CSCE 638 Final Review Sheet

Basic Text Processing (p11)

✰ Regex: a formal language for specifying text strings. **p.11 - p.13**

✓ Regular expressions play a surprisingly large role - sophisticated sequences of regular expressions are often the first model for any text processing task.

✰ Lemma/citation form: the canonical form, dictionary form, or citation form of a set of words (headword). e.g. run is the lemma of run, runs, ran and running.

✰ Wordform: the "inflected" word as it appears in text. e.g. ran/running/runs.

✓ **cat** and **cats** have the same lemma, but different wordforms.

✰ Type: an element of the vocabulary. Token: an instance of that type in running text.

✓ Think of type as a set and token as a list (can have duplicates).

✰ Casefolding: reduce all letters to lower cases.

✰ Lemmatization: reduce inflections or variant forms to base form. Context dependent, have to find correct dictionary headword form. e.g. am, is, are → be.

✰ Morphemes: the small meaningful units that make up words. Stems = the core meaning-bearing units, affixes = bits and pieces that adhere to stems.

✰ Stemming: a crude chopping of affixes. It's context independent but language dependent; reduce terms to their stems. e.g. automates, automatic, automation → automat.

✰ Sentence segmentation could be tricky (especially when period . is involved). May be helpful with the introduction of machine learning classifiers, e.g. decision tree, logistic regression, SVM etc.

Classification (p23)

✱ Applications: spam detection, authorship/age/gender/language identification, sentiment analysis, etc.

✱ Definition: given a document d and a fixed set of classes $C = \{c_1, c_2, ..., c_j\}$, output a predicted class $c \in C$.

✱ Manual rules: accuracy can be high, but building and maintaining these rules are expensive.

✱ Supervised machine learning: given a document d, a fixed set of classes $C = \{c_1, c_2, ..., c_j\}$, and a training set of m hand-labeled documents $(d_1, c_1), ..., (d_m, c_m)$, output a learned classifier $\gamma : d \to c$.

✱ Understand the 2-by-2 contingency table. Precision = % of selected items that are correct. Recall = % of correct items that are selected. F-measure = A combined measure that assesses the P/R tradeoff (weighted harmonic mean). Accuracy = Fraction of documents that are classified correctly.

✱ For more than one classes, use micro- and macroaveraging to combine multiple performance measures. Macroaveraging: Compute performance for each class, then average. **Average on classes**. Microaveraging: Collect decisions for each instance from all classes, compute contingency table, evaluate. **Average on instances**. Note that Microaveraged score is dominated by score on common classes.

✱ To combat overfitting: use unseen test set or cross-validation over multiple splits.

Naive Bayes (p29)

✱ Assumption: Bag of Words assumption: Assume position doesn't matter. Conditional Independence: Assume the feature probabilities $P(x_i|c_j)$ are independent given the class c.

✱ Bayes' rule: for a document d and a class c, $P(c|d) = \frac{P(d|c)P(c)}{P(d)}$. To maximize $P(c|d)$: one could maximize $P(d|c)P(c)$ instead ($P(d)$ has little use when comparing among different classes, so can safely drop it). Thus, $P(c|d) = P(x_1, x_2, ...x_n|c) * P(c)$ (where x_n is the nth word).

✱ To estimate the probabilities $\hat{P}(c_j)$ and $\hat{P}(w_i|c_j)$: frequency is fine, but fails when encountering an unseen word. Use **Laplace transform (aka add-1 smoothing)** to combat (add one extra word that stands for "unknown").

✱ To prevent underflow: use log transformation. $\therefore P(c|d) \approx \log P(c_j) + \sum_{i \in positions} \log P(x_i|c_j)$. Model is now just max of sum of weights!

✓ Pros: Robust to irrelevant features, very good in domains with many equally important features

✗ Cons: optimal only when the independence assumptions hold, accuracy usually low compared with other supervised ML methods

Discriminative Estimation (p35)

✱ Motivation: highly accurate, easy to incorproate lots of linguistically important features.

✱ **Joint model** (aka generative models): place probabilities over both the observed data and the hidden stuff. Essentially $P(c, d)$. Examples: n-gram, Naive Bayes, HMM, etc.

