Short paper

RETRO: Relation Retrofitting For In-Database Machine Learning on Textual Data

Michael Günther
Database Systems Group,
Technische Universität Dresden
Dresden, Germany
Michael.Guenther@tu-dresden.de

Maik Thiele
Database Systems Group,
Technische Universität Dresden
Dresden, Germany
Maik.Thiele@tu-dresden.de

Wolfgang Lehner
Database Systems Group,
Technische Universität Dresden
Dresden, Germany
Wolfgang.Lehner@tu-dresden.de

ABSTRACT

There are massive amounts of textual data residing in databases, valuable for many machine learning (ML) tasks. Since ML techniques depend on numerical input representations, word embeddings are increasingly utilized to convert symbolic representations such as text into meaningful numbers. However, a naïve one-to-one mapping of each word in a database to a word embedding vector is not sufficient since it would miss to incorporate rich context information given by the database schema, e.g. which words appear in the same column or are related to each other. Additionally, many text values in databases are very specific and would not have any counterpart within the word embedding. In this paper, we therefore, propose RETRO (RElational reTROfitting), a novel approach to learn numerical representations of text values in databases, capturing the information encoded by general-purpose word embeddings and the database-specific information encoded by the tabular relations. We formulate *relation retrofitting* as a learning problem and present an efficient algorithm solving it. We investigate the impact of various hyperparameters on the learning problem. Our evaluation shows that embedding generated for database text values using RETRO are ready-to-use for many ML tasks and even outperform state-of-the-art techniques.

1 INTRODUCTION

Due too their appealing properties, word embeddings techniques such as Word2Vec [7], GloVe [8] or fastText [3] have become conventional wisdom in many research fields such as machine learning, NLP or information retrieval. Typically, these embeddings are used to convert text values in a meaningful numerical representations presenting the input for ML tasks. However, a naïve application of a word embedding model is not sufficient to represent the meaning of text values in a database which is often more specific than the general semantic encoded in the raw text embedding.

Thus, we argue to incorporate information given by the disposition of the text values in the database schema into the embedding, e.g. which words appear in the same column or are related. Therefore, we develop a relational retrofitting approach called RETRO which is able to automatically derive high-quality numerical representations of textual data residing in databases without any manual effort.

Relational Retrofitting. Figure 1 provides a small example sketching the main principles of the relational retrofitting process. RETRO expects a database and a word embedding as input, e.g. a movie table T that should be retrofitted into a pre-trained

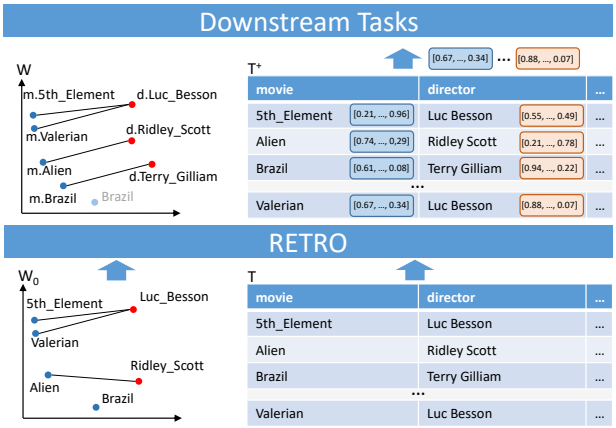


Figure 1: Relational Retrofitting: basis word embedding W^0 and relation T (left), retrofitted word embedding W and augmented relation T^+

word embedding W_0. To provide vector representations for textual information in databases one could simply reuse the vectors of pre-trained embeddings, e.g. map each term from T to a term-vector pair in W_0. However, often there will be a semantic mismatch between word embeddings and textual information in databases:

1) Given the movie table, it is known that all entities within the movie column must be movies, although some of the titles, such as “Brazil" or “Alien", may be interpreted differently by the word embedding model.

2) T provides a specific amount of relation types like the movie-director, whereas in the word embedding representation W_0 large amounts of implicit relations are modeled, e.g. if the director of a movie is also the producer or one of the actors this might be represented in the word embedding although not explicitly visible.

