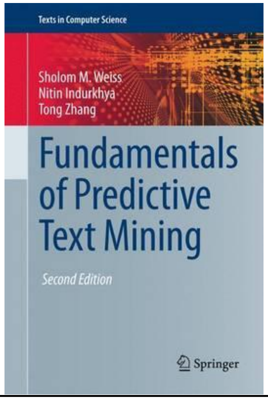
Text Mining for Social Media

CIS 700/CSE 791

Week 5: Named Entity Recognition (NER)

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

February 16, 18, 23, 2016



Sentence Boundary Determination

- For a Named Entity Recognition (NER) algorithm, or any information extraction algorithms for that matter, to perform well (or optimally), it usually requires a complete sentence as an input.

  → the sentences must be identified correctly.

Sentence Boundary Determination

- Sentence boundary determination is essentially the problem of deciding which instances of a **period** followed by whitespace are sentence delimiters and which are not, since we assume that the characters **?** and **!** are unambiguous sentence boundaries.

- Since this is a **classification problem**, one can naturally invoke standard classification software on training data and achieve accuracy of more than **98%**. (To be discussed next.)

- If training data are not available, one can use a handcrafted (rule-based) algorithm. (See next slide.)

Sentence Boundary Determination

- This algorithm will achieve an accuracy of more than **90%** on newswire text.
  - Adjustments to the algorithm for other corpora may be necessary to get better performance.
- Also, this algorithm is tailored for English.
  - A different language would have a completely different procedure but would still involve the basic idea of rules that examine the context of potential sentenc boundaries.

Input: a text with periods
Output: same text with End-of-Sentence (EOS) periods identified

Overall Strategy:
1. Replace all identifiable non-EOS periods with another character
2. Apply rules to all the periods in text and mark EOS periods
3. Retransform the characters in step 1 to non-EOS periods
4. Now the text has all EOS periods clearly identified

Rules:
All ? ! are EOS
If " or ' appears before period, it is EOS
If the following character is not white space, it is not EOS
If) }] before period, it is EOS
If the token to which the period is attached is capitalized and is < 5 characters and the next token begins uppercase, it is not EOS
If the token to which the period is attached has other periods, it is not EOS
If the token to which the period is attached begins with a lowercase letter and the next token following whitespace is uppercase, it is EOS
If the token to which the period is attached has < 2 characters, it is not EOS
If the next token following whitespace begins with $ ({ [" ' it is EOS
Otherwise, the period is not EOS

Sentence Boundary Determination

- If we want to treat sentence boundary determination as a **classification problem**, what kind of features should we generate?
  - Since the object to be classified is a period, each feature vector corresponds to a period occurring in the text.
  - Next, we need to consider what characteristics of the surrounding text are useful features.
  - From the previous algorithm, we can see that the useful features are the characters or character classes near the period, including:
    - the characters of the **token** to which the period is attached, and
    - the characters of the following **token**

Sentence Boundary Determination

- What features best predict sentence boundaries?
  - Is preceding token a known abbreviation?
  - How long is preceding token?
  - Is preceding token capitalized?
  - Is succeeding token capitalized?
  - ….
- Create feature vectors for each potential boundary (**.**)
- Apply ML algorithm to produce classifier
- Test on held-out data

Sentence Boundary Determination: NLTK

```
from nltk import *
from nltk.corpus import gutenberg
import pprint

text = gutenberg.raw('chesterton-thursday.txt')
sents = sent_tokenize(text)
# print sents[:10] # the entire list on one line
pprint.pprint(sents[:10]) # each item on a separate line
```

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

The Man Who Was Thursday

From Wikipedia, the free encyclopedia

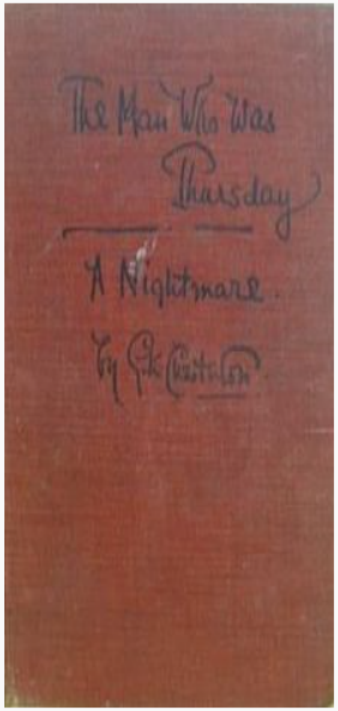
The Man Who Was Thursday: A Nightmare is a novel by G. K. Chesterton, first published in 1908. The book is sometimes referred to as a metaphysical thriller.

Contents [hide]

Plot summary [edit]

In Edwardian era London, Gabriel Syme is recruited at Scotland Yard to a secret anti-anarchist police corps. Lucian Gregory, an anarchistic poet, lives in the suburb of Saffron Park. Syme meets him at a party and they debate the meaning of poetry. Gregory argues that revolt is the basis of poetry. Syme demurs, insisting the essence of poetry is not revolution but law. He antagonizes Gregory by asserting that the most poetical of human creations is the timetable for the London Underground. He suggests Gregory isn't really serious about anarchism, which so irritates Gregory, that he takes Syme to an underground anarchist meeting place, revealing his public endorsement of anarchy is a ruse to make him seem

The Man Who Was Thursday: A Nightmare



First edition

Author	G. K. Chesterton
Country	United Kingdom
Language	English
Genre	Thriller

nltk.tokenize package

Submodules

nltk.tokenize.api module

Tokenizer Interface

class `nltk.tokenize.api.`**`StringTokenizer`** [source]

Bases: `nltk.tokenize.api.TokenizerI`

A tokenizer that divides a string into substrings by splitting on the specified string (defined in subclasses).

`span_tokenize`(*s*) [source]

`tokenize`(*s*) [source]

class `nltk.tokenize.api.`**`TokenizerI`** [source]

Bases: `object`

A processing interface for tokenizing a string. Subclasses must define `tokenize()` or `tokenize_sents()` (or both).

`span_tokenize`(*s*) [source]

TABLE OF CONTENTS

SEARCH

Go

Enter search terms or a module, class or function name.

Sentence Boundary Determination: OpenNLP

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;