✈ Attempts to maximize the joint likelihood; use relative frequencies as the weights (trivial solution)

✱ **Conditional model** (aka Discriminative models): place probabilities over the hidden structure given the data Essentially $P(c|d)$. Examples: logistic regression, maxent, SVMs, etc.

✈ Much harder to do, may have overlapped features.

✱ MaxEnt model: essentially calculating maximum entropy. (Formula could be found on **p.37**) It chooses the model which has the most uniform distribution.

✈ The Principle of Maximum Entropy states the rather obvious point that you should select that probability distribution which **leaves you the largest remaining uncertainty** (i.e., the maximum entropy) consistent with your constraints. That way you have not introduced any additional assumptions or biases into your calculations.

✈ Essentially, the derivative of maxent is equivalent to the difference between the actual count and the predicted count. The optimal solution would have each feature's predicted expectation equals to its empirical expectation. The optimal distribution is always unique and extant.

✈ NB may double count (multiplying each feature, disregarding whether these features are correlated or not). But maxent won't - it weights features differently so that correlated features won't have the same high weights.

✱ Perceptron: see the algorithm on **p.40**. A crude version of neural networks (SGD); hyperparameter will be *the number of iterations*, which also regularizes the model (not ideal if too big).

✈ Note that each shift is making a big change that tailors to this example only, but not generalizable. If not averaging, the model may perform well at the last few instances only (i.e. forgetting about the beginning). In practice, perceptron is also combined with shuffling.

Language Modeling (p41)

✲ Goal: assign a probability to a sentence.

✲ Definition: a model that computes either $P(W)$ or $P(w_n|w_1, w_2, ...w_{n-1})$ is called a language model. (A good estimation, though grammar is what we desire to get)

✲ To compute a prob. of a sentence: can use chain rules. But it's impossible to estimate prob. like P(the|its water is so transparent that) using frequencies (unable to enumerate all the possibilities).

✈ To resolve this: use the Markov assumption (that is, instead of looking at all prior words, limited yourself to the size of k windows. e.g. $P(w_i|w_1w_2...w_{i-1}) = P(w_i|w_{i-k}...w_{i-1})$, where k is a hyperparameter.

✲ $k = 0$: unigram model. Simplest, completely ignore the context. $k = 1$: bigram model, conditioning only on the previous word.
✈ Extending k → n-gram models. Generally not so well as language has long-distance dependencies; but n-gram is a good approximation. **(p.43)**
✲ Tradeoff between sparsity and preciseness. Length of a sentence is usually controlled; otherwise it's more biased towards shorter sentences. Normalization on sentence length is required then. Higher order models are usually more sparse.
✲ To estimate probabilities: can use the maximum likelihood estimate (count ratios). Note that log ratio is used to prevent underflow and speed up processing (adding is faster than multiplying).
✲ Perplexity: an intrinsic evaluation metric. **p.48** - the inverse probability of the test set, normalized by the number of words. The value will always be greater than 1. Lower perplexity = better model.
Why inverse? Otherwise the number will be too small to compare.
✲ Generalization: n-gram usually poor, as no new content is introduced. It works well for word prediction if the test corpus looks like the training corpus. Note - add-1 smoothing not applicable to handle unseen words, as it distorts the actual underlying distribution (see **p.51** for an example).
✈ Better generalization methods: backoff (starting with a higher order n-gram model, gradually reduces) and interpolation (mix different n-gram structures). Note that interpolation often works better than backoff.
✈ Advanced smoothing algorithm: Kneser-Ney. (Example on **p.55**) It leverages the count of things we have seen to estimate the count of things we have never seen. Trying to address this issue: "How likely is w to appear as a novel continuation?" Adds to the original n-gram estimate probability (e.g. $P(w_i|w_{i-1})$).
d: hyperparameter to be deducted. λ: normalization constant. Lower order n-gram is assumed to be more reliable (as the higher order ones are too sparse); use the training set wisely; useful when one has never seen a word following another word before.

