NER

Friday, August 31, 2018 2:19 PM

If you are thinking of writing a Named Entity Recognizer easily from scratch, do the following (Neural Networks might take some time to train, but the algorithm is pretty simple in their case) (This is the algorithm which was used to train Entity Extractor at ParallelDots , demo- ParallelDots) :
Step 1. Take a tagged dataset of sentences, with each word of a sentence tagged as entity(yes/no) and entity type(Person/Place/Organization). We have a data tagging team at ParallelDots that creates this type of tagged datasets for us. However, for getting started you can use openly available datasets like Tagged datasets for named entity recognition tasks
Step2: Choose an RNN or CNN sequence prediction architecture to use. [1702.01923] Comparative Study of CNN and RNN for Natural Language Processing . Its simple to write such architectures with Keras Documentation like libraries now-a-days. It requires some patience to train although.

From <https://www.quora.com/How-does-the-NER-training-work-I-want-to-build-my-own-trained-NEs>

https://www.commonlounge.com/discussion/2662a77ddcde4102a16d5eb6fa2eff1e

https://nlpforhackers.io/training-ner-large-dataset/

https://eli5.readthedocs.io/en/latest/tutorials/sklearn_crfsuite.html

http://www.stokastik.in/building-an-incremental-named-entity-recognizer-system/

https://eli5.readthedocs.io/en/latest/tutorials/sklearn_crfsuite.html

https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html#evaluation

https://www.codementor.io/jadianes/building-a-recommender-with-apache-spark-python-example-app-part1-du1083qbw
QnA bot
https://towardsdatascience.com/nlp-building-a-question-answering-model-ed0529a68c54

I have a semi-working system that solves this problem, open sourced using scikit-learn, with a series of blog posts describing what I'm doing. The problem I'm tackling is word-sense disambiguation (choosing one of multiple word sense options), which is not the same as Named Entity Recognition. My basic approach is somewhat-competitive with existing solutions and (crucially) is customisable.
There are some existing commercial NER tools (OpenCalais, DBPedia Spotlight, and AlchemyAPI) that might give you a good enough commercial result - do try these first!
I used some of these for a client project (I consult using NLP/ML in London), but I wasn't happy with their recall (precision and recall). Basically they can be precise (when they say "This is Apple Inc" they're typically correct), but with low recall (they rarely say "This is Apple Inc" even though to a human the tweet is obviously about Apple Inc). I figured it'd be an intellectually interesting exercise to build an open source version tailored to tweets. Here's the current code:https://github.com/ianozsvald/social_media_brand_disambiguator
I'll note - I'm not trying to solve the generalised word-sense disambiguation problem with this

approach, just **brand** disambiguation (companies, people, etc.) when you already have their name. That's why I believe that this straightforward approach will work.
I started this six weeks ago, and it is written in Python 2.7 using scikit-learn. It uses a very basic approach. I vectorize using a binary count vectorizer (I only count whether a word appears, not how many times) with 1-3 n-grams. I don't scale with TF-IDF (TF-IDF is good when you have a variable document length; for me the tweets are only one or two sentences, and my testing results didn't show improvement with TF-IDF).
I use the basic tokenizer which is very basic but surprisingly useful. It ignores @ # (so you lose some context) and of course doesn't expand a URL. I then train using logistic regression, and it seems that this problem is somewhat linearly separable (lots of terms for one class don't exist for the other). Currently I'm avoiding any stemming/cleaning (I'm trying The Simplest Possible Thing That Might Work).
The code has a full README, and you should be able to ingest your tweets relatively easily and then follow my suggestions for testing.
This works for Apple as people don't eat or drink Apple computers, nor do we type or play with fruit, so the words are easily split to one category or the other. This condition may not hold when considering something like #definance for the TV show (where people also use #definance in relation to the Arab Spring, cricket matches, exam revision and a music band). Cleverer approaches may well be required here.
I have a series of blog posts describing this project including a one-hour presentation I gave at the BrightonPython usergroup (which turned into a shorter presentation for 140 people at DataScienceLondon).
If you use something like LogisticRegression (where you get a probability for each classification) you can pick only the confident classifications, and that way you can force high precision by trading against recall (so you get correct results, but fewer of them). You'll have to tune this to your system.
Here's a possible algorithmic approach using scikit-learn:

- Use a Binary CountVectorizer (I don't think term-counts in short messages add much information as most words occur only once)
- Start with a Decision Tree classifier. It'll have explainable performance (see *Overfitting with a Decision Tree* for an example).
- Move to logistic regression
- Investigate the errors generated by the classifiers (read the DecisionTree's exported output or look at the coefficients in LogisticRegression, work the mis-classified tweets back through the Vectorizer to see what the underlying Bag of Words representation looks like - there will be fewer tokens there than you started with in the raw tweet - are there enough for a classification?)
- Look at my example code in https://github.com/ianozsvald/social_media_brand_disambiguator/blob/master/learn1.py fo r a worked version of this approach

Things to consider:

- You need a larger dataset. I'm using 2000 labelled tweets (it took me five hours), and as a minimum you want a balanced set with >100 per class (see the overfitting note below)
- Improve the tokeniser (very easy with scikit-learn) to keep # @ in tokens, and maybe add a capitalised-brand detector (as user @user2425429 notes)
- Consider a non-linear classifier (like @oiez's suggestion above) when things get harder. Personally I found LinearSVC to do worse than logistic regression (but that may be due to the high-dimensional feature space that I've yet to reduce).
- A tweet-specific part of speech tagger (in my humble opinion not Standford's as @Neil suggests - it performs poorly on poor Twitter grammar in my experience)
- Once you have lots of tokens you'll probably want to do some dimensionality reduction (I've not tried this yet - see my blog post on LogisticRegression l1 l2 penalisation)

Re. overfitting. In my dataset with 2000 items I have a 10 minute snapshot from Twitter of 'apple' tweets. About 2/3 of the tweets are for Apple Inc, 1/3 for other-apple-uses. I pull out a balanced subset (about 584 rows I think) of each class and do five-fold cross validation for

training.
Since I only have a 10 minute time-window I have many tweets about the same topic, and this is probably why my classifier does so well relative to existing tools - it will have overfit to the training features without generalising well (whereas the existing commercial tools perform worse on this snapshop, but more reliably across a wider set of data). I'll be expanding my time window to test this as a subsequent piece of work.

From <https://stackoverflow.com/questions/17352469/how-can-i-build-a-model-to-distinguish-tweets-about-apple-inc-from-tweets-abo>

2 flags :--Alphabets followed by numeric (True or False)
Extract max 2 alphabets from the word

Try without current word

Check metrics of test and train (K folds) have a set of unseen mails to check the scoring

https://www.kaggle.com/gagandeep16/ner-using-bidirectional-lstm

http://www.albertauyeung.com/post/python-sequence-labelling-with-crf/

https://www.analyticsvidhya.com/blog/2018/08/nlp-guide-conditional-random-fields-text-classification/

https://www.searchtechnologies.com/blog/natural-language-processing-techniques