Homework

- Read chapter 7
- Repeat confusable experiments using Timbl. Try different options settings to see what works best.
- **WARNING:** Timbl's idea of C4.5 format isn't quite the same as C4.5's
- Now try the same thing without using part-of-speech tags. You can use words, parts of words, capitalization, etc., but not tags
- How well does this work? What kinds of things does it still get wrong?

Memory-based Tagging

- MBT (Zavrel, Daelemans, van den Bosch) is a part-of-speech tagging system based on decision tree and instance based learners
- First step: construct a lexicon of words in training data, annotated with ‘ambitags’
- Next, construct classifiers for known and unknown words using a sliding window of features
- Classifier for known words uses two previously assigned tags, ambitag of current word, and ambitag of next word
- Classifier for unknown words uses previously assigned tag, ambitag of next word, and the first letter and last three letters of current word

Memory-based Tagging

- Case representation for known word IGTREE classifier:

	d	d	f	a	t
Pierre	=	=	np	np	np
Vinken	=	np	np	,	np
,	np	np	,	cd	,
61	np	,	cd	nns	cd
years	,	cd	nns	jj-np	nns
old	cd	nns	jj-np	,	jj

- Case representation for unknown word IB1 classifier:

	p	d	a	s	s	s	t
Pierre	P	=	np	r	r	e	np
Vinken	V	np	np	k	e	n	np
61	6	,	nns	=	6	1	cd
years	y	cd	jj-np	a	r	s	nns
old	o	nns	,	o	l	d	jj

Memory-based Tagging

- Tagging proceeds left to right, with no backtracking or global optimization
- Training and application of model is relatively cheap, not too far from HMM methods
- Results are competitive with the state of the art: 96.4% on WSJ
- Used as tagger for Corpus Gesproken Nederlands project
- Source code freely available, easy to use

Memory-based Tagging

- LOB corpus tagging experiment:

System	Acc (%)
Trigram	96.1
TBL	96.6
MBT	97.0
MaxEnt	97.4
Ensemble	97.9

- Tagging homework assignment:

System	Acc (%)
baseline	90.5
bigram	94.1
MBT (bigram)	94.1
TnT	96.1
MBT	96.5

Instance-based learning

- IBL classifiers relate query instances to most similar instance in training data
- Applies a very simple learning rule (use majority class) to a local neighborhood of the feature space
- Can be used for both discrete and continuous features and target functions
- Variants like IB2 and TRIBL can improve efficiency without reducing accuracy too much
- Can run into overfitting problems, but in a way (it's been argued) that is particularly well suited to NLP problems

Rule induction

- Early AI systems and not-so-early NLP systems used large collections of hand written rules
- Rule induction system try to produce sets of rules automatically from training data
- Classic rule-based systems: RIPPER (propositional logic), FOIL (first order)
- Inductive Logic Programming produces Horn clauses (Prolog programs)
- Grammar induction
- Decision trees, Transformation-based learning

