Ling 131A
Introduction to NLP with Python

Tagging

Marc Verhagen, Fall 2017

Contents

- Final project questions?
- Assignment 4 questions?
- Language models
- POS tagging
  - Chain rule
  - Bigram tagger
  - Brill tagger
- Train and test corpora; evaluation measures

Language Models

- Set of rules or a statistical model that underly some aspect of language
- Bigram statistics can model what sentences look like
  - What is the next word given the previous words in the sentence?
  - What is the probability of a sentence?
  - The probability of all sentences in a language must sum to 1.
- The Mutual Information score that we saw a few weeks ago can model collocations

Language Models

- Formal grammars (e.g. regular, context free) give a hard binary model of all the legal sentences in a language.
- For NLP, a probabilistic model of a language that gives a probability that a string is a member of a language can be more useful.

What is POS-tagging?

- Part of speech (POS): also known as lexical category or sometimes as word class
- POS-tagging: the process of assigning part-of-speech tags to words in a corpus (automatically or manually)
  - Usually one part-of-speech per word in that context
  - Thus resolving some lexical ambiguity.
- Tagset: The collection of tags for a particular task

Two scenarios

- Given a POS-tagged corpus, identify linguistically interesting phenomena
  - The Brown corpus, the Penn Treebank
- Given raw text, automatically assign POS tags to all words (POS tagging)

What you can do with POS tagged data

- What is the most common POS tag?
- What is the most frequent POS tag following another POS tag?
- What is the most frequent verb/noun in a POS-tagged corpus?
- Find the most ambiguous words in a pos-tagged corpus
- Some of these questions may be part of assignment 5

Automatic POS-tagging

- Two issues: ambiguity, unknown words
- Approaches:
  - Rule-based
    - Regular expressions
  - Machine learning
    - Learning statistics (n-gram models)
    - Learning rules (transformation-based learning)

Part of speech (POS) ambiguity

Words	Possible Tags	Example of Use
that	subordinating conjunction	***that he can swim is good***
	determiner	***that white table***
	adverb	***it is not that easy***
	pronoun	***that is the table***
	relative pronoun	***the table that collapsed***
round	verb	***round the usual suspects***
	preposition	***turn round the corner***
	noun	***a big round***
	adjective	***a round box***
	adverb	***he went round***
table	noun	***that white table***
	verb	***I table that***
might	noun	***the might of the wind***
	modal verb	***she might come***
collapse	noun	***the collapse of the empire***
	verb	***the empire can collapse***

Penn Treebank Tagset

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	*+, %, &*
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	*#*
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PRP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PRP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

Automatic POS-tagging: using context

- Water
  - *Water* the → verb
  - The *water* → noun
- Context is morphological, syntactic or semantic

Morphological context

- Inflectional morphology
  - Verb:
    - destroy, destroying, destroyed
  - Noun:
    - destruction, destructions
- Derivational morphology
  - Noun:
    - destruction

Syntactic context

- Verb:
  - The bomb destroyed the building.
  - He decided to water the plant.
- Noun:
  - The destruction of building

Semantic context

- Verb: action, activity
- Noun: state, object, etc.
- Not directly observable, therefore hard to exploit, but it's there:
  - A noun in one language is usually translated into a noun in another language

Baselines and toplines

- Baseline: what is the least you can do?
  - Assigning the most frequent tag to all words
  - Using regular expressions to exploit morphological information
  - Assigning to each word its mostly likely tag
- Topline: what is the best you can do?
  - Use a combination of morphological, syntactic, and semantic context
  - Determining the contribution of each type of context, and mixing them up in the optimal way

Baseline 1

- The Default Tagger assigns the most likely tag

```
import nltk
from nltk.corpus import brown

brown_news_tagged = brown.tagged_words(categories='news')
brown_sents = brown.sents(categories='news')

default_tagger = nltk.DefaultTagger('NN')
default_tagger.tag(brown_sents[10])
```

