Annals of Telecommunications (2020) 75:549–561
https://doi.org/10.1007/s12243-020-00800-4

Resource management for model learning at entity level

Christian Beyer[1] · Vishnu Unnikrishnan[1] · Robert Brüggemann[1] · Vincent Toulouse[1] · Hafez Kader Omar[1] · Eirini Ntoutsi[2] · Myra Spiliopoulou[1]

Received: 7 November 2019 / Accepted: 13 August 2020 / Published online: 29 August 2020

Abstract

Many current and future applications plan to provide entity-specific predictions. These range from individualized healthcare applications to user-specific purchase recommendations. In our previous stream-based work on Amazon review data, we could show that error-weighted ensembles that combine entity-centric classifiers, which are only trained on reviews of one particular product (entity), and entity-ignorant classifiers, which are trained on all reviews irrespective of the product, can improve prediction quality. This came at the cost of storing multiple entity-centric models in primary memory, many of which would never be used again as their entities would not receive future instances in the stream. To overcome this drawback and make entity-centric learning viable in these scenarios, we investigated two different methods of reducing the primary memory requirement of our entity-centric approach. Our first method uses the lossy counting algorithm for data streams to identify entities whose instances make up a certain percentage of the total data stream within an error-margin. We then store all models which do not fulfil this requirement in secondary memory, from which they can be retrieved in case future instances belonging to them should arrive later in the stream. The second method replaces entity-centric models with a much more naive model which only stores the past labels and predicts the majority label seen so far. We applied our methods on the previously used Amazon data sets which contained up to 1.4M reviews and added two subsets of the Yelp data set which contain up to 4.2M reviews. Both methods were successful in reducing the primary memory requirements while still outperforming an entity-ignorant model.

Keywords Entity-centric learning · Stream classification · Document prediction · Memory reduction · Text ignorant models

1 Introduction

Recent developments in hardware have made it easier to consider going beyond traditional machine learning methods, and challenge the concept of "more data is better." Especially in the case of data streams and time series, it can be the case that the streaming data is generated by some identifiable "entity." While a machine learning model trained on the entire stream may generalize and perform well, there are cases where learning the idiosyncratic properties of the exact entity that generated a data point would help the model to make better predictions. This would be particularly true in fields like healthcare, where each patient needs predictions and recommendations tailored to their exact case. Entity-specific learning may also bring additional benefit to social network analysis, IoT, and other data sets where sub-groups of entities may behave in ways that may not match the tendencies of the global stream. Entity-centric data mining is a new area of research when it comes to data streams. Depending on the domain, it can be very important to take the entity-instance relationship into account. A few entities with many instances can otherwise dominate the learned model. While there are entity-centric approaches in the time series mining and the data panel community, such approaches are usually not easily transferable to a stream mining setting where we have to deal with a potentially unlimited amount of instances and entities.

Our previous work [5] investigated the value of the one-model-per-entity paradigm in a rather simplistic scenario, where only the label of an instance arriving on a stream is available for learning and all its features are ignored. Surprisingly, even this limited information was in some cases sufficient for competitive predictions on the Amazon

The first-author position is shared between the first two authors Christian Beyer and Vishnu Unnikrishnan.

✉ Christian Beyer
christian.beyer@ovgu.de

Extended author information available on the last page of the article.

data set [10], compared to methods that used the review text (features) for prediction. We further followed-up our work with [6], where entity-centric classifiers now had access to the review texts. The entity-centric classifiers were combined with an entity-ignorant classifier that saw data from all entities to form a two-classifier ensemble that labels every incoming review. It was found that the ensemble improves the predictive performance when the ensemble votes are weighted on their previous errors over predictions for that entity.

This work improves on one of the main shortcomings of the entity-centric modeling paradigm, which are the increasing memory requirements (see Fig. 1). It was observed in [5] that the variation in entity lengths is extreme, which means we have a lot of very short entities. As a consequence, the vast majority of entity-centric classifiers that are trained in [6] are used to make very few predictions, providing almost no performance gain while adding to the memory overhead of maintaining the classifier in memory. In this work, we combine the ensembles from [6] with a lossy counting [14] method that keeps in primary memory only those classifiers of entities that appear frequently in the stream. The lossy counting algorithm is periodically invoked to ensure that entities which appear infrequently are removed from primary memory, and their models are stored in secondary memory. If a review appears for such an entity that has been offloaded to secondary memory, the entity-specific model for that review is simply loaded from secondary storage into RAM and is subsequently used for training/to make predictions. In our second approach, we replace the entity-centric models with much simpler models that only rely on the labels of seen instances and have a much smaller memory footprint. The paper is structured as follows: Section 2 introduces related work, followed by our methods in Section 3. Section 4 details our experiments, with results discussed in Section 5, and closing remarks with possible areas for improvement in Section 6.

