Feature Engineering and Explainability with Vadalog: A Recommender Systems Application

Jack Clearman[1], Ruslan R. Fayzrakhmanov[2], Georg Gottlob[2,3] and Yavor Nenov[2], Stéphane Reissfelder[2], Emanuel Sallinger[2,3], and Evgeny Sherkhonov[2]

[1] Meltwater Group
[2] University of Oxford
[3] TU Wien

1 Introduction

Vadalog [2] is an extension of Datalog that features existential rules and a rich set of functions, libraries, and methods for connecting to external data sources, which make it a powerful tool for building advanced industrial AI applications [1]. Vadalog forms the core of an ongoing research collaboration between the University of Oxford and the media intelligence company Meltwater, that aims at a recommender system for the most relevant insights about companies from outside data, including Meltwater's repository of millions of news articles. In this application paper, we demonstrate various aspects of such a recommender system in the movies domain, and show how Vadalog can be used for feature engineering and the computation of explainable recommendations.

Recommender Systems assist users in choosing the most relevant items they may be interested in, thus reducing the experienced information load. The typical methods used in recommender systems are based on the analysis of items the user has already selected and are usually limited to "low-level" features, i.e., metadata associated with an item. However, such methods are not able to provide suitable recommendations in the absence of discriminative low-level features or the presence of non-trivial combinations of features which capture discrepancy between liked and disliked items. In this paper, we approach this problem by building a new set of *high-level* features that can capture domain knowledge and non-trivial factors that influence user's decision in choosing movies. Vadalog is well suited for computing such high-level features, by having support for: (1) *aggregation*, for computing features such as total revenue of movies, (2) *graph traversal* for computing properties on the co-starring graph, (3) *integration of various data sources* for unified access to multiple sources, such as IMDB and RottenTomatoes, and (4) *existential rules* used in the computation of recommendations for new users. Furthermore, declarativeness of Vadalog allows developing high-level features rapidly (usually, a few hours per feature from conception to deployment) and easily maintaining the resulting programs. Finally, we demonstrate how to build an *explainable* ranking of recommendations, thus allowing Vadalog not only to provide explanations at reasoning level, i.e., why a particular high-level feature has a certain value, but also explanation at the machine learning level, i.e., why there is a particular ranking of items.

```
feature(User, "AwardWinningCast", Movie, Score) :-
    user(User),
    hasAwardWinningActor(Movie, Person, Award),
    awardScore(Award, AwardScore),
    Score = max(AwardScore).

hasAwardWinningActor(Movie, Person, Award) :-
    crew(Movie, PersonID, "Cast"),
    hasWonPrestigiousAward(Person, Award).

hasWonPrestigiousAward(Person, "Oscars") :-
   oscarsAward(Nomination, Person, Movie, Year).

hasWonPrestigiousAward(Person, "BAFTA") :-
   baftaAward(Nomination, Person, Movie, Year).

@input("oscarsAward").
@bind("oscarsAward", "postgres", "awards", "oscars").
@input("baftaAward").
@bind("baftaAward", "postgres", "awards", "bafta").
```

(a) Award Winning Cast

```
feature(User, "HighlyRatedDirector", Movie, Score) :-
    user(User),
    crew(Movie, Person, "Director"),
    directorWithHighRating(Person, Score).

directorWithHighRating(Person, AvgRating) :-
    crew(Movie, Person, "Director"),
    imdbRating(Movie, Rating),
    AvgRating = avg(Rating),
    AvgRating > 8.5.
```

(b) Highly Rated Director

```
feature(User, "Co-Production", Movie, Score) :-
    coProduced(User, LikedMovie, Movie, Hop),
    scoreTable(Hop, Score).

coProduced(User, Movie, Movie, 0) :-
    user(User),
    likedMovie(User, Movie).

coProduced(User, Movie, Movie2, NewHop) :-
    coProduced(User, Movie, Movie1, Hop),
    produced(Producer, Movie1),
    produced(Producer, Movie2),
    Movie1 != Movie2,
    Hop < 4,
    NewHop = Hop + 1.
```

(c) Co-Production

Fig. 1: High-Level Features

Due to space limitation we do not provide preliminaries for syntax and semantics of Vadalog and refer the reader to [2] for details. Note that in the below constructed programs negation and aggregate functions are restricted to be *stratified*.

2 High-Level Features

We consider data closely resembling IMDB, the largest movie industry database. The relation crew(Movie, Person, Role) represents crew members with their role in the production, imdbRating(Movie, Rating) represents the movies' ratings, produced(Producer, Movie) represents the production studio, and oscars(Nomination, Person, Movie, Year) as well as bafta(Nomination, Person, Movie, Year) are external data sources providing the Oscars resp. BAFTA award information. We assume that the common attributes Person and Movie in the latter two relations have been appropriately linked via a custom Vadalog program. Additionally we use relations user(User), likedMovie(User, Movie), ratedMovie(User, Movie, Rating) and friend(User1, User2) to provide users, their liked movies, the rating a user has given to a movie and the pairs of friends.

