**Knowledge base 知识库**

Text 的局限性：大型语料库的信息检索一开始是 keyword-based 现在是 entity-based

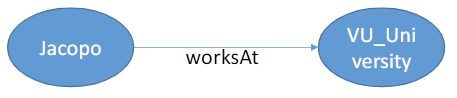
Data指的是给定的意思，而knowledge是理解的意思

解决text限制 用manifest建立知识库 knowledge base或者 graph 由 manual或者unstructured source建立 这是latent 潜在知识 是 hidden to people的，这些 latent model or feature 通常有ML学习得到

一个争论 knowledge 是 啥都该学 还是 能够理解

Manifest knowledge 人们表示knowledge的方式是logic

Opinion：知识是无歧义可断言的

Knowledge base：实体和关系间的links 

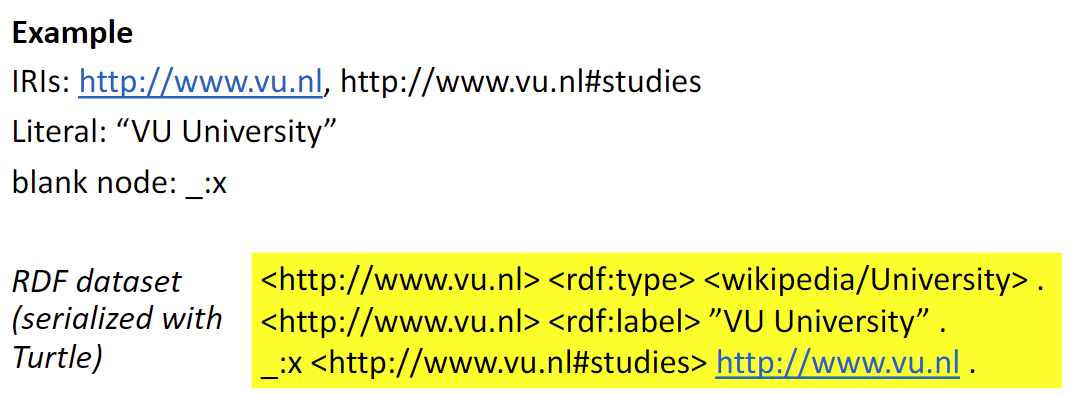
Latent knowledge 潜在知识

不用去解释知识，只要完成用知识做的事就行 deep learning就是 一种模仿

WordNet word有monospermous 一个意思 polysemous 多个意思

RDF Resource Description Framework 资源描述框架 W3C标化一种

用于 形容 资源的属性 triples表示 <s p o> subject predicate object 是一种连接图



SPARQL 一种query语言 SQL语法 用于 find all possible graph homomorphisms between the query and the graph 找query和rdf图的同态



DBpedia 将wiki转rdf的项目

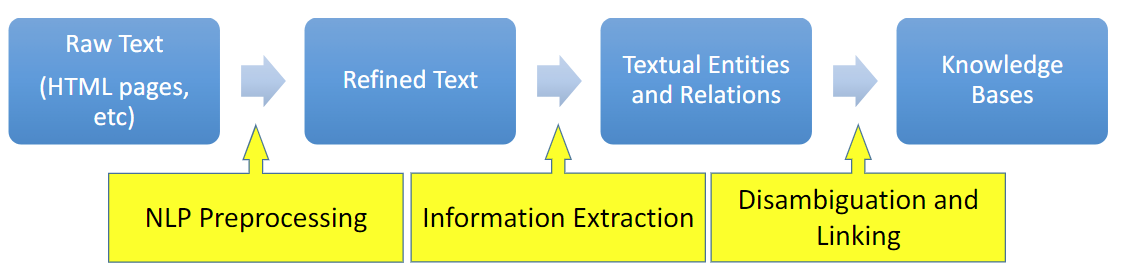
YAGO 统一wiki和wordnet，extract facts从wiki，check plausibility 合理性

Freebase 一个社群贡献的

Wikidata 主要text 难verify 有持久性 是data版本的wikipedia 高质量知识 用qualifiers表示属性超过3个的entity

**NLP**

Knowledge acquisition知识获取 从unstructured text or other data中extract 知识的 过程



提取的种类 提取entities和relations 另一种提取 events和 temporal experssions

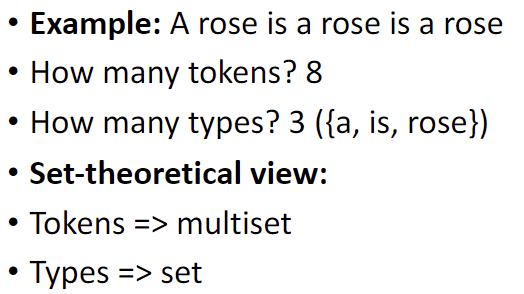
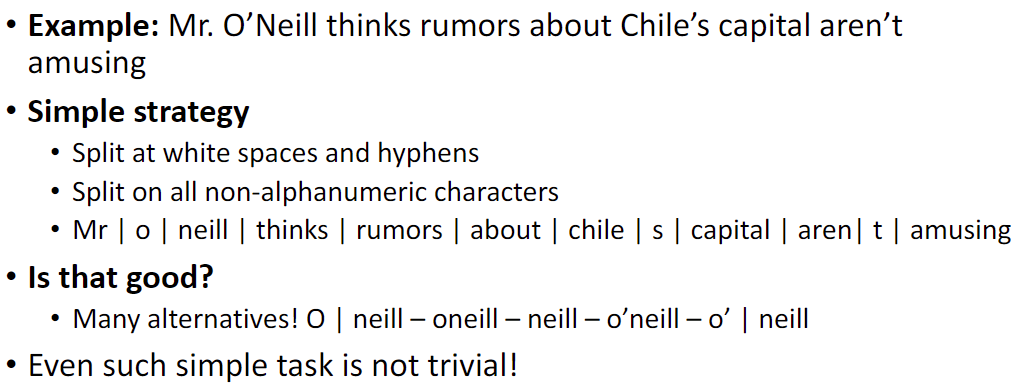
NLP处理 Unstructured data 如 text，结构数据为DB型，有Semi-structured data，含有title等，如json

预处理任务：**Tokenization** 切分 **Stemming or lemmatization** 词元化

**Stopword remova**l去除终止词 **POS tagging** 词性标注 **Parsing** 解析

Tokenization 切分：给个character句子，split into pieces 就是tokens 也就是words

Token：句子里的一个词汇实例 Type：相同的token构成的类

拆分 连字符 空格 和所有非字母字符 例子2不好使

Query和documents都要相同的预处理 保证拆分后和text相同 拆分决定了query的match

有潜在问题：连字符，名字，中文等

SOA：Stanford Tokenizer Apache OpenNLP NLTK

Stemming or lemmatization 词元化

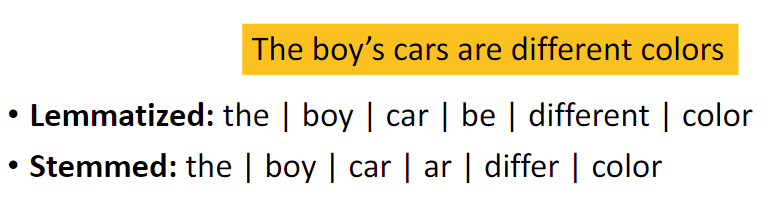
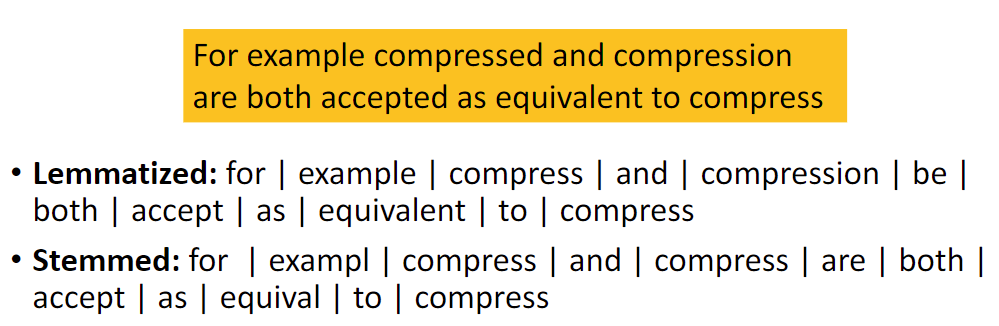
Lemmatization and Lemma：将所有变体form单词化为base form 如Am is be are 变为 be

Lemmatization 词性还原：适当还原为字典格式 Lemma：a set of words 的字典形式

Stem and Stemming：将terms变为他们的roots，如 are变ar

automate automatic automation变automat

Stemming 词干提取：粗糙的词汇切割 Stem词干：a set of words的root形式

常用的词干提取算法：Porter’s algorithm Snowball

**Stop words** 停止词：没有任何semantic的词，贼常见，每个文档都会出现的

如 a, an, and, are, as, at, be, by, for, from, has, he, in这些，基本都是介词副词等

方法：维护一个stop list，根据list去掉words（不属于IR系统字典）省memory，让query更快 现在的Trend：不再去掉停止词 因为好的压缩技术和查询优化技术

某些情况下需要停止词：let it be to be or not to be

**Part of speech:** 给每个token一个label说明他是啥

Function words: 让句子的语法正确 连词 语气词 代词 表达次

Content words：包含真正的意思 nouns verbs adj adv

种类：8到百种 而且一个word属于多种 part of speech

Tagging 打标签：使用更高级的抽象

amazing 可以再goalkeeper前 可以在scored前 Can 可以是 n 罐头 verb 能

自动打标签：Penn Treebank tagse 有36个lists 典型分类问题 目标给句子打一系列标签

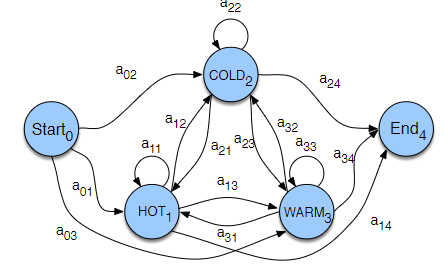
方式：1. 输入一系列token words 2. 输出 token chain with their tags

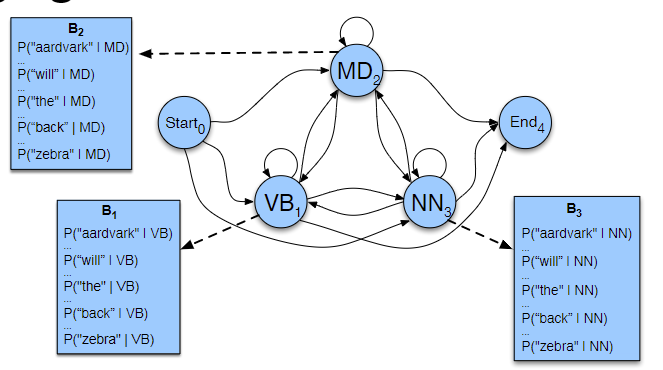
Not hard，SOA 97% baseline都有90%

How to work: rule-based taggers Stochastic taggers.