public class FindingSentences {
    private static String paragraph = "..."
    public static void main(String[] args) {
        usingOpenNLP();
    }
   // In Capter 3 of NLP with Java that I have uploaded to Blackboard
    private static void usingOpenNLP() {
            try (InputStream is = new FileInputStream(new File("en-sent.bin"))) {…}
    }
}
```

openNLP

Models for 1.5 series

Use the links in the table below to download the pre-trained models for the OpenNLP 1.5 series.

The models are language dependent and only perform well if the model language matches the language of the input text. Also make sure the input text is decoded correctly, depending on the input file encoding this can only be done by explicitly specifying the character encoding. See this Java Tutorial section for further details.

Note: All models are zip compressed (like a jar file), they **must not** be uncompressed.

Language	Component	Description	Download
da	Tokenizer	Trained on conllx ddt data.	da-token.bin
da	Sentence Detector	Trained on conllx ddt data.	da-sent.bin
da	Part of Speech Tagger	Maxent model trained on conllx ddt data.	da-pos-maxent.bin
da	POS Tagger	Perceptron model trained on conllx ddt data.	da-pos-perceptron.bin
de	Tokenizer	Trained on tiger data.	de-token.bin
de	Sentence Detector	Trained on tiger data.	de-sent.bin
de	POS Tagger	Maxent model trained on tiger corpus.	de-pos-maxent.bin
de	POS Tagger	Perceptron model trained on tiger corpus.	de-pos-perceptron.bin
en	Tokenizer	Trained on opennlp training data.	en-token.bin
en	Sentence Detector	Trained on opennlp training data.	en-sent.bin
en	POS Tagger	Maxent model with tag dictionary.	en-pos-maxent.bin
en	POS Tagger	Perceptron model with tag dictionary.	en-pos-perceptron.bin
en	Name Finder	Date name finder model.	en-ner-date.bin

Part-Of-Speech Tagging

- In order to recognize named entities (names of people, places, and organizations), it is usually desirable to perform additional linguistic analyses of the text and extract more sophisticated features.
- Toward this end, the next logical step is to determine the **part of speech** (POS) of each token

Parts Of Speech

- In any natural language, words are organized into grammatical classes or parts of speech.
  - Almost all languages will have at least the categories we would call **nouns** and **verbs**.
  - The exact number of categories in a given language is not something intrinsic but depends on how the language is analyzed by an individual linguist.

Part-Of-Speech Tagging

- In English, some analyses may use as few as six or seven categories and others nearly one hundred.
  - Most English grammars would have, at a minimum, **noun**, **verb**, **adjective**, **adverb**, **preposition**, and **conjunction**.
- A bigger set of **36** categories is used in the Penn Tree Bank, constructed from the Wall Street Journal corpus that I mentioned before. (See next 3 slides)

(This web page is permanently under construction.)

The Penn Treebank Project

The Penn Treebank Project annotates naturally-occuring text for linguistic structure. Most notably, we produce skeletal parses showing rough syntactic and semantic information -- a *bank* of linguistic *trees*. We also annotate text with part-of-speech tags, and for the Switchboard corpus of telephone conversations, dysfluency annotation. We are located in the LINC Laboratory of the Computer and Information Science Department at the University of Pennsylvania.
All data produced by the Treebank is released through the Linguistic Data Consortium.

Descriptions and samples of annotated corpora:

Wall Street Journal | The Brown Corpus | Switchboard | ATIS

On-line tgrep searches are now possible for those with LDC Online access.

Frequently Asked Questions (FAQs)

- tokenization
- NP heads and Base NPs in Treebank II bracketing

Annotation Style Manuals

Alphabetical list of part-of-speech tags used in the Penn Treebank Project:

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Part-Of-Speech Tagging

- Dictionaries showing word-POS correspondence can be useful but are not sufficient.
  - All dictionaries have gaps, but even for words found in the dictionary, several parts of speech are usually possible. (ambiguity)
    - Returning to an earlier example, "**bore**" could be a noun, a present tense verb, or a past tense verb.
  - The goal of **POS tagging** is to determine which of these possibilities is realized in a particular text instance.

Part-Of-Speech Tagging

- Although it is possible, in principle, to manually construct a part-of-speech tagger (the rule-based approach), the most successful systems are generated automatically by machine-learning algorithms from annotated corpora.
  - Almost all POS taggers have been trained on the Wall Street Journal corpus available from LDC (**Linguistic Data Corporation**, **www.ldc.upenn.edu**) because it is the most easily available large annotated corpus.

Part-Of-Speech Tagging

- Although the WSJ corpus is large and reasonably diverse, it is one particular genre, and one cannot assume that a tagger based on the WSJ will perform as well on, for example, e-mail messages, or tweets.
  - Because much of the impetus for work on information extraction has been sponsored by the military, whose interest is largely in the processing of voluminous news sources, there has not been much support for generating large training corpora in other domains.

POS Tagging: NLTK

```
from nltk import *
from nltk.corpus import Gutenberg

text = gutenberg.raw('chesterton-thursday.txt')
sents = sent_tokenize(text)

for i in range(len(sents)):
    tokens = word_tokenize(sents[i])

    # Part-of-speech tagging
    POS_tagged_tokens = pos_tag(tokens)
    print POS_tagged_tokens
```

POS Tagging: OpenNLP

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;
import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.tokenize.WhitespaceTokenizer;
import opennlp.tools.util.Sequence;

public class POS {
    public static void main(String[] args) {
            usingOpenNLPPOSModel(); // in Blackboard (Chapter 5 of NLP with Java)
    }
    …
}
```

Named Entity Recognition (NER)

- A specialization of phrase finding, in particular noun phrase finding, is the recognition of particular types of **proper noun phrases**, specifically **persons**, **organizations**, and **locations**, sometimes along with money, dates, times, and percentages.
  - From the point of view of technique, this is very like the **phrase recognition** problem.
  - In fact, one might even want to identify noun phrases as a first step, and then assign the correct categories to the proper noun phrases (person, location, etc.).

One of the many differences between *Robert L. James*, *chairman and chief executive officer of McCann-Erickson*, and *John J. Dooner, Jr.*, the agency's president and chief operating officer, is quite telling: Mr. James enjoys sailboating, while Mr. Dooner owns a powerboat.

Now, Mr. James is preparing to sail into the sunset, and Mr. Dooner is poised to rev up the engines to guide *Interpublic Group's McCann-Erickson* into the 21st century. Yesterday, *McCann* made official what had been widely anticipated: *Mr. James*, *57* years old, is stepping down as chief executive officer on *July 1* and will retire as chairman at the *end of the year*. He will be succeeded by *Mr. Dooner*, *45* ...

Fig. 6.1 WSJ text with entity mentions emphasized by italic fonts

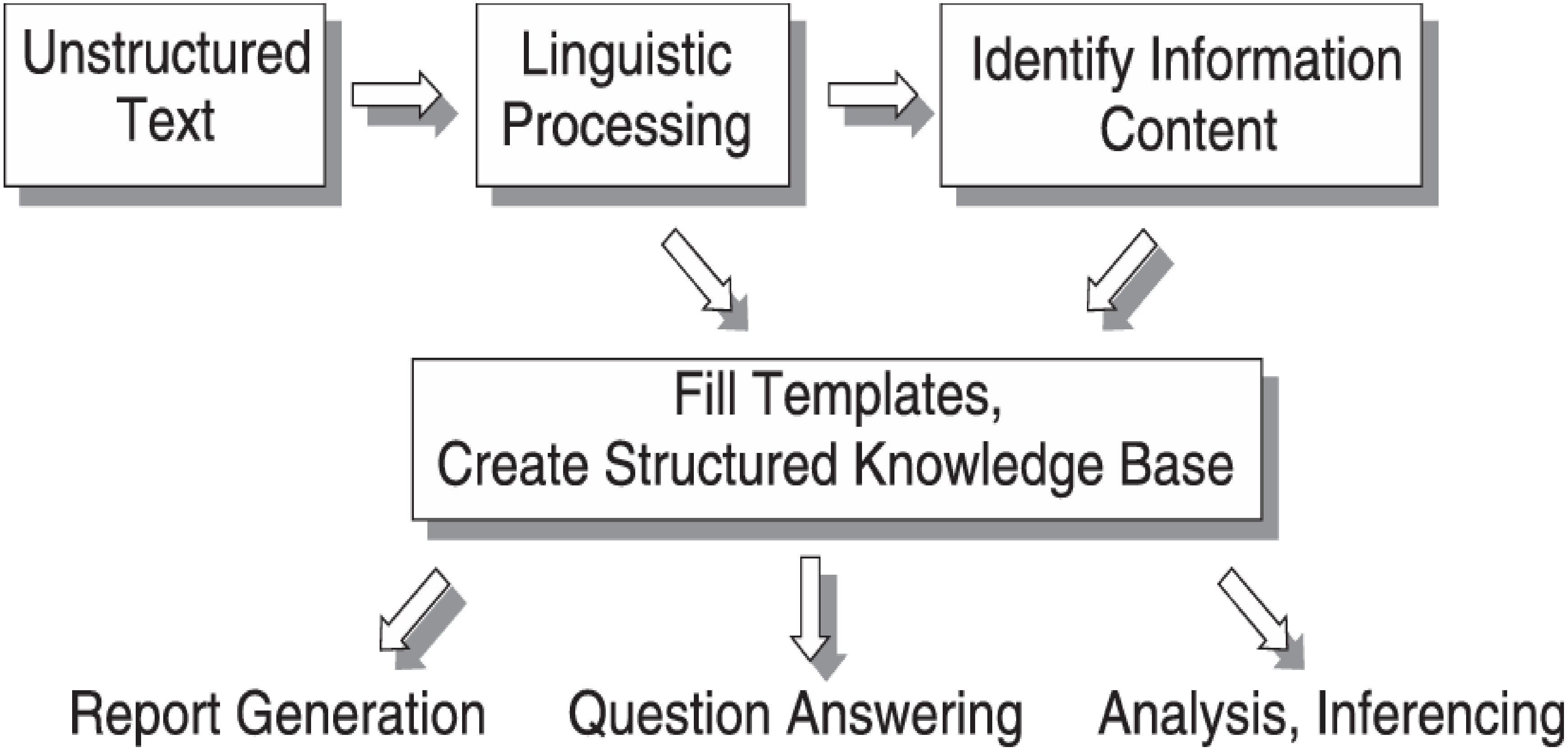
Table 6.1 Extracted position change information

