Extracting entities and relationships from vehicle manual

Summary

This document describes the entire process from the cleaning of the dataset, preparing golden-set, trying and evaluating different approaches, tuning parameters and approaches and the final approach and results for extracting entities and relationships from vehicle manual data.

Extraction of entities

Problem Description

The input data is given in the form of a json. A sample of the input data is as follows:

```
 [
      {
       "category": "Contents",
       "sub_category": "No Subcategory",
       "title": "No Title",
       "description": " The fastest way to find information on a particular topic or
     item is by using the index, refer to page 232. Notes",
       "image_url": "",
       "tags": "",
       "warning_text": "",
       "image_urls": []
      }, …..

]
```

From this dataset, we need to extract entities like “safety belt” etc. and relationships like “the child seat” “is fastened to” “backrest of seat” etc.

Challenges

The entity can be present in category, subcategory or description.

The input text is sometimes irrelevant to our task. For example: The following sentence: `" Thank you for choosing a BMW. The more familiar you are with your vehicle, the better control… “.`

The input might not be properly extracted. For example: “`Deactivated front passenger airbags If a child restraint fixing system`” should be “`Deactivated front passenger airbags. If a child` restraint fixing system" and "All around the steering wheel 12 ABS, Antilock Brake SysAll-season tires," which includes page numbers and data from different columns.

Preparing the Golden set

In order to evaluate any approaches and fine-tune them, we need to have a training dataset or the golden set. The whole dataset is too huge to tag. Hence, for this, we manually selected a few passages relating to "Transporting children safely" category. The input file is here. Once, this is fixed, we manually created a file containing entities of interest for us by going through each of the entries. The dataset for it is here. Once, we fine-tune and get the best entity and relationship extraction, we can then apply it on the larger dataset.

Clean up of text

The following are the different clean up strategies we used.

1. Add the texts of category, subcategory, title and descriptions into one sentence. (The category is .., the sub category is .. and so on.)
2. Remove double quotes.
3. Remove Unicode characters.
4. Add a period before a capslock (that is not a noun-phrase) for ending the premature sentence before it like described above.

We also tried removing numbers but it was affecting some genuine entity extraction.

Clean up of Entities

The following are the different clean up strategies we used.

1. Lowercase, strip end characters and remove double quotes from entities
2. Remove empty, integer entities
3. Remove pronouns from the entities
4. Remove entities that are just stop words or additional special words added by us like "category, title" etc.
5. Stemming of the entities.

Results of cleanup

All these approaches significantly improved the precision, recall, fscore on the goldenset from:

Spacy Entities stats (precision, recall, fscore): 0.10,0.07,0.08

Spacy Noun chunk stats (precision, recall, fscore): 0.22,0.69,0.34

Google API Entity stats (precision, recall, fscore): 0.29,0.62,0.40

To

Spacy Entities stats (precision, recall, fscore): 0.18,0.03,0.05

Spacy Noun chunk stats (precision, recall, fscore): 0.42,0.74,0.53

Google API Entity stats (precision, recall, fscore): 0.43,0.58,0.49

Different approaches tried for entity extraction.

There are a lot of different opensource and corporate tools to extract entities. Some of them I looked at are as follows:

1) NLTK entity extraction
2) Spacy entity extraction
3) Spacy noun chunk extraction
4) Microsoft Entity Linking service
5) Google Cloud NLP Entity Extraction
6) Stanford CoreNLP

When we tried them a lot of these focus on extracting highly precise real world entities (Microsoft Entity Linking service, NLTK entity extraction, Spacy entity extraction, Stanford CoreNLP) and many of the NER libraries we have seen. In our particular scenario, we would like to have a good recall and should not miss not so precise but important entities like (belt, windows, steering button etc.). This is why we started looking into libraries that can extract good noun phrases and then have a smarter filtering algorithm on our own which can develop to improve on precision. Spacy noun chunk extraction and Google Cloud NLP Entity Extraction seemed to do a good job at giving a high recall. We just included Spacy entity extraction in our evaluation just to prove this case. In addition, google also gives us what type of entity and the salience of the entity in the sentence. We use these to filter out additional entities that might not be useful for us (eg. We are probably not interested in organization entities or people entities).

