Training SRL Models

Dr. Asja Fischer, Prof. Jens Lehmann

Overview

- Last lecture we got to know latent distance models (as third example for latent feature models) which make the probability of the existence of a relation between two entities dependent on the distance of their latent representation.
- We then learned about graph feature models which explain the existence of triples from features directly observed in the KG.
- Today we will see, in the first part, how latent and graph feature models can be combined.
- In the second part we will learn some general aspects about model training that are specific for knowledge graph analysis.

Latent feature and graph feature models

- ▷ The strengths of latent feature and graph feature models are complementary, as both families focus on different aspects of relational data.
- ▷ Latent feature models are
  - well-suited for modeling **global relational patterns via newly introduced latent variables**.
  - computationally efficient if triples can be explained with a small number of latent variables.
- ▷ Graph feature models are
  - well-suited for modeling **local and quasi-local graph patters**.
  - computationally efficient if triples can be explained from the neighborhood of entities or short paths in the graph.

Latent feature and graph feature models

- ▷ The strengths of latent feature and graph feature models are complementary, as both families focus on different aspects of relational data.
- ▷ Latent feature models are
  - well-suited for modeling **global relational patterns via newly introduced latent variables**.
  - computationally efficient if triples can be explained with a small number of latent variables.
- ▷ Graph feature models are
  - well-suited for modeling **local and quasi-local graph patters**.
  - computationally efficient if triples can be explained from the neighborhood of entities or short paths in the graph.

⇒ **Idea: Combine latent and graph feature models!**

Additive relational effects (ARE) model [1]

- Idea: Combine RESCAL with PRA.
- The score function gets

$$f_{ijk}^{\text{RESCAL+PRA}} = \boldsymbol{r}_k^{(1)\mathrm{T}} \boldsymbol{x}^{\text{RESCAL}} + \boldsymbol{r}_k^{(2)\mathrm{T}} \boldsymbol{x}^{\text{PRA}}$$

  where $\boldsymbol{x}^{\text{RESCAL}} = \boldsymbol{e}_i \otimes \boldsymbol{e}_j$ and $\boldsymbol{x}^{\text{PRA}} = [P(\pi) : \pi \in \Pi_{e_i,e_j}]$.
- Can be trained by alternately optimizing RESCAL and PRA parameters.
- RESCAL only needs to model "residual errors" which can not be modeled by observable graph patterns.
- This allows for latent variables with lower dimensionality.

[1] Nickels et al. *Reducing the Rank in Relational Factorization Models by Including Observable Patterns*, NIPS 1014

Additive models

▷ Other additive models[2] combine latent feature models with an additive term to learn from latent and neighborhood based information:

$$f_{ijk}^{\text{ADD}} = \boldsymbol{r}_{k,j}^{(1)\text{T}} \boldsymbol{x}_i^{\text{SUB}} + \boldsymbol{r}_{k,i}^{(2)\text{T}} \boldsymbol{x}_j^{\text{OBJ}} + \boldsymbol{r}_k^{(3)\text{T}} \boldsymbol{x}_{ijk}^{\text{N}}$$

where

- $\boldsymbol{x}_i^{\text{SUB}}$ is the latent representation of the ith entity as a subject.
- $\boldsymbol{x}_j^{\text{OBJ}}$ is the latent representation of the jth entity as an object.
- $\boldsymbol{x}_{ijk}^{\text{N}} = [y_{ijk'} : k' \neq k]$ is for capturing patterns where the existence of some other triple $y_{ijk'}$ is predictive for the triple of interest.

[2] Jiang et al. *Link Prediction in Multi-relational Graphs using Additive Models*. CEUR Workshop Proceedings, 2012

Stacking: a general way to combine models

- Stacking[3] (sometimes called stacked generalization) corresponds to training a learning algorithm to combine the predictions of several other learning algorithms.
- First, some algorithms are trained on the data, then a combiner algorithm is trained to make a final prediction using the predictions of the other algorithms as (additional) inputs.
- E.g. train a binary classifier on the scalar output of PRA and the ER-MLP.
- Advantage: Flexibility in the kind of models that can be combined.
- Disadvantage: Individual models cannot cooperate, and thus need to be more complex than in combined models.
- E.g. When stacking based on RESCAL and PRA one will need more latent features than for joint training.