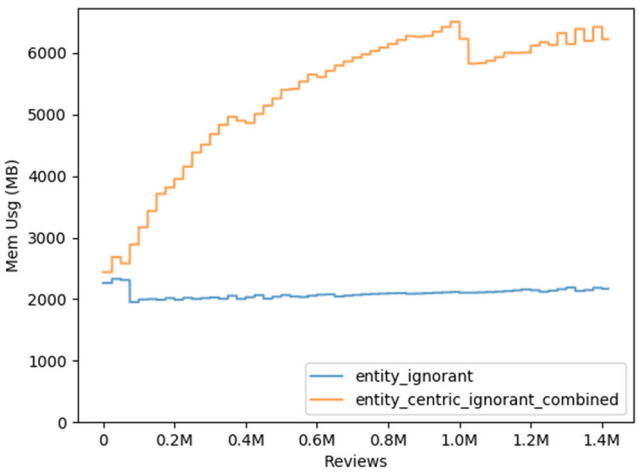

Fig. 1 We compare the memory requirements of an entity-ignorant model against a model that combines entity-ignorant with entity-centric predictions, without memory management. The figure shows an almost constant memory use of the entity-ignorant model at around 2GB, whereas the combined model rises sharply up to almost 6.5GB

2 Related work

This section briefly introduces some relevant literature. We begin with some of our previous work related to entity-centric learning, then discuss entity-centric learning approaches from different fields and further describe how frequent items in a stream can be tracked with a significant reduction in memory requirements. As we have mentioned before, entity-centric learning is a new research area in data streams which is why we have also included some related literature from the time series and data panel community.

2.1 Learning at entity level

In our first work [5], we investigated to what extent entity-level information can inform predictors. The first approach investigated the case where the predictors were only informed by the entity-ID, and no other information apart from the timestamp and label of the arriving review is provided to the ("text-ignorant") predictor. Several windowed and non-windowed predictors based on simple moving average, hidden Markov models, regression, and simply the prior likelihood of a given label were investigated. Additionally, [5] proposes a new entity-centric evaluation framework, where the $\kappa+$ measure is adapted to work at an entity level. The $kappa+$ statistic [8] measures how much better a classifier performs compared to a random classifier and a classifier that just propagates the last seen label in the stream or in our case last seen label of a particular entity. It was found that while no method could beat the entity-ignorant baseline, which had access to the text/features of a review, there were cases where even the text-ignorant entity-centric predictors outperformed the entity-ignorant model as measured by the $\kappa+$.

The work in [5] was improved upon in [6] by using entity-centric predictors which now also had access to the review text. Each entity-centric predictor was allowed to access only one entity's "sub-stream" of reviews, while an entity-ignorant classifier is trained on all arriving reviews regardless of the entity. Since each arriving review can be predicted by two classifiers, one entity-ignorant classifier and another entity-centric classifier, various methods were investigated to combine the predictions as ensembles of two voting members. The votes of each of the classifiers were either averaged or averaged weighted on their respective error rates over predictions for that particular entity. It was found that the error rate–weighted predictions of the two

ensemble members gave better results than using either classifier on their own. It is noted, however, that the entity-centric modeling paradigm is very inefficient in the use of primary memory. Over the 33,000 entities used in the study, the entity-centric models needed more than 6GB of RAM during execution; a figure that growths with an increase in the number of encountered entities. The entity-ignorant model on the other hand only needs around 2GB and stays constant (see Fig. 1). In our experiments, we vectorize all review texts by using a bag-of-words model with the 10,000 most popular words over all reviews in a data set. The features of a review will be the word counts of these 10,000 most popular terms. We have five different classes a review (instance) can belong to which are the star ratings from one to five.

Further work in [18] also considers entity-centric learning but focuses on the problem of most entities having too little information available for learning reliable models. This issue of most entities being too short is handled by augmenting the data of the current entity with the observations of other entities in its neighborhood. The neighborhood of an entity is computed over the static properties of an entity (product category/patient blood type, etc.), while the predictors use the combined information from the entity as well as its neighborhood to learn a predictive model. Various methods for augmenting entity data are considered, and the data augmentation method (pooling all data to train one model) preserving timestamp info was found to be better across data sets from three domains; eCommerce, meteorology, and mHealth. It is also noted that expert knowledge can be used to tune entity neighborhoods and that removing an entity that is deemed similar according to static variables but is known to not be predictive of entity trajectory does improve performance in some cases. Recently a new method for calculating the k-nearest neighbors in a data stream was published [3]. The authors use a new dimensionality reduction approach (UMAP) and a batch incremental approach in their work which enables the efficient calculation of neighborhoods in a data stream.

Melidis et al. [15] have also considered a tangential view of entity-centric ensembles, where each word in a text stream is considered to be an entity in its own right. In this work, an ensemble of predictors is built for each word to capture changes in word frequency and sentiment while maintaining a sketch that guarantees memory-bounds over the feature space computation. In addition to drifts of class priors, the likelihood of documents d given class c, and changes in posterior probability, this work also considers the case of *feature drifts* [4], where the relevance of features for the learning task changes over time. By considering words in the text stream as an entity, [15] shows that both changes in the feature space and changes in the relative contributions of a word to the final classification can be handled. In order to predict the values of each feature/word at the next time point, a Poisson model and its seasonal variant, an ARIMA model, and an exponential weighted moving average model are used. The predictions of each of these components are shown to "average out" potentially competing trends at different time scales. Spitz et al. [1, 2] consider a related view on entities, where they use named entities in documents to facilitate the exploration of news articles arriving in a stream. The authors suggest the creation of an entity-centric network which is a graph structure that links named entities (locations, organizations, actors, and dates) with the sentences, documents, and dates they appear in. They also create edges between entities that appear in the same document and weigh the edge with the number of sentences that separate the entities. The graph structure can then be used for various entity-centric information retrieval purposes, for instance, entity-centric topic exploration which was presented in [1].

Other work by Liu and Hauskrecht [12] investigates the effect of modeling each entity separately in the context of physiological data from patients. They first develop an entity-ignorant model where data from all patients are combined to create a linear dynamical system that captures the general trend in the population. As a next step, to capture individual variability, each patient's time series is converted to a multivariate residual time series, where each observation is the deviation from the predictions of the global model. A multi-task Gaussian process model is trained on this residual time series to create predictions personalized to a patient.