Part-of-speech Tagging (p57)

✳ Intuition: words often have more than one part of speech (POS) tag. The problem is to determine the POS tag for a particular instance of a word.
✈ Baseline: tag every word with its most frequent tag. Tag unknown words as nouns. Accuracy already 90%! Many words are unambiguous, but those that are ambiguous are mostly common words (e.g. that).
✳ Source of information: knowledge of neighboring words, knowledge of word probabilities (more useful).
✳ To improve supervised results: build more informative features. Examples can be found on **p.59**.
✳ Upper bound of POS tagging accuracy: around 98%.
✳ Summary could be found on **p.60**.

Hidden Markov Models (p63)

✶ Applications: the tagging problem, named entity recognition as tagging.
✶ HMM: starting from **p.66**. Basic idea: imagine that the unknown tag sequence is observable only through the sentence. Now what we are trying to do is to deduce this sequence using the words that we have. (Hence "hidden") Tags have limited dependency with each other.
A trigram HMM will be of the form
$p(x_1, ...x_n, y_1, ...y_{n+1}) =$
$\prod_{i=1}^{n+1} q(y_i|y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i|y_i)$. q = transition probability (the Markov part, only between tags), e = emission probability (only between words). Estimation could be found on **p.68**.
Why called HMM? Prior = Markov chain, emission = words that are then observed.
Definition: one tag can generate only 1 word, otherwise it will get messy. (Only looking at the previous tags - succeeding tags: different models, more complicated)
Compare with MEMM tagger: both use previous tags; errors can propagate if the previous tag was wrongly predicted.
✶ To deal with low-frequency words: split the vocabulary into two sets, then map low frequency words into a small, finite set (depending on prefixes, suffixes etc)
✶ Using HMM to solve POS: Viterbi algorithm. Can be found on **p.70**. A dynamic programming approach.
Runtime complexity: $O(n|S|^3)$. Space complexity: $O(n|S|^2)$
■ Compared with brute force complexity: $O(n^{|S|})$
✓ Pros: Simple to train, high accuracy ($\geq 90\%$ on name entity recognition)
✗ Cons: difficulty in modeling $e(word|tag)$ (especially if an unknown word is related to multiple tags)
✶ The forward algorithm: p.71. Really similar to the Viberti algorithm, but change the max to be the sum. Applicable for unlabeled set (i.e. unsupervised learning).

Context Free Grammar (p73)

✳ Parse tree is a syntactic structure. Information conveyed by a parse tree includes: 1) POS for each word, 2) phrases, 3) useful relationships.
✳ A level up than POS tagging: leaf level will be POS tags, but the upper nodes will show how the words interact with each other. (Noun phrases, verb phrases etc. Syntactically related groups)
✈ Think of POS tagging problem as 1-dimensional, and parse tree as multi-dimensional.
✳ What is "Context free": only 1 terminator on the left hand side. (Definition on **p.75**)
✳ Left-most derivation: definition on p.76. Starting with the sentence symbol S, it keeps expanding by greedily choosing the left-most derivation rule, until hitting non-terminals.
✳ Property of CFG: it defines a set of possible derivations. A string $s \in \sum^*$ is the *language* defined by the CFG if there is at least one derivation that yields s. Each string in the language generated by the CFG may have more than one derivation (so called "ambiguity").
✳ Brief overview of English syntax: details can be found on **p.80 - p.82**.
✳ Ambiguity may come from: 1) part of speech, 2) noun premodifiers

Probabilistic Context Free Grammar (p85)

Essentially CFG + probabilities. (rank each rule in the order of probability) Prob. of a parse tree = $\prod$ (prob. for the rules in the tree)
→ Each sub rules sharing the same non-terminals would have their probabilities sum to 1.
✶ Data: Penn WSJ Treebanks. Use maximum likelihood estimates - taking the ratios between counts to estimate the probability of a rule occurring. (No negative examples though)
Assumption: the training data is generated by a PCFG. When the size of the training set is large enough, the estimated PCFG distribution will be equivalent to the true distribution.
Assumption 2: each rule is derived independently, not impacted by the others.
✶ Chomsky Normal Form: defined on **p.87**. Note that a non-terminal may span to two more non-terminals, or a terminal symbol.
✶ Goal: Find the most likely parse tree given a sentence and a PCFG. How? Use dynamic programming. (Similar to the POS problem, a bottom up approach)

! In order to have the algorithm working, will need binarized rules. See examples on **p.88.**

✶ Pseudocode for the CKY algorithm: **p.89**. Extended version adds supports to both unaries and empties (messy, but not increasing algorithm complexity). A worked example could be found starting from **p.90**.