[3] Wolpert, *Stacked generalization*. Neural networks, 1992.

Training SRL models on KGs

Up until know we learned about different kind of models

- ▷ Latent feature models (based on tensor factorization, neural networks or distance minimization)
- ▷ Graph feature models (based on neighborhood or path/random walk information)
- ▷ Combinations of both.
- ▷ Markov logic networks.

Training SRL models on KGs

Up until know we learned about different kind of models

- ▷ Latent feature models (based on tensor factorization, neural networks or distance minimization)
- ▷ Graph feature models (based on neighborhood or path/random walk information)
- ▷ Combinations of both.
- ▷ Markov logic networks.

What else do we need to know for training the models?

- ▷ Where do the negative examples come from?
- ▷ Which loss-functions to take?
- ▷ How to perform model selection?
- ▷ How to evaluate the performance of a model?

Open vs. closed world assumption

- Existing triples always encode known true relationships (facts), but there different interpretations of non-existing triples.
- Under the **closed world assumption (CWA)** non-existing triples indicate false relationships.
- Under the **open world assumption (OWA)** non-existing triple are interpreted as unknown, i.e., the corresponding relationship can be either true or false.
- The open world assumption fits the fact that KGs are known to be very incomplete.
- E.g., even the place of birth attribute is missing for 71% of all people included in Freebase.

Ways to get negative examples

- Training is based on a set $\mathcal{D}$ of training examples $(x^n, y^n), n = 1, 2, \ldots,$ where x^n is a triple and $y^n \in \{0, 1\}$ indicates if the triple is true or false.
- It is easy to get positive training examples corresponding to true facts.
- Lets $\mathcal{D}^+ = \{x^n \in \mathcal{D} | y^n = 1\}$ denote the set of observed true triples.
- Training only on $\mathcal{D}^+$ is tricky and can lead to over generalization.
- Where do negative examples (corresponding to false facts) come from?
- Under the **closed world assumption** all triples $x^n \notin \mathcal{D}^+$ are false.
- However, for incomplete KGs the assumption is violated.
- Moreover, $\mathcal{D}^- = \{x^n \in \mathcal{D} | y^n = 0\}$ might be very large which can lead to scalability issues.

Getting negative examples by exploiting constraints

- Alternative approach: **exploit known constraints** on KG structure, such as
  - **type constraints** for predicates
    (e.g., persons are only married to persons)
  - **valid attribute ranges** for predicates
    (e.g., the height of humans is below 3 meters)
  - **functional constrains** such as mutual exclusion
    (e.g., a person is born exactly in one city).
- It is guaranteed that examples violating such hard constrains are indeed negative examples.
- But, functional constraints are scarce and negative examples based on type constraints and valid attribute ranges usually not sufficient to train useful models.

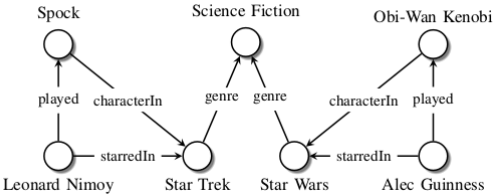
Getting negative examples by perturbing true triples

- Better approach: Generate examples by **"perturbing" true triples**, i.e., by replacing subject or object in true triples.
- We get

$$\mathcal{D}^- = \{(e_l, r_k, e_j) | e_l \neq e_i \wedge (e_i, r_k, e_j) \in \mathcal{D}^+\} \\ \cap \{(e_i, r_k, e_l) | e_l \neq e_j \wedge (e_i, r_k, e_j) \in \mathcal{D}^+\}$$

- This leads to a smaller $\mathcal{D}^-$ and to more plausible negatives than based on the closed world assumption.

Getting negative examples by perturbing true triples



- ▷ Close world assumption would generate
  - good negative examples such as (*LeonardNimoy*, *starredIn*, *StarWars*),(*AlecGuinness*, *starredIn*, *StarTrek*).
  - type-consistent but irrelevant negative triples such as (*BarackObama*, *starredIn*, *StarTrek*).
- ▷ The letter would not be generated by perturbation based generation, since there exist no triples (*BarackObama*, *starredIn*, ·).