Trees to rules

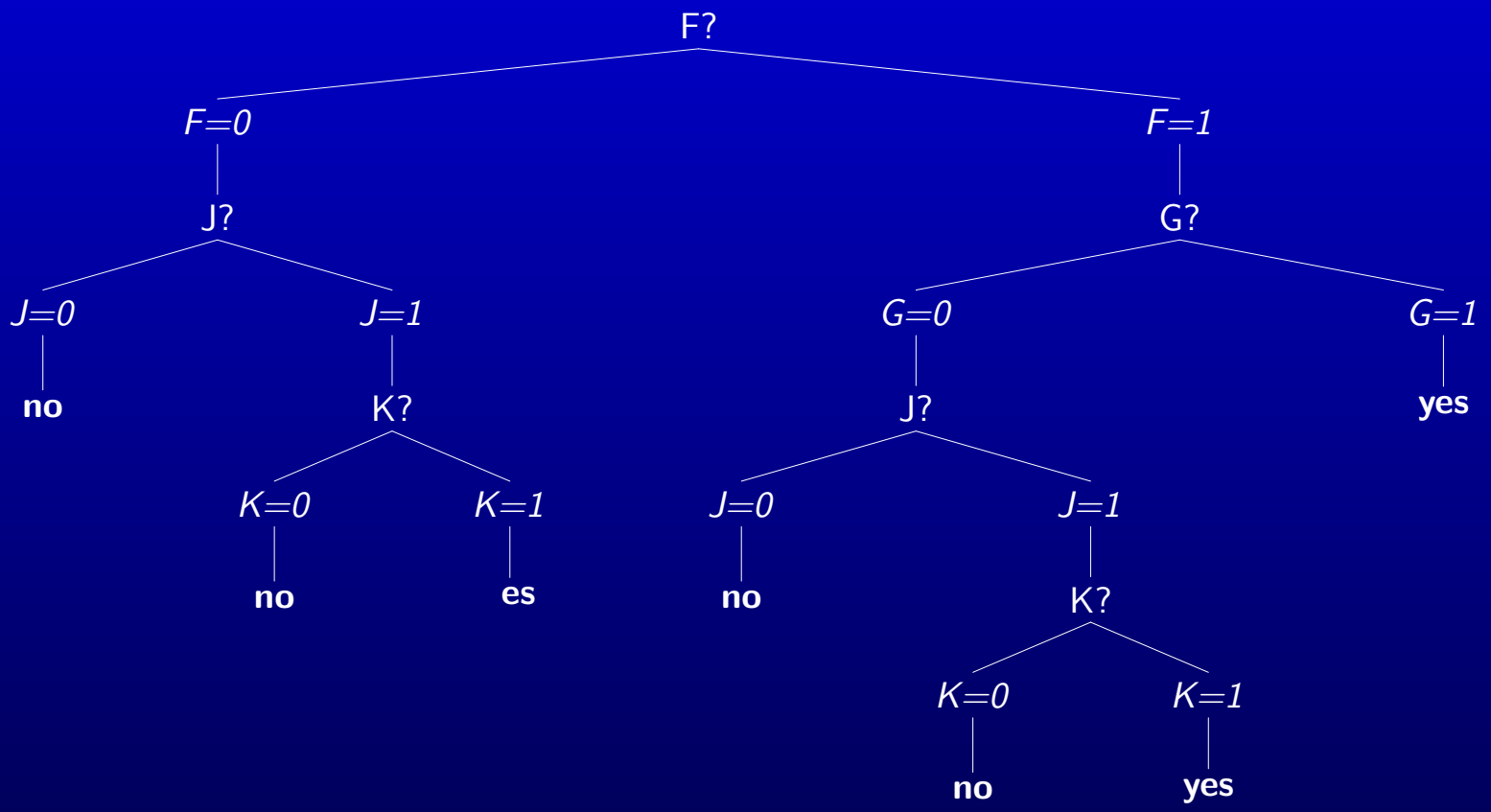
- Decision trees/lists/stumps have a natural interpretation as rule sets
- Each path from the root to a leaf is a rule of the form:

if outlook=rain &
wind=strong
then class=no

- Explicitly rewriting trees as rules may give model greater generalization power and transparency
- C4.5 includes an algorithm for performing this transformation (`c4.5rules`)

Trees to rules

- Decision tree for F=G=1 or J=K=1:



Trees to rules

- Some rules from this tree include superfluous conditions:

if F=1 &
G=0 &
J=1 &
K=1
then class=yes

- By deleting conditions, we can get to the rule we want:

if J=1 &
K=1
then class=yes

Deleting conditions

- Should we delete condition X from rule R?

	Class C	Class $\neg C$
Satisfies X	Y_1	E_1
Doesn't satisfy X	Y_2	E_2

- The original rule R assigns class C to $Y_1 + E_1$ examples that satisfy X (E_1 of them incorrectly)
- The new rule would assign class C to $Y_2 + E_2$ examples that don't satisfy X (E_2 of them incorrectly)
- Only retain X if the difference is statistically significant
- Or, we could use the pessimistic error rate estimate: remove X if doing so doesn't increase the upper bound on the expected error

Deleting conditions

- Given a decision tree, we can iterate over the paths deleting conditions (greedy search)
- This gives the result of **global** pruning and tree reconfiguration
- But, now each instance may have more than one rule that could apply to it, or none at all!
- We need a way to resolve conflicts between applicable rules
- Extrinsic ordering is always popular, but can be difficult to implement in a transparent manner

Resolving conflicts

- C4.5 organizes all rules into *sets*, one for each class
- Now within each set, rule order doesn't matter since all rules have the right-hand side
- Collectively a rule set is a binary classifier which can be evaluated by its precision and recall
- Each rule set can be optimized using the Minimum Description Length Principle