2.2 Error-weighted predictions and clustering entities

A very similar approach to our error-corrected ensembles comes from the time series community and was also published in 2019 [16]. The authors trained an entity-ignorant Gaussian process on the whole population in their data and combined it with entity-centric Gaussian processes trained on singular subjects' time series. The authors made the same observation as us that the individual models deliver subpar results but that combining the models can improve prediction quality. In contrast to us, they learn a regression model to combine the entity-centric and entity-ignorant models. Another related work comes from the data panel community where the authors also used a performance-based weighting scheme [9] but used AUC instead of RMSE. The authors build an ensemble of performance-weighted models based on AUC but not for different entities but for different time steps, e.g., they trained a model on the financial data of the year 1998 and used its performance on the data of the year 1999 to determine its weight when calculating its contribution for a prediction

task for the year 2001. In [17], the authors use dynamic ensembles of classifiers and create a weighted prediction using an error estimate like RMSE in a data stream but they do not incorporate the entity-instance relationship in their approach. Another ensemble-based approach using error-based weighting was presented in [11]. Here, the authors used a weighted ensemble based on mean squared error for dealing with different types of concept drift but they do not deal with entities.

Clustering similar entities in a data panel to train models for similar entities was done by Lu et al. in [13] and is an approach that follows our intuition that a one-model-fits-all approach might be suboptimal when we face instances from varying entities. The authors of [7] present an algorithm for the clustering of evolving sub-streams which are similar and which can be transferred to our case where we consider each entity's instances a sub-stream to the stream of all incoming instances.

2.3 Reducing memory footprint with lossy counting

Lossy counting was introduced by Manku et al. [14] in order to reduce the memory footprint of tracking frequent itemsets in a data stream. The algorithm can reduce the number of itemsets for which we have to keep counts while guaranteeing that the frequency estimate of any itemset will never be below a user-specified margin of error. The lossy counting algorithm uses a data structure D that keeps the counts of itemsets it has recently seen and then uses rules to determine if the counts of an itemset are sufficient to stay in D or if the statistics should be deleted from D. The user specifies the required minimum support s, and the allowed error ϵ. The algorithm guarantees that the memory requirement will be at most $\frac{1}{\epsilon} log(\epsilon M)$ with M being the current length of the stream. This is achieved by dividing the streams into buckets with size $w = \left\lceil \frac{1}{\epsilon} \right\rceil$ and each bucket gets an ID starting at 1. The ID of the current bucket is called $b_{current}$. Each entry in D consists of $\langle e, f, \Delta \rangle$ where e is an element in the data stream, f is the frequency estimate of e, and Δ is the maximum possible error in f. When an element e arrives, one first check if it is already in D. If that is the case, then f is incremented; otherwise, we create a new entry in D with $\langle e, 1, b_{current} - 1 \rangle$. Once a bucket is full, we delete all entries in D which fulfill the following condition $f + \Delta \leq b_{current}$.

3 Methods

In this section, we explain how the entity-centric learning with ensembles was realized in [6] as well as the methods for reducing the primary memory requirements. Section 3.1 is mostly taken over from [6] as the entity-centric learning part of our process stays the same (text that is the same is in blue for review purposes).

3.1 Entity-centric learning

As in conventional opinion stream classification, our learning task is to predict the label of each arriving review. To exploit the fact that some reviews refer to the same entity, we partition the stream into one sub-stream per entity, as explained in Section 3.1.1. In Section 3.1.2, we describe how model learning, adaption, and forgetting are done by the entity-centric learning algorithm and by the entity ignorant one, which cooperate in an ensemble. We then present our weighting schemes for the ensemble members in Section 3.1.5.

3.1.1 Entity-centric modeling of the stream

We model the data set DS as a stream of incoming reviews, where each review belongs to a specific product. From here on, we use the more general terms "entity" instead of product and "observation" or "instance" instead of a review. We denote as $t_1, t_2, \ldots, t_m, \ldots$ the time points of the arrivals of the observations, so that o_m stands for the observation which has arrived at time point t_m.

We denote the set of all entities as E, and the jth observation belonging to entity $e \in E$ as $obs_{e,j} \in DS$. This implies that all observations belonging to $e \in DS$ constitute a sub-stream T_e. Since the first observation for an entity may arrive at any time point t_m, and since the popularity of the entities varies, the sub-streams have different speeds, and the jth observation for entity e may arrive much later than the jth observation for entity e'.

Each observation consists of a text field (the review content) and the sentiment label from a set of labels $\mathcal{L}$. For example, the number of stars assigned to a review, or the set {pos, neg, neutral}.

Given the infinite stream DS of observations, the learning task is to build a model, which at each time point t_m receives an observation o_m belonging to entity $e \in E$ and predicts the label of this observation, given all reviews seen thus far for e and for all other entities.

3.1.2 An ensemble with two voting members

Our proposed ensemble has two voting members: a conventional stream classifier that treats all observations as independent, and the set of single-entity classifiers (SECs), one per entity. We explain the SECs first and describe the orchestration of the ensemble thereafter.