Getting negative examples from local CWA

- **Local-closed world assumption**: assume that a KG is only locally complete.
- If one has observed any triple for a subject-predicate pair e_i, r_k, assume that any non-existing triple $(e_i, r_k, \cdot)$ is indeed false and include it in $\mathcal{D}^-$.
- This assumption is valid for functional relations, such as *bornIn*, but not for set-valued relations, such as *starredIn*.
- If we have not observed any triple at all for the pair e_i, r_k, assume that all triples $(e_i, r_k, \cdot)$ are unknown and not include them in $\mathcal{D}^-$.

Getting negative examples from text extraction

- One can also make use of the candidate triples generated by **text extraction methods** run on the Web.
- Many of these triples will be false, due to extraction errors, but define a good set of plausible negatives.
- This technique is for example used in the Knowledge Vault project.

Recall: Probabilistic models for SRL

- Idea: model the knowledge graph by a joint probability distribution $P(Y)$.
- Let $\mathcal{D}$ be the set of all observed triples and N_e and N_r be the numbers of entities and relations respectively. Assuming that all y_{ijk} are independent of each other given a set of parameters Θ we can write

$$P(Y|\mathcal{D}, \boldsymbol{\theta}) = \prod_{n=1}^{N} \text{Ber}\Big(y^n|\sigma\big(f(x^n; \boldsymbol{\theta})\big)\Big)$$

with $\sigma = 1/(1 + e^{-u})$ and

$$\text{Ber}(y|p) = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{if } y = 0 \end{cases} .$$

Recall: Probabilistic models for SRL

- Idea: model the knowledge graph by a joint probability distribution $P(Y)$.
- Let $\mathcal{D}$ be the set of all observed triples and N_e and N_r be the numbers of entities and relations respectively. Assuming that all y_{ijk} are independent of each other given a set of parameters Θ we can write

$$P(Y|\mathcal{D}, \boldsymbol{\theta}) = \prod_{n=1}^{N} \text{Ber}\Big(y^n|\sigma\big(f(x^n; \boldsymbol{\theta})\big)\Big)$$

with $\sigma = 1/(1 + e^{-u})$ and

$$\text{Ber}(y|p) = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{if } y = 0 \end{cases} .$$

- **How can we train such a model?**

Recapitulation: Maximum Likelihood Estimate

- Let $x^1, \ldots, x^N$ be i.i.d samples drawn from an unknown probability distribution P_0.
- We assume that P_0 belongs to a certain family of distributions $P(\cdot|\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta} \in \Theta$, i.e., $P_0(\cdot) = p(\cdot|\boldsymbol{\theta}_0)$ for unknown parameters $\boldsymbol{\theta}_0$.
- How can we find an estimate $\hat{\boldsymbol{\theta}}$ which is as close to $\boldsymbol{\theta}_0$ as possible?
- Let us first note that (because of the independence of the samples) the joint probability distribution is given by

$$P(x^1, \ldots, x^N|\boldsymbol{\theta}) = \prod_{n=1}^{N} P(x^n|\boldsymbol{\theta})$$

Recapitulation: Maximum Likelihood Estimate

- Lets change the perspective: Consider the samples $x^1, \ldots, x^N$ as fixed, and let $\boldsymbol{\theta}$ be the function variables.
- This function is called the **likelihood**

$$\mathcal{L}(\boldsymbol{\theta}; x^1, \ldots, x^N) = P(x^1, \ldots, x^N | \boldsymbol{\theta}) = \prod_{n=1}^{N} P(x^n | \boldsymbol{\theta}) \ .$$

- The method of **maximum likelihood** estimates $\boldsymbol{\theta}_0$ by finding the parameters that maximize $\mathcal{L}(\boldsymbol{\theta}; x^1, \ldots, x^N)$.
- Thus the **maximum likelihood estimate** is given by

$$\hat{\boldsymbol{\theta}}_{MLE} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; x^1, \ldots, x^N) \ .$$

Recapitulation: Maximum Likelihood Estimate

- In practice its often more convenient to work with the **log-likelihood**

$$\log \mathcal{L}(\boldsymbol{\theta}; x^1, \ldots, x^N) = \log P(x^1, \ldots, x^N | \boldsymbol{\theta}) = \sum_{n=1}^{N} \log P(x^n | \boldsymbol{\theta}) \ .$$

- This does not change the maximum likelihood estimate, since the logarithm is a monotone increasing function and thus

$$\hat{\boldsymbol{\theta}}_{MLE} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; x^1, \ldots, x^N) = \arg\max_{\boldsymbol{\theta}} \log \mathcal{L}(\boldsymbol{\theta}; x^1, \ldots, x^N) \ .$$