Baseline #2

- The Regular Expression Tagger uses a couple of morphological rules


```
import nltk
from nltk.corpus import brown
```



```
brown_news_tagged = brown.tagged_words(categories='news')
brown_sents = brown.sents(categories='news')
```



```
patterns = […]
regexp_tagger = nltk.RegexpTagger(patterns)
regexp_tagger.tag(brown_sents[10])
```

Baseline #2

- Regular expression tagger rules
  - VB: base form, e.g., ‘go’
  - VBZ: 3rd person singular present, e.g., goes
  - VBN: past participle, e.g., ‘gone’
  - VBG: gerund, e.g., ‘going’
  - VBD: simple past, ‘went’
  - VBP: non-3rd person singular present

Baseline #3

- The Unigram Tagger tags a word token with its most likely tag, regardless of the context


```
import nltk
from nltk.corpus import brown
```



```
brown_news_tagged = brown.tagged_words(categories='news')
brown_sents = brown.sents(categories='news')
```



```
# training step
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)

# running the tagger
unigram_tagger.tag(brown_sents[10])
```

Today

- Final project questions?
- Assignment 4 questions?
- POS tagging
  - Chain rule
  - Bigram tagger
  - Brill tagger

N-Gram Tagging

- Based on statistics derived from a set of n-grams (unigrams, bigrams, trigrams).
- Bigram example:
  - Given the frequencies of
    - <The/DET water/NOUN>
    - <The/DET water/VERB>
  - What is the most likely tag for water if it follows <The/DET>

N-Gram Model Formulas

- Chain rule of probability

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)\ldots P(w_n \mid w_1^{n-1}) = \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

- Bigram approximation

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-1})$$

Chain Rule

- Calculating the probability of a sequence of words given conditional probabilities
- Conditional probability
  - measure of the probability of an event given that another event has occurred
  - P(cat|the) =
    the probability of "cat" if we already have "the"
  - P(the) . P(cat|the) =
    the probability of "the cat"