Organization	*McCann-Erickson*
Position	*Chief executive officer*
Date	*July 1*
Outgoing person name	*Robert L. James*
Outgoing person age	*57*
Incoming person name	*John J. Dooner, Jr.*
Incoming person age	*45*

Information Extraction

- Information extraction typically consists of the following tasks: (See also next slide)
  - Tokenization
  - Sentence segmentation
  - POS Tagging
  - **NER**
  - Relationship extraction
    - Parsing
    - Semantic Analysis
    - Discourse Analysis
  - Template filling/merging

Information Extraction Systems

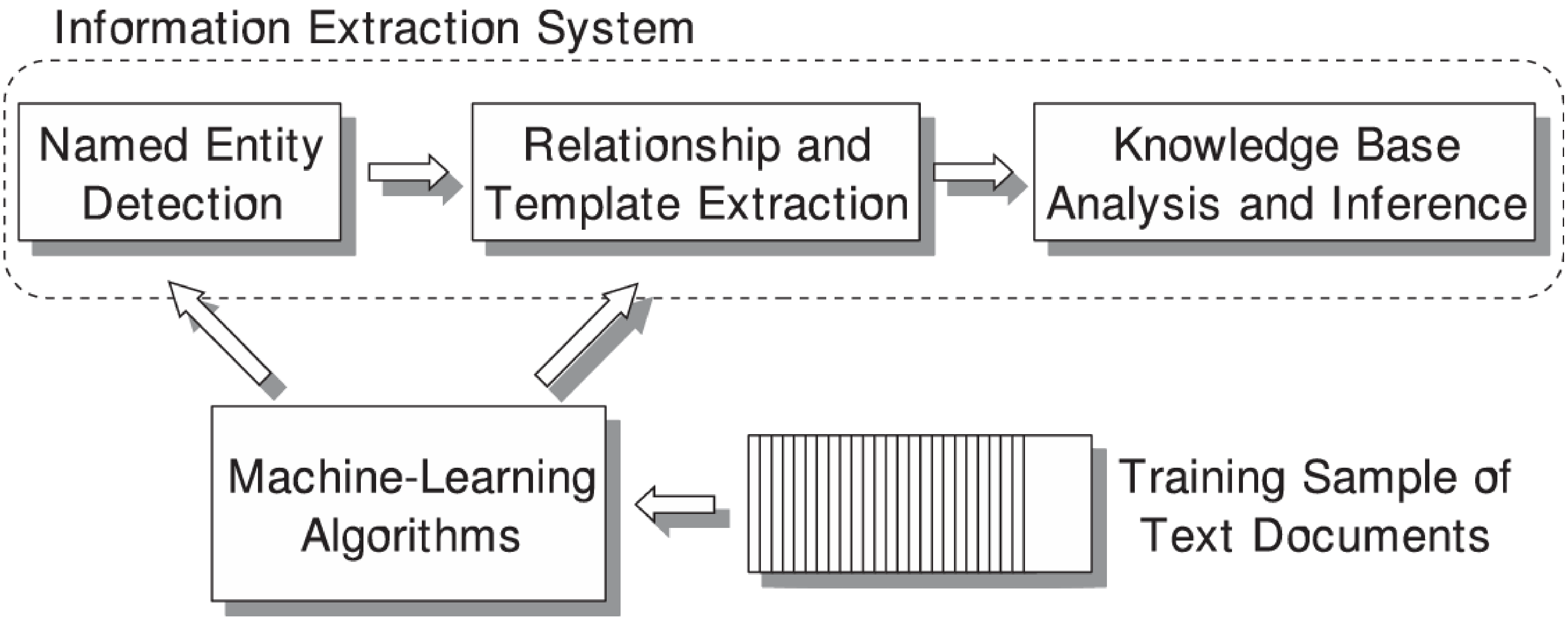


Information Extraction

- Although the most accurate information extraction systems often involve handcrafted language processing modules, substantial progress has been made in applying **machine-learning** techniques to a number of processes necessary for extracting information from text.

- The application of machine-learning techniques to information extraction is motivated by the time-consuming process needed to handcraft these systems.
  - Special expertise in linguistics, artificial intelligence and computational linguistics, as well as domain-specific information may be required.

Trainable Information Extraction Systems



NER as Sequential Tagging

- One way of looking at NER is to annotate **chunks** of text strings with some pre-specified types.
  - If you want to find mentions of **people**, **locations**, **organizations**, etc. in text, then our task is to divide text into syntactically related non-overlapping groups of words (**chunks**).

One of the many differences between Robert L. James, chairman and chief executive officer of McCann-Erickson,and John J. Dooner, Jr., the agency’s president and chief operating officer, is quite telling …

→

One of the many differences between **[PER Robert L. James]**, **[POS chairman and chief executive officer]** of **[ORG McCann-Erickson]**, and **[PER John J. Dooner, Jr.]**, the agency’s **[POS president and chief operating officer]**, is quite telling…

NER as Sequential Tagging

- The most successful machine-learning-based approach to this task regards the problem as a **token-based tagging** problem.
  - The idea is to divide text into tokens (words) and then assign each token a tag value that encodes the chunking information.
  - There are many different encoding schemes. One commonly used method is to represent chunks by the following three types of tags:

    B-X: first word of a chunk of type X,

    I-X: non-initial word in a chunk of type X,

    O: word outside of any chunk.
  - As an example, the sentence considered before in this section can be tokenized and annotated as shown in Fig. 6.4. of our textbook. (Next slide)

NER as Sequential Tagging

One	of	the	many	differences	between	Robert	L.	James	,
O	O	O	O	O	O	B-PER	I-PER	I-PER	O

chairman	and	chief	executive	officer	of	McCann-Erickson	,
B-POS	I-POS	I-POS	I-POS	I-POS	O	B-ORG	O

and	John	J.	Dooner	,	Jr.	,	...
O	B-PER	I-PER	I-PER	I-PER	I-PER	O	

NER as Sequential Tagging

- Given the previous encoding scheme, we can now view NER as a sequential prediction problem:
  - We predict the **class label** associated with every **token** in a sequence of tokens.
  - In our case, each token is either a word or punctuation in the text.
- The advantage of this approach is that the task now becomes a simpler classification problem, where the goal is to predict the class label associated with every token.