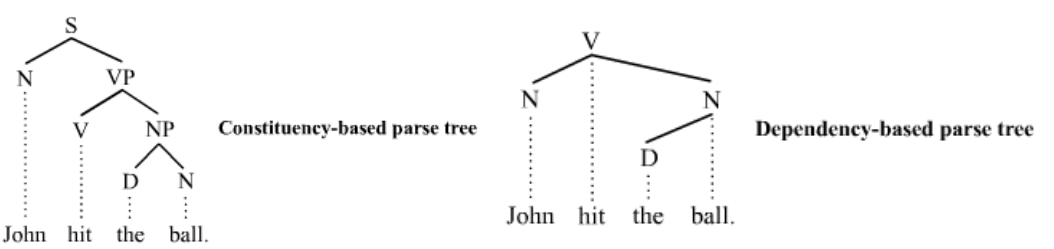
**HMM for POS Tagging** 隐藏markov模型： sequence model 给句子中每个unit分个label

典型的概率图模型 HMM func：给sequence of units，对每个unit选 最高概率的label 依靠 **Markov chain**：权重自动机模型，边表示概率

一个问题：Markov的假设是事件全部独立，无之前的知识，输出独立

  
训练好model，给个words就能显示其POS，过程叫做解码，使用viterbi算法

**Parsing** 解析：根据语法，构建string的语义树

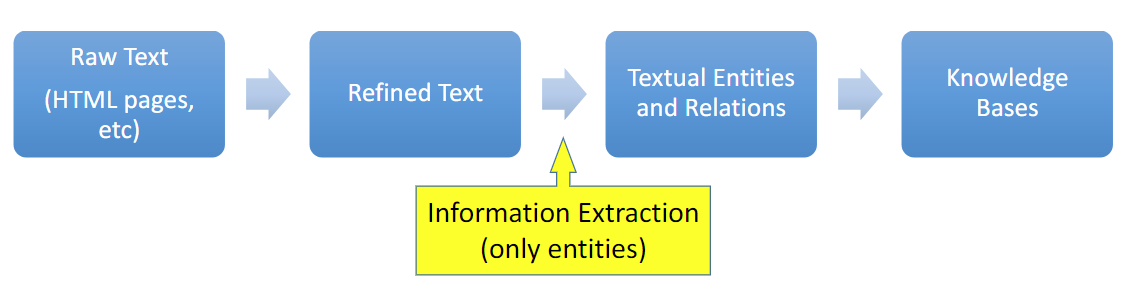


Constituency parsing 区解析：将phrase化成sub-pharse，树的node是短语，leaf是word

Dependency parsing 依赖解析：根据关系把words连成树。

Co-reference resolution 同义分析: 在一个text中找到 相同entities的所有的表达

**Entity Link1**



Information Extraction 将Refined文本变为 文字entities和relations的过程

两种提取办法：**Named Entity Recognition (NER) Relation Extraction (RE)**

**NER 命名实例识别**

Named entity：任何东西，指定一个名字，用改名字引用 person，location，song这种类

Named entity Recognition：find并classify names在text中

如：我昨天在荷兰是傻逼 person：我 data：昨天 location：荷兰

Detect entities很关键 named entities 有很多relations，问题的答案是named entities

Like POS，NER也是 sequence labeling问题

三种办法：1. Hand-written regular expressions 手写正则表达式

2. 使用classify：通用贝叶斯，最大熵model

3. Sequence models HMM CMM (Conditional Markov Models) CRF (Conditional Random Fields)

典型的SOA NER只能detect gross-grained 粗粒度实例类型 如 location org person等

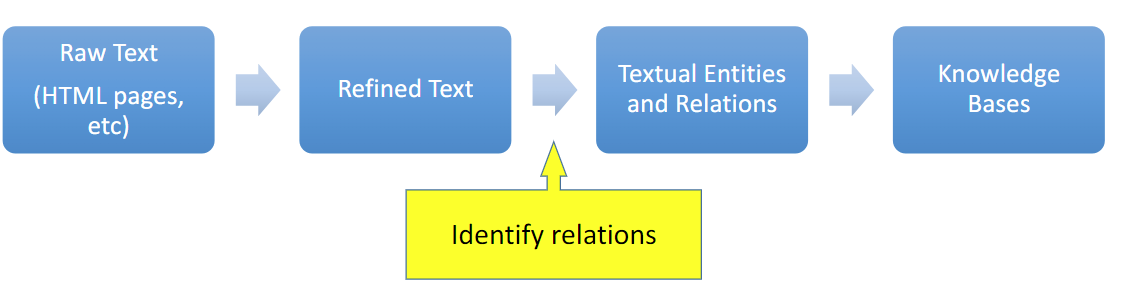
现在厉害的NeuroNER，用Bi（双）LSTM做实例识别，deep learning

Token (Character) Embeddings：将 每个character，n-gram 和 token 都映射到 实数向量

向量在一个高维域中，学习算法将计算相似的words并将他们映射到close向量上

用LSTM 长短期记忆：常用的学习算法

**Relation Extraction 关系提取**



从文本中提取关系，给定一段text，从中可以extract complex relations 如

Company-Founding：Company IBM Location New York Date June 16, 1911

更简单的任务，relation triples 上面找的是company关系，这个就找year和loc关系

如 Founding-year(IBM,1911) Founding-location(IBM, NewYork)

RE的类型：

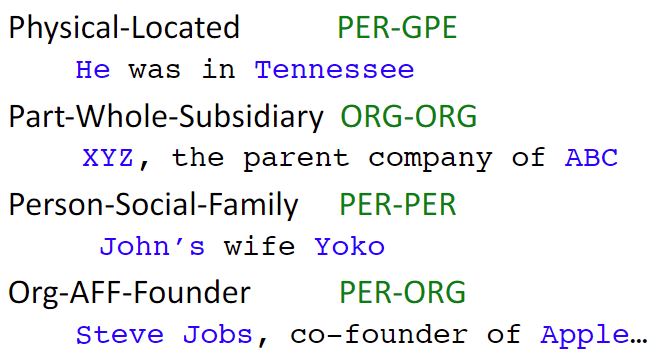
1. **Traditional Extraction 传统提取** 有已知的关系和带注释的input

2. **Open Extraction 开放提取** 没有任何prior info的知识提取

**Traditional extraction**

Automated Content Extraction (ACE) 自动内容提取：detect **Entities Relations Events**

ACE是测试新IE算法的benchmark

例子 UMLS: Unified Medical Language System 医药领域的ACE

Ontological relations本体关系：IS-A (hypernym 上位词): subsumption between classes

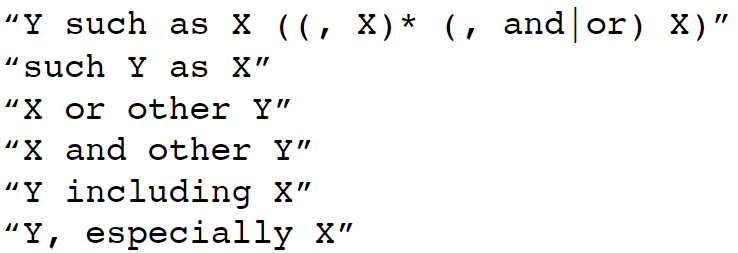
哺乳动物-脊椎动物-动物

Instance-of: relation between individual and class 荷兰 instance-of nation

**Types of traditional relational extraction methods**

1.Hand-written patterns 2. Supervised machine learning 3. Semi-supervised：Bootstrapping 自举法 (using seeds) 和 Distant supervision 远程监督

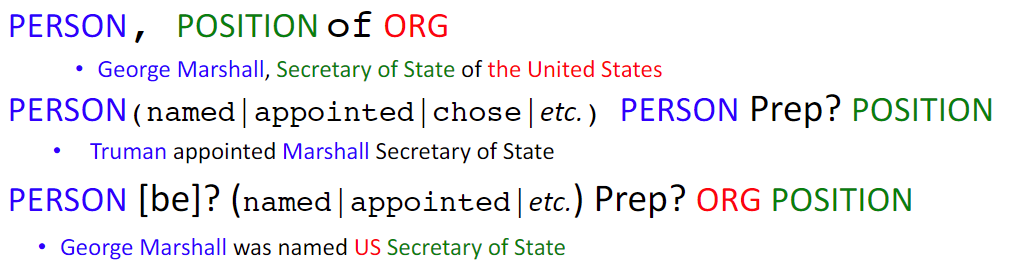
Hand-written patterns 手写所有可能的表达 自动获取 hyponyms（IS-A）



提取更多的relations：Intuition：关系经常发生在实例之间 cures (DRUG, DISEASE)

located-in (ORGANIZATION, LOCATION)； founded (PERSON, ORGANIZATION)

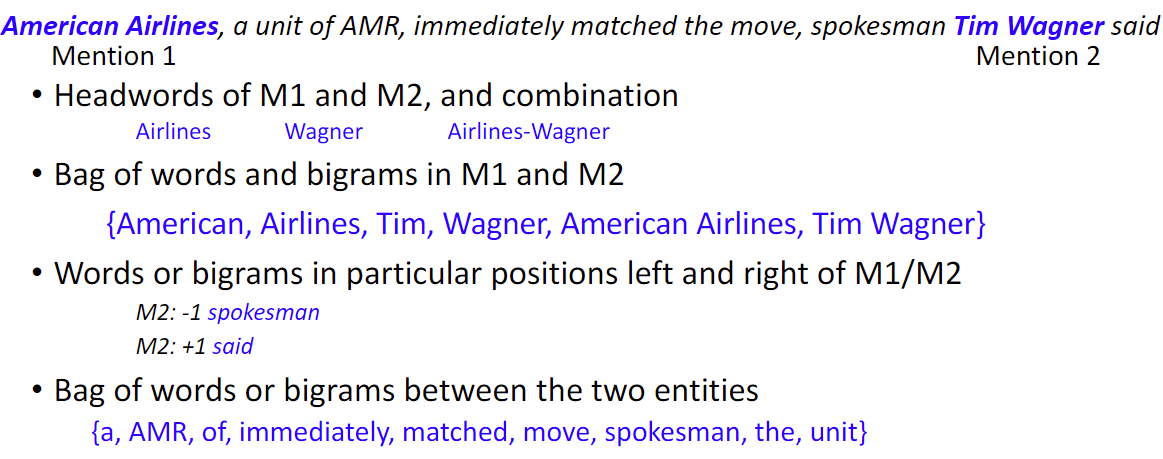
但是 多个 PERSON 或者 不明确的 后半部分 会有歧义

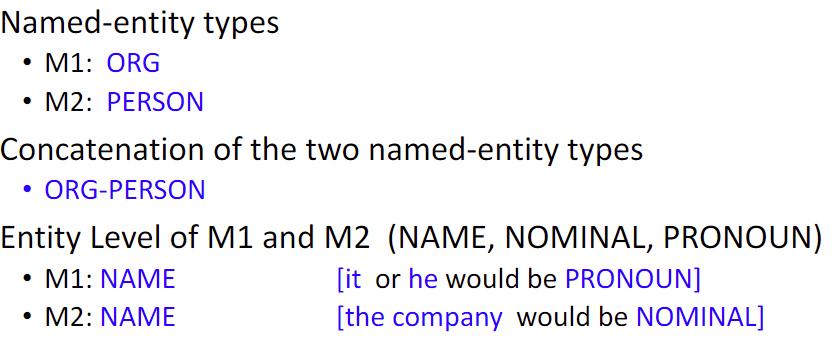


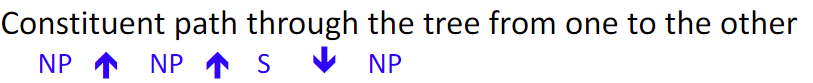
好处：1. 人写的 high-precision 2. 可以specific domain 定制

坏处：1. Often low re-call 2. 所有patterns要很多works 3. 不可能写到全部relation

Supervised machine learning：找需要extract的relation集合，再找相关的entities，find and label data，最后训练。在 find and label中，在sentence中分类两个entities的relation。







组成的路径通过M1在树上下走，抵达M2

常用的分类器：max entropy，简单贝叶斯，SVM等

Training set上训练，dev set上 调试，test set上test

好处：1. 如果有足量的 hand-labeled training data 且 test 和 training相似，有高精确度

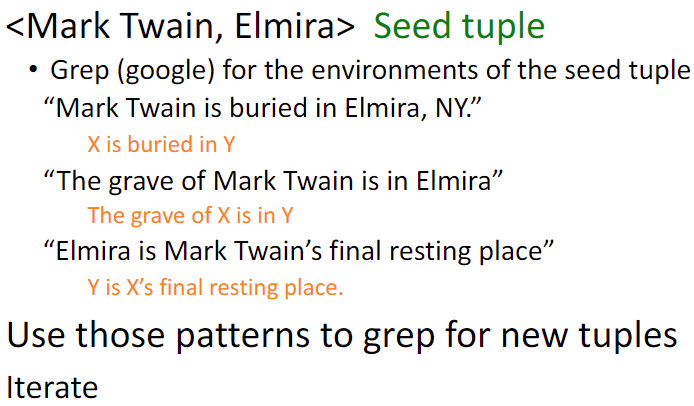
坏处：1. 给大量数据集 labeling很贵的 2. 监督学习模型很脆弱，没有鲁棒性

