid approach that merges computational linguistics and deep learning as a means to overcome the problems that we have found in the literature, namely: it does not rely on user-defined patterns, it does not require any specific-purpose dictionaries, taxonomies, or heuristics, and it can mine conditions in both factual and opinion sentences. Furthermore it relies on a number of components that are readily available, namely: a stemmer, a dependency parser, and a word embedder. We have also performed a comprehensive experimental analysis on a dataset with 3 779 000 sentences on 15 common topics in English and Spanish. Our results confirm that our proposal can beat the state-of-the-art proposals in terms of recall and F_1 score.

References

1. Aravkin, A.Y., Burke, J.V., Chiuso, A., Pillonetto, G.: Convex vs non-convex estimators for regression and sparse estimation: the mean squared error properties of ARD and GLasso. J. Mach. Learn. Res. **15**(1), 217–252 (2014)
2. Caruana, R., Lawrence, S., Giles, C.L.: Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. In: NIPS, pp. 402–408 (2000)
3. Chikersal, P., Poria, S., Cambria, E., Gelbukh, A.F., Siong, C.E.: Modelling public sentiment in Twitter. In: CICLing, vol. 2, pp. 49–65 (2015)
4. Chung, J., Gülçehre, Ç., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR abs/1412.3555 (2014)
5. Cummins, R.: On the inference of average precision from score distributions. In: CIKM, pp. 2435–2438 (2012)
6. Etzioni, O., Fader, A., Christensen, J., Soderland, S., Mausam: Open information extraction: the second generation. In: IJCAI, pp. 3–10 (2011)
7. Garcia, S., Herrera, F.: An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. J. Mach. Learn. Res. **9**, 2677–2694 (2008)
8. Han, H., Zhang, S., Qiao, J.: An adaptive growing and pruning algorithm for designing recurrent neural network. Neurocomputing **242**, 51–62 (2017)
9. Mandt, S., Hoffman, M.D., Blei, D.M.: Stochastic gradient descent as approximate Bayesian inference. J. Mach. Learn. Res. **18**, 134:1–134:35 (2017)
10. Mausam, Schmitz, M., Soderland, S., Bart, R., Etzioni, O.: Open language learning for information extraction. In: EMNLP-CoNLL, pp. 523–534 (2012)
11. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS, pp. 3111–3119 (2013)
12. Mitchell, T.M., Cohen, W.W., Hruschka, E.R., Talukdar, P.P., Betteridge, J., Carlson, A., Mishra, B.D., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E.A., Ritter, A., Samadi, M., Settles, B., Wang, R.C., Wijaya, D.T., Gupta, A., Chen, X., Saparov, A., Greaves, M., Welling, J.: Never-ending learning. In: AAAI, pp. 2302–2310 (2015)
13. Nakayama, Y., Fujii, A.: Extracting condition-opinion relations toward fine-grained opinion mining. In: EMNLP, pp. 622–631 (2015)
14. Narayanan, R., Liu, B., Choudhary, A.N.: Sentiment analysis of conditional sentences. In: EMNLP, pp. 180–189 (2009)
15. Nußbaum-Thom, M., Cui, J., Ramabhadran, B., Goel, V.: Acoustic modeling using bidirectional gated recurrent convolutional units. In: Interspeech 2016, pp. 390–394 (2016)
16. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: ICML, vol. 3, pp. 1310–1318 (2013)
17. Ravi, K., Ravi, V.: A survey on opinion mining and sentiment analysis: tasks, approaches and applications. Knowl. Based Syst. **89**, 14–46 (2015)
18. dos Santos, C.N., Xiang, B., Zhou, B.: Classifying relations by ranking with convolutional neural networks. In: ACL, vol. 1, pp. 626–634 (2015)
19. Schouten, K., Frasincar, F.: Survey on aspect-level sentiment analysis. IEEE Trans. Knowl. Data Eng. **28**(3), 813–830 (2016)
20. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Trans. Signal Process. **45**(11), 2673–2681 (1997)

21. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Drop-out: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
22. Zhang, D., Wang, J., Zhao, X.: Estimating the uncertainty of average F_1 scores. In: ICTIR, pp. 317–320 (2015)
23. Zhang, X., Zhao, J.J., LeCun, Y.: Character-level convolutional networks for text classification. In: NIPS, pp. 649–657 (2015)