3.1.3 The entity-centric ensemble member

For each entity $e \in E$, we train and gradually adapt a single-entity classifier SEC_e. This classifier sees only the sub-stream of observations $T_e = \{obs_{e,1}, obs_{e,2}, \ldots\}$. Since the set of entities E over the stream DS is not known in advance, we perform a single initialization step for the whole DS. Then, whenever a new entity e shows up, we launch a new SEC_e. A SEC_e is invoked for classification and adaption only if an observation on e arrives. In the initialization step, we build a single feature space F of size N over DS, selecting the top-N words (for a very large N). When the first observation of an entity e, $obs_{e,1}$ appears, a new SEC_e is created and trained. In [6], we kept all SECs in primary memory (see Fig. 1), whereas in this work, we are using two different memory management strategies which are explained in Section 3.2.

As learning core for each SEC, we consider a Multinomial Naive Bayes with "gradual fading" (MNBF), proposed in [19]: this algorithm decays the word counts per class, depending on how long it has been since a word has been encountered for a given class. For SEC_e, we perform gradual fading on the sub-stream T_e, wherein the count of a word per class is decayed proportionately to the last time point at which the word appeared in observation of e for this class. Since for each entity e, SEC_e is trained only within the sub-stream T_e, the conditional word counts per class, faded to different extents, differ among entities.

3.1.4 The entity-ignorant ensemble member

The second member of our ensemble is a conventional stream classifier that ignores the entity to which each observation belongs. We denote this classifier as "Entity IGnorant Classifier" (EIGC).

The EIGC uses the same feature space F as the SECs. Since it sees all observations of the stream DS, it is initialized as soon as the first observation arrives and can be used for learning and classification thereafter. As learning core of the EIGC, we use again the gradual fading MNB of [19], wherein the word counts are modified for each arriving observation and the fading of a word refers to the whole stream DS, as opposed to the sub-stream used by each SEC.

3.1.5 Ensemble variants based on weighting

We consider three weighting schemes for the ensemble members, each of them corresponding to an ensemble variant.

Variant 1: the entity-centric-classifier-ensemble $ECCE$ builds upon the fact that a minimum number of training observations x is necessary before a classifier can deliver reliable predictions. Hence, when the stream starts, $ECCE$ initializes $EIGC$ and one SEC for the entity e of each arriving observation. As soon as the minimum number of observations x has been reached for the SEC of an entity e, this classifier SEC_e can be used for predictions. Obviously, $EIGC$ is the first learner to start, since it is trained on all observations. Thus, $ECCE$ uses $EIGC$ to deal with the cold-start problem for new entities and for rarely referenced ones. As soon as x observations have been seen for entity e, $ECCE$ switches from $EIGC$ to the dedicated SEC_e for observations on e. The $EIGC$ is still trained in parallel so that it can benefit from the knowledge as well.

Variant 2: the entity-centric-weighted-ensemble $ECWE$ uses $EIGC$ for the observations of some entities, even after the cold-start is over. In particular, $ECWE$ assigns a weight w to the SECs of the ensemble and $1 - w$ to $EIGC$. These two weights are applied to the votes of the ensemble members for the label of each arriving observation.

Variant 3: the entity-RMSE-weighted-ensemble $ERWE$ replaces the fixed weights used by $ECWE$ with a weight emanating from the classification error of each ensemble member, thus assigning a higher voting weight to the member that has a lower error. In particular, for each arriving observation o, let e be the entity to which o belongs. We define the weight assigned to SEC_e as

$$wSEC(e) = \frac{RMSE(EIGC_e)}{RMSE(EIGC_e) + RMSE(SEC_e)}$$

The weight assigned to $EIGC$ for that entity e is

$$wEIGC(e) = \frac{RMSE(SEC_e)}{RMSE(EIGC_e) + RMSE(SEC_e)}$$

where we use the root mean square error (RMSE) as misclassification error, assuming ordinal labels. If the labels have no internal order, as would be the case for positive and negative labels only, the misclassification error can be used instead of RMSE.

Note that the weight of the $EIGC$ vote for observation depends on the entity to which this observation belongs. This allows the ensemble variant $ERWE$ to assign higher weights to $EIGC$ on entities, for which the SEC does not perform well (yet), while giving preference to the SECs, as soon as they show superior performance. Furthermore, this method is parameter-free in contrast to the $ECWE$ where we have to pick the weights in advance.

3.2 Memory reduction

In our work, we investigate two methods for reducing the memory requirements of entity-centric learning. The first method uses the lossy counting algorithm [14] to determine whether an entity-centric model should be kept in primary memory or should be stored in secondary memory for future retrieval. The second approach replaces complex entity-centric models with much simpler models that only rely on the label but not the text of an observation. These simple models have a much smaller memory footprint than the MNBFs [19] that we used in [6].

3.2.1 Entity management with lossy counting

In our case, we do not want to track frequent itemsets but the percentage of observations in a data stream that refer to a specific entity, so our elements e would be entity identifiers which in our case are the IDs of the Amazon products, which the incoming reviews refer to. At the end of a bucket, we collect the IDs of all the entities which would be deleted from D and save their models in secondary memory and delete them from the primary memory. We keep in primary memory the models of all entities that are in D. Models that have been saved to disk can later be retrieved and put back in primary memory in case future observations belonging to that entity arrive in the data stream. To realize this function, we need a second data structure L which simply tracks if we have seen the entity before. If an entity is in L but not in D, then we know we have to retrieve the model from disk. We can ignore the user parameter s as we are not interested in reporting frequent itemsets or frequent entities.