✶ A PCFG parser would include two parts: 1) learned PCFG from the treebank, 2) parsing a test sentence using the CKY algorithm.

✶ Runtime complexity: $O(n^3|R|) = O(n^3|N|^3)$.

✶ Evaluation: by count the number of brackets that match with the gold standard. Can calculate precision, recall, F1 and accuracy.

✶ PCFG is robust but not use in practice. As the tree grows higher to the root, the connection with the actual words lessens. In other words, the parsed three has been detached from the input sequence of words. (Also the independence assumption is too strong)

✈ To improve PCFG: non terminal rules + some usages of the actual words (head-driven lexicalization). Still, the result won't be significantly better than trigrams.

Dependency Parsing (p95)

✷ Arrows: representing binary asymmetric relations. AKA dependencies. Connect a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate).

✈ Heads are usually verb and prepositional phrases.

✈ Note: one head may have multiple dependents. (Not true vice versa)

✷ Dependency tree: a connected, acyclic, single-head graph that consists of words and dependencies (e.g. nsubjpass, prep). In practice more useful.

✷ Relationship with the parse tree/CFG: **p.95**. CFG does not have the idea of "head" but dependency parsing does. Can do a conversion between the two.

✷ Methods of dependency parsing: **p.95**. Generally 3 choices: DP, graph, or "deterministic parsing" (a greedy approach).

✷ Source of information: **p.96**. Bilexical affinities, dependency distance, intervening material, and valency of heads.

✷ Projectivity: no arc within a dependency tree crosses with each other. In reality, though, non-projective (e.g. questions) is usually allowed (otherwise cannot parse a sentence). Projectivity is important for maximum spanning tree type of algorithms.

✷ Evaluation: calculating the accuracy (number of correct dependencies / total number of dependencies).

❛ Unlabeled attachment score (UAS): without the labels (e.g. nsubj) on the right, count how many dependency pairs have the correct relationship.

❛ Labeled attachment score (LAS): with the labels taken into account. Generally more strict.

✷ Dependency parsing is useful in extracting relations. e.g. protein interaction.

Word Meaning and Similarities (p99)

✹ Definitions:

Sense: a discrete representation of an aspect of a word's meaning. A word may have multiple senses.

Homonyms: words that share a form but have unrelated, distinct meanings. If sharing the same word representation (and the same POS tag), it's called homographs. Homophones mean "sharing the same pronunciation but different spelling in writing".

polysemy: a word that contains multiple meanings, and these meanings are related to each other. e.g. bank = a financial institution, or the building. Polysemy can be systematic, aka metonymy.

Zeugma test: identifies if a word has multiple senses through conjunction. **p.100**

Synonyms: word that have the same meaning in some or all contexts. Note that synonymy is a relation between senses, not words.

Perfect synonymy: synonyms that can substitute each other in all situations. Same propositional meaning; *Rare*.

✈ Synonymy is a binary relation; two words are either synonymous or not.

Antonyms: senses that are opposites with respect to one feature of meaning. Surprisingly - other than the difference in one dimension, the words are pretty similar. (can define a binary opposition or be reversions)

Hyponym: one sense is a subclass of the other sense. e.g. *car* is a hyponym of *vehicle*. Extensional, entailment, transitive. IS-A hierarchy.

Hypernym: one sense is a superclass of the other sense. e.g. *vehicle* is a hypernym of *car*.

✈ An instance belong to a class. We talk about hyponymy on the class level. Member might still be referring to a general concept (e.g. faculty → professor)

✹ WordNet: a hierarchically organized lexical database. It defines word senses using the synset (synonym set) with glosses (the definition/description of a sense).

✹ Similarity: a looser metric than synonymy. Technically it measures the relationship between senses, not words. But in this class both are considered.

Why similarity: summarized on **p.105**.

Note: word relatedness **is not** word similarity. Two words can be related but not similar, e.g. car and gasoline.