Tag Prediction as Classification

- In order to determine the **label** of a token, we create a **feature vector** for this token.
  - The label is determined by the feature vectors.

- We use $\boldsymbol{x} = [x_1, \ldots, x_d]$ to denote a *d*-dimensional **feature vector** associated with a token and $\boldsymbol{t}$ to denote the tag value (lable) of the token. (See next slide for an example.)

- The task is to estimate the conditional probability $\mathbf{Pr}(t|x)$

- We can then assign the token a label that has the largest conditional probability score $\mathbf{Pr}(t|x)$

A Feature Vector for 'German'

Token Sequence: EU rejects German call to boycott British lamb .

Feature Type	Nonzero Features for *German*
Previous two labels	$Tok-2$ is labeled I-ORG
	$Tok-1$ is labeled O
Initial capitalizations in window of ± 2	Current token starts with a capital letter
	$Tok-2$ starts with a capital letter
All capitalizations in window of ± 2	$Tok-2$ is all capitalized
Prefix strings of length ≤ 4 of current token	G
	Ge
	Ger
	Germ
Suffix strings of length ≤ 4 of current token	rman
	man
	an
	n
Positional tokens in window of ± 2	*German* at position 0
	call at position $+1$
	to at position $+2$
	rejects at position -1
	EU at position -2

Naive Bayes Classifiers

- Is a statistical classifier
  - It performs probabilistic prediction, i.e., predicts class membership probabilities
- Is well-founded
  - Based on **Bayes' Theorem**.
- Is simple
  - A simple Bayesian classifier, **naïve Bayesian** classifier, has comparable performance with decision tree and neural network classifiers
- Is standard
  - They provide a standard of optimal decision making against which other methods can be measured

Bayes' Theorem

$$P(H\,|\,\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H)\times P(H)/P(\mathbf{X})$$

- Informally, this can be written as

  posterior = likelihood * prior / evidence

- Let **X** be a data sample ("*evidence*"): class label is unknown
- Let H be a *hypothesis* that X belongs to class C
- Classification is to determine P(H|**X**), *posterior probability*, the probability that the hypothesis holds given the observed data sample **X**
- P(H): *prior probability*, the initial probability
- P(**X**): probability that sample data is observed
- P(**X**|H): *likelihood*, the probability of observing the sample **X**, given that the hypothesis holds

Naïve Bayes Classifiers

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an **n** attribute vector $\mathbf{X} = (x_1, x_2, \ldots, x_n)$
- Suppose there are *m* classes $C_1, C_2, \ldots, C_m$.
- Classification is to find the maximum posterior probability, i.e., the maximal $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since $P(\mathbf{X})$ is constant for all classes, only

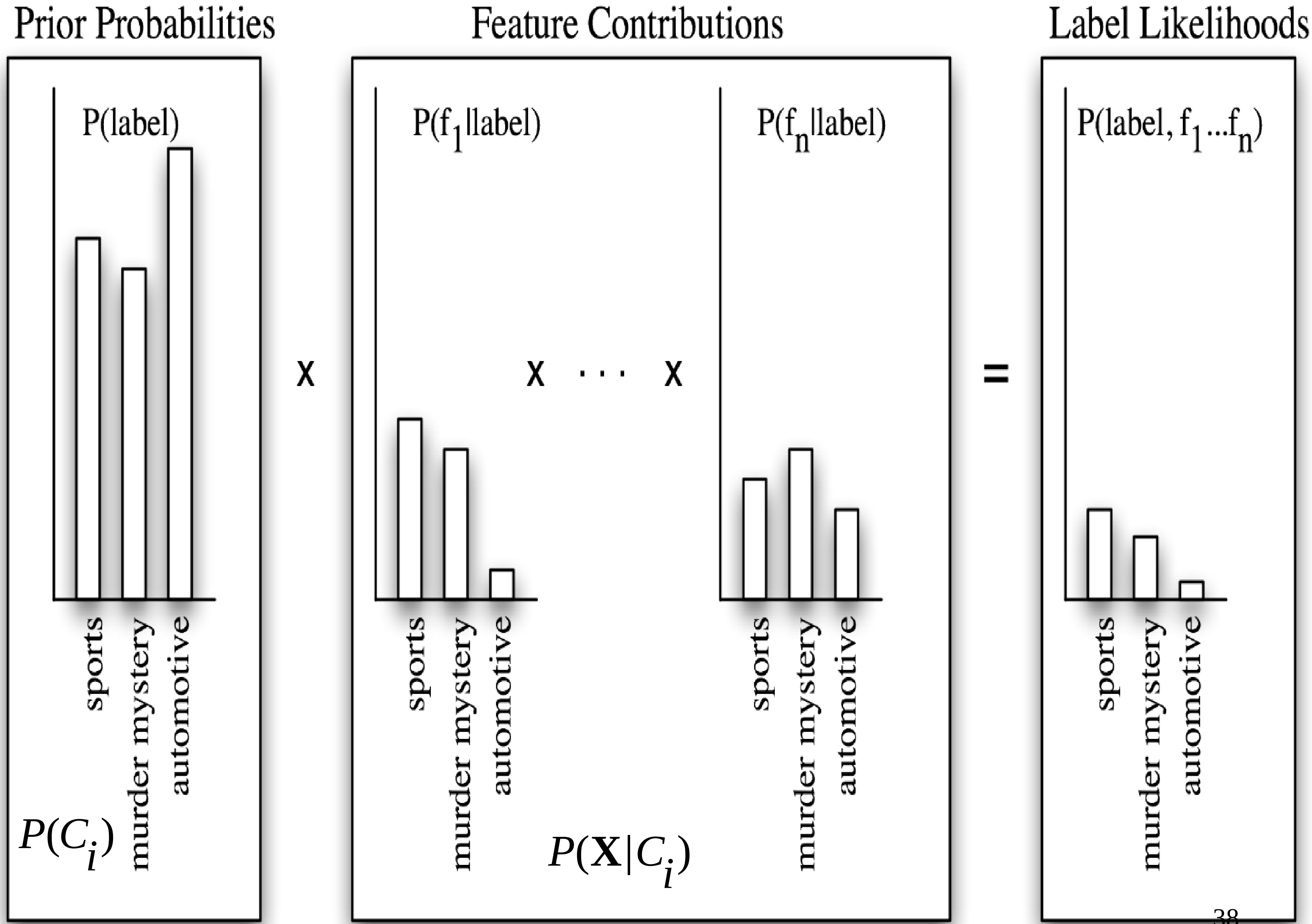
$$P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Naïve Bayes Classifiers

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$



Prior Probabilities
Feature Contributions
Label Likelihoods
P(label)
P(f1|label)
P(fn|label)
P(label, f1...fn)
x
x · · · x
=
sports
murder mystery
automotive
sports
murder mystery
automotive
sports
murder mystery
automotive
sports
murder mystery
automotive
$P(C_i)$
$P(\mathbf{X}|C_i)$

Zero Counts and Smoothing

- As mentioned earlier, the simplest way to calculate ***P(f|label)*** is to take the percentage of training instances with the given label that also have the given **feature.**

$$P(f|label) = count(f, label) / count(label)$$

- This approach becomes problematic when a feature never occurs with a given label in the training set.
- In this case, our calculated value for *P(f|label)* will be zero, which will cause the label likelihood for the given label to be zero.
- Thus, the input will never be assigned this label, regardless of how well the other features fit the label

Zero Counts and Smoothing

- The basic problem here is that, just because we haven't seen a feature/label combination occur in the training set, it doesn't mean it's impossible for that combination to occur.
  - For example, we may not have seen any murder mystery documents that contained the word "**football**," but we wouldn't want to conclude that it's completely impossible for such documents to exist.