Chain Rule Formula

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k | w_1^{k-1})$$

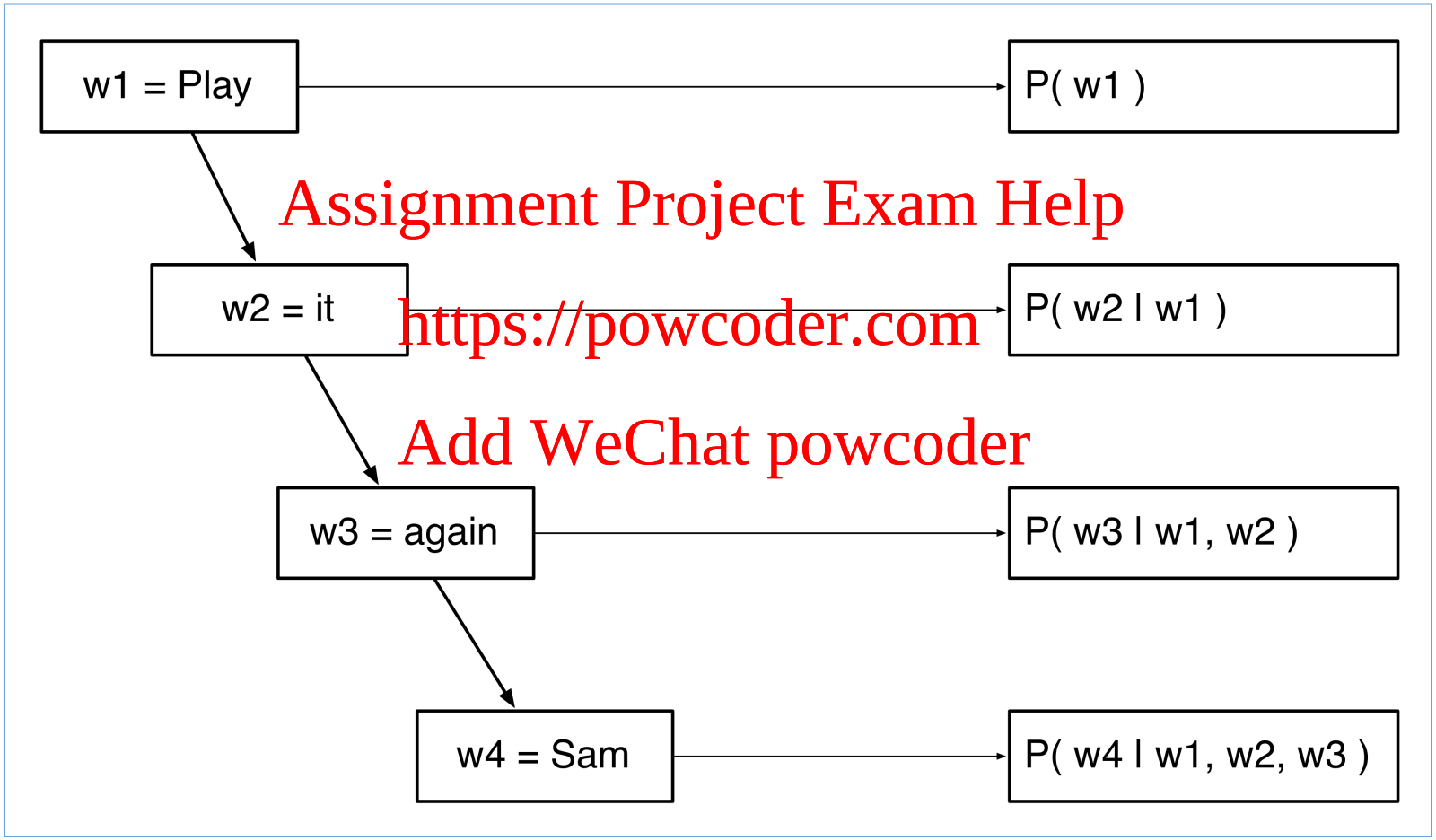
where

w_k	=	the word at index k
w_1^n	=	the sequence $w_1, w_2, ...w_n$
$P(A\|B)$	=	the probability of A given B
$\prod_{k=1}^{n} \Phi$	=	the product of all formulas Φ with k ranging from 1 through n