Once we ran a bunch of them, we got some good feedback in terms of the false positives and false negatives for each algorithm. The results of this are here and here.

As we can see the entities that we are interested in and not interested in are slightly different (eg. Some of the false positives are: travel, height, injury, stability as opposed to: car seat, safety belt, backrest etc.)

Classifiers

To improve the precision on this, we created a training dataset from the above samples and tested out various featurizers and classifiers with different parameters using 5-fold cross validation on the dataset (with 30% test hold-out). We used sklearn for this.

The various featurizers we tried are:

1) CountVectorizer (bag of words with counts)
2) TfIDFVectorizer (bag of words with TFIDF scores)
3) MeanWordVectors (Average spacy word embeddings)

The various classifiers we used are:

1) Nearest Neighbors
2) Linear SVM
3) RBF SVM
4) Guassian Process
5) Decision Tree
6) Random Forest
7) Neural Network
8) Adaboost
9) Naïve Bayes
10) QDA (Quadratic Discriminant Analysis).

The code for doing this is here and the results are here. The results contain the accuracy as well as time taken.

F-Score of different Algorithms

Bag of words - Counts	
AdaBoost	0.86
Decision Tree	0.64
Gaussian Process	0.89
Linear SVM	0.89
Naive Bayes	0.68
Nearest Neighbors	0.76
Neural Net	0.88
Quadratic Discriminant Analysis	0.67
Random Forest	0.6
RBF SVM	0.86
Bag of Words - TFIDF	
AdaBoost	0.88
Decision Tree	0.64
Gaussian Process	0.87
Linear SVM	0.89
Naive Bayes	0.68
Nearest Neighbors	0.76
Neural Net	0.86
Quadratic Discriminant Analysis	0.67
Random Forest	0.59
RBF SVM	0.84
Average Word Vectors	
AdaBoost	0.83
Decision Tree	0.81
Gaussian Process	0.86
Linear SVM	0.73
Naive Bayes	0.7
Nearest Neighbors	0.7
Neural Net	0.74
Quadratic Discriminant Analysis	0.68
Random Forest	0.78
RBF SVM	0.8

Avg Times Taken for different feature vectors:

Features	Avg Time Taken (seconds
Bag of words - Counts	39.12
Bag of Words - TFIDF	38.86
Average Word Vectors	465.3

As we can see, Linear SVM is performing pretty well with bag of words and TFIDF and Word Vectors generally seem to perform bad. Also, the time taken for it is an order of magnitude more. This linear classifier seem to work pretty well for the small dataset that we have and hence we chose that.

For SVMs too, we tried a grid search to determine the best possible parameters with the following grid.

```
grid = [{'classifier__C': [1, 10, 100, 1000],'classifier__kernel': ['linear']},
{'classifier__kernel': ['rbf'], 'classifier__C': [1, 10, 100, 1000], 'classifier__gamma':
[0.001, 0.0001]}]
```

The code for doing this is here and the results are here.

0.842 (+/-0.063s for {'classifier__kernel': 'linear', 'classifier__C': 1}

0.889 (+/-0.056) for {'classifier__kernel': 'linear', 'classifier__C': 10}

0.877 (+/-0.099) for {'classifier__kernel': 'linear', 'classifier__C': 100}

0.874 (+/-0.098) for {'classifier__kernel': 'linear', 'classifier__C': 1000}

0.560 (+/-0.088) for {'classifier__kernel': 'rbf', 'classifier__gamma': 0.001, 'classifier__C': 1}

0.560 (+/-0.088) for {'classifier__kernel': 'rbf', 'classifier__gamma': 0.0001, 'classifier__C': 1}

0.560 (+/-0.088) for {'classifier__kernel': 'rbf', 'classifier__gamma': 0.001, 'classifier__C': 10}

0.560 (+/-0.088) for {'classifier__kernel': 'rbf', 'classifier__gamma': 0.0001, 'classifier__C': 10}