✹ Measuring similarities using thesaurus-based algorithms: find if the words are closed to each other in the hypernym hierarchy, or they have similar definitions. Focus less on verbs.

1) Path based: the shortest distance between two senses.

✗ Assumed each link represents a uniform distance, not addressing the fact that going up is not the same as going left/right (lower level more concrete).

2) Resnik: based on the theory of information content, use the lowest common subsumer. The idea is that two words are similar if they have a lot in common.

Idea of information content: word that appears more frequently would have a higher probability, thus less informative.

✗ Differnt nodes on one branch will have the same similarity score with nodes in another branch, due to sharing the same LCS. Specific words are not considered.

3) Lin: improved upon Resnik - also consider the differences between the two words. Despite having a lot in common, if two words also have a lot of differences, they are NOT similar.

4) Lesk: given two words, the algorithm computes a score based on the number of common words in their glosses.

Formula: see **p.109**.

To evaluate similarity metrics: **p.110**.

✗ No thesaurus for every language; recall sucks.

✹ Measuring similarities using distributional algorithms: aka vector-space models of meaning. Higher recall than the hand-built thesauri (though precision suffers).

Idea: two words are similar in meaning if their context vectors are similar. (Turn a word into a vector over counts of context words; term-context matrix)

✈ Positive Pointwise Mutual Information (PPMI) is used other than raw counts, which may be biased towards frequent words like "it". It measures whether two words co-occur more often than they occurring independently. PPMI has its lower end bounded by 0. Formula and examples can be found on **p.113**.

! PMI is biased towards infrequent events. Can be alleviated using weighting schemes or add-1 smoothing.

✈ Vectors might also be calculated using syntactic dependencies (**p.114 - 115**)

→ Actual similarity metric: see **p.116**. Cosine, Jaccard, Dice, JS. PPMI is a way to represent a word context vector.

Vector Semantics (p119)

✺ Unlike PPMI vectors that are long and sparse, want to learn short and dense vectors. Why? easy to use as features in ML; more generalizable; better at capturing synonymy.
✺ How to get short dense vectors? SVD, neural networks (skip-grams and CBOW), and Brown clustering.
✺ SVD: **p.120 - p.122**. An effective way to reduce dimensions (like PCA). Note that we only use the reduced column W matrix (aka the first one) as embeddings.
→ Each row of W matrix is a k-dimensional representation of each word w.
✓ Why dense SVD embeddings? Better at tasks like word similarity! Denoising; more generaliable to new data; proper weighting on smaller dimensions; better at capturing higher order co-occurrences.
✺ Neural network based: skip-grams and CBOW. Learn embeddings as part of the process of word prediction; a NN is trained to predict neighboring words.
✓ fast, easy to train; package available; pretrained embeddings are offered; also these embeddings capture relational meanings.
✺ Brown clusering: each word forms a cluster itself. At each iteration, merging clusters that have similar probabilities of preceding and following words. Stop when all words are in a big cluster. (Visualization could be found on **p.124**)
→ Essentially builds a binary tree in a bottom-up manner.

Semantic Role Labeling (p125)

✻ Idea: Predicates (bought, sold, purchase) represent an event. Semantic roles express the abstract role that arguments of a predicate can take in the event.
✻ Why semantic role labeling? A useful shallow semantic representation (between parses and full semantics); improves NLP tasks like question answering, machine translation, etc.
✻ Definition: SRL is the task of finding the semantic roles of each argument of each predicate in a sentence.
Thematic roles: a way to capture semantic commonality. e.g. breaker and opener are both *agent*. (a typical set is on **p.127**)
✗ Hard to create standard set of roles or formally define them; often roles need to be fragmented to be defined.
Intermediary instruments: can appear as subjects.
Enabling instruments: cannot appear as subjects.
✻ 2 common architectures other than thematic roles:
1) Fewer roles - generalized semantic roles, defined as prototypes from PropBank.
Two parts: proto-agent and proto-patient. A verb usually has its sense defined in 5 arguments. A parse tree may turn to a propbanked structure.
2) More roles - define roles specific to a group of predicates (FrameNet). Frame-specific roles.
✻ A simple SRL algorithm: **p.132**. Note that it classifies everything locally, where each decision about a constituent is made independently of all others. THIS IS NOT TRUE. Constituents in FrameNet and PropBank must be non-overlapping.
✻ Determining if a verb is a predicate: propbank - choose all verbs, possibly removing light verbs. FrameNet - choose every word that was labeled as a target in the training data.
✻ Features: see the last slide on **p.132**. Note that path feature is equivalent to the path in the parse tree from the constituent to the predicate.
✻ To do joint inference: reranking! Use the first classifier to produce multiple possible labels for each constituent. The second classifier then chooses the best global label for all constituents. The second classifier would be a bit more complicated - more features are added (e.g. sequences of labels).
✻ Current systems first parse a sentence, then find predicates in it. For each predicate, they classify each parse tree constituent.