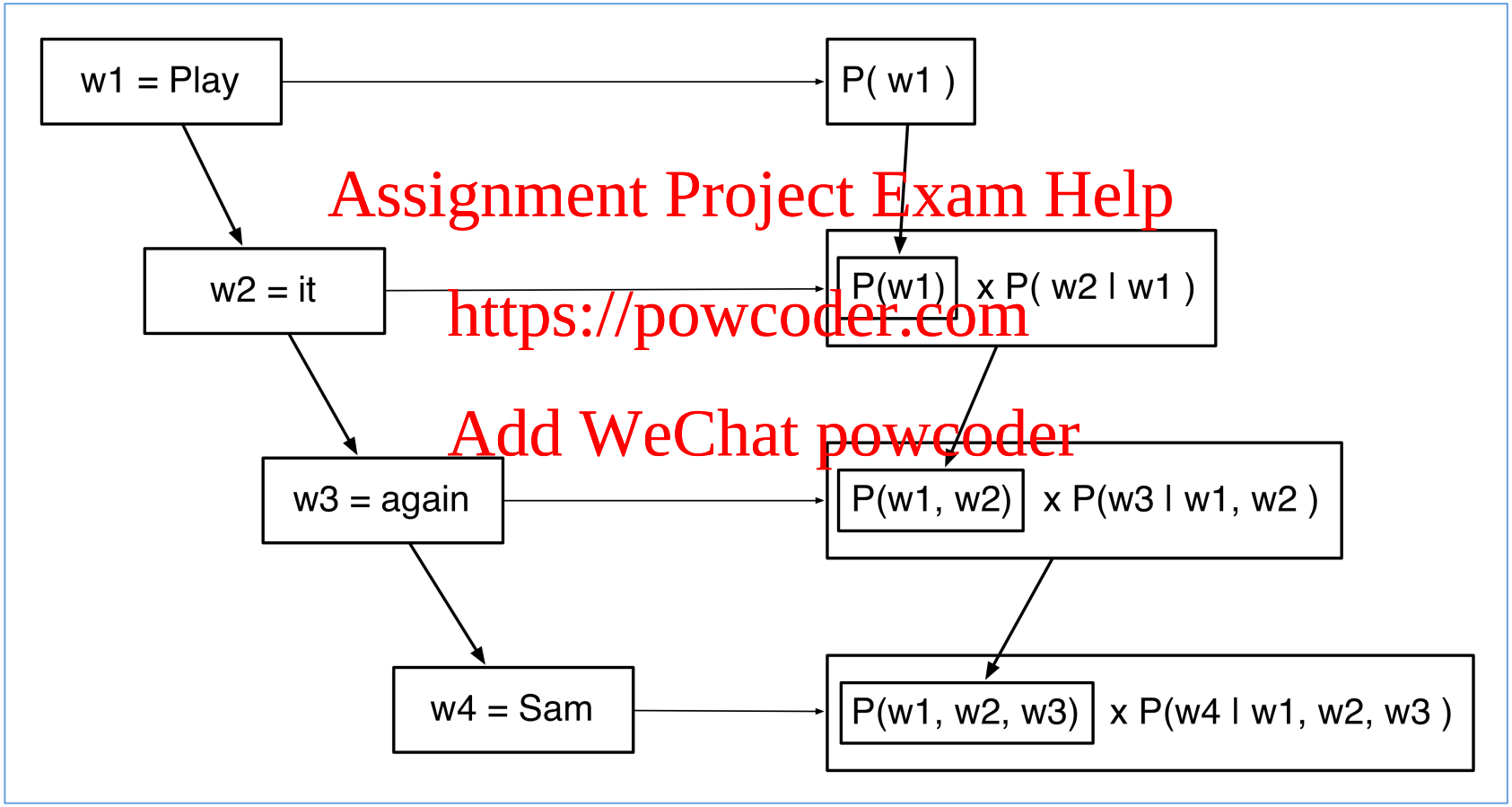
Chain Rule Example

- Input: “Play it again Sam”
- What is the probability that
  - "it" will follow "play"?
  - "again" will follow "play it"?
  - "Sam" will follow "play it again"?

Chain Rule Example



Chain Rule Example



Beyond words

- We used w_k to stand in for just the word
- Instead we could use something like $<w_k, t_k>$, that is, a pair of a word and its tag
- For example
  - instead of looking for the probability of "cat" following "the"
  - we can look for the probability of the noun "cat" following the determiner "the"
- Or we can use any combination of features

Problems

- Need very big memory
- Combinatorics of storing the probabilities of words given any preceding sequence is $O(2^n)$

N-Gram Models

- With the regular chain rule we estimates the probability of each word given prior context.
- An N-gram model uses only N-1 words of prior context.
  - Unigram: P(sleeps)
  - Bigram: P(sleeps | cat)
  - Trigram: P(sleeps | the cat)
- **Markov assumption**: the future behavior of a system only depends on its recent history
  - in a **kth-order Markov model**, the next state only depends on the *k* most recent states (an N-gram model is a (N-1)-order Markov model)

Bigram Model

- The probability of the sequence ABCD using all context is
  - P(A) . P(B|A) . P(C|AB) . P(D|ABC)
- But with the Markov assumption you can throw out your long-term memory
  - P(A) . P(B|A) . P(C|B) . P(D|C)
  - P(play it again sam)
    = P(play) . P(it|play) . P(it|again) . P(sam|again)
    = P(play|S).* P(it|play) . P(it|again) . P(sam|again)

Estimating Probabilities

- N-gram conditional probabilities can be estimated from raw text based on the ***relative frequency*** of word sequences.

