Interview Assignments for Lead Insights Digital Incubator @ PMI

Use Case #4

Giovani Bernardes Merlin
giovani.merlin@mines-paristech.fr

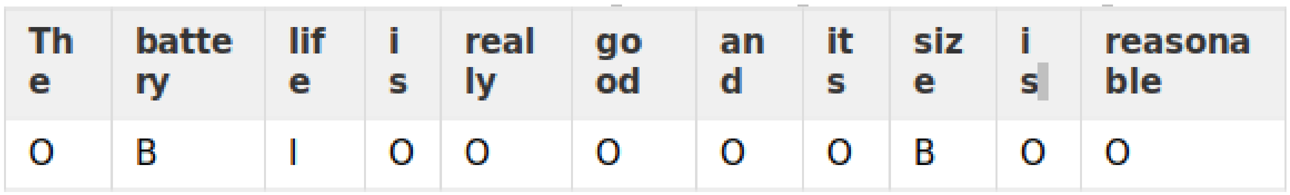
14 novembre 2021

Contents

1 Problem statement

Companies that provide services or products to their customers would like to know which aspects of their products preoccupy most of the customers. This knowledge usually helps to improve product portfolio. Are people talking the most about the quality of the food, the price of an item or the battery life of a computer? All those terms are what we call aspect term. You are asked to build an Aspect Term Extractor (ATE). An ATE is a model that extracts aspect terms from a review, i.e. for each word of a review, your model should predict if the word is an aspect term or not of the reviewed product. The input data has the following format:

review → list of aspect terms, e.g. "The battery life is really good and its size is reasonable"→ "battery life", "size". We recommend to change it and use the BIO format instead. Example:



The	battery	life	is	really	good	and	its	size	is	reasonable
O	B	I	O	O	O	O	O	B	O	O

With this format, we can see that the role of an ATE is to assign to each word one of the three possible classes:

- O = not an aspect (Outside)
- B = first word of an aspect (Beginning)
- I = second, third, ... word of an aspect (Inside)

For this task I will use a dataset made up of laptop reviews.

1.1 Objectives

Please prepare the presentation with your results. In it you'll have to justify and explain:

- The feature extraction process: what is the intuition behind your method, which features are the most important, etc.?
- The algorithm you selected (we ask you to explore at least two different algorithms)
- The metric that you used for quantifying the performance of your model
- The performance evaluation of the algorithms you selected

Your model must be **interpretable**, i.e. you are not allowed to use any kind of deep learning method (LSTM, CNN, . . .).

2 Dataset

The dataset consists of laptop reviews in XML format, first, we will convert this format to JSON format to make it easier to use. I.e, I'm converting the data from:

```
<sentence id="2339">
    <text>I charge it at night and skip taking the cord
    with me because of the good battery life.</text>
    <aspectTerms>
        <aspectTerm term="cord" polarity="neutral" from="41" to="45"/>
        <aspectTerm term="battery life" polarity="positive" from="74" to="86"/>
    </aspectTerms>
</sentence>
<sentence id="812">
    <text>I bought a HP Pavilion DV4-1222nr laptop and
    have had so many problems with the computer.</text>
</sentence>
```

to:

```
{
    "id": "2339",
    "text": "I charge it at night and skip taking the cord
    with me because of the good battery life.",
    "terms_position": [
        [41, 45],
        [74, 86]
    ],
    "polarity": [2, 1]
}, {
    "id": "812",
    "text": "I bought a HP Pavilion DV4-1222nr laptop and
    have had so many problems with the computer.",
    "terms_position": [],
    "polarity": []
}
```

3 Algorithms

Deep learning algorithms reach the state of the art in ATE but are not interpretable. To use interpretable models it is necessary to manually choose the features that will be used by each algorithm.

When thinking about Aspect Term Extraction we are thinking about *nouns* (or *noun phrases*) being classified by *adjectives* (positive, negative, etc...). So, logically I need to extract grammatical properties from the phrases. For that, I will use Spacy using the *"en_core_web_sm"* model which is a lightweight and CPU optimized model.

I'll present here the following Algorithms:

1. Classical machine learning approach using individual tokens to classify its label
2. Sequential model - CRF (Conditional Random Field) model. This model is a sequence of conditional probabilities, it gets better the natural sequential order of texts.

Each one will use a specific set of features, crafted directly from the sentence and with grammatical features. To extract these grammatical features and use the text I need to tokenize it. In all the tests presented here, I've tokenized the text using *Spacy* but pre-processed it to put a space between init/end of words (defined by having a space between them) to make it easier for spacy to tokenize the words. For example, the phrase "I like "sales" team of this -company" is transformed to "I like " sales " team of this - company".