Semi-supervised Relation Bootstrapping

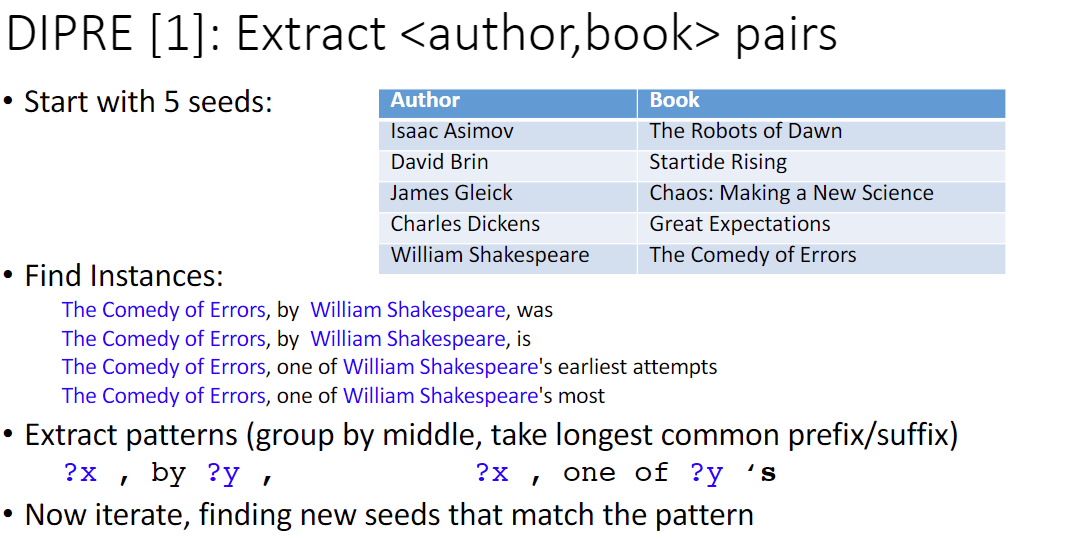
没有training set，只有一些 seed tuples和一些 高精确的 patterns

直接使用这些seed去populate a relation，先gather seed pairs有relation R

迭代：1. 在pairs中找到 sentences 2. 在pair附近看context，概括上下文以生成patterns 3. 用这些patterns去grep更多的pairs



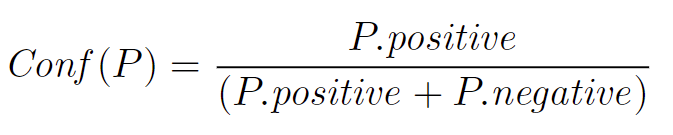
怎么概括？用google搜索，然后替换pair关键词，得到关系 . 例子

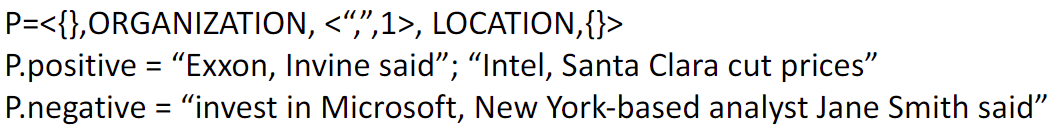


升级版DIPRE 是 SNOWBALL 使用相似前中后缀作为 extract pattern



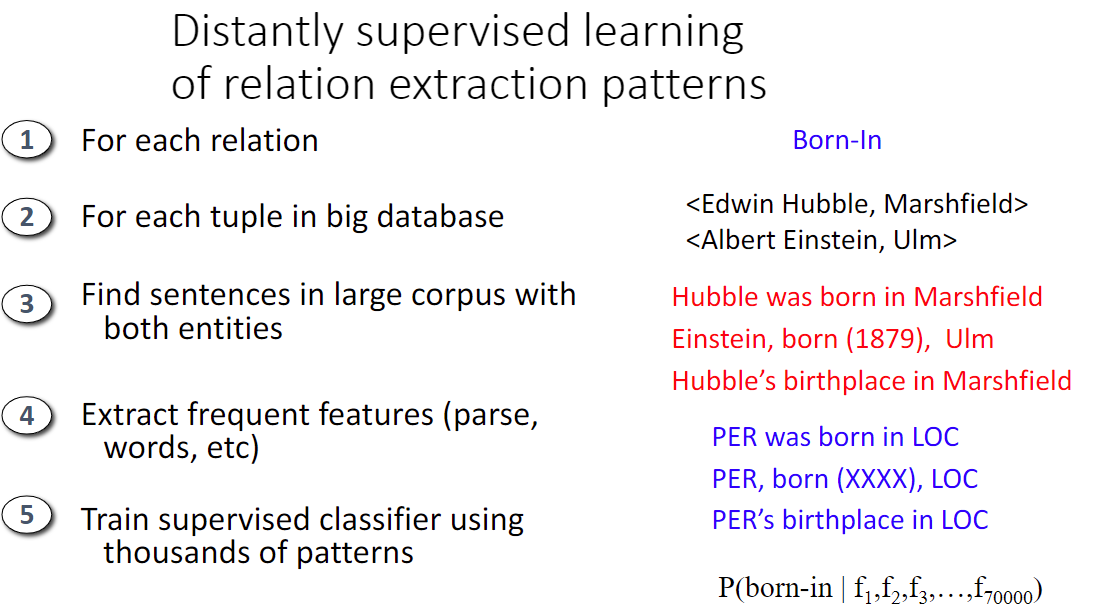
Snowball计算pattern的置信:





Distant Supervision 远程监督 在bootstrapping中结合监督学习 不再使用few seeds, 使用大型seed sample库, 用这些创建features, 结合监督分类器

例子: supervised classification: 监督使用手写的知识 unsupervised classification: 使用unlabeled data, 对语料中的genre 体裁问题不敏感



**Open (relation) extraction 开放关系提取 aka Open Information Extraction (Open IE)**

目标：从web中提取relation without training data 和 relation lists 并 把information 放到一个语义精确的环境

Open IE的动机：web大且多样 所以relation经常是出乎意料的 有三大chllenges：

1.Automation：自动生成合适的training without监督2. Corpus heterogeneity 语言的异质性：NER等重语言技术有限3. Efficiency：传统的IE耗时长

**TextRunner** 第一个开放IE系统，目标：发现所有文本内的关系，高scalable，给提取到的关系一个概率 3个key模块：1. Self-supervised learner 2. Single-pass extractor 单通过提取器 3. Redundancy-based assessor 冗余评估器

自监督学习：输入：小语料例子 输出: 分类器将候选标记为 可信 or not

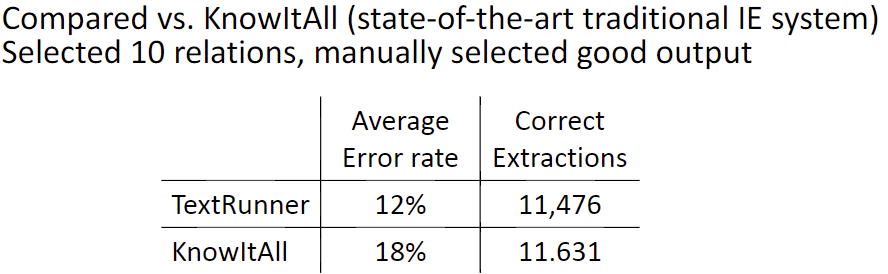
步骤:1. 自动在training data上label data 做 pos/neg extraction 2. 使用这些labeled data连一个简单贝叶斯分类器

Single-pass extractor 遍历语料库 提取所有可能的 relations

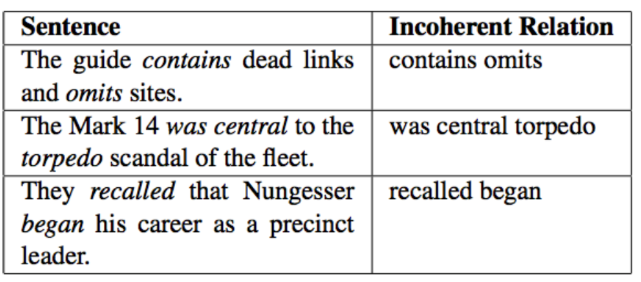
步骤: 1. 先找entities 给每个words打上概率最大的tags 2. 在名词短语之间 exam text来找关系 3. 给分类器送去candidate tuples

Redundancy-based Assessor 提取表示相同东西的不同的tuples

步骤: 1. 正则化tuples通过去掉不必要的modifier 2. 吧相同的tuples merged together, 并统计重复数 3. 根据重复的数量, 给<e1 r e2>分配概率



**ReVerb** 开放IE的问题: Incoherent 非语义: 解析出来的关系没有语义

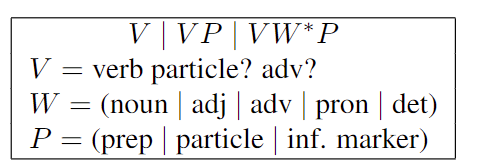
 Up to 13-30% output is incoherent

Uninformative 非信息: 提取的关系 忽略了 关键信息

extractor handles improperly verb-noun relation phrases 动名词短语处理不当

LVC: 动名词短语,多由名词承担语义

解决: 使用 语法 限制. Relation 必须 由verb开始, 跟一个POS



只有一个 verb则 invented “发明”

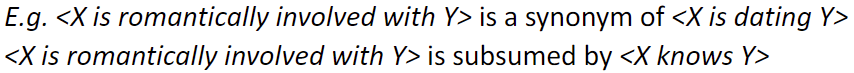
在preposition后面的verb则是 located in

在noun后面的verb则是 has atomic weight of

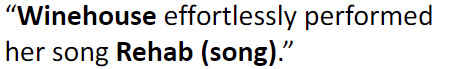
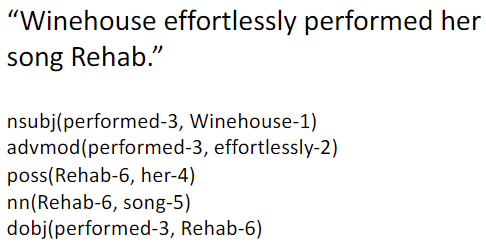
这种LVC会capture一些很奇怪的relations,要用 lexical constraint 词汇对比限制: 每个关系必须出现至少n次在不同的情况下

**PATTY** reVerb主要针对verb和动词短语 patty直接从语料库中编译关系模式, 并加入语义类型结构

Wordnet将名词 动词 形容词都划成 同义词. 在关系上做同样的事,



先用standord parser处理 独立语句 获得dependency path



然后在 sentences中 detect named entities, 对找到的entities 使用 dependency tree找到最短路径

关系: 

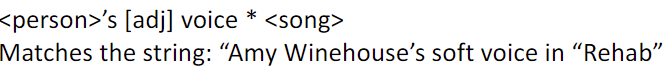
会施加一些 constraints 使得 结构是 subject-relation-object

在最短路径中的 words就是 textual pattern 这种textual pattern和 text的surface form一般都有关联 PATTY再将 这些 textual pattern转为 syntactic-ontologic-lexical patterns (SOL) SOL定义: 连接两个named entities的textual pattern的抽象

SOL pattern 包含: **[pos]** = a word of a certain POS pos 一个包含具体POS(adj啥的)的word

\* = a sequence of zero or more words **<class>** = an instance of the class

每个pattern包含至少两个instances



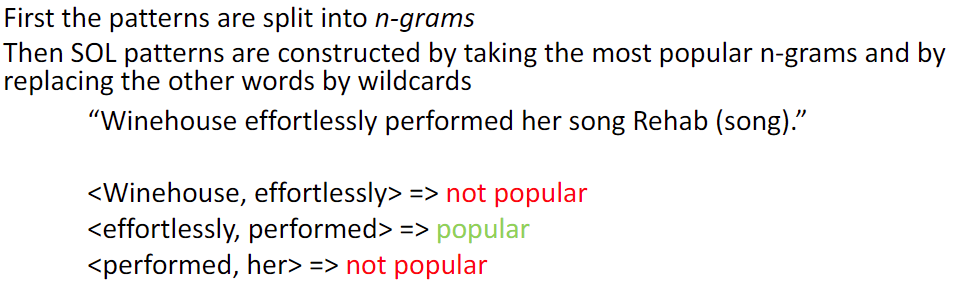
定义: 1. P在语法上more general than Q, if 每个能被Q match的string都能被P match