Bigram: $P(w_n \mid w_{n-1}) = \dfrac{C(w_{n-1}w_n)}{C(w_{n-1})}$

N-gram: $P(w_n \mid w_{n-N+1}^{n-1}) = \dfrac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$

To have a consistent probabilistic model, append a unique start (<s>) and end (</s>) symbol to every sentence and treat these as additional words.

Examples

- P(<s> i want english food </s>)
  = P(i | <s>) P(want | i) P(english | want)
    P(food | english) P(</s> | food)
  = .25 x .33 x .0011 x .5 x .68 = .000031
- P(<s> i want chinese food </s>)
  = P(i | <s>) P(want | i) P(chinese | want)
    P(food | chinese) P(</s> | food)
  = .25 x .33 x .0065 x .52 x .68 = .00019

Unknown Words

- How to handle words in the test corpus that did not occur in the training data, i.e. ***out of vocabulary*** (OOV) words?
- Train a model that includes an explicit symbol for an unknown word (<UNK>).
  - Choose a vocabulary in advance and replace other words in the training corpus with <UNK>.
  - Replace the first occurrence of each word in the training data with <UNK>.

Smoothing

- Many rare yet impossible combinations never occur in training (sparse data), so many parameters are zero
  - If a new combination occurs during testing, it is given a probability of zero and the entire sequence gets a probability of zero.
- In practice, parameters are smoothed (regularized) to reassign some probability mass to unseen events.
- Adding probability mass to unseen events requires removing it from seen ones in order to maintain a joint distribution that sums to 1.

Laplace Smoothing

- Aka "Add-One Smoothing"
- "Hallucinate" additional training data in which each possible N-gram occurs exactly once and adjust estimates accordingly.

Bigram: $$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

V is the total number of possible (N-1)-grams (i.e. the vocabulary size for a bigram model).

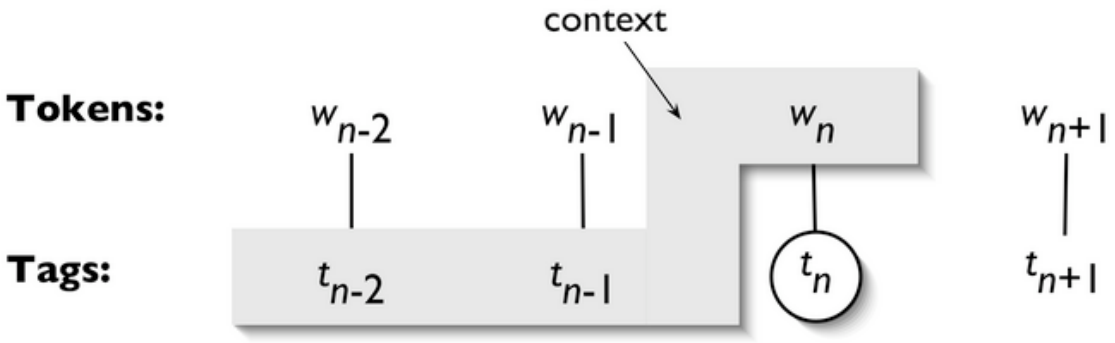
Backoff

- Use a simpler model for those cases where your more complicated (yet better) model has nothing to say

But wait a minute…
none of this was about tagging

Context

- Unigram tagging
  - Using one item of context, the word itself.
- N-Gram tagging
  - context is the current word together with the pos tags of the n-1 preceding tokens



context can include more than just the tag itself

Conditional probability

- $P(tag_i \mid tag_{i-1}, w_i)$
- So we are looking for the probability of tag_i given that the previous tag was tag_{i-1} and the word itself is w_i