3) Terms in T which occurring infrequent in the general domain can not be modeled accurately by word embedding models. For instance Word2Vec has a limited vocabulary according to a frequency threshold. Many terms appearing in a database will therefore have no counter-part within the embedding.

We present RETRO[1], a novel relational retrofitting approach addressing all these challenges. RETRO augments all terms in database tables by dense vector representations encoding the semantics given by the relation T and the word embedding W_0. In the context of our movie example, RETRO would generate a new embedding for “Terry Gilliam" which should be close to other directors and their respective vectors. Furthermore, RETRO would create vectors for all other textual values in the movie table that encode the semantic given of the pre-trained word embeddings and the database. On the one hand, this ensures that vectors

[1] https://github.com/guenthermi/postgres-retrofit

appearing in the same column, such as movies or directors, are close to each other. On the other hand, this ensures that the difference vectors between movie-director pairs are similar. These vectors are ready-to-use for a wide range of ML, retrieval and data cleaning tasks such as classification, regression, null value imputation, entity resolution and many more.

Outline. In Section 2, we give an overview of the problem and a briefly introduce the original retrofitting problem. We then present our novel *relation retrofitting* and formulate the underlying learning problem in Section 3. In Section 4, we show the feasibility of Retro in automatically creating vector representations by defining different classification task and conclude in Section 5.

2 RETROFITTING AND PROBLEM SCOPE

We aim at leveraging powerful word embedding models to generate good vector representations for text values residing within relational databases. We therefore extend the notion of retrofitting which was initially proposed by Faruqui et al. [5]. Retrofitting is performed as a post-processing step and allows to inject additional information into word embeddings. The approach of Faruqui et al. took a matrix $W^0 = (\boldsymbol{v'_1}, \ldots, \boldsymbol{v'_n})$ of word embeddings and a graph $G = (Q, E_F)$ representing a lexicon as input. The retrofitting problem was formulated as a dual objective optimization function: The embeddings of the matrix W^0 are adapted to $W = (\boldsymbol{v_1}, \ldots, \boldsymbol{v_n})$ by placing similar words connected in the graph G closely together, while at the same time the neighborhood of the words from the original matrix W^0 should be preserved. Hereby, $Q = \{q_1, \ldots, q_n\}$ is a set of nodes where each node q_i corresponds to a word vector $\boldsymbol{v_i} \in W$ and $E_F \subset \{(i,j)|i,j \in \{1,\ldots,n\}\}$ is a set of edges. The graph is undirected, thus $(i,j) \in E_F \Leftrightarrow (j,i) \in E_F$. The authors specified the retrofitting problem as a minimization problem of the following loss function:

$$\Psi_F(W) = \sum_{i=1}^{n} \left[\alpha_i ||\boldsymbol{v_i} - \boldsymbol{v'_i}||^2 + \sum_{j:(i,j)\in E_F} \beta_i ||\boldsymbol{v_i} - \boldsymbol{v_j}||^2 \right] \quad (1)$$

The constants α_i and β_i are hyperparameters. $\Psi_F(W)$ is convex for positive values of α_i and β_i. Thus, the optimization problem can be solved by an algorithm, which iterates over every node in Q and updates the respective vector in W.

However, while retrofitting is typically used to improve the vector quality of general-purpose word embeddings by using lexical knowledge graphs, we aim at learning vector representations for text entries in database tables. Here the objective is to 1) reflect the semantics of the text value specifically referred to in the database and 2) to fit into the vector space of the given basis word embedding model.

3 RELATIONAL RETROFITTING

In this paper, we extend idea proposed in [5] and formulate the *relational retrofitting* approach that learns a matrix of vector representations $W = (\boldsymbol{v_1}, \ldots \boldsymbol{v_n})$ corresponding to text values $T = (\boldsymbol{t_1}, \ldots \boldsymbol{t_n})$ where each $\boldsymbol{v_i} \in \mathbb{R}^D$ represents a unique text value in a specific column of the database. To find an initial vector representation for every text value, we tokenize the text values based on the vocabulary of the basis word embedding model and build centroid vectors which is a convenient way to obtain a representation of text values consisting of multiple tokens [1, 11]. These vectors are stored in a matrix $W^0 = (\boldsymbol{v'_1}, \ldots \boldsymbol{v'_n})$ forming the basis for the retrofitting process. Besides, *columnar and relational connections* are extracted from the database (see Section 3.1). This encompasses semantic relations between text values, which are derived from the relational schema. Those connections are used to create a representation capturing the context of the text value in the database (e.g. "Brazil" in the column "movie.title" is considered as a movie) and thus helps to preserve their semantics more accurately compared to a plain word embedding representation. The core procedure of the relational retrofitting is the adaption of the basis vectors W^0. This is performed by solving an optimization problem detailed further in Section 3.2.