2. P在语法上more general than Q, if 所有能被 Q match的 entities set 能被P match

3. 如果 P在语法上more general than Q 且 Q在语法上more general than P, 则说明 PQ是 synonymous 同义的

PATTY extraction

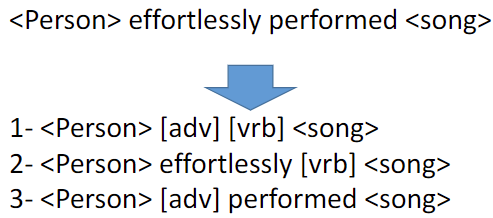
上述的SOL pattern是从 textual pattern中提取的



然后将 entities replaced by 他们属于的class

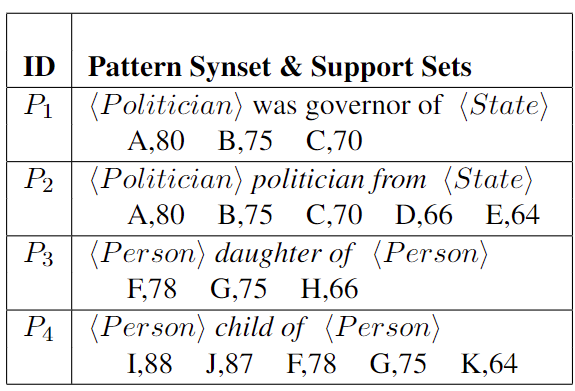
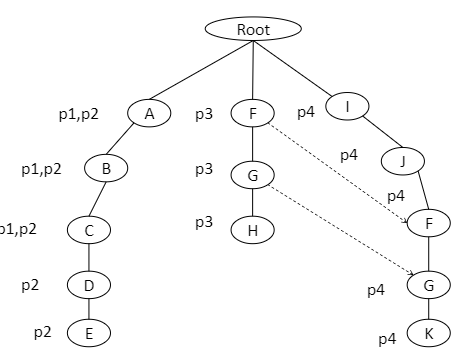


可以用 syntactic generalization 语法生成 来替换 words 为 他们的 POS



不是所有的generation是好的, 必须check matched entities是相同的

Pattern Taxonomy 分类: 最后将patterns分到 taxonomy

根据分类 可以构建 prefix-tree 前缀树, 遍历 tree 通过 bottom-up 可以得到 subsumption between patterns, P2比P1更加 generic

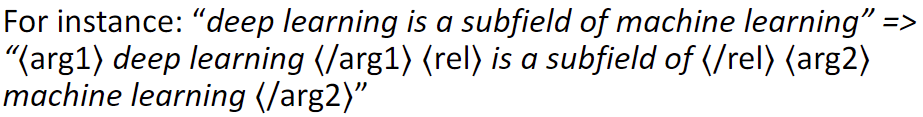
其他系统: NELL: 目标: 从web中提取 info 并 populate 一个大的KB

ClausIE: 发掘关于英语语法的语言知识, to detect clauses 从句 在 语句中

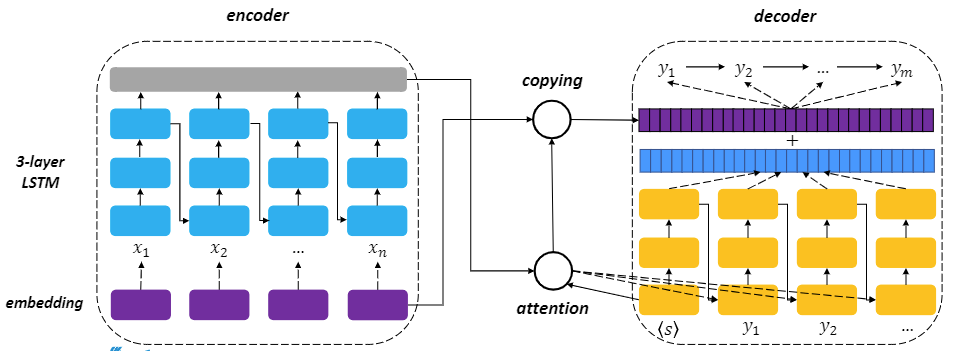
能 extract more info than ReVerb, WOE 和 TextRunner

**Neural Open Relation Extraction** 神经式提取:

Sequence to sequence model在RE中有效, 训练一个NN, 将序列 translate 为相应提取

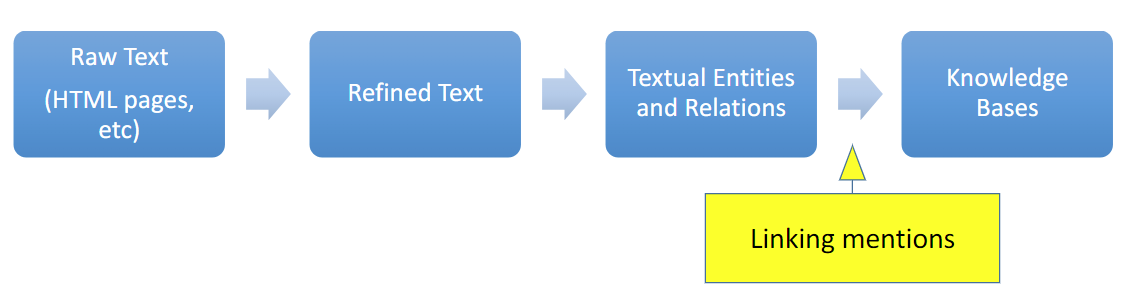


重新用 encoder-decoder 神经架构 来做 自动text translation



最后用 卷积提取器 训练 network

**Entity Linking**



将关系和实例 link 到 知识库中

主要问题（assignment）：给了个source doc，在其中identify并query他表示哪个Wikipedia。

任务：在text中找到mention的一个mapping to entities in KB

主要步骤：1. Candidate Entity Generation 候选实例生成 2. Candidate Entity Ranking 候选实例的排序 3. Unlinkable Mention Prediction 不可连预测

**Candidate Entity Generation** 目标：将any entity mention link到KB中的候选entities集合

Recall 很关键：因为如果link missed，没有办法recover

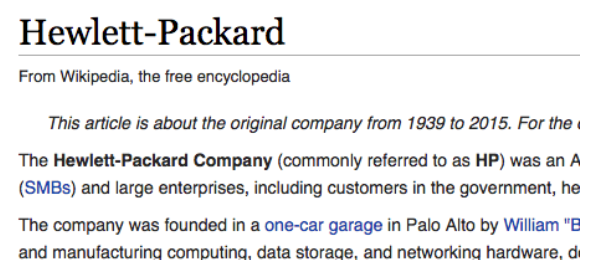
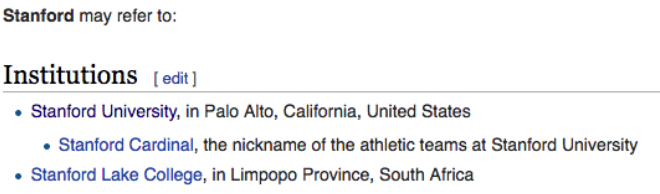
3中技术：1. Dictionary-based techniques 字典技术 2. Surface form expansion 名称扩展 3. Based on search-engines 搜索引擎

**Dictionary-based techniques**：构造一个 offline 的字典D between 多个names 和 KB中的潜在entities。 有了name，直接在字典里找到entities就行 常用wikipedia作为KB

Wikipedia entity pages 就是描述entity的page，其title就是name of entity

对于 别名 alternative name，在redirect那里。

当有重名的entity时候，有Disambiguation pages 来refer不同的entities



第一段的粗体，也是别名



描述中的超链接可能是同义词

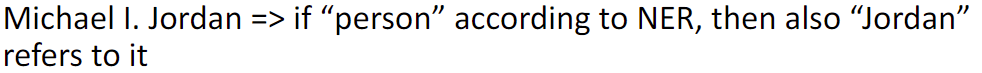
Text中的entities matched with dictionary keys 用 exact matching 精确匹配 和 string similarity measures 字符串相似度评估 如 hamming distance ， dice score相似评估

有时候 entities 会拼错，用算法纠正

**Surface form expansion**：一些entities是 acronyms 缩写 和 或者是字典 key 的 part of full names 两种办法：1. Heuristic-based methods 启发办法 2. Supervised learning methods

启发办法：搜索entities周围的text，用 pattern matching， 连续check N个连续的word（word的开头要和acronym 搜字母缩略词一样的）





监督学习：一些缩写很难用启发办法搜到，如 DOD=Department of Defense，这时候，用启发式得到的答案和SVM等分类器学习

**Based on search engines**：Google上搜，top k的网站就是了 坏处：not scale！

**Candidate Entity Ranking** 候选实例排序

在benchmark database上，实例基本都会有13个links to other entities，rank这些实例

两种方法：1. Supervised ranking 2. Unsupervised ranking

别的分类器：1. Independent Ranking：mention都是独立的

2. Collective Ranking：document refers to coherent entities 一致的实例

3. Collaborative Ranking： Leverage cross-document context to disambiguate 用跨上下文 文档 消除歧义

Feature：正确的Leverage mapping between mention 和 entities

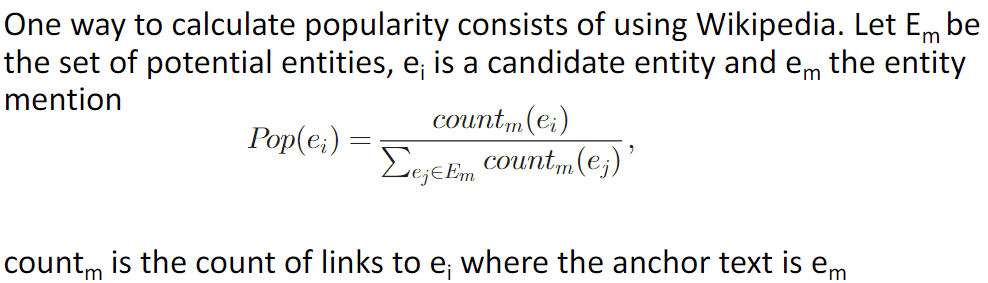


**Feature**

两种feature：1. Context-independent features 内容独立特征：只依靠 surface-form of entity mention 2. Context-dependent features 内容以来特征：也看entity mention周围的内容

**Context-independent features**：Name string comparison：简单check mention and the entity label in the KB match 和上面dictionary-based很像 有 Exact matching， Dice， Hamming distance 这些衡量办法

Entity Popularity：给定一个mention，选择跟这个mention有关的most popular entity 方法



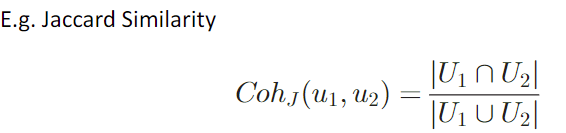
Entity type：NER 会返回一个大的mention类 如person song 检查 这个实例类型在KB中和NER识别的是否一样

**Context-dependent features**：实例周围的context提供valuable information： Bag of words：和enetity相关的所有words，并match这个words到entity

根据这些包含mention的docs，可以提取key-phrase，anchor text，named entities等，创建一个vector包含这些entities和mention，entity和mention的相似度计算方式：



Coherence between mapping 映射的连贯性：一个doc中的entity coherent 1个或多个 topics。 假设两个mentions有两个 candidate 实例 count articles link to both