3.1 Machine learning approach

To put such algorithms into practice, I will tokenise the text, extract the POSTag of each token, the type of relationship it has with another token (dependency) and the POSTag of the dependency token. To make this clearer, we can observe the following image:

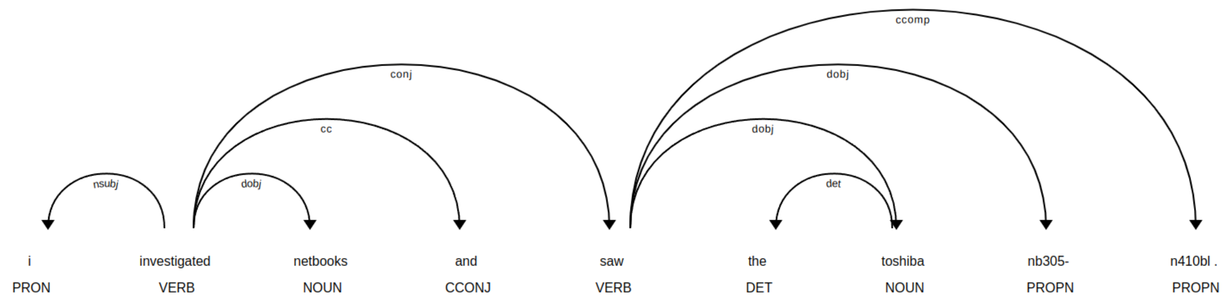


Figure 1: The grammatical characteristics of a sentence extracted by *Spacy*, image by *displacy*

For example, the token "netbooks" has the following properties:
POSTag=NOUN, dependency=dobj, head tag=VERB.
And the token "investigated" :
POSTag=VERB, dependency=ROOT, head tag=VERB.
For more information on tags and dependencies, see Universal Dependencies Website.

After collecting these features for each token, I used Scikit Learn's "DictVectorizer" function to create the vocabulary for each feature and trained different models (Multinomial Naive Bayes, Linear Support Vector Machine, DecisionTrees and LogisticRegression). Here, I will only comment on the Logistic Regression and Linear SVC models as they performed better. The code to reproduce the results can be found in the file "1_count.py".

To evaluate the models, firstly, I will only judge the F1 score on the B and I classes, because the dataset has many more O tags (which are not really interesting for us).

In a first try, doing only accent normalisation, stripping spaces from sentences and removing multiple spaces as the pre-processing step, these features where transformed in sparse vectors of size 4238, i.e. 4160 for tokens (vocabulary size), 45 for dependencies, 18 for head tags, and also 18 for tags, resulting in:

Linear SVC f1-score: 0.69 B, 0.49 I
Logistic Regression f1-score : 0.64 B, 0.40 I

By adding text lowering and removing punctuation, I was able to reduce the size of the vocabulary of tokens to 4125 (35 less), which did not result in any significant improvements.

Linear SVC f1-score: 0.69 B, 0.49 I
Logistic Regression f1-score : 0.65 B, 0.42 I

Finally, by using lemmas instead of tokens, I was able to successfully reduce the vocabulary size to 3301. The lemmas are "word roots", for example the lemma for "netbooks" is "netbook" (singular form). In this way, we can associate words that have virtually the same meaning (such as plural and singular) in the same feature space, making it easier for the model to understand the data. By doing all these processes, it was possible to obtain the best results with the svc model:

Linear SVC f1-score: 0.67 B, 0.52 I
Logistic Regression f1-score : 0.62 B, 0.46 I

Which improved the result quite a bit for the logistic regression but not much for the SVC model.

To further investigate the difference (and choose the best steps), we can look at the "partial match" and "full match" scores. Full match is defined as "out of all the entities, how many times I got a full match" and "partial match" as "out of all the entities, how many times I got at least one matching token". For the model using lemmas and performing all preprocessing steps, the total match is 0.523, the partial match 0.876. Without preprocessing and lemmas I've gotten full match score: 0.535 and partial match score 0.876. So, without pre-processing or lemmatization it was possible to obtain the best result. The detailed result of such a model can be seen in the following table:

	Precision	Recall	F1-Score
O	0.97	0.98	0.98
B	0.71	0.65	0.67
I	0.57	0.44	0.49
Macro avg	0.75	0.69	0.71

Moreover, the model is interpretable and we can analyse its coefficients. For reasons of time, I will not discuss them here, but I print the most important features to define each of the tags in the code. As a quick example, for the tag I BIO, we have that the most important TAG to classify the token as it is the "ADP" (adposition, forms the compound nouns) with coef = 0.702.

4 Sequential CRF model

Such a section will be quicker as most of the details have already been discussed.

Conditional random field (CRF) is a graph-based model, which can successfully model sequences, where each prediction depends on the prediction before (and the next one) thus It can model easily that it's impossible to have an I tag after an O tag for example.

For this model, I used the same features as before more the head token in a window size of 3, that is, I use all these pieces of information for the current token, the before token, and the after token. The algorithm is implemented in 2_crf.py .

Training the model with lower case (and again, just considering the B and I tags) I could get 0.742 for the B tag and 0.694 for the I one. For this model, using lemmas in the place of tokens resulted in good improvements, as can be seen in the following table:

	Precision	Recall	F1-Score
B	0.896	0.656	0.757
I	0.913	0.588	0.715
Macro avg	0.904	0.622	0.740

We can thus observe the enormous gain in the score when using a model adapted to the types of data in question. Previously, the best f1-B was 0.69 and 0.52 for the f1-I. With such a model it was possible to gain almost 7 points on f1-B and almost 20 for f1-I!

Again, the model is interpretable, running the code I print the most important features. Only for giving an example here, the transition O $\rightarrow$ I have weight -9.5, getting the impossibility of this transition.