3.1 Extracting Relational Information

One can derive different structural relations from the alignment of text values in the relational schema.

Columnar Connections: Text values with the same attribute, i.e. appearing in the same column, usually form hyponyms of a common hypernym (similar to subclass superclass relations). Thus, they share a lot of common properties which typically leads to similarity. We capture this information and assign each text value t_i to its column $C(i)$.

Relational Connections: Relations exhibit from the co-occurrence of text values in the same row as well as from foreign key relations. Those relations are important to characterize the semantics of text value in the database. We define a set of relation types R for each specific pair of related text value columns. Those columns are related because they are either part of the same table or there exists a foreign key relationship between their tables. For every relation type $r \in R$ there is a set E_r containing the tuples of related text value ids. Relation types are directed. Accordingly, there is an inverted counterpart $\bar{r}$ for each relation r with $E_{\bar{r}} = \{(j,i)|(i,j) \in E_r\}$.

3.2 Optimization Problem

Retro considers relational and columnar connections (see Section 3.1) to retrofit an initial embedding. Accordingly, we define a loss function Ψ adapting embeddings to be similar to their basis word embedding representation W^0, the embeddings appearing in the same column, and related embeddings.

$$\Psi(W) = \sum_{i=1}^{n} \left[\alpha_i ||\boldsymbol{v_i} - \boldsymbol{v'_i}||^2 + \beta_i \Psi_C(\boldsymbol{v_i}, W) + \Psi_R(\boldsymbol{v_i}, W) \right] \quad (2)$$

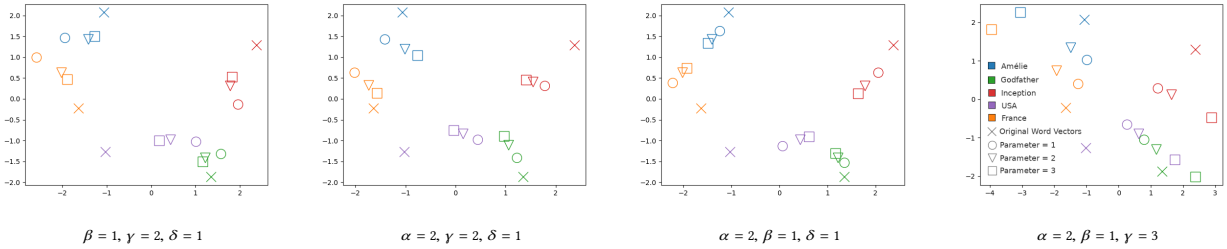
The columnal loss is defined by Ψ_C and treats every embedding $\boldsymbol{v_i}$ to be similar to the constant centroid $\boldsymbol{c_i}$ of the basis embeddings of text values in the same column $C(i)$.

$$\Psi_C(\boldsymbol{v_i}, W) = ||\boldsymbol{v_i} - \boldsymbol{c_i}||^2 \quad \boldsymbol{c_i} = \frac{\sum_{j \in C(i)} \boldsymbol{v'_j}}{|C(i)|} \quad (3)$$

The relational loss Ψ_R treats embeddings v_i and v_j to be similar if there exists a relation between them and dissimilar otherwise. E_r is the set of tuples where a relation $r \in R$ exists. $\widetilde{E_r}$ is the set of all tuples $(i,j) \notin E_r$ where i and j are part of relation r. Thus, each of both indices has to occur at least in one tuple of E_r.

$$\Psi_R(\boldsymbol{v_i}, W) = \sum_{r \in R} \left[\sum_{\substack{j:(i,j) \\ \in E_r}} \gamma_i^r ||\boldsymbol{v_i} - \boldsymbol{v_j}||^2 - \sum_{\substack{k:(i,k) \\ \in \widetilde{E_r}}} \delta_i^r ||\boldsymbol{v_i} - \boldsymbol{v_k}||^2 \right] \quad (4)$$

