
Russian NP-Chunker

Benjamin Rozonoyer

Goals and Motivations

The goal of this project is to create a NP-Chunker for Russian data. The data that is used for the development of this Chunker is the SynTagRus Corpus -- a fully annotated subset of the Russian National Corpus. This project, under the guidance of Alex Luu, is meant to aid the syntactic parsing of the BiRCh (Bilingual Russian Children) Corpus by performing a "shallow parsing" of noun phrase chunks (which will then be easy to incorporate into a syntactic tree for a full sentence). This project was suggested to me by Sophia Malamud and Alex Luu when I asked them how I could be helpful to the corpus collection project.

Design

The following general pipeline is used (and explained below):

1. **Remove non-projectivity** from dependency trees of SynTagRus
2. Convert projective dependency trees of SynTagRus **into IOB-based NP chunks** stored in Conll-style files (`train.txt` and `test.txt`)
3. Develop a variety of statistical **NP-Chunkers** on converted data
4. **Evaluate** the performance of these chunkers on the test data, experimenting with different feature sets

Step 1
A text file consisting of 49,420 projective sentences (equivalent to 708,480 tokens excluding punctuation marks) in Conll2007 format was provided by MILa research group[1].

Step 2
I use the procedure outlined in the article *Conditional Random Field in Segmentation and Noun Phrase Inclination Tasks for Russian* (Kudinov et al.) to determine the "base NPs which are the fragments of noun phrases excluding recursive parts". For Russian, a base NP has the following structure: obligatory noun, optionally preceded by agreeing adjectives/numerals, optionally followed by a dependent base NP in the genitive case. We use Kudinov's augmented version of the IOB labels: "To provide the opportunity of detection of heads of the phrase we augmented the standard BIO-alphabet with two labels: BH (token is the beginning of the base NP and is the head) and IH (token is the phrase head)" (Kudinov et al.)

Step 3
I develop several NP-Chunkers based on the following models: *nltk's* `Unigram`, `Bigram`, and `Trigram` `SequentialBackoff` `Tagger`, *sklearn's* Maximum Entropy (Logistic Regression) and linear SVM, and *pycrfsuite's* CRF. This step of the pipeline trains and saves all of the models that can be successfully pickled and unpickled.

Step 4
I use various combinations of lexical and morphological features for the evaluation process.

[1] people.brandeis.edu/~smalamud/MILa/about.html

Challenges and Solutions

I began the process practicing running different models on the English Conll2000 data and structuring the program's architecture. I eventually settled on a single `Chunker` class that would contain the methods to extract the train data and the test data, to train the model, pickle it, and evaluate the test data in another method. Originally I tried to split the test data from the train data myself, but encountered more problems than benefits to this approach. The chunker therefore works with two files: `train.txt` and `test.txt`, which are split during the process of data conversion (*Step 2*).

Originally the `Unigram`, `Bigram`, and `Trigram` chunkers weren't susceptible to pickling, but I took them apart and pickled the corresponding tagger which each of them uses, and the models saved and loaded fine. Next, the `Bigram` and `Trigram` chunkers assigned a "None" tag to words in only 4 sentences of `test.txt` during evaluation. The nltk `DefaultTagger` which they extend has a default backoff to "None". I fixed this problem by giving the `Unigram` tagger a backoff `DefaultTagger("O")`, which assigned the *outside* label to any tag not encountered during training, and that fixed the problem.

Another issue had to do with the `sklearn` training techniques, which gave me the error during testing: `X has 8449 features per sample, expecting 19478`. I vectorize both the training and testing data for `sklearn`. I originally used `DictVectorizer`, and that was the problem, since it created a sparse matrix which collapsed any unencountered names among the features. For this reason `X_train` and `X_test` were bound to have different matrix sizes. I fixed this problem by using the `FeatureHasher`, which placed the feature values into a dense matrix (numpy array), so that the training and testing matrices were the same.