3.2.2 Replacing entity-centric models with text-ignorant models

Our second approach for reducing the memory footprint was inspired by our earlier work on entity-centric models [5] which only rely on the label of observation and ignore all the other features, in this case, the review texts. We could show that only using the labels of an entity could yield better predictions for some entities compared to the entity-ignorant MNBF but the entity-ignorant model is still the best overall. In our follow-up work [6], we show that we can improve the entity-ignorant MNBF by combining it with entity-centric MNBFs. In this work, we combine these two findings and now enrich the entity-ignorant MNBF with entity-centric models that only use labels, as these have a much smaller memory footprint than an MNBF. The label-only model that was most successful in [5] is a model that predicts the most frequent label of an entity seen so far which is why we also use this model in this study and call it the majority label of an entity.

Table 1 Description of the *tools* and *watches* data sets

Name	#Ent.	#Inst.	#Feat.	#Classes
tools	33,990	1,417,499	10,000	5
watches	78,220	487,907	10,000	5
bars5	25,110	2,224,710	10,000	5
barsFull	59,372	4,198,061	10,000	5

4 Experiments

In our experiments,[1] we compare the primary memory requirements of our two approaches (discussed in Section 3) on parts of the Amazon review data set[2] [10]. We use reviews from the "Tools and Home Improvement" category and the "Watches and Jewellery" category. They will be called *tools* and *watches*. On *tools*, we removed all products (entities) that have less than 10 reviews (observations) because of memory constraints when using no memory management strategy, as in [6]. The here presented methods do not have this limitation and also run on the whole data set but it would make the results incomparable which is why we refrained from this step. This leaves us with 1,412,499 observations (instances) and 33,990 entities on the *tools* data set and 487,907 observations and 78,220 entities on the *watches* data set. The features are the word counts of the 10,000 most popular words in all reviews and we have 5 classes which are the star ratings from one to five. We also used parts of the Yelp[3] data set, namely the "bars and restaurants" section. The Yelp data set was too big for some of our methods without memory management which is why we use the data from the top five cities (Toronto, Las Vegas, Phoenix, Montréal, Calgary) to assess the memory management functionality and reconfirm the ensembles classification performance on the full "bars and restaurants" data set, which contains 4,198,061 observations and 59,372 entities. This leaves us with the following four data sets (see Table 1).

[1] All our code can be found here: https://github.com/m-vishnu/entity-memory-management