α_i, β_i, γ_i and δ_i are hyperparameters. Ψ should be a convex function. In [6] we proved the convexity of Ψ for hyperparameter



$\beta = 1, \gamma = 2, \delta = 1$ $\alpha = 2, \gamma = 2, \delta = 1$ $\alpha = 2, \beta = 1, \delta = 1$ $\alpha = 2, \beta = 1, \gamma = 3$

(a) Influence of $\alpha = 1, 2, 3$ **(b) Influence of $\beta = 1, 2, 3$** **(c) Influence of $\gamma = 1, 2, 3$** **(d) Influence of $\delta = 0, 1, 2$**

Figure 2: Examples for Different Hyperparameter Settings

configurations fulfilling the following inequation:

$$\forall r \in R, i \in \{1, \ldots, n\} \quad (\alpha_i \geq 0, \quad \beta_i \geq 0, \quad \gamma_i^r \geq 0) \qquad (5)$$
$$\forall \boldsymbol{v_i} \in W \quad (4\alpha_i - \sum_{r \in R} \sum_{j:(i,j) \in \widetilde{E_r}} \delta_i^r \geq 0)$$

In practice, however, other parameter configurations that do not comply might work as well. The impact of the hyperparameter values on the retrofitting result is shown in Section 4.1. The retrofitting algorithm iteratively executes for all $\boldsymbol{v_i} \in V$ the following equation, which is derived from the root of the partial derivative $\frac{\partial \Psi(W)}{\partial \boldsymbol{v_i}}$.

$$\boldsymbol{v_i} = \frac{\alpha_i \boldsymbol{v_i'} + \beta_i \boldsymbol{c_i} + \sum_{r \in R} \left[\sum_{\substack{j:(i,j) \\ \in E_r}} (\gamma_i^r + \gamma_j^{\bar{r}}) \boldsymbol{v_j} - \sum_{\substack{k:(i,k) \\ \in \widetilde{E_r}}} (\delta_i^r + \delta_k^{\bar{r}}) \boldsymbol{v_k} \right]}{\alpha_i + \beta_i + \sum_{r \in R} \left[\sum_{\substack{j:(i,j) \\ \in E_r}} (\gamma_i^r + \gamma_j^{\bar{r}}) - \sum_{\substack{k:(i,k) \\ \in \widetilde{E_r}}} (\delta_i^r + \delta_k^{\bar{r}}) \right]} \qquad (6)$$

Given the property of convexity, such an iterative algorithm can be used to minimize Ψ illustrated in more details in the next section.

3.3 Retrofitting Algorithm

The retrofitting algorithm can be expressed as a set of matrix operations that can be solved with linear time complexity according to the number of text values in W. We update all vectors at once using a recursive matrix equation. $\Psi(W)$ can be minimized by iteratively calculating W^k according to (7).

$$W_R = \sum_{r \in R} \left[((\gamma_{ij}^r) + (\gamma_{ij}^{\bar{r}})^T) - ((\delta_{ij}^r) + (\delta_{ij}^{\bar{r}})^T) \right] W^k$$
$$W' = \alpha W^0 + \beta \boldsymbol{c} + W_R$$
$$D = diag\left(\alpha + \beta + \sum_{r \in R} \left[\sum_{\substack{j:(i,j) \\ \in E_r}} (\gamma_i^r + \gamma_j^{\bar{r}}) - \sum_{\substack{k:(i,k) \\ \in \widetilde{E_r}}} (\delta_i^r + \delta_k^{\bar{r}}) \right] \right)$$
$$W^{k+1} = D^{-1} W'$$
$$\boldsymbol{c} = (\boldsymbol{c_1}, \ldots, \boldsymbol{c_n}) \quad \alpha = (\alpha_1, \ldots, \alpha_n) \quad \beta = (\beta_1, \ldots, \beta_n) \qquad (7)$$

More details are outlined in [6].