Regarding the process of converting the dependency trees to IOB format…

I ended up removing about 600 sentences from the data because they contained recursive base NPs (i.e. a base NP with two heads). Here is an example:

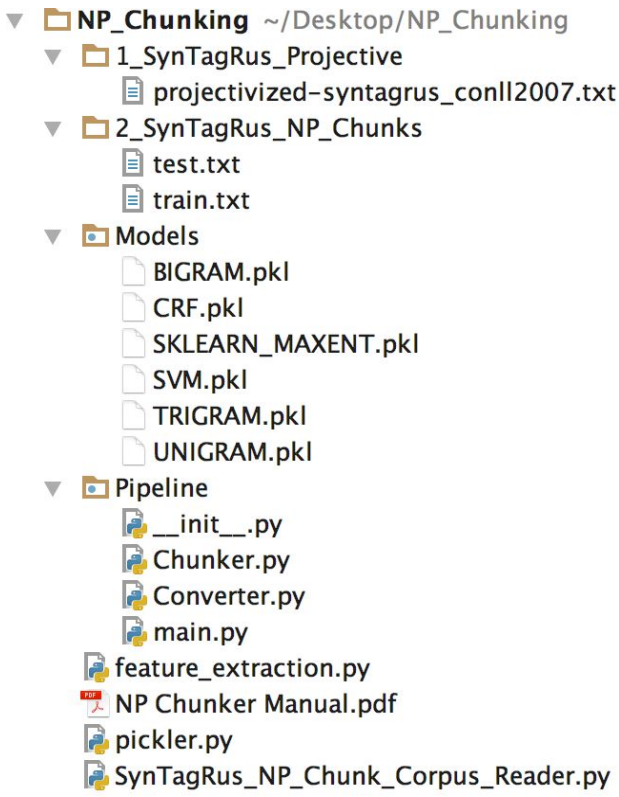
Chunk	*Russian*	*Gloss*
B	Первую	First
I	в	in
BH	Сибири	Siberia
I	частную	private
IH	типографию	typography

Translation: *…the first private typography in Siberia…*

In the Russian, the base NP *Siberia* is inserted into the overarching NP as a component of a PP.

This example demonstrates the diversity of Russian NP structure in comparison with English structure. For this reason I decided not to include regular expression chunking in this project.

Code Structure and Usage



- *1_SynTagRus_Projective* contains the text file with the dependency trees of SynTagRus
- *2_SynTagRus_NP_Chunks* contains the test and train data for the Chunker
- *Models* contains the models that will be pickled during Chunker training
- The *Pipeline* package contains:
  - *Converter.py*: the code for converting the dependency trees from *1_SynTagRus_Projective* into the Conll-style train and test data, storing it in *2_SynTagRus_NP_Chunks*
  - *Chunker.py*: the Chunker class with the methods to extract the train and test data and apply the different models to it. A Chunker object must be instantiated inside *main.py* with one of the following parameters:
    {'UNIGRAM', 'BIGRAM', 'TRIGRAM', 'SKLEARN_MAXENT', 'CRF', 'SVM'}
    (See commented code in *main.py*)
  - *main.py*: the code used to train and evaluate models
- *feature_extraction.py* contains the code to extract the different features from data. Edit features inside the *features* method of this script
- *SynTagRus_NP_Chunk_Corpus_Reader.py*: adapted from nltk conll corpus reader to read converted train and test data
- *pickler.py* is basic code to pickle and load models

Required Installations

- *pycrfsuite* (https://python-crfsuite.readthedocs.io/en/latest/)
- *sklearn* (http://scikit-learn.org/stable/)

Experiment

1. Baseline: we only take the pos into consideration.

Note: the Unigram, Bigram and Trigram chunkers use only this feature set, and won't be included in the other tables.

	avg precision	**avg recall**	**avg f-measure**
Unigram	0.58	0.71	0.63
Bigram	0.87	0.84	0.85
Trigram	0.87	0.84	0.85

For this experiment, Bigram and Trigram chunkers have very similar performance.