Intro to Information Extraction (p135)

✻ Definition: extracting pieces of information from text and assigning some meaning to the information. Most applications focus on turning unstructured text into a "structured" representation.
Normally needs to: find text snippets to extract; assign semantic meaning to entities or concepts; find relations between entities or concepts
✻ Common applications and general techniques: see **p.135**. Some problems including named entity recognition (NER), semantic class identification, semantic lexicon induction, etc.
✻ IE systems often rely on low-level NLP tools for basic language analysis in a pipeline architecture. Wide applications; some tasks like named entity recognition are relatively well-understood, but many challenges exist.
✻ Turney algorithm: **p.140**.
1) Extract two-word phrases based on 5 rules. These phrases are called "phrasal lexicon".
2) Measure polarity of each phrase using PMI (pointwise mutual information). $Poloarity(phrase) = PMI(phrase, "excellent") - PMI(phrase, "poor")$. Prob. of each words occurring is calculated using Altavista, the query search engine. (Formula **p.141**)
3) For each review, calculate the average polarity score based on all phrases extracted. If the average score is greater than 0: the review is positive. Otherwise negative.
Performance: 74% accuracy. The algorithm uses phrases other than words, and it learns domain-specific information.

Discourse, Pragmatics, Coreference Resolution (p143)

✻ Coreference resolution: identify all noun phrases (mentions) that refer to the same real world entity.
... challenging as many pairs of noun phrases co-refered, with some nested inside others.
✻ Types of reference: referring expressions (more common in newsire, generally harder in practice), free and bound variables (more interesting grammatical constraints, more linguistic theory, easier in practice).
! Note that not all noun phrases are referring, e.g. *Every dancer* twisted *her knee*. "Every dancer" is non-referential, thus "her knee" refers to nothing.
✻ Supervised machine learning for pronominal anaphora resolution: given a pronoun and an entity mentioned earlier, classify whether the pronoun refers to that entity or not given the surrounding context (yes/no). (A binary classification task)
→ Usually ignoring pleonastic pronouns like "it is raining".
→ Any classifier is fine; get positive examples from the training data, negative examples by pairing each pronoun with other incorrect entities.
→ Pairwise classification can be in conflict with each other! (Not representing the cluster)
✈ Features: **p.145 - 146**.
✻ Supervised machine learning for coreference: data = positive examples that corefer and negative examples that don't corefer. Note that the distribution will be very skewed; most of the mention pairs do not refer to each other.
✈ Models: mention pair models, mention ranking models, and entity mention models.
Mention pair models: treat coreference chains as a collection of pairwise links. Make independent pairwise decisions and reconcile them in some way (e.g. clustering or greedy partitioning)

Mention ranking models: explicitly rank all candidate antecedents for a mention.
Entity-mention models: a cleaner but less studies approach. (Seems to be what the prof prefers :) Posit single underlying entities. Each mention links to a discourse entity.
✻ Stanford deterministic coreference: a rule-based model, no ML. 7 rules. Process texts through multiple passes. After applying each rule, precision ↓ bur recall ↑. Note that decisions cannot be modified in later passes (hence "deterministic").
✓ Simple approach but gives state of the art performance; easy insertion of new features or models; a typical representation of "easy first" model. (ML is still working though)
✻ Evaluation: multiple metrics, like MUS Score, CEAF etc. All focusing on different parts; the CoNLL challenge used a combination approach.