Drawbacks of N-Gram approaches

- Conditioned on just the previous tags, even though word tokens can be useful as well
  - Conditioning on word tokens is unrealistic
- Size of the n-grams (bi-gram and tri-gram tables) can be large, and the n-grams are hard to interpret
- Transformation-based learning addresses these issues

Transformation-based Tagging

- Transformation-Based Error Driven Learning (TBL)
  - Aka the Brill tagger, after its inventor Eric Brill
  - a simple rule-based approach to automated learning of linguistic knowledge
- Idea:
  - first solve a problem with a simple technique
  - then apply transformations to correct mistakes

Tagging with transformation rules

Phrase	to	increase	grants	to	states	for	vocational	rehabilitation
Unigram	TO	*NN*	NNS	*TO*	NNS	IN	JJ	NN
Rule 1		*VB*						
Rule 2				*IN*				
Output	TO	VB	NNS	IN	NNS	IN	JJ	NN
Gold	TO	VB	NNS	IN	NNS	IN	JJ	NN

Tagging with transformation rules

The phrase *The can rusted* has two readings

Let's suppose that *can*/modal is more frequent than *can*/noun in our corpus

First step: Assign the most likely POS

The/art **can**/modal **rusted**/verb

Second step: Apply rules

Change the tag from modal to noun if one of the two previous words is an article

This is the idea of the Brill tagger.

Sample Rule Templates

Rules	Explanation
`alter(A, B, prevtag(C))`	Change A to B if previous tag is C
`alter(A, B, prev1or2or3tag(C))`	Change A to B if previous one or two or three tag is C
`alter(A, B, prev1or2tag(C))`	Change A to B if previous one or two tag is C
`alter(A, B, next1or2tag(C))`	Change A to B if next one or two tag is C
`alter(A, B, nexttag(C))`	Change A to B if next tag is C
`alter(A, B, surroundingtag(C, D))`	Change A to B if surrounding tags are C and D
`alter(A, B, nextbigram(C, D))`	Change A to B if next bigram tag is C D
`alter(A, B, prevbigram(C, D))`	Change A to B if previous bigram tag is C D

Templates vs rules

- A rule is an instance of a template
- Template: alter (A, B, prevtag(C))
- Rules:
  - alter (modal, noun, prevtag(article))
  - alter (noun, verb, prevtag (to))
  - alter (verb, noun, prevtag(article))
  - alter (verb, noun, prevtag(adjective))

Is there a way to learn these rules automatically?