Zero Counts and Smoothing

- Thus, although **count(f,label)/count(label)** is a good estimate for *P(f|label)* when *count(f, label)* is relatively high, this estimate becomes less reliable when *count(f)* becomes smaller.

- Therefore, when building naive Bayes models, we usually employ more sophisticated techniques, known as **smoothing** techniques, for calculating *P(f|label)*, the probability of a feature given a label.
  - **Laplacian Estimation:** 1 + c*ount(f, label)*
  - **Expected Likelihood Estimation:** 0.5 + c*ount(f, label)*
  - The **nltk.probability** module provides support for a wide variety of smoothing techniques.

The Maximum Entropy Principle

- Suppose we are assigned the task of picking the correct word sense for a given word, from a list of ten possible senses (labeled A-J).
- At first, we are not told anything more about the word or the senses.
- There are many probability distributions that we could choose for the ten senses, such as:

	A	B	C	D	E	F	G	H	I	J
i	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
ii	5%	15%	0%	30%	0%	8%	12%	0%	6%	24%
iii	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%

The Maximum Entropy Principle

- *(i)* is considered more "fair" than the other two, based one the concept of **entropy**.
- If a single label dominates then entropy is low, but if the labels are more evenly distributed then entropy is high.
- In our example, we chose distribution *(i)* because its label probabilities are evenly distributed - in other words, because its entropy is high. → the **Maximum Entropy principle:**
  - Among the distributions that are consistent with what we know, we should choose the distribution whose entropy is

	A	B	C	D	E	F	G	H	I	J
i	**10%**	**10%**	**10%**	**10%**	**10%**	**10%**	**10%**	**10%**	**10%**	**10%**
ii	**5%**	**15%**	**0%**	**30%**	**0%**	**8%**	**12%**	**0%**	**6%**	**24%**
iii	**0%**	**100%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**

The Maximum Entropy Principle

- Next, suppose that we are told that sense A appears 55% of the time.
- Once again, there are many distributions that are consistent with this new piece of information, such as:

	A	B	C	D	E	F	G	H	I	J
iv	55%	**45%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
v	**55%**	**5%**	**5%**	**5%**	**5%**	**5%**	**5%**	**5%**	**5%**	**5%**
vi	**55%**	**3%**	**1%**	**2%**	**9%**	**5%**	**0%**	**25%**	**0%**	**0%**

The Maximum Entropy Principle

- Finally, suppose that we are told that the word "**up**" appears in the nearby context **10%** of the time, and that when it does appear in the context there's an **80%** chance that sense A or C will be used.
- In this case, we will have a harder time coming up with an appropriate distribution by hand; however, we can verify that the following distribution looks appropriate:

	A	B	C	D	E	F	G	H	I	J
vii+up	**5.1%**	**0.25%**	**2.9%**	**0.25%**	**0.25%**	**0.25%**	**0.25%**	**0.25%**	**0.25%**	**0.25%**
vii -up	**49.9%**	**4.46%**	**4.46%**	**4.46%**	**4.46%**	**4.46%**	**4.46%**	**4.46%**	**4.46%**	**4.46%**

Maximum Entropy Classifiers

- The Maximum Entropy classifier uses a model that is very similar to the model employed by the naive Bayes classifier. The main difference is, rather than using **probabilities** to set the model's parameters, it looks for the set of parameters that maximizes the **total likelihood** of the training corpus, which is defined as:

$$P(features) = \Sigma_{x}\, P(label(x)|features(x))$$

where *P(label|features)*, the probability that input features, *features(x)*, will have the class label, *P(label(x)*, is defined as:

$$P(label|features) = P(label, features) \,/\, \Sigma_{label}\, P(label, features)$$

- See the handout (from **Foundations of Statistical NLP,** pp. 589-594) for more math details.

Maximum Entropy Classifiers

- Because of the potentially complex interactions between the effects of related features, there is no way to directly calculate the model parameters that maximize the likelihood of the training set.
  - Therefore, the model parameters are chosen based on using **iterative optimization techniques**, which initialize the model's parameters to random values, and then repeatedly refine those parameters to bring them closer to the optimal solution. (See the handout for the standard **GIS** algorithm.)
  - They guarantee that each refinement of the parameters will bring them closer to the optimal values.
    - It has been shown that the **GIS** algorithm always converges to a probability distribution that maximizes the entropy.
    - However, due to its iterative nature, it can take a long time to learn.
      - This is especially true when the size of the training set, the number of features, and the number of labels are all large.

Generative vs Conditional Classifiers

- The naive Bayes classifier is an example of a **generative** classifier, which builds a model that predicts *P(input, label)*, the joint probability of a *(input, label)* pair
- As a result, generative models can be used to answer the following questions:
  1. How likely is a given label for a given input?
  2. What is the most likely label for a given input?
  3. How likely is a given input value?
  4. What is the most likely input value?
  5. How likely is a given input value with a given label?
  6. What is the most likely label for an input that might have one of two values (but we don't know which)?

Generative vs Conditional Classifiers

- The Maximum Entropy classifier, on the other hand, is an example of a **conditional** (or **discriminative**) classifier.
- **Conditional/discriminative** classifiers build models that predict *P(label|input)* - the probability of a label *given* the input value.
- Thus, conditional models can still be used to answer questions 1 and 2, but not 3-6.

Generative vs Conditional Classifiers

- In general, generative models are strictly more powerful than conditional models, since we can calculate the conditional probability *P(label|input)* from the joint probability *P(input, label)*, but not vice versa.
  - However, this additional power comes at a price.
  - Because the model is more powerful, it has more "free parameters" which need to be learned. However, the size of the training set is fixed.
  - Thus, when using a more powerful model, we end up with less data that can be used to train each parameter's value, making it harder to find the best parameter values.
  - As a result, a generative model may not do as good a job at answering questions 1 and 2 as a conditional model, since the conditional model can focus its efforts on those two questions.

Generative vs Conditional Classifiers

- The difference between a generative model and a conditional model is analogous to the difference between a **topographical map** and a **picture of a skyline**. (NLP with Python)
  - Although the topographical map can be used to answer a wider variety of questions, it is significantly more difficult to generate an accurate topographical map than it is to generate an accurate skyline.