Maximum likelihood learning for SRL models

- Lets look again at the probabilistic SRL model

$$P(Y|\mathcal{D},\boldsymbol{\theta}) = \prod_{n=1}^{N} \mathrm{Ber}\Big(y^n|\sigma\big(f(x^n;\boldsymbol{\theta})\big)\Big) \ .$$

- The maximum likelihood estimate is given by

$$\hat{\boldsymbol{\theta}}_{MLE} = \arg\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log \mathrm{Ber}\Big(y^n|\sigma\big(f(x^n;\boldsymbol{\theta})\big)\Big) \ .$$

- They can for example be found by performing stochastic gradient ascent on the (log-)likelihood.
- The negative (log-)likelihood defines a loss-function for probabilistic models.
- Depending on $f(x^n;\boldsymbol{\theta})$ there might be several (local) optima.

Maximum likelihood learning for SRL models

▷ We can write

$$\hat{\boldsymbol{\theta}}_{MLE} = \arg\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log \mathrm{Ber}\Big(y^n | \sigma\big(f(x^n;\boldsymbol{\theta})\big)\Big)$$
$$= \arg\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} y^n \log \sigma(f(x^n,\boldsymbol{\theta})) + (1-y^n)\log(1-\sigma(f(x^n,\boldsymbol{\theta}))) \ .$$

▷ Recall, that we have seen this before for the example of RESCAL

$$\arg\max_{\boldsymbol{A},\boldsymbol{R}} \sum_{ijk} y_{ijk} \log \sigma(\boldsymbol{a}_i \boldsymbol{\mathcal{R}}_{:,:,k} \boldsymbol{a}_j^{\mathrm{T}}) + (1-y_{ijk}) \log(1-\sigma(\boldsymbol{a}_i \boldsymbol{\mathcal{R}}_{:,:,k} \boldsymbol{a}_j^{\mathrm{T}})) \ ,$$

where $x_{ijk} = 1$ if triple (e_i, r_k, e_j) exists and $x_{ijk} = 0$ otherwise.

Recapitulation: Maximum a-posteriori estimate

- ▷ But what if we have some prior knowledge about the parameters?
- ▷ In Bayesian ML we use the **Bayes rule** to infer model parameters $\boldsymbol{\theta}$ from the data $\mathcal{D} = \{x^1, \ldots, x^N\}$ by

$$P(\boldsymbol{\theta}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} \quad .$$

where

- $P(\boldsymbol{\theta}|\mathcal{D})$ is called the **posterior**.
- $P(\mathcal{D}|\boldsymbol{\theta})$ is called the **likelihood**.
- $P(\boldsymbol{\theta})$ is called the **prior**.
- and $P(\mathcal{D})$ is the probability of observing the data which we can not compute.

Recapitulation: Maximum a-posteriori estimate

- Since $P(\mathcal{D})$ is the same for all models, we can equivalently write

$$P(\boldsymbol{\theta}|\mathcal{D}) \propto P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta}) \ .$$

- The **maximum a-posteriori estimate (MAP)** is given by

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{MAP} &= \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{\theta}|\mathcal{D}) \\ &= \arg\max_{\boldsymbol{\theta}} P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}} \log P(\mathcal{D}|\boldsymbol{\theta}) + \log P(\boldsymbol{\theta}) \ .\end{aligned}$$

- With a uniform prior $P(\boldsymbol{\theta})$ it is the same as the maximum likelihood estimate.

Maximum a-posteriori for SRL models

- The MAP estimate for the probabilistic SRL model is given by

$$\hat{\boldsymbol{\theta}}_{MLE} = \arg\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log \mathrm{Ber}\Big(y^n | \sigma\big(f(x^n;\boldsymbol{\theta})\big)\Big) + \log P(\boldsymbol{\theta}) \ .$$

- The prior can also be interpreted as a regularization term (for including additional information to prevent overfitting).
- With $\mathcal{L}(f(x^n;\boldsymbol{\theta}), y^n) = -\mathrm{Ber}\Big(y^n | \sigma\big(f(x^n;\boldsymbol{\theta})\big)\Big)$ we can state this as regularized loss minimization problem

$$\arg\min_{\theta} \mathcal{L}(f(x^n;\boldsymbol{\theta}), y^n) + \lambda \mathrm{reg}(\boldsymbol{\theta})$$

where λ depends on the prior.