We next demonstrate how Vadalog can be used to build a number of high-level features in the movies domain. Here we show only those that are aimed at demonstrating the (combinations of) three main use cases mentioned above: integration of various data sources, aggregates, and graph traversal. All feature values are stored in a predicate with the following signature: feature(User, FeatureName, Movie, Score), where User is the user ID and Score is the value of the feature FeatureName for each movie.

Award Winning Cast. Often one of the factors in choosing a movie is whether the movie features a star cast. A cast member is usually considered a "star" if they have won a prestigious award such as an Oscars or BAFTA award. Such a

high-level feature is described in Figure 1a, where awardScore stores predefined scores for each type of award. The first rule generates a score for a given user and a movie if the movie has a cast member that has won an award. This is encoded in the predicate hasAwardWinningActor defined in the second rule. The need for max is justified by the fact that the input movie can have multiple actors with different awards, but we assign the best resulting score. The rules for hasWonPrestigiousAward integrate two different data sources: the Oscars and BAFTA datasets. In particular, the last four lines declare how external datasources (in this case PostgreSQL tables) are bound to predicates.

Highly Rated Director. Another possible factor for choosing a movie is whether the movie's director has a good track record. This can be modelled by choosing directors whose average movie ratings exceeds a certain threshold. This is formalised in Figure 1b where the predicate directorWithHighRating stores all directors with their high average ratings. This feature computation demonstrates the need for aggregate queries, such as average rating. Other similar features can be built using aggregates, e.g., the producer's total revenue of all movies they produced and average sentiment about the movie in the social media.

Co-Production. This feature captures the following intuition: a producer related to a movie liked by a user is likely to produce movies that the user will like as well. Assume that we have a list of movies that the user has already liked, which we refer to as a *seed list*. Based on this list we can build *relative* features, i.e., their values are relative to the seed list. Our feature builds on the co-production relation: two movies are in the relation if they were produced by the same producer or a company. The co-production relation can then be transitively closed and the feature value reflects how "far" a movie is from the seed list in the resulting relation. This is formalised in Figure 1c, where the last two rules demonstrate a (limited depth) recursive definition of the predicate coProduced. Similarly, other relative features such as *Co-direction* and *Co-starring* can be computed by traversing the corresponding relations.

Cold start. The relative features above assume that a given user has a set of movies they liked. This information however is not available for new users who have not liked any movies yet. This problem is known as "cold start" in Recommender Systems. One way to overcome this is by creating "placeholder" movies that have attributes such as Producer or Actor that are populated by most popular producers and actors from movies liked by the user's friends. Vadalog is particularly suitable to model such a situation as it supports existential quantifiers. The rules are shown in Figure 2a, where the first two compute the most featured actors and the top rated directors from the movies that the user's friends have liked. Then the next rule creates the placeholder movies for a user in case they have not liked any movie. In particular, the variable Movie is existentially quantified. The last rule defines an extended relation likedMovieExt that stores a placeholder movie for a user that has not liked any movie yet as well as known liked movies for each user.

```
topOccuringCastMembers(User, Person) :-
    friend(User, User1),
    likedMovie(User1, Movie),
    crew(Movie, Person, "Cast"),
    Occurrence = count(Movie),
    occThreshold(User, Threshold),
    Occurrence > Threshold.

topRatedDirectors(User, Person) :-
    friend(User, User1),
    ratedMovie(User1, Movie, Rating),
    crew(Movie, Person, "Director"),
    AvgRating = avg(Rating),
    rateThreshold(User, Threshold),
    AvgRating > Threshold.

likedMovieExt(User, Movie),
cast(Movie, Cast),
crew(Movie, Director, "Director") :-
    user(User),
    topOccuringCastMembers(User, Cast),
    topRatedDirectors(User, Director) .
    not likedMovie(User, _).

likedMovieExt(User, Movie) :- likedMovie(User, Movie).
```

(a) Cold Start

```
trained(User, Trained) :-
    trainingData(User, Features, Label),
    Trained = ml:train(User, Features, Label).

predictedRankingScore(User, Movie, RankingScore) :-
    trained(User, Trained),
    movieFeatures(User, Movie, Features),
    RankingScore = ml:predict(User, Features).
```

(b) Training and Predicting

```
modifiedScore(User, Movie, Feature, FeatureIndex, ModifiedScore) :-
    trained(User, Trained),
    movieFeatures(User, Movie, Features),
    sampleValues(Feature, FeatureIndex, Value),
    ModifiedFeatures = col:setAt(Feature, FeatureIndex, Value),
    ModifiedRankingScore = ml:predict(User, ModifiedFeatures).

minModifiedScore(User, Movie, MinModifiedScore) :-
    modifiedScore(User, Movie, Feature, FeatureIndex, ModifiedScore),
    MinModifiedScore = min(ModifiedScore).

explainedRanking(User, Movie, RankingScore, ProminentFeatures) :-
    predictedRankingScore(User, Movie, RankingScore),
    minModifiedScore(User, Movie, MinModifiedScore),
    modifiedScore(User, Movie, Feature, FeatureIndex, MinModifiedScore),
    movieFeatures(User, Movie, Features),
    ProminentFeatures = list(Feature) .
```

(c) Ranking Explanation

Fig. 2: Cold Start, Learning, Training, and Explanations.

