- pos_tag help us to define each word in text (noun - verb - …)
- A tagger can also model our knowledge of unknown words; for example, we can guess that scrobbling is probably a verb, with the root scrobble, and likely to occur in contexts like he was scrobbling
- We can create one of these special tuples from the standard string representation of a tagged token, using the function str2tuple():
- Several of the corpora included with NLTK have been tagged for their part-of-speech. and we should use tagged_words function to read it
- the python shows the different language like chinese for example in ascii code
- The simplified noun tags are N for common nouns like book, and NP for proper nouns like Scotland
- You will see that there are many variants of NN; the most important contain $ for possessive nouns, S for plural nouns (since plural nouns typically end in s), and P for proper nouns. In addition, most of the tags have suffix modifiers: -NC for citations, -HL for words in headlines, and -TL for titles (a feature of Brown tags).
- some info related to dictionary , and how to use it also there is option of dictionary in nltk
- Default taggers assign their tag to every single word, even words that have never been encountered before
- For instance, we might guess that any word ending in ed is the past participle of a verb, and any word ending with ’s is a possessive noun just to give them tags
- The final regular expression «.*» is a catch-all that tags everything as a noun. This is equivalent to the default tagger (only much less efficient).
- We evaluate the performance of a tagger relative to the tags a human expert would assign Since we usually don’t have access to an expert and impartial human judge, we make do instead with gold standard test data. This is a corpus which has been manually annotated and accepted as a standard against which the guesses of an automatic system are assessed
- Unigram taggers are based on a simple statistical algorithm , . A unigram tagger behaves just like a lookup tagger (Section 5.4), except there is a more convenient Figure 5-4. Lookup tagger
- The NgramTagger class uses a tagged training corpus to determine which part-of-speech tag is most likely for each context. Here we see a special case of an n-gram tagger, namely a bigram tagger
- Notice that the bigram tagger manages to tag every word in a sentence it saw during training, but does badly on an unseen sentence. As soon as it encounters a new word (i.e., 13.5), it is unable to assign a tag. It cannot tag the following word (i.e., million), even if it was seen during training
- combining tagger is helpful too by using different tagger (default tagger , unigram tagger and bigram tagger)
- we can use <unk> as definition for unknown words
- A potential issue with n-gram taggers is the size of their n-gram table (or language model). If tagging is to be employed in a variety of language technologies deployed on mobile computing devices, it is important to strike a balance between model size and tagger performance. An n-gram tagger with backoff may store trigram and bigram tables, which are large, sparse arrays that may have hundreds of millions of entries.
- Brill tagging is a kind of transformation-based learning, named after its inventor. The general idea is very simple: guess the tag of each word, then go back and fix the mistakes. In this way, a Brill tagger successively transforms a bad tagging of a text into a better one

-