NER: NLTK

```
from nltk import *
from nltk.corpus import gutenberg
import pprint

text = gutenberg.raw('chesterton-thursday.txt')
sents = sent_tokenize(text)
for i in range(len(sents)):
        tokens = word_tokenize(sents[i])
        # Part-of-speech tagging
        POS_tagged_tokens = pos_tag(tokens)
         # Named Entity Recognition
        print ne_chunk(POS_tagged_tokens, binary=False)
```

nltk.chunk package

Submodules

nltk.chunk.api module

class `nltk.chunk.api.`**`ChunkParserI`** [source]

Bases: `nltk.parse.api.ParserI`

A processing interface for identifying non-overlapping groups in unrestricted text. Typically, chunk parsers are used to find base syntactic constituents, such as base noun phrases. Unlike `ParserI`, `ChunkParserI` guarantees that the `parse()` method will always generate a parse.

`evaluate`(*gold*) [source]

Score the accuracy of the chunker against the gold standard. Remove the chunking the gold standard text, rechunk it using the chunker, and return a `ChunkScore` object reflecting the performance of this chunk peraser.

Parameters: **gold** (*list(Tree)*) – The list of chunked sentences to score the chunker on.
Return type: ChunkScore

`parse`(*tokens*) [source]

Return the best chunk structure for the given tokens and return a tree.

TABLE OF CONTENTS

SEARCH

Go

Enter search terms or a module, class or function name.

nltk.classify package

Submodules

nltk.classify.api module

Interfaces for labeling tokens with category labels (or "class labels").

`ClassifierI` is a standard interface for "single-category classification", in which the set of categories is known, the number of categories is finite, and each text belongs to exactly one category.

`MultiClassifierI` is a standard interface for "multi-category classification", which is like single-category classification except that each text belongs to zero or more categories.

class `nltk.classify.api.`**`ClassifierI`** [source]

Bases: `object`

A processing interface for labeling tokens with a single category label (or "class"). Labels are typically strs or ints, but can be any immutable type. The set of labels that the classifier chooses from must be fixed and finite.

Subclasses must define:

- `labels()`

TABLE OF CONTENTS

SEARCH

Go

Enter search terms or a module, class or function name.

NER: OpenNLP

```
import java.io.File; import java.io.FileInputStream; import java.io.InputStream;
import java.util.ArrayList;
import opennlp.tools.namefind.NameFinderME;
import opennlp.tools.namefind.TokenNameFinderModel;
import opennlp.tools.tokenize.Tokenizer;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;
import opennlp.tools.util.Span;

public class NER {
    private static final String sentences[] = {"Joe was the last person to see Fred. ",...}
    public static void main(String[] args) { usingOpenNLP(); }
        private static void usingOpenNLP() {
            usingOpenNLPNameFinderME(); // in Chapter 4
            usingMultipleNERModels(); // in Chapter 4
        } …
    }
}
```

openNLP

Models for 1.5 series

Use the links in the table below to download the pre-trained models for the OpenNLP 1.5 series.

The models are language dependent and only perform well if the model language matches the language of the input text. Also make sure the input text is decoded correctly, depending on the input file encoding this can only be done by explicitly specifying the character encoding. See this Java Tutorial section for further details.

Note: All models are zip compressed (like a jar file), they **must not** be uncompressed.

Language	Component	Description	Download
da	Tokenizer	Trained on conllx ddt data.	da-token.bin
da	Sentence Detector	Trained on conllx ddt data.	da-sent.bin
da	Part of Speech Tagger	Maxent model trained on conllx ddt data.	da-pos-maxent.bin
da	POS Tagger	Perceptron model trained on conllx ddt data.	da-pos-perceptron.bin
de	Tokenizer	Trained on tiger data.	de-token.bin
de	Sentence Detector	Trained on tiger data.	de-sent.bin
de	POS Tagger	Maxent model trained on tiger corpus.	de-pos-maxent.bin
de	POS Tagger	Perceptron model trained on tiger corpus.	de-pos-perceptron.bin
en	Tokenizer	Trained on opennlp training data.	en-token.bin
en	Sentence Detector	Trained on opennlp training data.	en-sent.bin
en	POS Tagger	Maxent model with tag dictionary.	en-pos-maxent.bin
en	POS Tagger	Perceptron model with tag dictionary.	en-pos-perceptron.bin
en	Name Finder	Date name finder model.	en-ner-date.bin

NER: Stanford CoreNLP

```
import edu.stanford.nlp.ie.crf.CRFClassifier;
import edu.stanford.nlp.ling.CoreAnnotations;
import edu.stanford.nlp.ling.CoreLabel;
import java.util.List;

public class NER_Stanford {
    private static final String sentences[] = {"Joe was the last person to see Fred. ",…}
    public static void main(String[] args) { usingStanfordNER(); }
    private static void usingStanfordNER() { // in Chapter 4
        String model = "english.conll.4class.distsim.crf.ser.gz";
        CRFClassifier<CoreLabel> classifier = CRFClassifier.getClassifierNoExceptions(model);

        String sentence = "";
        for (String element : sentences) {
                        sentence += element;
        } …
    }
}
```

An Introduction to Conditional Random Fields

Charles Sutton
University of Edinburgh
csutton@inf.ed.ac.uk

Andrew McCallum
University of Massachusetts Amherst
mccallum@cs.umass.edu

17 November 2010

Abstract

Often we wish to predict a large number of variables that depend on each other as well as on other observed variables. Structured prediction methods are essentially a combination of classification and graphical modeling, combining the ability of graphical models to compactly model multivariate data with the ability of classification methods to perform prediction using large sets of input features. This tutorial describes *conditional random fields*, a popular probabilistic method for structured prediction. CRFs have seen wide application in natural language processing, computer vision, and bioinformatics. We describe methods for inference and parameter estimation for CRFs, including practical issues for implementing large scale CRFs. We do not assume previous knowledge of graphical modeling, so this tutorial is intended to be useful to practitioners in a wide variety of fields.

Software > Stanford Named Entity Recognizer

Stanford Named Entity Recognizer (NER)

About | Getting started | Questions | Mailing lists | Download | Extensions | Models | Online demo | Release history | FAQ

About

Stanford NER is a Java implementation of a Named Entity Recognizer. Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names. It comes with well-engineered feature extractors for Named Entity Recognition, and many options for defining feature extractors. Included with the download are good named entity recognizers for English, particularly for the 3 classes (PERSON, ORGANIZATION, LOCATION), and we also make available on this page various other models for different languages and circumstances, including models trained on just the CoNLL 2003 English training data.

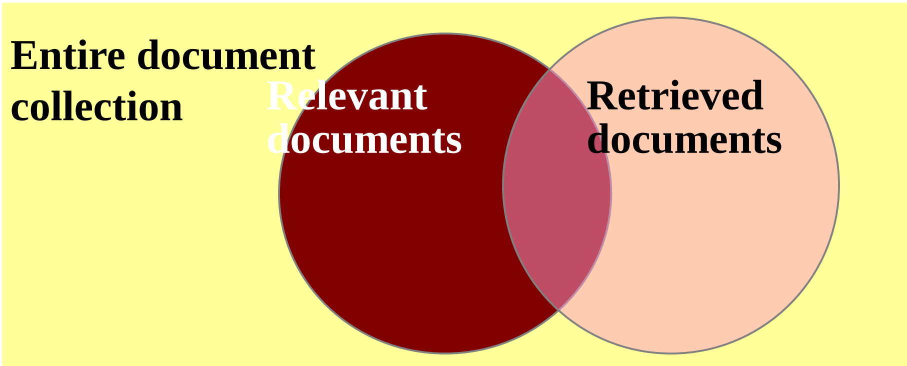
Stanford NER is also known as CRFClassifier. The software provides a general implementation of (arbitrary order) linear chain Conditional Random Field (CRF) sequence models. That is, by training your own models on labeled data, you can actually use this code to build sequence models for NER or any other task. (CRF models were pioneered by Lafferty, McCallum, and Pereira (2001); see Sutton and McCallum (2006) or Sutton and McCallum (2010) for more comprehensible introductions.)