3 Explainable Ranking

In this section, we show how Vadalog can be used to perform explainable ranking of movies based on their precomputed features. For each user we perform the following steps: training a machine learning ranking model; use the trained model to rank all movies; and, finally, perform feature value analysis to compute a tailored explanation for the ranking of each movie. We next provide more detail for each of the these steps.

Setting. Let the predicate movieFeatures(User, Movie, Features) contain all feature values computed for each user and movie. Concretely, each movie has one entry in movieFeatures, Features is the list of feature values for that movie. Furthermore, assume that we have computed a relation trainingData(User, Features, Label), which contains the training data for each user. The trainingData contains two types of records. For each user and for each movie that they liked, the relation associates the feature vector of the movie with label 1. Furthermore, for each user and each movie from a predefined set of *sample* movies, the relation associates the feature vector of the movie with label 0. Thus, labels 1 and 0 represent positive and negative examples respectively.

Training And Prediction. We use a machine learning regression model to perform ranking of movies based on their features. The regression model takes as input the feature values of a movie and produces a ranking score, which we then use to rank all movies. The Vadalog system exposes several open-source machine learning libraries, such as Weka[4], that can seamlessly be used during reasoning. We use a separate instance of the regression model for each user. To train an ML model, we invoke a dedicated aggregate function ml:train, as shown in the first rule of Figure 2b. The first argument of ml:train is the identifier of the trained model (in our case the user), the second argument is the input vector,

[4]https://www.cs.waikato.ac.nz/~ml/weka/

and the third argument is the value to be learned. Next we can use the trained model to compute a user-specific ranking of all movies. To this end we use the library function ml:predict, which takes the identifier of the trained model and the input feature vector to produce the model prediction, as shown in the second rule of Figure 2b.

Ranking Explanation. We next show how one can use Vadalog to produce explanations about the rank of each individual movie. We adapt an approach described in [3]. The goal is to identify for each movie the feature that has the highest contribution to its position in the ranked list, and consequently report the explanation of its computation to the user. To identify the most prominent feature for the rank of a given movie, we first identify a range of interesting values (e.g. minimum, maximum, average, etc.) for each feature. For each such value, we compute a modified ranking score for the given movie by replacing its original value with the selected one. The feature that gives the lowest modified ranking score is then used as an explanation for the movie's rank. Assume we have precomputed with Vadalog the interesting feature values in relation sampleValues(Feature, FeatureIndex, Value), where Feature is the feature name, FeatureIndex is the index of the feature in the movie's feature vector, and Value is a value to be used for computing the modified scores. Computing the minimal modified scores for a movie is then performed using the trained model, as shown in the first rule in Figure 2c, which makes use of the collections function col:setAt(Vector, Index, Value) which returns the result of replacing the value at position Index in Vector with Value. Finally in the last two rules, we identify for each movie the features yielding the lowest modified scores, and collect those in a list using the list aggregate.

4 Conclusion

In this application paper we reported on how a Recommender System can be rapidly developed using Vadalog. As part of ongoing work, since the obtained explainable recommender system is agnostic to the underlying machine learning model, we intend to perform an evaluation of our approach using different models. We believe our approach is useful in the scenarios when explanation of recommendation is crucial. The transparency of such a system also enables users to give feedback, incorporation of which in our model is future work.

Acknowledgements. This work is supported by the EPSRC programme grant EP/M025268/1 VADA, the WWTF grant VRG18-013, and the EU Horizon 2020 grant 809965.

References

1. L. Bellomarini, R. R. Fayzrakhmanov, G. Gottlob, A. Kravchenko, E. Laurenza, Y. Nenov, S. Reissfelder, E. Sallinger, E. Sherkhonov, and L. Wu. Data science with Vadalog: Bridging machine learning and reasoning. In *Proc. MEDI*, 2018.
2. L. Bellomarini, E. Sallinger, and G. Gottlob. The Vadalog system: Datalog-based reasoning for knowledge graphs. *PVLDB*, 11(9):975–987, 2018.
3. M. ter Hoeve, A. Schuth, D. Odijk, and M. de Rijke. Faithfully explaining rankings in a news recommender system. *CoRR*, abs/1805.05447, 2018.