[2] The data set can be found here: http://jmcauley.ucsd.edu/data/amazon/t

[3] The data set can be found here: https://www.yelp.com/dataset

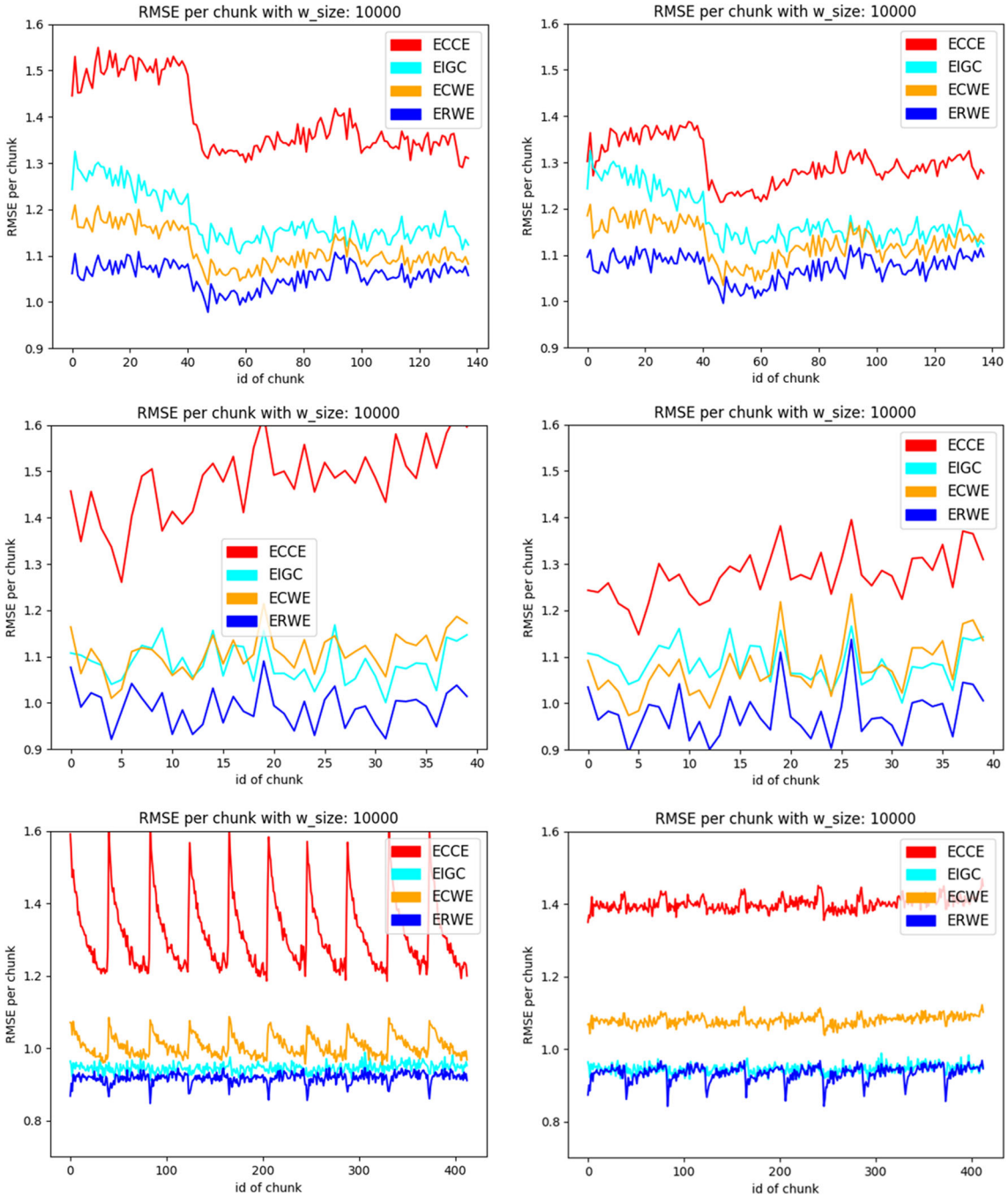

Fig. 2 We compare the RMSE of a model that only uses entity-centric predictions (ECCE), against a model that uses only the entity-ignorant predictions (EIGC), a model that combines both predictions by calculating the mean (ECWE) and a model that combines both predictions but with error correction (ERWE). We calculate the RMSE on non-overlapping chunks each containing 10k reviews. The top row shows the results on the *tools* data set, the middle row on *watches* and the bottom row on *barsFull*. The left graphics show the results using MNBFs as entity-centric models whereas the graphics on the right show the version using the majority label of the entity. In all cases, the error-corrected ensemble methods outperform the entity-ignorant classifier

4.1 Evaluation

We train our models prequentially which means every time an observation arrives we first predict its label using our models and afterwards we train the respective entity-centric model and the entity-ignorant model. Our evaluation focuses on two parts; the primary focus lies on how well our two approaches are suited for reducing the primary

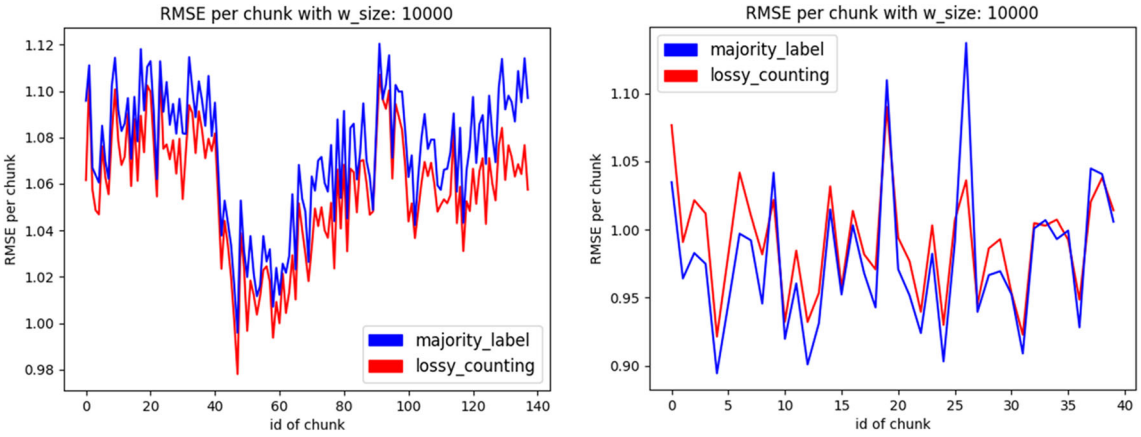

Fig. 3 We compare the RMSE of entity-centric MNBFs using lossy counting against entity-centric majority label on non-overlapping chunks of 10k reviews on the *tools* data set on the left and *watches* on the right. We can see that the RMSE is almost the same but that the lossy counting performs slightly better on *tools* and majority label performs slightly better on *watches*

memory footprint of our entity-centric models. For this, we plot the memory consumption over time and compare the two approaches to the memory requirements of a version which only uses an entity-ignorant model and to a version that combines entity-ignorant and entity-centric models but without memory management. For the lossy counting, we use $\epsilon = 0.001$.

The secondary focus is on whether replacing the entity-centric MNBFs with the majority label still improves performance compared to having only an entity-ignorant model and how the performance compares to having one MNBF per entity. While the main focus lies in the first question, we have to answer the second question first because it would not make sense to use the majority label if it would come at the cost of a huge performance drop.

To evaluate the performance, we use $RMSE(\hat{y}, y)$:

$$RMSE(\hat{y}, y) = \sqrt{\sum_{n}^{i=1} \frac{(\hat{y_i} - y_i)^2}{n}}$$

where $\hat{y}$ are the predicted labels, y are the true labels and n is the number of predictions. We chose $RMSE$ as an evaluation metric because we are dealing with ordinal labels and because we wanted to punish predictions that are far away from the true label. We calculate the $RMSE$ over 10,000 observations in non-overlapping chunks while ignoring the first observation for each entity as we do not have an entity-specific model at that time yet and need at least one observation to initialize one.

5 Results

In this section, we first compare the predictive performance of entity-centric models using MNBFs against entity-centric models that use the majority label of an entity. The second part investigates the reduction in memory requirements which lies at the heart of this work. We begin by reintroducing our acronyms for the ensembles to facilitate the discussion of our results. We compare a model that only uses entity-centric predictions (ECCE), against a baseline model that uses only the entity-ignorant predictions (EIGC), a model that combines both predictions by calculating the mean (ECWE) and a model that combines both predictions but with error correction (ERWE).