0.677 (+/-0.242) for {'classifier__kernel': 'rbf', 'classifier__gamma': 0.001, 'classifier__C': 100}

0.560 (+/-0.088) for {'classifier__kernel': 'rbf', 'classifier__gamma': 0.0001, 'classifier__C': 100}

0.864 (+/-0.047) for {'classifier__kernel': 'rbf', 'classifier__gamma': 0.001, 'classifier__C': 1000}

0.677 (+/-0.242) for {'classifier__kernel': 'rbf', 'classifier__gamma': 0.0001, 'classifier__C': 1000}

The best model was **Linear SVM with C=10**. This was what was finalized in our final model.

With this our accuracy on the Goldenset significantly increased from:

Spacy Entities stats (precision, recall, fscore): 0.18,0.03,0.05

Spacy Noun chunk stats (precision, recall, fscore): 0.42,0.74,0.53

Google API Entity stats (precision, recall, fscore): 0.43,0.58,0.49

To

Entities stats (precision, recall, fscore): 1.0,0.03,0.06

Noun chunk stats (precision, recall, fscore): 0.92,0.74,0.82

Google API Entity stats (precision, recall, fscore): 0.97,0.58,0.72

Different approaches tried for relation extraction.

Approaches

Relationship extraction is a traditionally done via distance supervision and focusing on one relation to extract. There are supervised and semi-supervised approaches for the same. In supervised approaches, we have the sentence containing the relationship as input and the relationship as the label. It is sometimes pretty hard to get this quality data. Hence, people resorted to semi supervised approaches. The idea is that we have a small seed of sentences for a relationship. This helps us to determine the pattern to extract. We use this pattern to get more examples from the web for this relationship and so on. We also use distance supervision to generate the training data. For example: If we know that Barack Obama and Michelle Obama are a couple, we find all the sentences having these two people and use that as a training data for learning the marriage relationship. A good survey paper is here and a good software to do this is here.

Open IE and Filtering

However, in our particular scenario the relationships are pretty ad-hoc and for each relationship we don't have a lot of data to train a model that is interesting. To extract such ad-hoc relatonships, we can look more into the structure and dependency tree of the sentences and try to extract the relationships. OpenIE is a wonderful tool for doing this. In addition to extracting ad-hoc relations in any given text, it does also extract the types of relationships and contexts. Good examples for this are given here. This seems to be a great fit for our purposes. Running this takes a huge amount of memory so I have used a VM in azure to generate the jar that we use for running it.

Results

The results of running it on our small dataset is here. It has around 75 relations and not all of them are very interesting. Hence, we further improve the precision of our results by considering only those relations for which both subject and object have one of the entities extracted by the algorithm. As a result of this, we get some nice results here for nounchunks (13 relations) and here for google api (11 relations).

Co-occurring Entities

Another approach is to see what entities co-occur together in a sentence. This kind of gives us an idea that they are related. The results of this are here for noun-chunks (19 cooccurring entities) and here for google (15 cooccurring entities). As we can see, these are also very indicative that these entities are related and are also precise enough since the entities we detected are of good precision.

Some of the relations extracted are:

Subject	**Predicate**	**Object**
a child restraint fixing system	is used	in the front passenger seat
the child seat	is securely fastened	to the backrest of the seat
the upper fixing point of the safety belt	is located	before the belt guide of the child seat

Results on running on the entire dataset.

The results on the entire dataset are here.

The **cooccur_*algo*.tsv** files contain all the entities that co-occur together for that algo in our dataset.

The **count_*algo*.tsv** files contain all the entities that were discovered by the algorithm.

The **rel_*algo*.csv** files contain all the relations extracted by open ie and the algorithm.

Future Work

1) The extraction from pdf to json is bad in many of the places. Especially when we have column data. This needs to be improved.
2) Improve our classifiers by getting more training data and expanding the seed of our initial dataset. This should be easy to see based on the entities extracted and filtered out or in in the run on whole dataset.
3) There are some false negatives in the small dataset. This is because even the noun chunks and google api are missing out a few good potential entities. Add an additional algorithm to extract these kind of entities.