Context dependent计算很贵，因为有很多种可能

**Ranking**

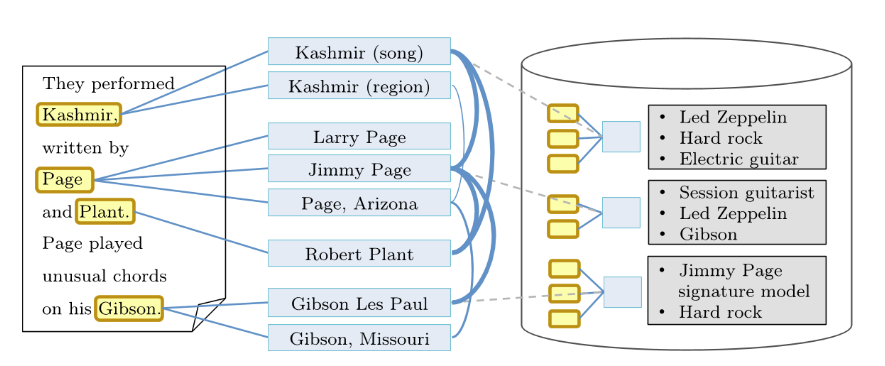
Supervised Ranking：

Binary Classification 二分分类：给input <mention, entity> 训练分类器 return 1 if 合适，0 不合适 常用分类器：SVM，简单贝叶斯等

Probabilistic Methods：不适用分类器，使用概率模型，表示likelihood of given mention and its surrounding connected to entity

Unsupervised Ranking：

**Graph Based** 图方法：**AIDS** system entity-mention 和 entity-entity 的 关系 用 图表示 links上有相似度值 Uses YAGO2 as KB, Stanford NER



找子图 – entity-mention的edge有最大权重

Steiner-tree problem NP-hard 用 贪心算法

**VSM based model：DBPedia Spotlight** 问题：好的training data是hard expensive的，Uses DBpedia as KB, LingPipe as NER 用vector 表示 mention 和entity，选择有相似measure的entity， 那就是closest to mention了

**Unlinkable Mention Prediction**：

如果候选entity压根儿就没有合适的，detect不到合适的mention-entity咋办 办法

1. Ignore this problem CEG就是1
2. 如果候选实例集是空，就是unlinkable
3. 用一个threshold value控制ranking score

4. Train a binary classifier Input <mention, top\_entity> output 1 if ok, 0 is bad

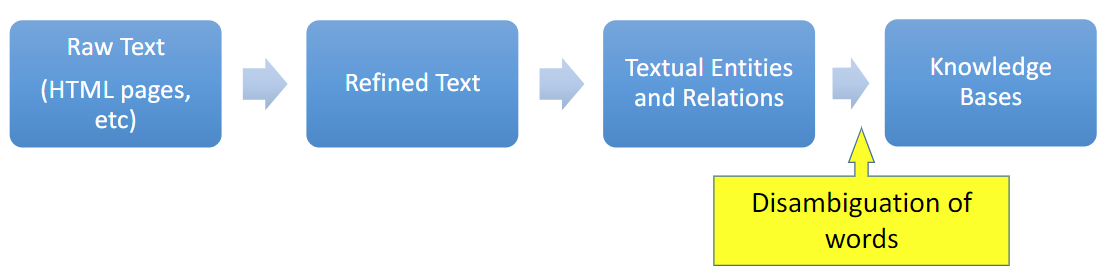
5. Add NIL as a special entity 采用 特殊实例，如果NIL在候选实例集中是最高得分 则是 unlinkable

State-of-art：

AIDA： YAGO2 as KB Stanford NER to detect the entities

DBPediaSpotlight： Uses DBpedia as KB LingPipe as NER 用VSM消歧义

**Disambiguation 消歧**



**Word Sense Disambiguation WSD词义消歧**

给定一个word in context, 和 一个固定的潜在词义表, 决定word到底是哪个意思

两种WSD**任务**: 1. Lexical sample 词汇样本: 一个小的目标words预选择集合 和 每个word对应的意思, 用监督学习炼一个 classifier给 each word 2. All-words: 每个word都有完整文本, 每个word都有a lexicon with sense, 因为数据稀疏, 无法训练分类器

WSD**方法**: 1. 监督学习 2. 字典 3. Semi-监督学习

**Supervised Methods** - Supervised Machine Learning Approaches

方法: 1. 一个训练语料库 带有有标记的上下文 和 他们的意思 2. 训练一个分类器在新文本里做 tag words 所需4样: the tag set, the training corpus语料库, 从语料库中提取出来的feature, 一个分类器

Supervised WSD 1: WSD Tags 一个tag就是 字典sense 比如 apple就有很多意思 水果 公司等 对于给定的sentence, 要确定apple是哪个 sense

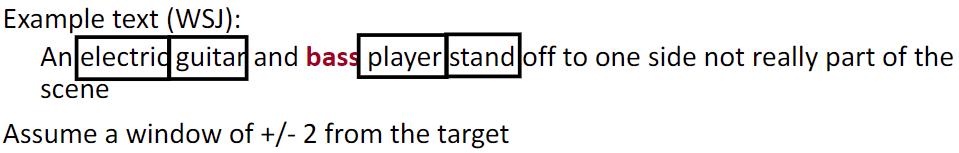
Supervised WSD 2: Get a corpus 获得语料库

对于所有的words 有 Semantic concordance: 语义一致性 在同一语料库中, 一对 开放类的 word只有一个字典意思 如 He 只有一个意思 POS

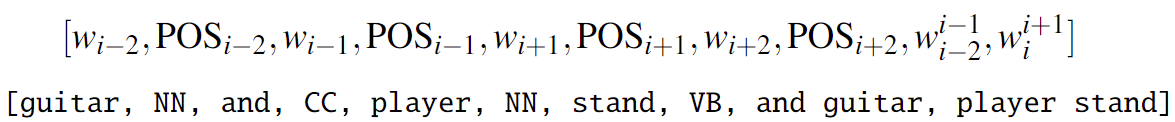


Supervised WSD 3: 提取特征向量

向量中的两证特征: 1. Collocational 并置: 在目标词附近的 specific position的 words 通常是 word 和 POS 2. Bag-of-word: BOW 关于在文本中任意位置出现的words 的 频数 通常是 frequencies count 例子:



Collocational features 是 根据window和目标词 bass +- 3的位置选词, 有:



Bag-of-words features: 就是计算文本中每个独立词出现的 frequency 是文本word频数直方图的一种

Co-Occurrence 同现 Example 下图是text中提取出来的单词



对特定的 words set的 vector 有: guitar and bass player stand [0,0,0,1,0,0,0,0,0,0,1,0]

原单词表中, 出现了 特定words set中的词 则 在vector中设置为1 否则为0

Classification Methods: Supervised Machine Learning

输入: 1个word 和 其文本窗doc 一个固定的类c 一个手工标记的训练集 将c和doc对应起来 输出: 分类器: 根据d 获得c

IMS (It makes sense) 最常用的算法 使用SVM

WSD Evaluations and baselines 最好的评估 extrinsic (‘end-to-end’, `task-based’) evaluation 外在评估 端对端 或者 基于task 但是常用的是 intrinsic evaluation 内在评估 1. Exact match sense accuracy 匹配的精度 2. 使用相同语料库的 held-out data 保留数据

Baseline: Most frequent sense (Most Popular) 就是 出现频数排第一的sense

**Dictionary and Thesaurus Methods** 字典和同义词典法

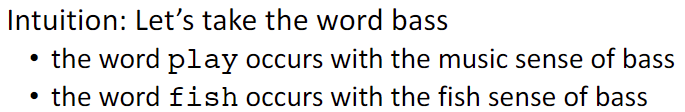
The Simplified Lesk algorithm: 找文本 和 字典定义 的相似词 有相似词就是

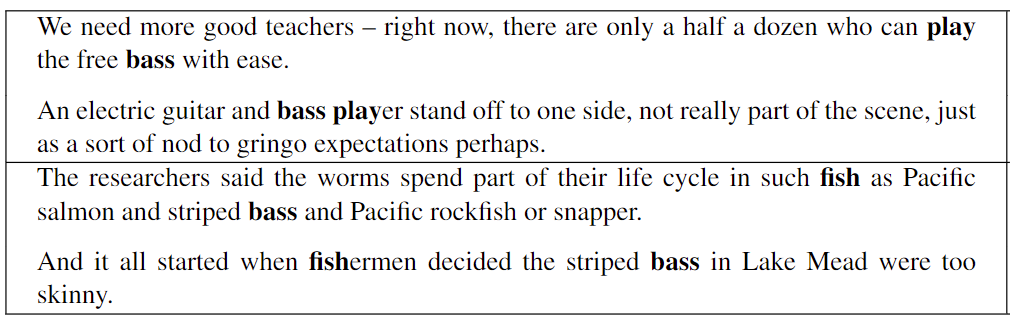
基于图的方法: WordNet是一个KB图

将目标词和其周围的词插入 KB中 查询他们之间的距离 选择 pagerank算法最高的node

**Semi-supervised** Methods: 监督学习和字典法都需要hand made数据集很麻烦 解决: Bootstrapping 使用自举法 生成一个小的种子集合

Yarosky ’s algorithm: 给target word专门学习一个分类器 然后使用自举法生成更多

 提取单词 bass 鲈鱼 贝斯



上图是bass两个意思的文本 生成seed: One sense per collocation 一个意思一个搭配: 一个单词在搭配中出现多次就一定有相同的关系

One sense per discourse 一个意思一个演讲: 在同一个doc里的相同word有相同的意思

Google’s的方法: 使用DNN解决

**General problem of WSD: in general it’s hard 通常来说很难**

需要人来做注解 做80%左右的人工wordnet风格的注解

SOA 60-80% 不存在能解决长尾效应的方法

**Knowledge Consumption 知识消费**

从web获得的知识的非trivial处理: 1. Latent knowledge 潜在知识 2. Manifest Knowledge 显性知识 process的input从 text 变为 graphs

**Latent knowledge 潜在知识**

原理: 1. Meaning is use – 常用于NLP的统计研究, 根据word的usage来判断词义,而不是根据所指的对象

2.啥都是能学会的 – 最新的ML可以捕捉到大数据和概念之间的non-trivial关系

**Word2vec**: 一类将words map到高维连续向量空间的办法 每个word都有个 real value vector, 这些vector包含的编码就是 embeddings 嵌入 由google提出,广泛应用

构建原则: You shall know a word by the company it keeps 根据他的伙伴来判断word

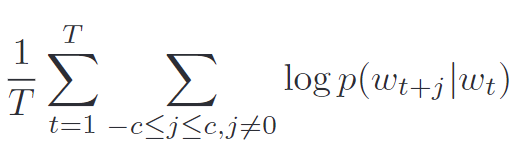
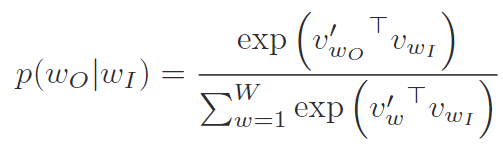
两种模型: 1. Skip-gram 2. Continuous bag-of-words (CBOW)

将word转成vector可以有很多用处: 计算相似度 分析等

Skip-gram model: 目标: 给了一个word 预测他周围的words

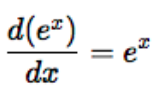


根据现有的语料库, 在w周围的单词, 用model去学习 然后return一个周围词的概率

最大化函数:  

和计算条件概率P, 将两个向量作为实数对待 in their vocabulary W

用函数的梯度 来计算 优化的方向 x轴为 w y轴为最大化的函数 用导数计算优化

因为概率和为1, 计算梯度就转化为  可以简化计算

Continuous Bag-of-Words (CBOW) 连续词袋模型: 另一种方法来执行相似的任务



目标: 给定w1到5, 预测w6, 和w6的概率 使用CNN硬怼就完事儿了

**Statistical Inference on KGs 知识图谱的统计推断**

目的: 1. **Growth** 知识图谱是高度 未完成的 (Knowledge extraction, Link prediction) 2. **Validation** 知识图谱不是 一直正确的 (Entity resolution: 将相同实例合并, 错误合并的实例分开 Error detection: 去掉错误论断)3. **Interface** 怎么更加简单的访问知识图谱 (Semantic parsing: 解释查询含义 Question answering: 使用图谱计算答案) 4. **Intelligence** AI 能否从知识图谱中出现 (自动解释和计划, 生成和抽象)