Squared loss and mean squared error

- For score-based models there exists a variety of other loss functions.
- The **squared loss** (as mean over training examples also referred to as **mean squared error (MSE)**) is given by

$$(f_{ijk} - y_{ijk})^2 \ .$$

- Recall, that based on this the (regularized) minimization problem for RESCAL becomes

$$\arg\min_{\boldsymbol{A},\boldsymbol{\mathcal{R}}} ||\boldsymbol{\mathcal{T}} - \boldsymbol{\mathcal{R}} \times_1 \boldsymbol{A} \times_2 \boldsymbol{A}||_F^2 + \lambda_A||\boldsymbol{A}||^2 + \lambda_R||_F\boldsymbol{\mathcal{R}}||_F^2 \ ,$$

  where $||\cdot||_F$ is the Frobenius norm (the tensor/matrix variant of the Euclidean norm).
- This is build on the **closed world assumption**
- and can be minimized by alternating least squares (ALS).

Logistic Loss

- ▷ Another example of an loss function that can be used under the **closed word assumption** is the **logistic loss.**
- ▷ In this case one assumes that $y_i \in \{-1, +1\}$.
- ▷ The regularized loss minimization problem with logistic loss is then given by

$$\arg\min_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log(1 + \exp(-y_i, f(x_i : \boldsymbol{\theta})) + \lambda ||\boldsymbol{\theta}||^2 \ .$$

Pairwise loss function

- In the case of an **open world assumption** in which generated negative samples are not guaranteed to be really negative, it can make more sense to use a **pairwise loss function**.
- General idea: Encourage larger score function values for positive samples from $\mathcal{D}^+$ than for negative samples from $\mathcal{D}^-$:

$$\arg\min_{\boldsymbol{\theta}} \sum_{x^+ \in \mathcal{D}^+} \sum_{x^+ \in \mathcal{D}^-} \mathcal{L}(f(x^+, \boldsymbol{\theta}), f(x^-, \boldsymbol{\theta})) + \lambda \text{reg}(\boldsymbol{\theta}) \ ,$$

where $\mathcal{L}$ is a **margin based ranking loss function** such as

$$\mathcal{L}(f(x^+, \boldsymbol{\theta}), f(x^-, \boldsymbol{\theta})) = \max(0, \gamma + f(x^-, \boldsymbol{\theta}) - f(x^+, \boldsymbol{\theta})) \ .$$

Model selection

- ▷ Remember, that training success of a lot of models depends on choosing good values for regularization parameters and other hyper parameters, like
  - the dimension of latent features (all latent distance models)
  - the dimension of the hidden layer (all MLP based models)
  - the length of relation paths (for PRA)
  - etc.
- ▷ The choice should be made based on cross validation or on the performance on a separate validation set.

Evaluation Criteria

- For a specific relation link prediction can be seen as binary classification of entity pairs (does link between entities exist or not).
- Let us consider a binary classification problem in which outcomes are either *positive* (e.g. "link exists") or *negative* (e.g. "link does not exist") .
- For a binary classifier there exist four different outcomes:
  - **true positive**: actual label is positive and prediction is positive.
  - **false positive**: actual label is negative but prediction is positive.
  - **true negative**: actual label is negative and prediction is negative.
  - **false negative**: actual label is positive but prediction is negative.

True/false positive rate

- **True positive rate (TPR)** (also known as **sensitivity** or **recall**): "probability of detection".

$$\frac{\#\text{true positive}}{\#\text{samples with positive label}}$$

- **False positive rate (FPR)** (also known as **fall-out**): "probability of false alarm"

$$\frac{\#\text{false positive}}{\#\text{samples with negative label}}$$

Receiver operating characteristic (ROC) curve

- ▷ In binary classification, the class prediction for each instance is often made based on a score (e.g. f_{ijk}).
- ▷ Given a threshold parameter T the instances (e.g. (e_i, r_k, e_j)) are classified as positive if the score is larger than T (e.g. $f_{ijk} > T$) and negative otherwise.
- ▷ The TPR and FPR vary with T, which can be indicated by writing TPR(T) and FPR(T).
- ▷ The **ROC curve** is produced by plotting TPR(T) on y-axis against FPR(T) on x-axis.