Minimum Description Length

- Suppose S and R both have a copy of the training data, but only S has the values of the target function
- S could just transmit a list of classes to R
- Or, S could send a classification theory (e.g., a decision tree), plus a list of exceptions
- Which is better, a simple theory with lots of exceptions, or a complex theory with no exceptions?
- The Minimum Description Length (MDL) Principle (Rissanen 1983) says the best theory minimizes the sum of the lengths of the theory and the exceptions:

$$h_{\text{MDL}} = \underset{h \in H}{\operatorname{argmin}} L_{C_1}(h) + L_{C_2}(D|h)$$

Minimum Description Length

- Remember *maximum a posteriori* estimation:

$$\begin{aligned} h_{\text{MAP}} &= \operatorname*{argmax}_{h \in H} P(D|h)\,P(h) \\ &= \operatorname*{argmax}_{h \in H} \left(\log_2 P(D|h) + \log_2 P(h)\right) \\ &= \operatorname*{argmin}_{h \in H} \left(-\log_2 P(D|h) - \log_2 P(h)\right) \end{aligned}$$

- And remember the Source Coding Theorem:

$$L_{C_X}(x) = -\log_2 P(x)$$

- So, we can rewrite the MAP estimate:

$$h_{\text{MAP}} = \operatorname*{argmin}_{h \in H} L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

and $h_{\text{MAP}} = h_{\text{MDL}}$ when $C_1 = C_H$ and $C_2 = C_{D|h}$

Minimum Description Length

- What is the ‘length’ of a rule set?
- It depends on the encoding, but we can find an approximate lower bound
- The RHS of the rules are all the same, and can be dropped
- The length of the code for the LHS of a rule depends on the number of conditions x, and the length of each condition depends on the total number of features and values
- But since the order of conditions doesn’t matter, we give ourselves $\log_2(x!)$ bits credit

Minimum Description Length

- The cost of a set of rules is the sum of the discounted cost of their LHS's, but again the order doesn't matter so we give ourselves credit
- Say the rules cover r out of n cases, with f_p false positives and f_n false negatives. Then cost of the exceptions is bounded by:

$$\log_2 \binom{r}{f_p} + \log_2 \binom{n-r}{f_n}$$

- This tends to overestimate the length of the theory, so we discount it:

$$L = \textit{exception bits} + W \times \textit{theory bits}$$

Minimum Description Length

- We want to find a subset of our rule set which minimizes L
- Description length L has lots of local minima, so hill climbing fails. Instead, Quinlan uses *simulated annealing*
- Choose a rule at random
- If removing the rule (or putting it back in) would reduce L, do so
- If it would increase L by d bits, do so with probability $e^{-d/K}$
- Repeat, gradually reducing the 'temperature' K until convergence

Trees to rules

- Once the MDL subset of each rule set has been found, we need to order the subsets
- False positives are worse then false negatives, because they prevent later rule sets from applying
- So, we choose as the first rule set the one which makes the fewest false positive errors, recompute the error rates, and repeat
- Next, the default class is the one which is most common among those training examples to which no rule applies
- Finally, if there are any rules whose removal would reduce the error rate on the training data, get rid of them

Trees to rules

- C4.5's method of producing rule sets from trees allows more effective pruning than a simple bottom-up strategy
- The resulting rules are more compact and more easily understood than a decision tree
- Rule systems can in theory be edited by hand (though I'm not sure under what circumstances that would be a good idea)

Transformation-based learning

- To keep things reasonable, HMM taggers can take only a small window of context into account
- This leads them to make predictable errors which can be corrected by a set of fix-up rules
- Post-processing can also be used to *adapt* a model to a different situation (e.g., written $\rightarrow$ spoken language)
- These rules can be written by hand, but can be expensive to write and especially to maintain
- Wouldn't it be cool if we could do it automatically?

Transformation-based learning

- Brill (1992) proposed TBL as a solution to this problem (pp. 361–370)
- Start with a very simple baseline tagger
- Randomly generate a set of transformations which replace tags under certain conditions
- Choose the one which decreases the error rate the most, and add it to the system
- Iterate until you've had enough
- The collection of transformations can be converted to FST's, composed, determinized, and minimized for very efficient storage and application (much faster than HMMs)