Feature types = {pos}

	avg precision	**avg recall**	**avg f-measure**
Unigram	0.58	0.71	0.63
Maxent	0.58	0.71	0.63
Linear SVM	0.58	0.71	0.63
CRF	0.81	0.82	0.81

Even though Maxent and SVM are a more complex model, they can't outperform Unigram due to the limitation of the feature set. We also see the power of the CRF model.

Feature types = {prevpos, pos}

	avg precision	**avg recall**	**avg f-measure**
Bigram	0.87	0.84	0.85
Maxent	0.87	0.86	0.86
Linear SVM	0.87	0.86	0.86
CRF	0.87	0.87	0.87

Here we see that all of the models have a similar result, so this simple feature set doesn't demonstrate the difference between the predictional power of these different models.

Feature types = {prevprevpos, prevpos, pos}

	avg precision	**avg recall**	**avg f-measure**
Trigram	0.87	0.84	0.85
Maxent	0.88	0.87	0.87
Linear SVM	0.88	0.87	0.87
CRF	0.88	0.88	0.88

The observation here is similar to the previous experiment.

Conclusion: based on all of the experiments with the baseline, the Maxent and SVM had similar performance and were slightly better than the Unigram, Bigram and Trigram models in each experiment, respectively. We see that the CRF model generally outperforms all the other models, so we will use this model for experiments with more advanced feature sets. Regarding features, we see that the immediately preceding word (closest context to current word) has the most influence on the current word. This explains why we achieved significant improvement from Unigram to Bigram, but not from Bigram to Trigram.

2. CRF with more advanced features

Feature Set	**avg precision**	**avg recall**	**avg f-measure**	***Note***
baseline	0.87	0.87	0.87	
+{word, prevword}	0.93	0.93	0.93	{word} *helps*
+{nextword, nextpos}	0.93	0.93	0.93	{next} *doesn't help*
+{lemma}	0.93	0.93	0.93	{lemma} *doesn't help*
+{gender}	0.93	0.93	0.93	{gender} *doesn't help*
+{case}	0.95	0.95	0.95	{case} *obviously helps*
+{number}	0.95	0.95	0.95	{number} *doesn't help*
+{animacy}	0.95	0.95	0.95	{animacy} *doesn't help*
+{prevcase, nextcase}	0.95	0.96	0.96	*slight, but important improvement*

baseline = {pos, prevpos}

Conclusion: among the additional morphological features, {case} seemed to contribute the best improvement to the performance of the CRF model. The best results I achieved are comparable to the results reported in the most relevant work by Kudinov et al.[2]

Future Work

I will need to do an error analysis on the predicted results to find out in which cases this CRF model doesn't work well and to figure out which feature we should add to the feature set to overcome these weaknesses and improve the model's performance. After that, we will try to apply this model to the data of the BiRCh corpus in an effort to aid its syntactic parsing.

The result of the heuristic conversion process from SynTagRus dependency trees to conll-style IOB format is considered a "silver standard" (in the words of Marcus Verhagen), because its correctness is not exhaustively verified (by human annotators or otherwise). A future direction of work will be to manually correct the converted data and create the gold standard for a formal evaluation of the implemented conversion.

[2] However, we might not be using the same configuration of training and testing data.

References

Kudinov, M. S., A. A. Romanenko, and I. I. Piontkovskaja. "Conditional Random Field in segmentation and Noun Phrase inclination tasks for Russian." *Annual International Conference" Dialogue", Computational Linguistics and Intellectual Technologies*. Vol. 13. 2014.

Luu, Alex, Sophia A. Malamud, and Nianwen Xue. "Converting SynTagRus Dependency Treebank into Penn Treebank Style." *Proceedings of the Tenth Linguistic Annotation Workshop*. 2016.