Area under the curve (AUC)

- If one randomly picks a negative and a positive example from, one wants the score to be higher for the positive than the negative example.
- The **area under the ROC curve (ROC-AUC)** is the percentage of randomly drawn pairs for which this is true.
- Under some additional assumptions it corresponds to the probability that the the classifier will rank a randomly chosen positive sample higher than a randomly chosen negative sample.
- The AUC can for example be numerically estimated based on constructing trapeziods under the curve as an approximation of the area.

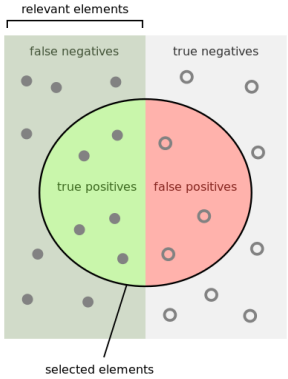
Precision and recall

▷ **Precision**:

$$\frac{\#\text{true positive}}{\#\text{samples classified as positive}} = \frac{\#\text{true positive}}{\#\text{true positive} + \text{false positive}}$$

▷ **Recall**:

$$\frac{\#\text{true positive}}{\#\text{samples with positive label}} = \frac{\#\text{true positive}}{\#\text{true positive} + \text{false negatives}}$$



Area under precision-recall curve

- The **precision-recall curve** is produced by plotting precision (in dependence on T) on y-axis against recall (in dependence on T) on x-axis.
- As before, the **area under the precision-recall curve (AUC-PR)** is good evaluation criterion.
- It has been shown that for data with a large number of negative examples (as its typical the case for KGs) the AUC-PR can give a clearer picture of an algorithms performance than the AUC-ROC.[4]

[4] Davis and Goadrich. *The relationship between precision-recall and ROC curves*. ICML. 2006.

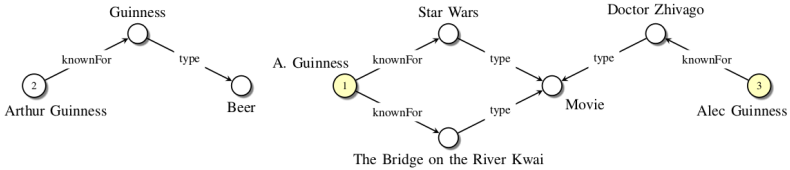
Mean reciprocal rank

- Recall the task of **entity resolution**: The problem of identifying which objects in relational data refer to same underlying entities.
- Assume a SRL system that given an entity returns scores for a set of candidate entities which could refer to the same object (as higher the score as higher the likeliness of referring to the same object).
- E.g. a system returning the entities corresponding to the k-nearest neighbors of the query entity in latent space and the distances as scores.
- Then the entities are ordered by decreasing score (i.e. the lower the score the higher the rank).
- The **mean reciprocal rank (MRR)** is given by the average of the reciprocal ranks of the correct results

$$MRR = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{rank_i}$$

where n is the number of query entities where candidates are estimated for.

Mean reciprocal rank - example



- For the query entity *A. Guenniss* the system returns to candidates: *Alec Guinniss* and *Arthur Guinness*.
- If *Alec Guinniss* has a higher score than *Arthur Guinness* the reciprocal rank is 1.
- If it is the other way around the reciprocal rank is $\frac{1}{2}$.

Summary

- Latent feature models (good in modeling global relational patters) and graph feature models (good in modeling (quasi-)local graph patterns) can be combined, e.g. the ARE model combines RESCAL with PRA.
- Stacking corresponds to training a learning algorithm to combine the predictions of other learning algorithms.
- Negative training examples can be generated by exploiting known constraints or the local CWA, by perturbing true triples, or using negative examples resulting from text extraction methods.
- The maximum likelihood and the maximum a-posteriori principle can be used to train probabilistic models.
- Score-based models building on the CWA can use the mean squared error or the logistic loss for training.
- Score-based models building on the OWA can use a pairwise margin based ranking loss.
- The AUC-ROC and the AUC-PR are good evaluation criteria for link prediction models.
- The mean reciprocal rank a good option for entity resolution models.

References

- Image on slide 32 was taken from
  https://en.wikipedia.org/wiki/Precision_and_recall