The CRF code is by Jenny Finkel. The feature extractors are by Dan Klein, Christopher Manning, and Jenny Finkel. Much of the documentation and usability is due to Anna Rafferty. The CRF sequence models provided here do not precisely correspond to any published paper, but the correct paper to cite for the software is:

> Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, pp. 363-370. `http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf`

The software provided here is similar to the baseline local+Viterbi model in that paper, but adds new distributional similarity based features (in the `-distSim` classifiers). The distributional similarity features in some models improve performance but the models require considerably more memory. The big models were trained on a mixture of CoNLL

Precision and Recall:



	retrieved	not retrieved
relevant	retrieved & relevant	Not retrieved & relevant
irrelevant	retrieved & irrelevant	not retrieved but irrelevant

$$recall = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ relevant\ documents}$$

$$precision = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ documents\ retrieved}$$

To consolidate Precision & Recall into 1 measure: **F = 2PR/(P+R)**

F-score / F-measure [edit]

Main article: F-score

The weighted harmonic mean of precision and recall, the traditional F-measure or balanced F-score is:

$$F = \frac{2 \cdot \mathrm{precision} \cdot \mathrm{recall}}{(\mathrm{precision} + \mathrm{recall})}$$

This is also known as the F_1 measure, because recall and precision are evenly weighted.

The general formula for non-negative real β is:

$$F_\beta = \frac{(1 + \beta^2) \cdot (\mathrm{precision} \cdot \mathrm{recall})}{(\beta^2 \cdot \mathrm{precision} + \mathrm{recall})}$$

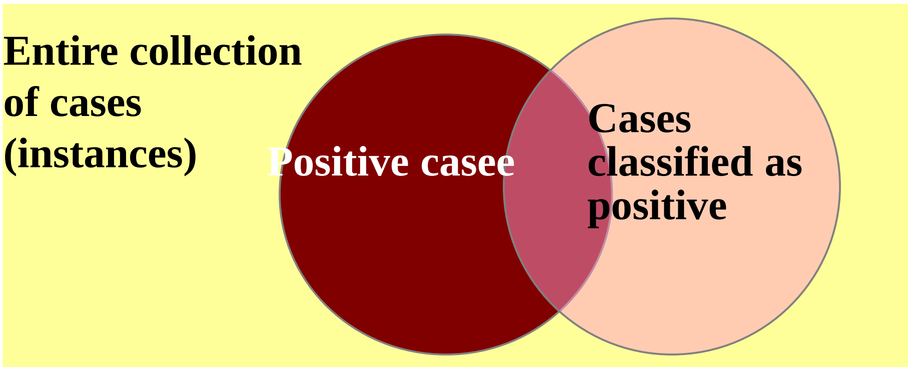
Two other commonly used F measures are the F_2 measure, which weights recall twice as much as precision, and the $F_{0.5}$ measure, which weights precision twice as much as recall.

The F-measure was derived by van Rijsbergen (1979) so that F_β "measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision". It is based on van Rijsbergen's effectiveness measure $E = 1 - \frac{1}{\frac{\alpha}{P} + \frac{1-\alpha}{R}}$. Their relationship is:

$$F_\beta = 1 - E \text{ where } \alpha = \frac{1}{1 + \beta^2}$$

F-measure can be a better single metric when compared to precision and recall; both precision and recall give different information that can complement each other when combined. If one of them excels more than the other, F-measure will reflect it.[citation needed]

Precision and Recall: Classification



	Classified as Positive	Classified as Negative
Positive	True Positive	False Negative
Negative	False Positive	True Negative

$$recall = \frac{True\ \ positive\ cases}{Total\ number\ of\ \ positive\ cases}$$

$$precision = \frac{True\ \ positive\ cases}{Total\ number\ of\ \ cases\ classified\ \ as\ \ positvie}$$

Assignment #2 (15 points): Due March 11

- What are some of the key factors that can affect the quality of results from a NER system? (Exercise 6-3)
  1. Select a NER system that you want to use for this assignment
  2. Select at least 50 sentences (with NEs) from a ‘news’ type dataset
  3. Select at least 50 tweets (with NEs) from your collection
  4. Run both the **news data** and **tweets data** through the NER system of your choice
  5. Evaluate their results separately (in terms of **Precision**, **Recall** and **F-score**). You may choose to do just 1 category, **Person**, or more.
  6. Report your evaluation results (from Step 5), and then do Exercise 6-3, based on your evaluation results, and perhaps additional experiments of your own design. Suggest at least **3 factors**.
  7. Submit your input, output, evaluation results and the report in a zipped folder, as before, via Blackboard.

List of text mining software

From Wikipedia, the free encyclopedia

Text mining computer programs are available from many commercial and open source companies and sources.

Contents [hide]

Commercial [edit]

- Angoss – Angoss Text Analytics provides entity and theme extraction, topic categorization, sentiment analysis and document summarization capabilities via the embedded Lexalytics Salience Engine. The software provides the unique capability of merging the output of unstructured, text-based analysis with structured data to provide additional predictive variables for improved predictive models and association analysis.
- Attensity – hosted, integrated and stand-alone text mining (analytics) software that uses natural language processing technology to address collective intelligence in social media and forums; the voice of the customer in surveys and emails; customer relationship management; e-services; research and e-discovery; risk and compliance; and intelligence analysis.
- AUTINDEX - is a commercial text mining software package based on sophisticated linguistics by IAI (Institute for Applied Information Sciences), Saarbrücken.
- Autonomy – text mining, clustering and categorization software
- Averbis – provides text analytics, clustering and categorization software, as well as terminology management and enterprise search
- Basis Technology – provides a suite of text analysis modules to identify language, enable search in more than 20 languages, extract entities, and efficiently search for and translate entities.

Open source [edit]

- Carrot2 – text and search results clustering framework.
- GATE – General Architecture for Text Engineering, an open-source toolbox for natural language processing and language engineering
- Gensim - large-scale topic modelling and extraction of semantic information from unstructured text (Python)
- OpenNLP - natural language processing
- Natural Language Toolkit (NLTK) – a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for the Python programming language.
- Orange with its text mining add-on.
- Text Mechanic – Simple, single task, browser based, text manipulation tools.[4]
- The programming language R provides a framework for text mining applications in the package *tm*.[5] The Natural Language Processing task view contains *tm* and other text mining library packages.[6]
- The KNIME Text Processing extension.
- The PLOS Text Mining Collection[7]

References [edit]

1. ^ Alba, Davet (12 February 2015). "The Startup That Helps You Analyze Twitter Chatter in Real Time". Wired. Retrieved 4 March 2015.
2. ^ Lohr, Steve (27 June 2014). "The U.S.-Germany Match Through a Social Media Lens". New York Times. Retrieved 4 March 2015.
3. ^ ":: Welcome to Abzooba ::". Abzooba.com. Retrieved 2013-10-13.
4. ^ "Text Mechanic". TextMechanic.
5. ^ Introduction to the tm Package: Text Mining in R
6. ^ CRAN Task View: Natural Language Processing
7. ^ "Table of Contents: Text Mining". PLOS.