Relation Extraction (p151)

✻ Can be complicated, but the focus of this class is a bout extracting relation triples.
✻ Why? Create new structured knowledge bases, useful for any apps; augment current knowledge bases; support question answering.
✻ Automatic content extraction (ACE): consists of 17 relations.
✻ How to build relation extractors?
1) Hand-written rules: utilize name entity tags, manually identifying relationships between entities.
✓ High precision, can be tailored to specific domains
✗ Low recall, hard to enumerate all patterns, expensive for every relation, accuracy is not ideal
2) Supervised machine learning: choose a set of relations we'd like to extract and a set of relevant named entities, find and label data. Train a classifier on the training set.
→ Entities are mostly in the same sentence.
☛ Normally involve the following steps: first, find all pairs of named entities. Use a classifier to tell if 2 entities are related; if so, use another classifier to find the relation. (Two-step process enables faster classification training by eliminating most pairs. Moreover, one can use distinct feature-sets appropriate for each task.)
Possible features: see **p.156**. Entity-based, word-based, syntactic features. Any classifier would work then.
✓ Accuracy may be high if: training data is large enough, and the testing data is similar to the training set.
✗ Expensive to do manual labeling, poor generalization to different genres.
3) Semi-supervised/unsupervised ML: training set not required!
✉ Relation boostrapping: gather a set of seed pairs that have relation R. Iterate: find sentences with these pairs → Look at the context between or around the pair and generalize the context to create patterns → Use the patterns to get more pairs.
✉ Distant supervision: combine bootstrapping with supervised learning. Use a large database to get huge number of seed examples; create lots of features from all these examples; then combine with a supervised classifier. → Exhibit both characteristics from the unsupervised and supervised classification.
✉ Unsupervised relation extraction: applicable to the web. First use parsed data to train a "trustworthy tuple" classifier; single pass extract all relations between noun phrases, keep those trustworthy ones; use an assessor to rank relations based on text redundancy.
✻ Evaluation (on semi-supervised and unsupervised relation extraction): no gold standard - calculate approximate precision only. Precision would be the ratio between the number of correctly extracted relations and the total number of extracted relations.
✈ May get different precision@k, but no way to measure recall.

Event Extraction (p161)

✾ Definition: extracting role fillers associated with events. Extracted information can then be used to fill corresponding event template.
✾ Differences with named entity recognition (NER): NER identifies types of entities, where event extraction identifies role relationships associated with events.
✾ Unstructured text: just like the normal text that we encountered every day, entirely depending on language understanding. Semi-structured text has some structure (layout) that can aid in understanding.
✾ Supervised learning: needs annotated texts. (Tradeoff compared with manual knowledge engineering: see **p.162**)
Manual text annotation: ✗ time consuming, tedious, deceptively tricky, have to annotate new corpus in each domain!
✾ Pattern-based systems: use patterns/rules on text. Pipeline typically involves syntactic analysis, pattern extraction, coreference resolution, and template creation.
✗ Event information is sparse!
✾ Sequence tagging models: classify individual tokens as to whether or not they should be extracted.
✈ Builds a classifier as a seq. tagging model; labels each token as extraction or non-extraction. Use simple features.
✗ time consuming, tedious, deceptively tricky, need annotations in each new domains.
✾ Weakly supervised learning: similar to the ones in relation extraction. Input: piles of texts that are marked as relevant and irrelevant. Human will then review the ranked patterns later. (Only classifying the type, not the actual text, so much easier)
→ Note: extracted patterns do not always apply. Some patterns may be generic and not tailored to a specific domain (e.g. "one of <np>" is not limited to the terrorism domain only).
✈ Keywords along are not as reliable! e.g. "exploded in anger" does NOT imply something actually explodes. Might need secondary contexts to help.
✾ More challenges: contextual effects, inference, metaphor etc. (**p.166**)

Evaluations

▮ Extrinsic evaluation: build an external system to evaluate (i.e. a real task). Time consuming, can be slow. Unclear if subsystem is the problem, other subsystems, or internal interactions. If replacing subsystem improves performance, the change is likely good.
✓ More expensive, but the real test bed.
▮ Intrinsic evaluation: Evaluation on a specific, intermediate task. It's fast to compute the performance and helps understand the subsystem. One would need positive correlation with real task to determine usefulness.