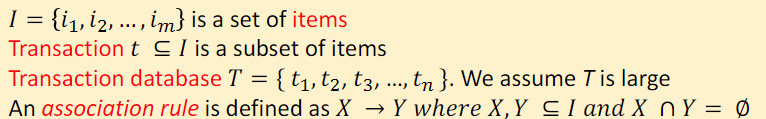
**Link Prediction**

目标: 从已存在的图中再增加知识 没有外部源 从图中推理

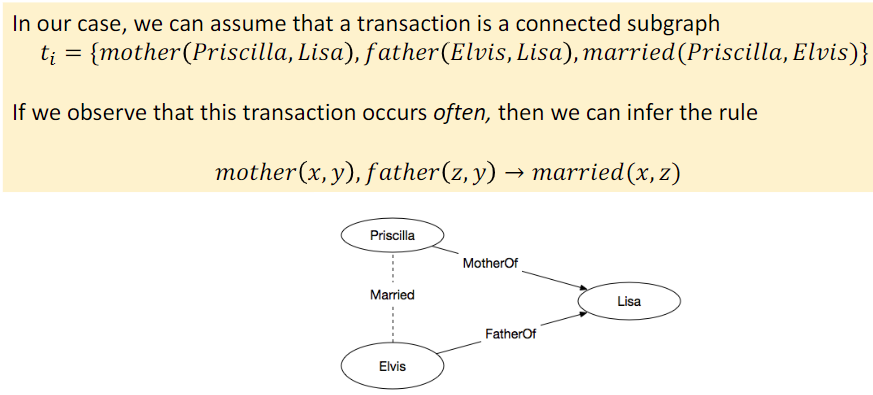
4种: Rule-based methods Probabilistic models Factorization models Embedding models

**Rule-based** methods **- Link Prediction with Rules** 通过规则来推理预测 rule mining

Association rule mining 关联规则挖掘: 将相关联的项 在一起发现规则 的过程



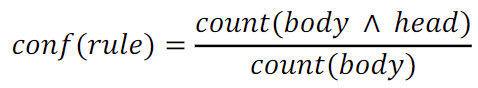
相关算法: Apriori algorithm



Inductive Logic Programming 归纳逻辑: 从事实中, 归纳逻辑的ML 从Web中提取rules

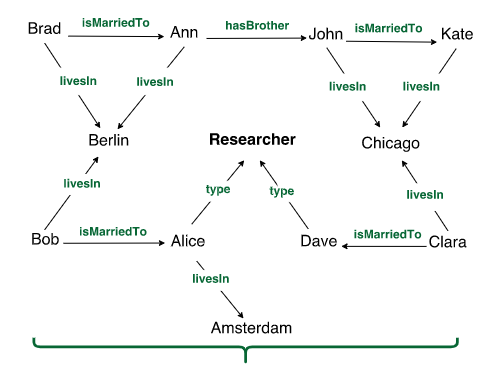
**AMIE**: 从系统中提取关系的 rule-association mining mindset

使用单原子原则, **fatherOf**(X,Y) 再加一个atom 共享一个变量 **motherOf**(Z,Y) 评估置信度



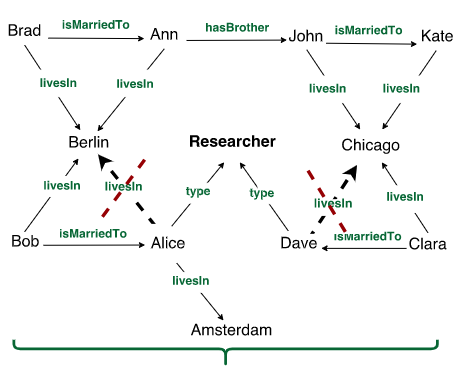
AMIE的创新点: 部分完整性假设 假设KG里包含所有正确的facts 哪怕KG本身并不清楚

算法: Non-monotonic learning 单调学习 专注 单一规则



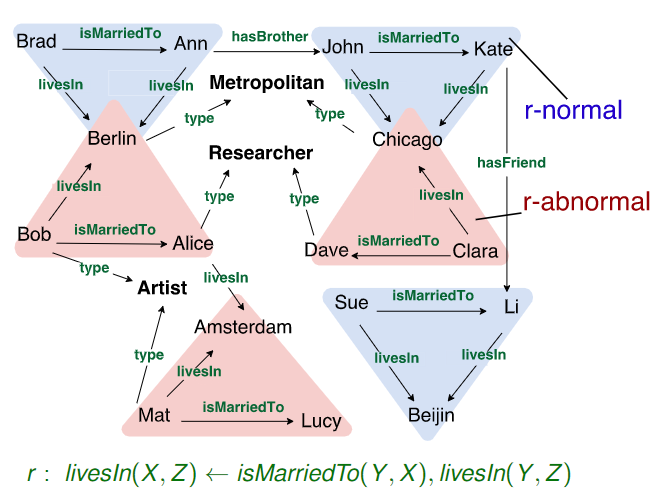


新的关系:





步骤: 1. 挖掘 关联关系 从 KG. 用FPGrowth 2. 确定正常和不正常的关系 3. 找到所有rules 的 candidates 4. Rank并选 best



**Probabilistic** models - **Link Prediction with Random Walks**

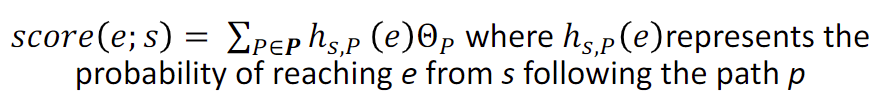
Rules的方法 计算贵 很脆弱 新方法 路径排序法(在图里随机移动, 记录path, 比较长度)

不再是单纯的随机, 而是有rules:



根据 head 的谓语来执行随机移动 并用 训练的triples来评估概率

假设有N 个规则 以playsInLeague为head, 构建线性回归模型 一个relation一个model



评估: 用NELL KG, 依赖人工(5个人有4个觉得ok)

**Factorization** 分解 models - **Link Prediction with Tensors 张量关系预测**

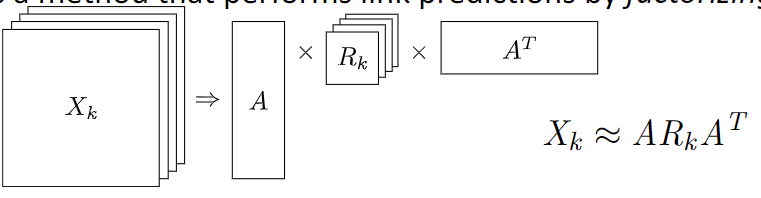
超过三维的array就是tensor

Tensor是ML中常见的数据结构 假设KG有m个关系和n个实例

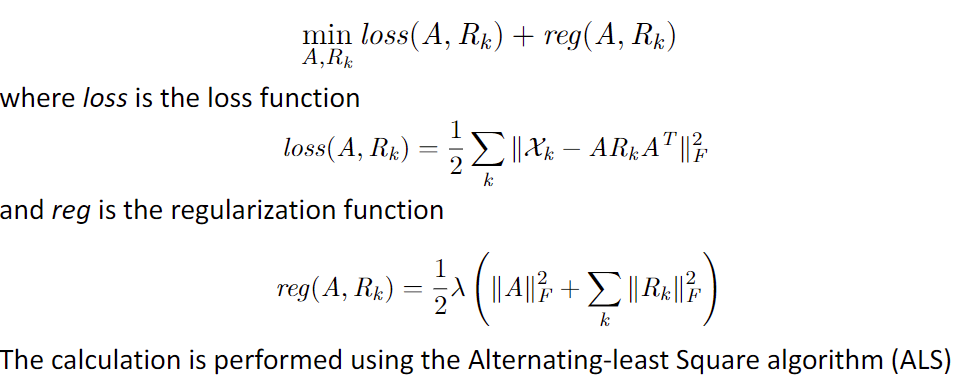
在实例间 用2modes 在关系间 1mode 当两个实例之间的关系存在, 则为1; 否则为0

(2,3,1) = 1 if R1 (e2,e3) exists in the KG

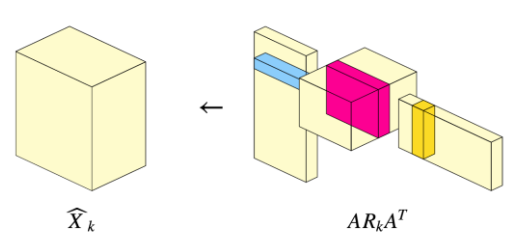
RESCAL 是 分解张量的link预测 n是实例数 r是维度 A是nxr 矩阵 表示 全局实例关系 R\_k 是 rxr 矩阵 表示预测



分解问题 会转化为 优化问题 ALS算法



RESCAL的基本步骤 1. 先计算 rank-reduced 分解 2. 重建的新实例就是模型的置信度



**Embedding** models 嵌入模型 - **Latent-features model** 延迟特性模型

RESCAL 就是 延迟特性模型 预测是有条件独立的, 要给全局延迟特性set

RESCAL 的参数数量: = 𝑟𝑑^2+𝑛𝑑 d是维度数, r关系数, n实例数 需要降低

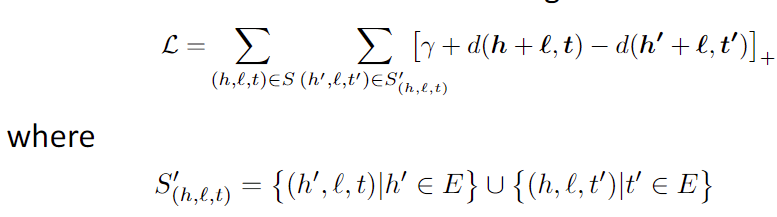
Latent distance models 延迟原型模型是 latent feature models 的 子集, link的相似度由实例的 latent表示之间的距离决定 如 one relation friendsOf two entities Mark and John

𝑓𝑟𝑖𝑒𝑛𝑑𝑠𝑂𝑓(𝑀𝑎𝑟𝑘, 𝐽𝑜ℎ𝑛)=𝑑(e\_mark, e\_john) d就是两实例的任意距离评估 (欧几里得距离)

**TransE** 是比rescal参数更少的模型(𝑟𝑑+𝑛𝑑) 每个实例和关系都有一个vector of d elements

𝑑(𝒆i+𝒓k, 𝒆j)是TransE 的距离, 给定一个实例和关系, 他们和的最近vector是哪个

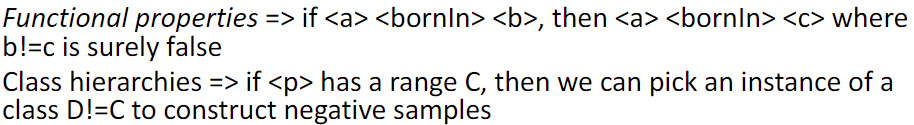
Train: 1. 给每个实例 和 每个关系 创建一个向量 d 2. 作为输入三元组 (h, l, t)属于S S是已知的KG 3. 最小化函数使用 随机梯度下降



Test: 1. 给定一个未知的triple 属于S` 计算𝑑(𝒉+𝒍, 𝒆v) 对于所有v属于E, 并rank 结果

2. check 平均的 位置k(mean rank), 以及t是否出现在top k的位置中 3. 看t的位置, 看看是否在KB里面

Open problem: 1. Negation 反例 – 一般用事实创建triples, 但是OWA不允许这样的假设, 用一些办法创建反例. 会引入偏见, 效果一般



2. Data sparsity 数据稀疏 – 只有一点可以训练的数据, 无解

3. Lack of established evaluation methodology 缺少既定的评估办法 – 没有简单的办法来直接评估所有性能, 数据集差异性也很大, 最新的办法是用重复推理来验证

4. Scalability 扩展性: Hogwild 可以并行的学习 sparse model, 扩展性还凑合

**Reasoning 推理**

提取推理的原因: 1.直接从统计模型看output, 很难理解 2.确定output是对的

**Ontologies/the Semantic Web** 本体 语义网

Ontologies 本体: formal explicit specification of a shared body of concepts 共享的明显指定