4 EVALUATION

Retro is a fully functional system built on top of PostgreSQL. Given an initial configuration including the connection information for a database and the hyperparameter configuration, Retro fully automatically learns the retrofitted embeddings and adds them to the given database. We created two databases based on the Movie Database[2] (TMDB) and the Google Play Store Apps dataset[3] (GPSA). TMDB consists of 15 tables containing 493,751 unique text values, whereas the GPSA database has 7 tables and 27,571 unique text values (details are outlined here[4][5]). Both of them are available as CSV files and are imported in our Retro PostgreSQL database system.

One baseline we compare our retrofitted embeddings to, are plain word vectors (PV) that have no notion of the relational schema. The counterpart to this would be embeddings that just rely on the the structural information given by the database. Here we use the node embedding technique DeepWalk [9] (DW) that is learned based on a graph representation of the database relations. Moreover, we applied the original retrofitting approach [5] leading to another baseline embedding dataset (MF).

4.1 Hyperparameter Analysis

The influence of the hyperparameters is visualized in Figure 2: We learned 2-dimensional embeddings for a small example dataset containing three movies and the country where those movies have been produced. Accordingly, there are two columnar (movie and country) and one relational connection (see Section 3.1). "Amélie" was produced in "France", the other movies in the "USA". Usually the hyperparameters for each vector are derived from four global hyperparameters α, β, γ, and δ as detailed in [6]. We set the hyperparameters α, β, γ, and δ to different values and performed the relational retrofitting.

As shown in Figure 2a, the learned embeddings stay closer to their original embeddings when the α values increasing. Higher values of β make it easier to cluster the categories from each other, e.g. reduce the distances between the movie vectors of "Inception" (red), "Godfather" (green), and "Amélie" (blue). The γ value controls the influence of relational connections. This brings the representations of text values which share a relation closer together. The δ factor causes vectors with different relations to separate and thus prevent concentrated hubs of vectors with different semantic. One can see in Figure 2d how $\delta = 0$ causes all vectors to concentrate around the origin of the coordinate system. If δ is set to a high value like $\delta = \alpha = 2$, the algorithm places the vectors far from the origin of the coordinate system. However, related text values still get assigned to similar representations. In the example, the retrofitting algorithm is still converging for this configuration. Our analysis shows, that the exposed hyperparameters allow to steer the relational retrofitting

[2] https://www.kaggle.com/rounakbanik/the-movies-dataset
[3] https://www.kaggle.com/lava18/google-play-store-apps/
[4] https://github.com/guenthermi/the-movie-database-import
[5] https://github.com/guenthermi/google-play-dataset-import

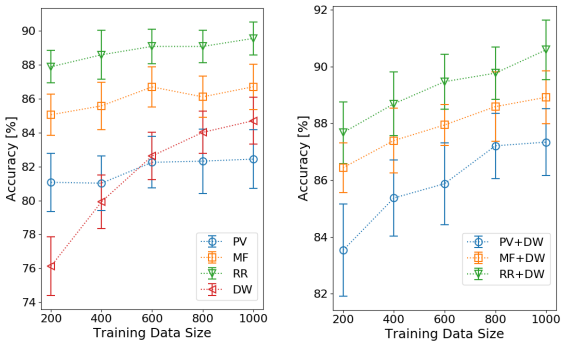


Figure 3: Classification of Birth Places of US-American Directors with Increasing Sample Size

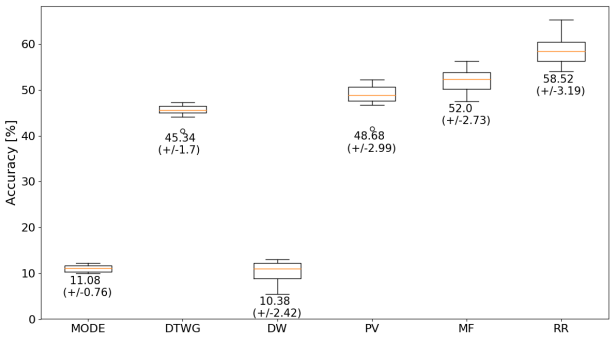


Figure 4: Imputation of Categories of Android Apps

process into different directions in a fine-grained manner, i.e. to adapt to different downstream tasks.