Transformation-based learning

- The LHS's of candidate rules are generated from a set of templates:
  - ⋆ word-1 (word$+1$) is tagged t
  - ⋆ word-2 (word$+2$) is tagged t
  - ⋆ one of word word$-1 \ldots -2$ (word$+1 \ldots +2$) is tagged t
  - ⋆ one of word word$-1 \ldots -3$ (word$+1 \ldots +3$) is tagged t
  - ⋆ word-1 is tagged t_1 and word$+1$ is tagged t_2
  - ⋆ word-1 (word$+1$) is tagged t_1 and word-2 (word$+2$) is tagged t_2
- For each rule template T, we get a set of rules of the form:

  if T **then** replace tag X with tag Y
- This yields an astronomical number of rules, but we don't really have to construct them all. Clever indexing will let us only consider the tiny subset of rules which actually apply to something in the training data

Transformation-based learning

- When applied to the WSJ, the first rules it learns are:

From	To	When
NN	VB	previous tag is TO
VBP	VB	one of the previous three tags is MD
NN	VB	one of the previous two tags is MD
VB	NN	one of the previous two tags is DT
VBD	VBN	one of the previous three tags is VBZ
VBN	VBD	previous tag is PRP
VBN	VBD	previous tag is NNP
VBD	VBN	previous tag is VBD
VBP	VB	previous tag is TO
POS	VBZ	previous tag is PRP
VB	VBP	previous tag is NNS

Transformation-based learning

- The same strategy can be used to learn lexical relationships:

From	To	When
IN	RB	word two positions to the right is *as*
VBP	VB	one of the two previous words is *n't*

- Some results on *closed vocabulary* experiments:

System	Training size	Model size	Accuracy
HMM	64,000	6,170	96.3
HMM	1,000,000	10,000	96.7
Lexical TBL	64,000	215	96.7
Lexical TBL	600,000	447	97.2
Non-lexical TBL	600,000	378	97.0

Unknown words

- TBL can also be used for inducing an unknown word guesser
- Call capitalized words proper names, all others common nouns
- Add transformations which are sensitive to context, prefix, suffix, and whether adding or removing a prefix or suffix yields a known word
- Gives open vocabulary accuracy of 96.6% (recall MBT is 97.0% and MaxEnt is 97.4%)

Unknown words

- Some unknown word rules:

From	To	When
NN	NNS	word ends in *-s*
NN	CD	word contains character *.*
NN	JJ	word contains character *-*
NN	VBN	word has suffix *-ed*
NN	VBG	word has suffix *-ing*
??	RB	word has suffix *-ly*
??	JJ	adding suffix *-ly* results in a known word
NN	CD	word *$* can appear to the left

Unsupervised learning

- TBL as described so far is a supervised learning method
- TBL can also be used *unsupervised* (well, semi-supervised) to learn adaptation rules
- Base classifier tags each word with its ‘ambitag’
- Proceed as before, but with a new scoring procedure which takes advantage of unambiguous words

Unsupervised learning

- Suppose we have the rule *Replace* X *with* Y *in context* C, where $Y \in X$. Now consider each tag $Z \in X$ where $Z \neq Y$ and find:

$$R = \operatorname*{argmax}_z \frac{\text{freq}(Y)}{\text{freq}(Z)} \times \text{incontext}(Z, C)$$

  The score for the rule is:

$$\text{incontext}(Y, C) - \frac{\text{freq}(Y)}{\text{freq}(R)} \times \text{incontext}(R, C)$$

- This is supposed to prefer transformations that pick out tags which occur more frequently in this context, correcting for relative frequencies of tags

Unsupervised learning

- For instance *can* could be a noun, a verb, or a modal. In the context *The can will be crushed*, we know it must be a noun because when an unambiguous word occurs after *the,* it's almost always a noun

- Some unknown word rules:

From	To	When
NN_VB_VP	NN	previous tag is NNS
NN_VB	VB	previous tag is MD
JJ_NNP	JJ	following tag is NNS

- In closed vocabulary experiments, WSJ accuracy improved from 90.7% to 95.1% and Brown accuracy improved from 89.9% to 95.6%

- In similar experiments, Baum-Welch estimation yields 83–92% accuracy

Transformation-based learning

- TBL automatically induces a set of error-correcting rules for a simple statistical classifier
- Training is slow, but the final model is very fast
- TBL seems to be immune to overtraining – later rules don't help much, but also don't seem to hurt
- This may be because it is directly minimizing the error rate, rather than a proxy (likelihood, entropy, etc.)
- There are lots of extensions imaginable (e.g., more sophisticated base taggers, multiple base taggers, genetic algorithms for 'evolving' rule templates,. . .)