5.1 Entity-centric MNBF vs. majority label

The first question was if we can replace the entity-centric MNBFs with majority label classifiers while still improving prediction quality of the ensemble over a sole entity-ignorant MNBF. Our results on the *tools* data set show

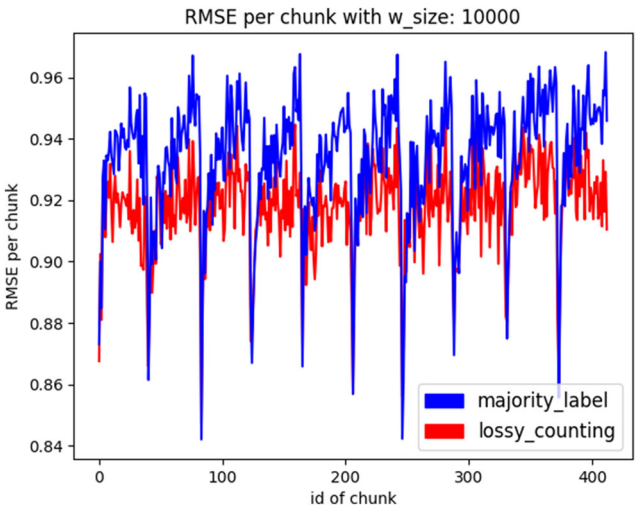

Fig. 4 We compare the RMSE of entity-centric MNBFs using lossy counting against entity-centric majority label on non-overlapping chunks of 10k reviews on the *fullBars* data set. We can see that the RMSE is close but that the lossy counting performs slightly better

that this is indeed the case for the error-corrected ensemble (ERWE) and that using the majority label classifier is only slightly worse than having one MNBF per entity. On the *watches* data set, the results of the majority label approach even slightly outperform the MNBFs (see Figs. 2 and 3). It is remarkable that a simple model like the majority label classifier can compete on these data sets with the much more complex MNBF while having a much smaller memory footprint which we will see in the next section. On the *barsFull* data set, the majority label also works well and leads to an overall improvement when used in an error-corrected ensemble (see Figs. 2 and 4). Due to the special nature of the data set, where we have a periodical arrival of many entities (see Fig. 8), we have a small rise of the RMSE above the entity-ignorant model EIGC whenever a huge number of entities arrives. This rise quickly disappears and is almost unnoticeable when looking at the graph. The ensemble using only the entity-centric models (ECCE) has of course a much worse performance than all the other ensembles which is consistent with our prior work [5]. This observation is independent whether we only use entity-centric MNBFs or the entity-centric majority label which shows that an entity-ignorant model which is trained in many instances is useful and necessary in both cases (Fig. 5).

5.2 Comparing the memory footprint

Both methods reduce primary memory requirements substantially (see Figs. 6 and 7). The second approach, using the majority label, is the better method when it comes to reducing memory requirements as it has almost the same memory footprint as the sole entity-ignorant model. When using lossy counting, the number of entities in memory drops very fast and remains almost constant from then on (see Fig. 8). Still, memory requirements rise very slowly. This is inevitable, since our data structure L tracks all entities seen so far and therefore keeps increasing even if the number of entities in memory stays constant. The memory requirements of the lossy counting method depend on how the parameter ϵ is set, as it affects the window size of the lossy counting buckets. In our experiments, we used an ϵ of 0.001 which equals a window size of 1000. The lossy counting algorithm stores models on the disk if they do not appear once per window on average. This means smaller windows (larger epsilon values) will lead to more models being stored on the disk, whereas larger windows will lead to more models being kept in primary memory. More models stored on the disk comes with an increased I/O-overhead, as more models have to be loaded back into primary memory. In Fig. 5, we can see that a window size of 10,000 almost matches the memory requirements of having no memory management at all and that a window size of 100 almost matches the memory requirements of having a sole entity-ignorant model. In the latter case, most entity-centric models are quickly stored to disk which frees up primary memory and in the former case, almost all entity-centric models are kept in primary memory. The choice of the parameter depends on the available hardware (Do we have a lot of RAM or a lot of disk space?), the frequency of new entities arriving on the stream, and how often these entities get new instances. In a real-world scenario, a suitable value can be computed quite easily by monitoring the used memory and when in doubt smaller windows would always ensure that most entity-centric models will be stored on the disk.

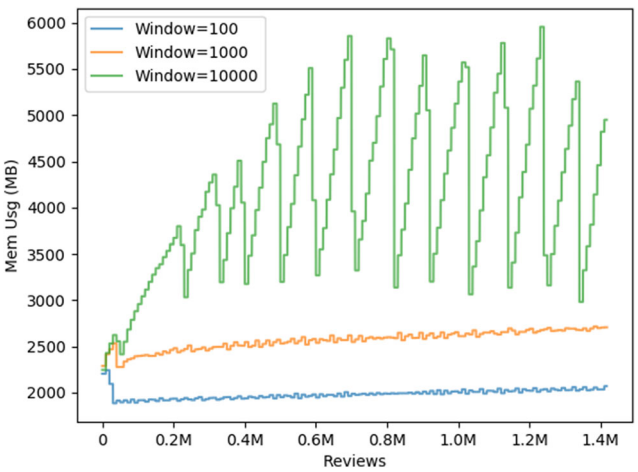

Fig. 5 We compare the primary memory usage of the Lossy Counting approach on the *tools* data set using different window sizes