Two Components in Transformation

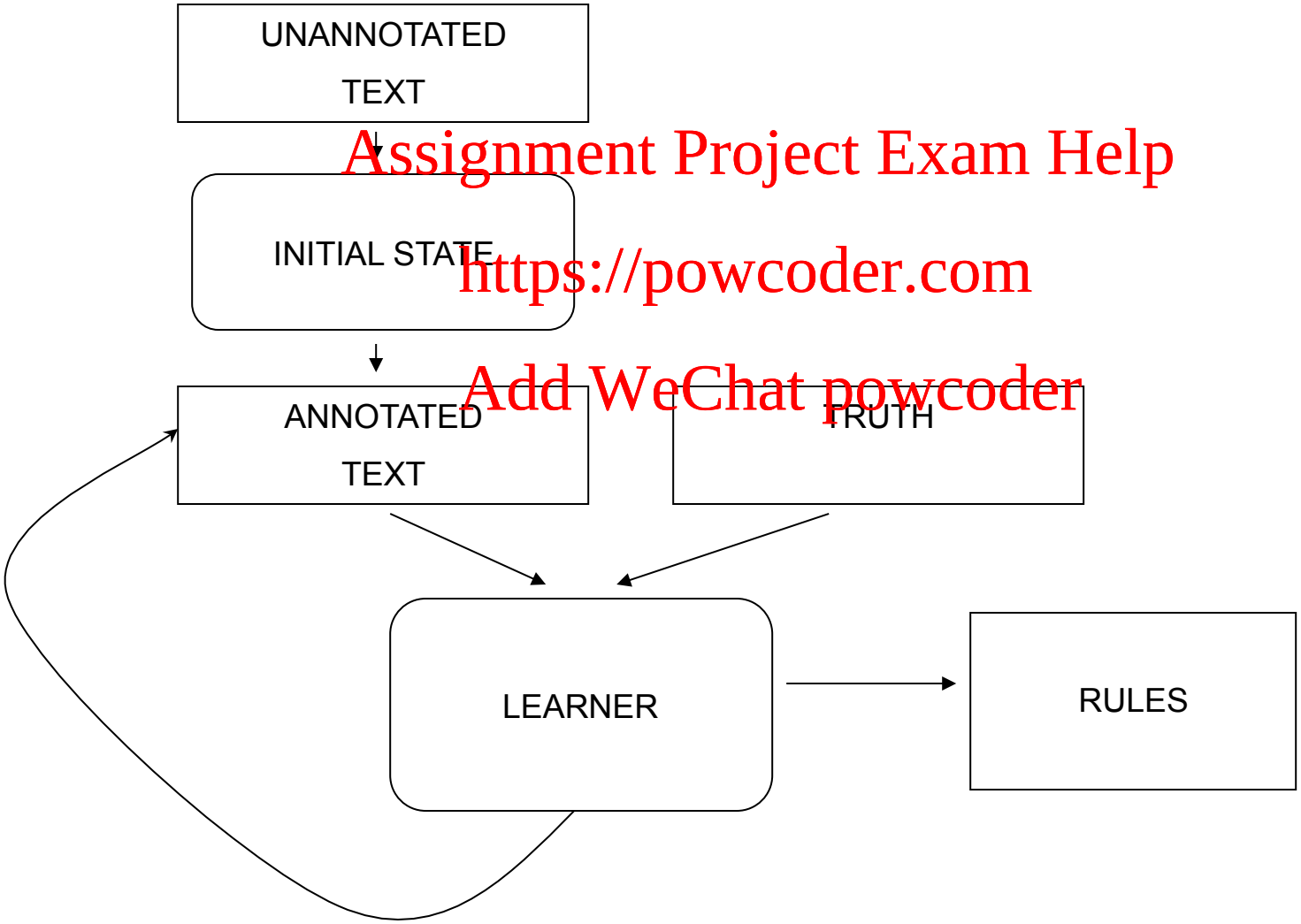
- A rewrite rule (Action)
  - ex. Change the tag from modal to noun
- A triggering environment (Condition)
  - ex. The preceding word is a determiner

The/det can/modal rusted/verb ./.

to

The/det can/noun rusted/verb ./.

Transformation-Based Error-Driven Learning



Learning procedure

- Initial tagger:
  - Assigning the most likely tag to each word
  - Deciding the most likely tag based on a training corpus
  - Tag a test corpus with this initial tagger
- Transformation
  - Write some rules based on the templates
  - Using the training corpus to spot patterns
  - Applying transformations to the test corpus
- Evaluate your accuracy

Train and Test Corpora

- A language model must be trained on a large corpus of text.
- Model can be evaluated based on its ability to generate good results on a held-out test corpus
  - testing on the training corpus gives an optimistically biased estimate
- Training and test corpus should be representative of the actual application data.
  - May need to adapt a general model to a small amount of new in-domain data by adding highly weighted small corpus to original training data.

Evaluation Measures

- Accuracy: percentage of correct tags
- Precision, recall and f-measure
  - FP - false positives
  - TP - true positives
  - FN - false negatives


```
John lives in London
ENT  -     -  ENT
-    ENT   -  ENT
FN   FP    TN TP
```

Evaluation Measures

- Precision = tp / (tp + fp)
- Recall = tp / (tp + fn)
- F-measure = 2 . (P . R) / (P + R)
  - Harmonic means
  - Penalty if P or R is pretty bad