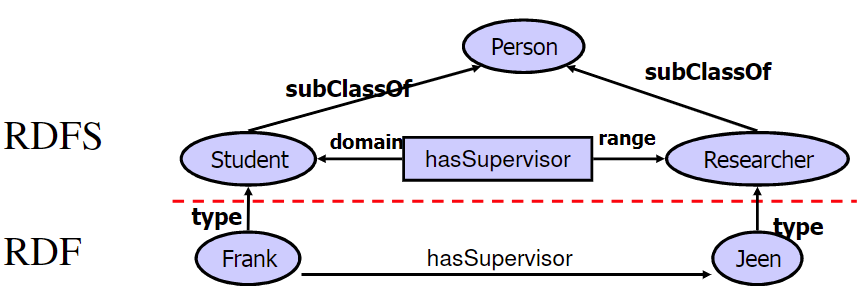
说白了就是 在一个领域里的 更精确的 意思



Ontological languages: 一种用于无歧义解释本体的语言 在web中多应用

The Semantic Web 语义网: 让机器理解 网络数据

RDF schema: 一种数据结构 rdf model是 rdf triples 的 set triple=(subject, predicate, object)



RDF具有基本的表达能力 但是不够高级

Web Ontology Language (OWL): 一种主体语言, 用于描述web上的内容 是rdf的高级设计

可以使用更高级的语义 <p> <rdf:type> <owl:Transitive>

OWL2 更高级的 有三种: EL-大量类和实体 QL-高效率query重写 RL-用rules表达推理

**Reasoning 推理**

在本体中的data如已被标注, 则可以使用他们 推理出新的 知识和结论

推理的不同点: 1. 算法 2. scope(分类, link prediction 等)

Notation: 假设 𝑇(𝐴,𝐵,𝑦)∧𝑃(𝑥,𝐵)→𝑄(𝑥,𝐴) PTQ都是要谓词, AB是标量, xy是常量

T(A,B,y), P(x,B), Q(x,A) 是原子, 没有变量的原子就是fact事实

T(A,B,y), P(x,B) is the body of the rule. 这俩是规则

Q(x,A) is the head of the rule. 用于判断是否开始应用规则

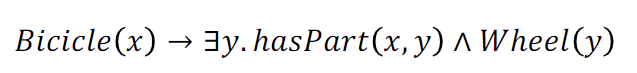
Substitution 𝜎 是替换式, 用于根据规则 产生新的 替换(知识)

Given 𝜎= {A/x, B/y }, T(A,B,z) -> 𝜎= T(x,y,z) 将T本来的AB换为xy

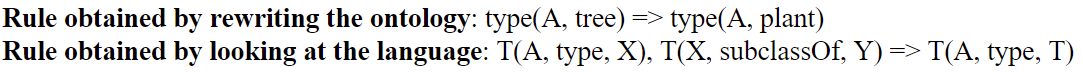
**Rules,** 规则有很多种, 常用的规则有: **Horn clauses 喇叭形**



和 Existentially quantified rules 存在量化规则: head中有的变量在body里没有

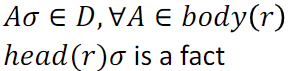


两种rules reasoning:



**Materialization** 具体化 – safe rules

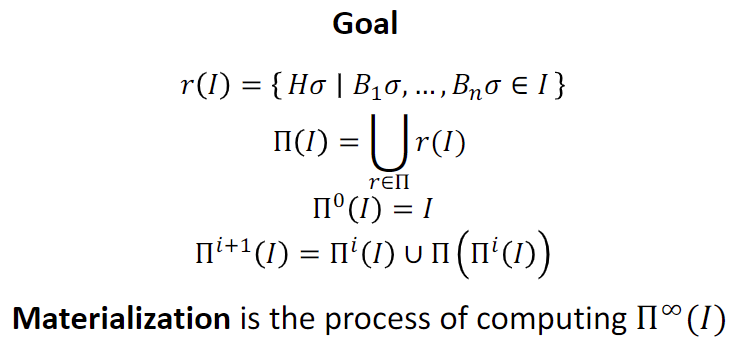
假设有个数据库有 D个facts, 执行 Horn rule r 去找所有替换, 则有



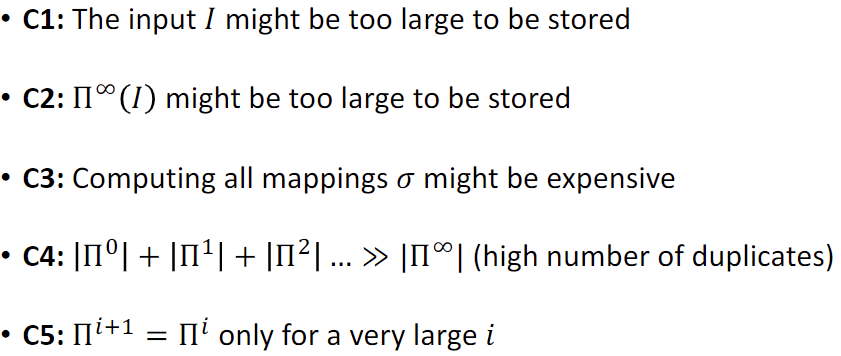
有规则 r 在 D 上的实例化



给定一个数据集, 和规则集 Π, 返回所有从该规则生成的结论



上述问题的 scale很大, 是个大问题: 5个难点:



**Datalog 数据分析**

Datalog and Prolog 是两种常用的 语言 用来执行规则的materialization Prolog AI社区开发 Datalog Database社区开发

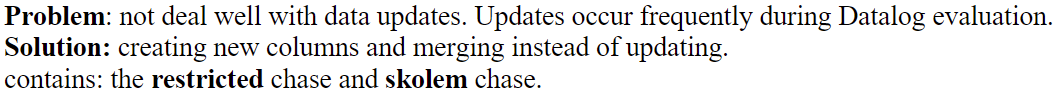
很多Datalog 被用来 在 webdata中 推理 有 RDFox Vlog等

**Vlog** 自己学校开发的SOA, 特点: 使用柱状存储策略 降冗 省内存 不是分布式 但是可以用不同的数据库后端 Log按col排列而不是row排列

R1: p(a,b) :- q(b,a) R2: q(a,b) :- p(a,b)

在R2中, 可以去掉R1的p表, 因为重复了 列是immutable 不变的, 可以重复使用

用列是一种更有效的压缩方式



SociaLite: 针对社群网路优化的datalog

内存消耗:best 2x less worst: 14x less

Alternatives to full materialization 完全具体替换: 完全materialization 具体化是不现实的, 主体会corrupted损坏, KG是动态的

很多工作都是在 **incremental reasoning** 和 **reasoning at query time**的

**reasoning at query time 在query时候的推理**: 增加条件限制 让 答案只和 想要的有关

Select ?s {?s rdf:type:Person ? :lives :Amsterdam} 只select住在阿姆的人

**top-down evaluation 自上而下的评估**: **先check head 再check body**, 以符合query要求

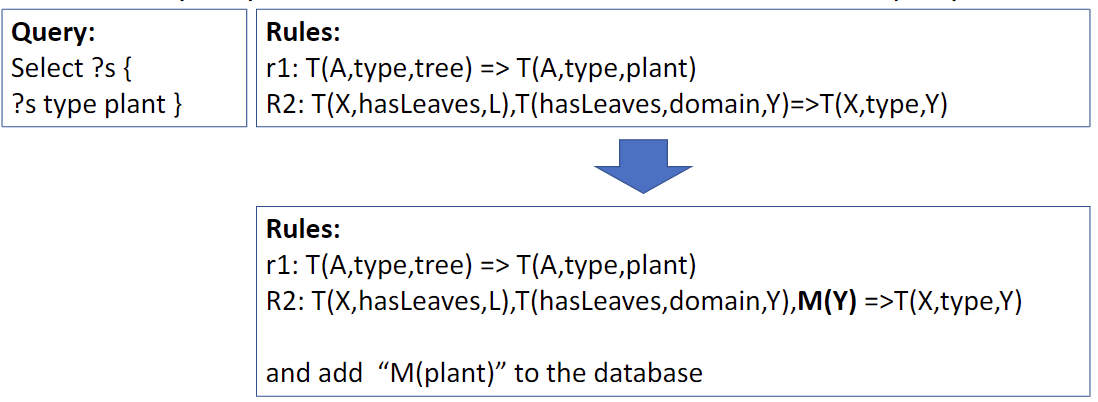


要回答query, 先检查r1的head T(A, type, tree) 是否match, 然后到body check是否符合query, 再到R2同样步骤. 问题: 要跟踪rules的推导, 不然可能无限

QSQ (query-subquery) 算法, 用暂时table来存子query和中间推导

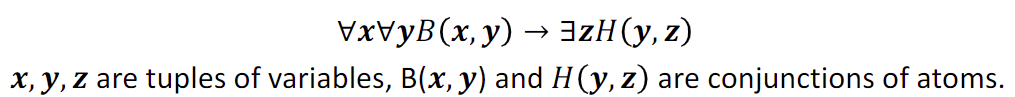
**bottom-up evaluation 自下向上的评估:** **重写rules,** 这样结论只和query有关

可以重写 query 让 他们推导的结论只和query内容有关 新的rules M(Y) = M(plant)



这种重写算法是 magic-sets 给query Q和 规则集P, magic set生成新的规则集, 专门生成query结果Q, 这种重写是一种暂时预测

**Reasoning with Existentials** 推理与存在



对任意有关系B的x和y, 都能退出来存在一个z, 和y有关系H 推理存在

这种东西是useful且在多领域的 但执行这种规则的前提是要引入**新的个体** z

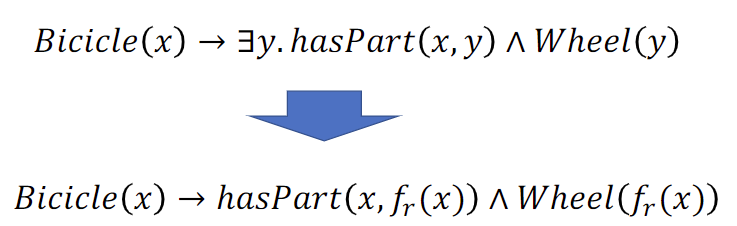


自行车x, 在body里知道了x 但是不知道y

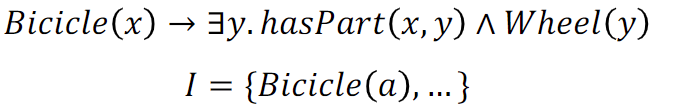
这种例子的推导 是不会停止的, 且detect也是 无法确定的 但是 有很多现实场景 且 他们的条件是足够停止的 这些条件叫做 (a)cyclicity conditions 回旋条件

**The Chase** 算法: 一种**自下而上的**针对存在规则的推理算法 两个变量: skolem restricted

**Skolem** chase: 去掉存在的量化变量, 然后换成has函数中的f(x)



**Restricted** chase: 在head没有实例化时候, 使用



在 hasPart(a, \*) 时 触发 𝜎= {𝑥 -> a}

**Restricted** chase 停了, 但**Skolem** chase 不停, 因为前者计算昂贵

Vlog里的chasing: 上述都实现了, restricted chase 用 列来存储 并 检查部分的实例化

Skolem chase 用 hashmap 来实现 skolem func

**Fact Checking 事实检查**

定义: 确定一个factual statement实际语句的正确性 来自: 从外部源找证据, 评估并聚合证据

Web上的错误信息有很多, 如: fake news 现实问题: WEF认为数字错误信息是个大问题

Falsehood 谬误的类型: 1. “Technical” falsehood: 提取错误, 提取出来的facts就是错的 或者 有噪音 2. Factual falsehood 事实错误, 恶作剧 hoaxes, 阴谋论 conspiracy, 错误的科学发现 等.