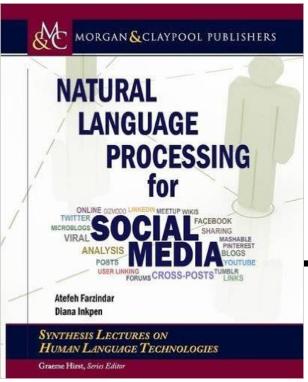
External links [edit]

- Text Mining APIs on Mashape
- Text Mining APIs on Programmable Web

Categories: Data mining and machine learning software | Lists of software

NER on Tweets

- Named entity recognition methods typically have **85-90**% accuracy on long and carefully edited texts, but their performance decreases to **30-50**% on tweets.
  - Ritter et al. [2011] reported that the Stanford NER obtains **44**% accuracy on Twitter data.
  - They also presented new NER methods for social media texts that allowed their NER system to reach an accuracy of **67**%.



Named Entity Recognition in Tweets: An Experimental Study

Alan Ritter, Sam Clark, Mausam and Oren Etzioni
Computer Science and Engineering
University of Washington
Seattle, WA 98125, USA
`{aritter,ssclark,mausam,etzioni}@cs.washington.edu`

Abstract

People tweet more than 100 Million times daily, yielding a noisy, informal, but sometimes informative corpus of 140-character messages that mirrors the zeitgeist in an unprecedented manner. The performance of standard NLP tools is severely degraded on tweets. This paper addresses this issue by re-building the NLP pipeline beginning with part-of-speech tagging, through chunking, to named-entity recognition. Our novel T-NER system doubles F_1 score compared with the Stanford NER system. T-NER leverages the redundancy inherent in tweets to achieve this performance, using LabeledLDA to exploit Freebase dictionaries as a source of distant supervision. LabeledLDA outperforms co-training, increasing F_1 by 25% over ten common entity types.

Our NLP tools are available at: `http://github.com/aritter/twitter_nlp`

the size of the Library of Congress (Hachman, 2011) and is growing far more rapidly. Due to the volume of tweets, it is natural to consider named-entity recognition, information extraction, and text mining over tweets. Not surprisingly, the performance of "off the shelf" NLP tools, which were trained on news corpora, is weak on tweet corpora.

In response, we report on a re-trained "NLP pipeline" that leverages previously-tagged out-of-domain text, [2] tagged tweets, and unlabeled tweets to achieve more effective part-of-speech tagging, chunking, and named-entity recognition.

1	The Hobbit has FINALLY started filming! I cannot wait!
2	Yess! Yess! Its official Nintendo announced today that they Will release the Nintendo 3DS in north America march 27 for $250
3	Government confirms blast n nuclear plants n japan...don't knw wht s gona happen nw...

Table 1: Examples of noisy text in tweets.

https://github.com/aritter/twitter_nlp
Identified by Digi...
aritter/twitter_nlp: Twitter ...

This repository Search
Pull requests Issues Gist

aritter / twitter_nlp

Watch 77 | Star 383 | Fork 168

Code | Issues 7 | Pull requests 1 | Wiki | Pulse | Graphs

Twitter NLP Tools

57 commits | 2 branches | 0 releases | 1 contributor

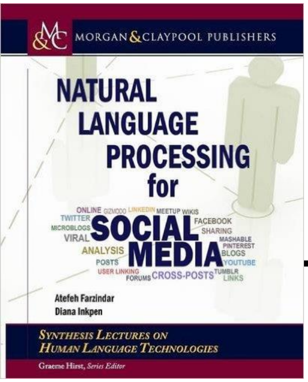
Branch: master | New pull request | New file | Find file | HTTPS | https://github.com/aritte | Download ZIP

aritter updated contact info — Latest commit b5b4f2c on Nov 9 2015

data	removed a name by request	4 months ago
hbc	added labels for weakly supervised NE categorization	2 years ago
lib	added README.md	4 years ago
mallet-2.0.6	re-importing to blow away some large files in the history	5 years ago
models	Fixed a bug in computing brown clusters reported by Yiye Ruan and Lu ...	2 years ago
python	Fixed a bug in computing brown clusters reported by Yiye Ruan and Lu ...	2 years ago
LICENSE	re-importing to blow away some large files in the history	5 years ago
README.md	updated contact info	3 months ago
TinySVM-0.09.tar.gz	build.sh should now work to compile libraries, etc...	4 years ago
build.sh	moved -lm switch to end of the line	4 years ago

NER on Tweets

- ❖ Derczynski et al. [2013b] reported that NER performance drops from **77%** F-score on newspaper text to **60%** on Twitter data, and that after adaptation it increases to **80%** (with the ANNIE NER system from GATE) .
  - ❖ The performance on newspaper data was computed on the CoNLL 2003 English NER dataset.
  - ❖ The performance on social media data was computed on the Ritter tweet dataset [Ritter et al., 2011], which contains of 2,400 tweets comprising 34,000 tokens.



Twitter Part-of-Speech Tagging for All: Overcoming Sparse and Noisy Data

Leon Derczynski
University of Sheffield
leon@dcs.shef.ac.uk

Alan Ritter
University of Washington
aritter@cs.washington.edu

Sam Clark
University of Washington
ssclark@cs.washington.edu

Kalina Bontcheva
University of Sheffield
kalina@dcs.shef.ac.uk

Abstract

Part-of-speech information i
many NLP algorithms. How
is difficult to part-of-speech ta
with linguistic errors and idiosync
We present a detailed error analysis o
ing taggers, motivating a series of ta
mentations which are demonstrat
performance. We identify a
niques for improving Eng
tagging performance in this ge

Further, we present a novel appro ys-
tem combination for the case where available
taggers use different tagsets, based on vote-
constrained bootstrapping with unlabeled data.

of label make microblog part-of-speech tag-
gin . Alongside the genre's informal
age "compressed" utterances,
not only needless words but also
tical or contextualising function.
ch tagging is a central problem in natu-
processing, and a key step early in manly
Machine learning-based part-of-speech
xploit labeled training data to adapt
languages, through supervised
phistication apart, the perfor-
man gers is reliant upon the quantity
and qua vailable training data. Consequently,
lacking larg PoS-annotated resources and faced with
prevalent noise, state-of-the-art PoS taggers perform
poorly on microblog text (Derczynski et al., 2013),

Passive-Aggressive Sequence Labeling with Discriminative Post-Editing for Recognising Person Entities in Tweets

Leon Derczynski
University of Sheffield
leon@dcs.shef.ac.uk

Kalina Bontcheva
University of Sheffield
kalina@dcs.shef.ac.uk

Abstract

Recognising entities in social media text is difficult. NER on newswire text is conventionally cast as a sequence labeling problem. This makes implicit assumptions regarding its textual structure. Social media text is rich in disfluency and often has poor or noisy structure, and intuitively does not always satisfy these assumptions. We explore noise-tolerant methods for sequence labeling and apply discriminative post-editing to exceed state-of-the-art performance for person recognition in tweets, reaching an F1 of 84%.

2 Background

Named entity recognition is a well-studied problem, especially on newswire and other long-document genres (Nadeau and Sekine, 2007; Ratinov and Roth, 2009). However, experiments show that state-of-the-art NER systems from these genres do not transfer well to social media text.

For example, one of the best performing general-purpose named entity recognisers (hereon referred to as Stanford NER) is based on linear-chain conditional random fields (CRF) (Finkel et al., 2005). The model is trained on newswire data and has a number of optimisations, including distributional similarity measures and sam-