4.2 Machine Learning Tasks

Binary Classification. We implemented a binary classifier to label a set of directors of the TMDB dataset according to there citizenship. The classifier should decide between US-American and non-US-American directors. Since this information is not available from the TMDB dataset, we extract the citizenship from Wikidata [10] by using the SPARQL query service. We trained a feed-forward neural network (one hidden layer with 600 neurons; applying dropout and L2 regularization; Nadam optimizer [4]) on the different 300-dimensional embedding representations of the director names (full names). We used for the training 200 to 1,000 samples and validate the accuracy with 1,000 test samples. We compared the accuracies achieved when using plain word embeddings (PV), node embeddings (DW), simple retrofitted (MF) and relational retrofitted embeddings (RR). We ran the training and testing on the ANNs 20 times for each configuration with different sample sets.

The accuracy values and their standard deviation achieved by the classifiers are shown on the left in Figure 3. The best results are achieved with our relational retrofitting approach (RR) utilizing word embedding features of the directors name but indirectly also word embedding features of related text values like the movie titles directed by them. The influence of the training sample size is at lowest for the plain word embeddings (PV). DeepWalk (DW) needs a larger amount of training data to achieve comparable results. The right side of Figure 3 shows the accuracies achieved by running the same experiment but combining the previous embeddings with node embeddings by concatination. This leads to better results for all methods. Notably, the accuracies of the retrofitting methods are much better compared to methods where node embeddings (DW) and plain word embeddings (PV) are just concatenated.

Missing Value Imputation. Further, we built classifiers to predict app categories within GPSA database which can be used to impute missing values. Here,a feed-forward neural network (two hidden layer with 600 and 300 neurons; applying dropout and L2 regularization; Nadam optimizer [4]) is applied on the embeddings of the application names. The network was trained 10 times on 400 random samples to predict the one out of 33 categories. The category information and the genre information (which is often redundant) are omitted for the retrofitting. We trained the network on all embedding types and compared it to MODE imputation, choosing always the most frequent category in the training data, and Datawig [2]. Figure 4 shows that best accuracy is achieved by relational retrofitting (RR).

5 CONCLUSION

In this paper, we presented Retro, a system that augments all terms in database tables by dense vector representations. Therefore, we employed the notion of retrofitting to modify word embedding representations to specialize for given relational schemas. We validated Retro experimentally by building standard feed-forward neural networks for different classification tasks. Our evaluation showed that the generated relational embeddings are ready-to-use for different ML tasks and even outperform state-of-the-art techniques such as the approach of Faruqui et al. or DeepWalk [9].

ACKNOWLEDGMENTS

This work is funded by the German Research Foundation (DFG) within the Research Training Group “Role-based Software Infrastructures for continuous-context-sensitive Systems” (GRK 1907) and by Intel® AI Research.

REFERENCES

[1] Abdulaziz Alghunaim, Mitra Mohtarami, Scott Cyphers, and Jim Glass. 2015. A Vector Space Approach for Aspect Based Sentiment Analysis. In *Proc. of NAACL-HLT*. 116–122.

[2] Felix Biessmann, David Salinas, Sebastian Schelter, Philipp Schmidt, and Dustin Lange. 2018. Deep Learning for Missing Value Imputation in Tables with Non-Numerical Data. In *Proc. of the CIKM 2018*. ACM, 2017–2025.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *TACL* 5 (2017), 135–146.

[4] Timothy Dozat. 2016. Incorporating Nesterov Momentum into Adam. In *ICLR 2016 Workshop*.

[5] Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. 2015. Retrofitting Word Vectors to Semantic Lexicons. In *Proc. of NAACL-HLT 2015*. 1606–1615.

[6] Michael Günther, Maik Thiele, and Wolfgang Lehner. 2019. RETRO: Relation Retrofitting For In-Database Machine Learning on Textual Data. (2019). arXiv:1911.12674

[7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS 2013*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3111–3119.

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *EMNLP*. 1532–1543.

[9] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online Learning of Social Representations. In *Proc. of the 20th ACM SIGKDD*. ACM, 701–710.

[10] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A Free Collaborative Knowledge Base. *Commun. ACM* 57, 10 (2014), 78–85.

[11] Xinjie Zhou, Xiaojun Wan, and Jianguo Xiao. 2015. Representation Learning for Aspect Category Detection in Online Reviews. In *29th AAAI Conference on Artificial Intelligence*.