如 在Blog上发布misleading 视频, 数据输入导致的错误, 数据过时导致的错误

错误的信息比没有信息还坏

Misinformation on the Web 分析两种假新闻的先行: 1. conspiracy theories 阴谋论 – 想象, 乱猜, 降低现实复杂性,来源未知 2. scientific information 科学信息 – 传播科学的东西, 知道来源 阴谋论更具有市场些, 随时间而增加

**为啥fact checking很难:** 1. Sparsity 稀疏: 数据太少了, 而且要verified数据更难

2. Conflicts 矛盾: 数据高度不一致, 还是在容忍了1%的不同下

3. Trustworthiness 信任度: 权威数据来源的数据准确度也不行

4. Semantic Ambiguity 语义歧义: 在不同的语义下, 歧义率近半

5. Instance Ambiguity 实例模糊: 一个实例很多links

6. change overtime 随时改变: 数据经常改变

7. Text understanding 文本理解: 文本离散, 很难理解

8. Inference 推论: 数据有时候是推导出来的, 并不能实现很好的对应

**Check啥:** 1. Different formats: 结构triples: (Obama, born\_in, Kenya)

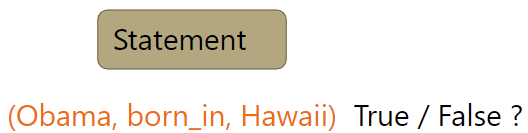
textual claims: “balabala” Entire articles

2. Different types: string, categorical 副词, numerical 数字

**在哪儿check:** 1. 图data 2. 结构data 3. 非结构data

**Fact Checking from Text and Social Networks**

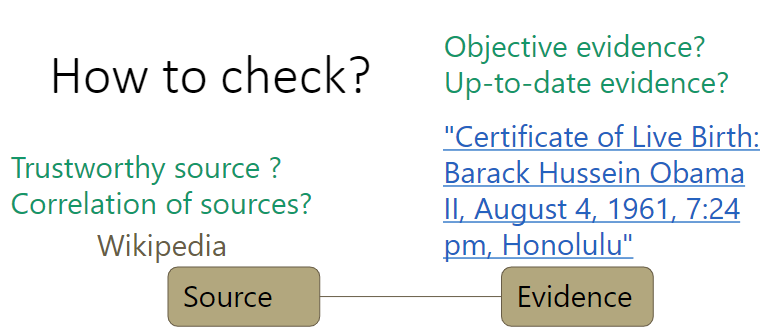
4步: 1. 收集证据 2. 评估证据 3. 链接模型 4. 预测正确性 例子:



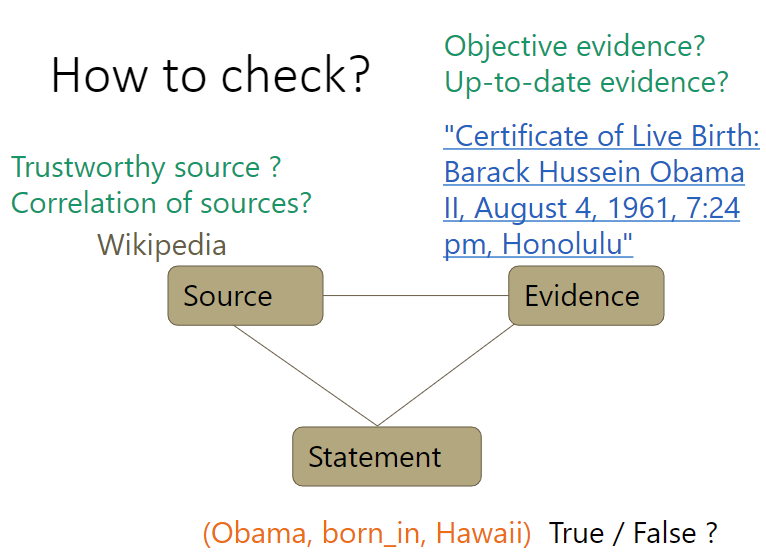
收集证据:



评估证据: 可信不? 时效性? 正确不?



链接和预测



Fact checker的三个目标: 1. Be accurate, 要真 2. Be scalable, 要能扩展 3. Be interpretable 要能解释

**Information Credibility on Twitter** 推特信息可信度

最著名的fact-checking案例, 数据在事件发生后产生大量的 交流, 这些交流有真有假

提出的方法: 监督分类 – 1. 找到带有标签的event – 用亚麻的事件集2. 训练分类器 – 先看是不是event, 再看真不真

事件特征分类: 1. Message-based features: 看发的message 2. User-based features: 看用户画像 3. Topic-based features: 看话题关键词 4. Propagation-based features: 看都是咋传播的

**DeClarE**: Evidence-aware Deep Learning

给一段文本, 来确定是不是真的 (credible or non credible)

4步: 1. 给定claim 2. 看下claim的source 3. 看下真不真的证据 4. 证据的来源

Key idea: 用LSTM 和attention机制

**ExFaKT** Explaining Facts over Knowledge Graphs and Text: 在text和KG上的是事实解释

以前的都是返回数值, 这个返回 有意义的解释

Key-idea: 用rules构建人能读懂的解释

Main features: 1. Consult both KGs and textual documents 同时访问KG和文本

2. 使用rules构建解释

解释必须是: 1. Concise 简洁 (few atoms) 2. Close to the query (chain不能太长) 3. Reliable 靠谱 (基于高质量数据)

这里的rules使用从上到下, 推理算法: 1. Explain 计算一个query q 的解释 2. Process goal: 回答query或者重写规则 不能使用as-is格式

**Privacy and Fairness on the Web 隐私和公平**

Online post或多或少都会暴露个人信息 会被用来打广告

假设: 广告会rank users并标记出来最可能买东西的人

目标: 告诉这帮傻逼, 上网冲浪少暴露点信息

办法: 定义sensitive状态: 看他的online, 是否用更多的words在sensitive topic上

1. First possibility (ENTROPY 熵): 先计算一个社群里用户的平均值, 看每个用户产生的KL divergence measure 相关信息熵, 来看他们有多不同 >平均值

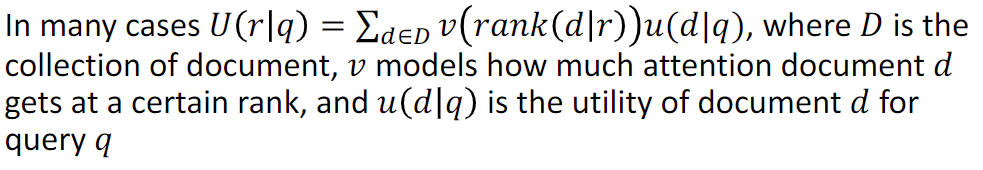
2. Second possibility (DIFF-PRIV): 累加用户们的数据, check敏感信息概率分布,如果加了一个人数据, 突然分布增长, 这人有问题 (叫做: differential-privacy 差分隐私)

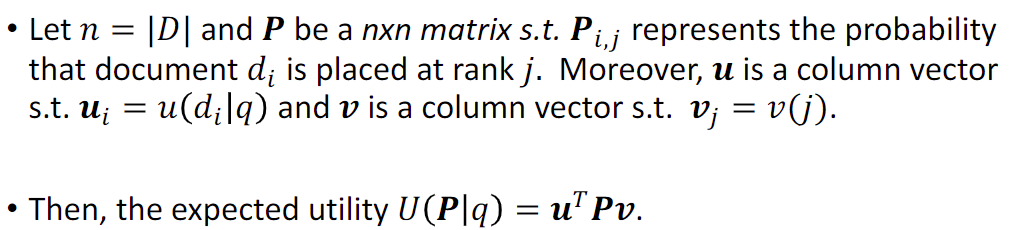
3. Third possibility: 构建topic users和post的分布表示, vector可以是 bag-of-words, LDA, W2V等, 风险计算就是算用户vector和目标分布vector的相似度

评估: 用 crowdsourcing service 众源信息来定义一系列敏感信息, 因为人们不可能同意同意通过, 所以抓取用户的真实judge作为验证的标准

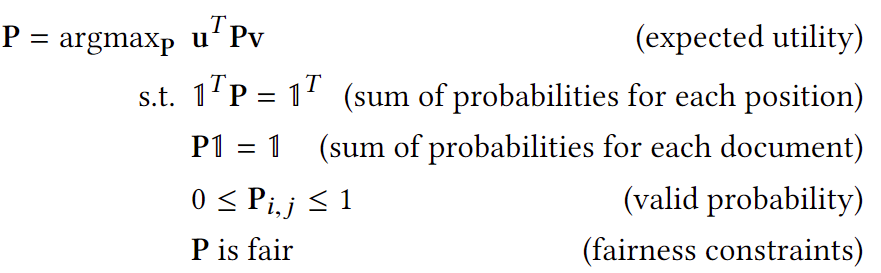
Fairness in Ranking: 所有的ranking都是unfair的

目标: 给一个query q, 选择一个fair ranking, 最大化U, U是utility function 效用函数





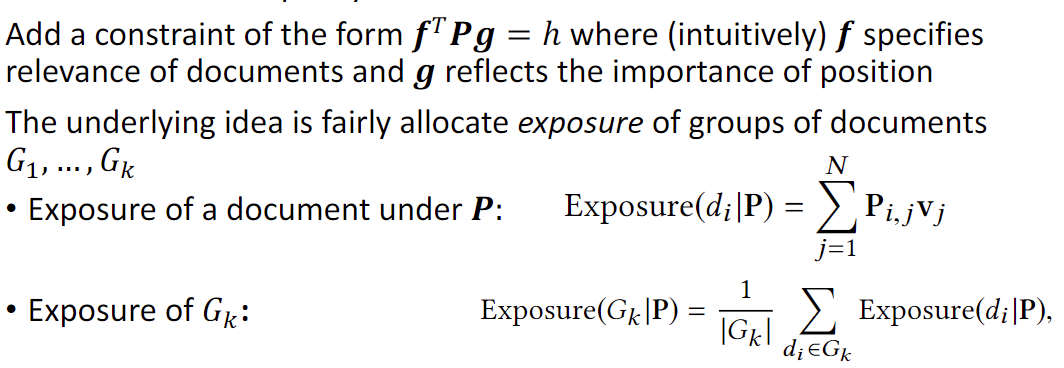
算法: 1,计算



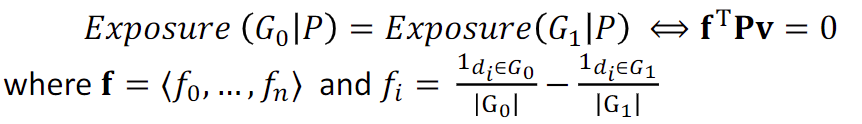
2. 一旦计算出来P, 可以计算概率ranking R, 用 Birkhoff-von Neumann decomposition

3. sample ranking r ~ R

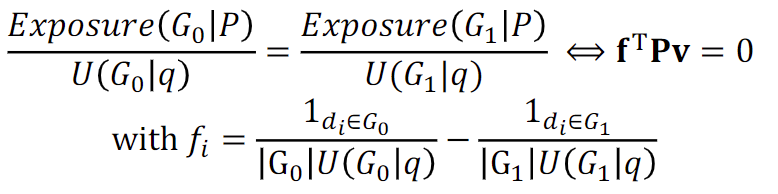
问题: 能算了, 怎么评估P是公平的, 加点限制算个曝光Gk



第一种选择: **Demographic parity constraint** 平等对待约束 只是用2个G, G0和G1



第二种选择: **Disparate treatment constraint** 区别对待约束



总结:

1. Various techniques for performing NLP preprocessing 多种NLP

2. One system to perform entity recognition (**NeuroNER**) 一种NER

3. Two systems to perform entity linking (**AIDA, DBPedia spotlight**) 两种linking

4. Two systems to do traditional relation extraction (**DIPRE**, **Snowball**) 两种关系提取

5. Three systems to do word sense disambiguation (**IMS, PPR, Google**) 三种消歧

6. One system to calculate word embeddings (**word2vec**) 一种word嵌入

7. Four systems for statistical inference (**AMIE, PRA, RESCAL, TransE**) 四种统计推断

8. One system to do logic-based reasoning (**VLog**) 一种逻辑推理

9. One system to protect privacy and technique for fairness in ranking 一种隐私公平