# Machine Learning for NLP
# Lecture 5 part 2: Learning rules

UNIVERSITY OF GOTHENBURG

Richard Johansson

October 1, 2015

# classifiers as rule systems

- assume that we're building a function tagger by hand
- how would it look?
- probably, you would start writing rules like this:
  - IF the current node is an NP, THEN
    - IF its parent is an S, THEN return the function tag SBJ
    - IF its parent is a VP, THEN return the function tag OBJ
    - ...
- a human would construct such a rule system by trial and error
- could this kind of rule system be learned **automatically**?

# learning rules

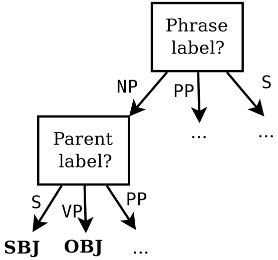
- so far, the learning algorithms we have seen have been different variations of the idea of scoring features
  - tends to work well, but it might not be the most intuitive way of viewing the task of learning
- rule learning algorithms produce results that are interpretable and editable
  - but perhaps not as mathematically well-understood as the linear classifiers

# rule learning algorithms: examples

- learning decision trees for classification
  - the next slides
- learning transformation rules for e.g. tagging
  - the Brill tagger in NLTK
  - see also Torbjörn's course,
    `http://www.ling.gu.se/~lager/Mutbl/course/index.html`
- inductive logic programming
- Markov Logic
- ...

# decision tree classifiers

- a decision tree is a tree where
  - the internal nodes represent how we choose based on a feature
  - the leaves represent the return value of the classifier
- like the example we had previously:
  - IF the current node is an NP, THEN
    - IF its parent is an S, THEN return the function tag SBJ
    - IF its parent is a VP, THEN return the function tag OBJ
    - ...



# general idea for learning a tree

- Occam's razor intuition: we'd like a small tree
- also, it should make few errors on the training set
- however, finding the smallest tree is a complex computational problem
  - it is NP-hard
- instead, we'll look at an algorithm that works top-down by selecting the "most useful feature"
- the basic approach is called the **ID3** algorithm
  - see e.g. Daumé III's book or `http://en.wikipedia.org/wiki/ID3_algorithm`

# greedy decision tree learning (pseudocode)

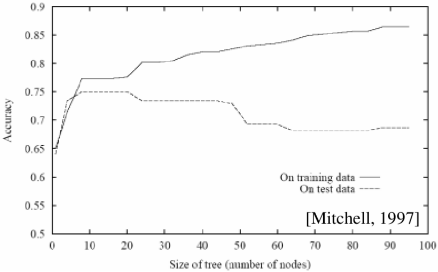
**def** TrainDecisionTree($T$)
  **if** $T$ is unambiguous
    **return** the class of the examples in $T$
  **if** $T$ has no features
    **return** a leaf with the majority class of $T$
  $F \leftarrow$ the "*most useful feature*" in $T$
  **for** each possible value $f_i$ of $F$
    $T_i \leftarrow$ the subset of $T$ where $F = f_i$
    remove $F$ from $T_i$
    $\text{tree}_i \leftarrow \text{TrainDecisionTree}(T_i)$
  **return** a tree node that splits on $F$,
    where $f_i$ is connected to the subtree $\text{tree}_i$

# how to select the “most useful feature”?

- there are many rules of thumb to select the most useful feature
  - idea: a feature is good if the subsets $T_i$ are unambiguous
- in Daumé III’s book, he uses a simple score to rank the features:
  - for each subset $T_i$, compute the frequency of its majority class
  - sum the majority class frequencies
- however, the most well-known ranking measure is the **information gain**
  - this measures the reduction of entropy (statistical uncertainty) we get by considering the feature

# problems with the naive approach

- ID3 and similar decision tree learning algorithms often have troubles with large, noisy datasets
- often, performance decreases with training set size!




- can be improved by using a separate development set:
  - **prune** the tree by removing the nodes that don't seem to matter for accuracy on the development set

# implementations: a small sample

- C4.5, C5: `https://www.rulequest.com/see5-info.html`
- `DecisionTreeClassifier` in NLTK
- `DecisionTreeClassifier` in scikit-learn
- NLTK's decision trees are more interpretable, since they work with symbolic features directly instead of numerical vectors