5.3 Discussion

As both methods were successful in reducing the memory requirements while still improving prediction quality, the question of which method to choose arises. Based on our experiments on this data set, we would tend towards the majority label approach as the gains in memory reduction seem to outweigh the almost negligible drop in predictive performance compared to the lossy counting approach on the *tools* data set and it even outperforms the lossy counting method on *watches*. A potential reason for this might be that a lot of entities in the data sets have few observations so most information is stored in the prior distribution of the label and not in the word counts. This would also explain why it performs even better on *watches* as the average entity length is smaller compared to *tools*, 41.7 versus 6.2. This situation appears to be very specific to the data, which is why we refrain from giving a general recommendation

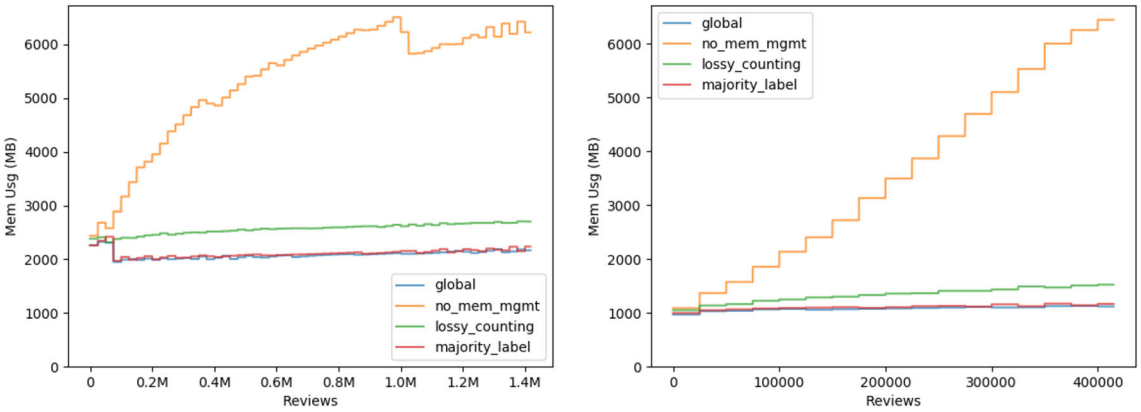

Fig. 6 We compare the primary memory usage of the entity-ignorant model against a combined model without memory management, a combined model with Lossy Counting, and a combined model which uses the majority label of the entity. Results for *tools* are on the left and for *watches* on the right. The figures show that the sole entity-ignorant model has constantly the lowest memory usage. The combined model using entity-centric majority labels almost matches the line of the entity-ignorant model. The combined model with lossy counting rises very slowly, whereas the combined model without memory management rises steeply

as more research is needed especially with different data sets with different entity lengths. In theory, it would also be possible to combine the two approaches and store the majority label models in secondary memory using lossy counting but we refrained from such a step as the memory requirements of the majority label approach already almost matched the memory requirements of a sole entity-ignorant model and the expected gain would therefore be minimal.

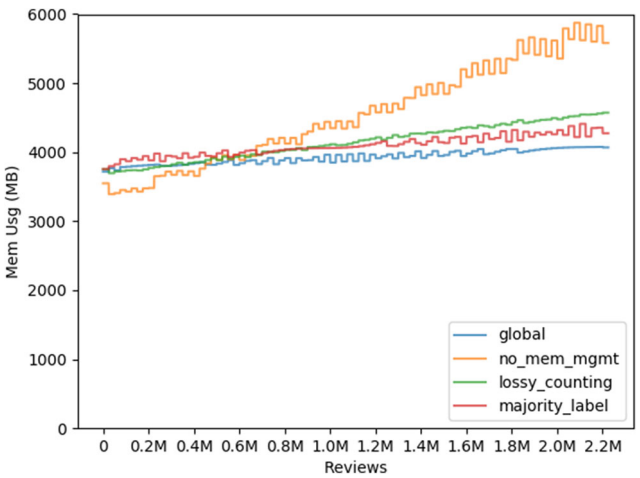

Fig. 7 We compare the primary memory usage of the entity-ignorant model against a combined model without memory management, a combined model with lossy counting, and a combined model which uses the majority label of the entity on the *bars*5 data set. The figure shows that the combined model without memory management uses the most memory and is steadily rising whereas the entity-ignorant model rises very slowly. The combined model using majority labels is slightly above the entity-ignorant model and the combined model using lossy counting uses even a bit more than that but still much less than using no memory management at all

6 Summary and conclusion

In this work, we have investigated two stream-based methods for reducing memory requirements of entity-centric models while keeping the predictive performance above a sole entity-ignorant model. The experiments were conducted on around 1.4M reviews from the "Tools and Home Improvement" category and 0.48M reviews from the "Watches and Jewellery" category on Amazon. Additionally, both methods were employed on two versions of the "Bars and Restaurants" Yelp data set which contain either 2.2M or 4.2M reviews.

Our first approach uses lossy counting to identify entities, whose models can be stored in secondary memory from where they could be retrieved in case observations belonging to these entities arrive again in the future. Our second approach replaces the entity-centric MNBFs (Multinomial Naive Bayes with Fading) with a much simpler majority label classifier which has a much smaller memory footprint.

6.1 Conclusion

Both methods are successful in reducing the memory requirements of the entity-centric models to an almost a constant rate, while still outperforming the entity-ignorant model. The best memory reduction method is the majority label approach, which uses around 2GB compared to the 2.6GB of the lossy counting approach and the 6.5GB of the previously published approach without memory management on the *tools* data set. On all presented data sets, the majority label approach almost matched the memory requirements of a sole entity-ignorant model. This indicates that we can improve prediction quality at almost

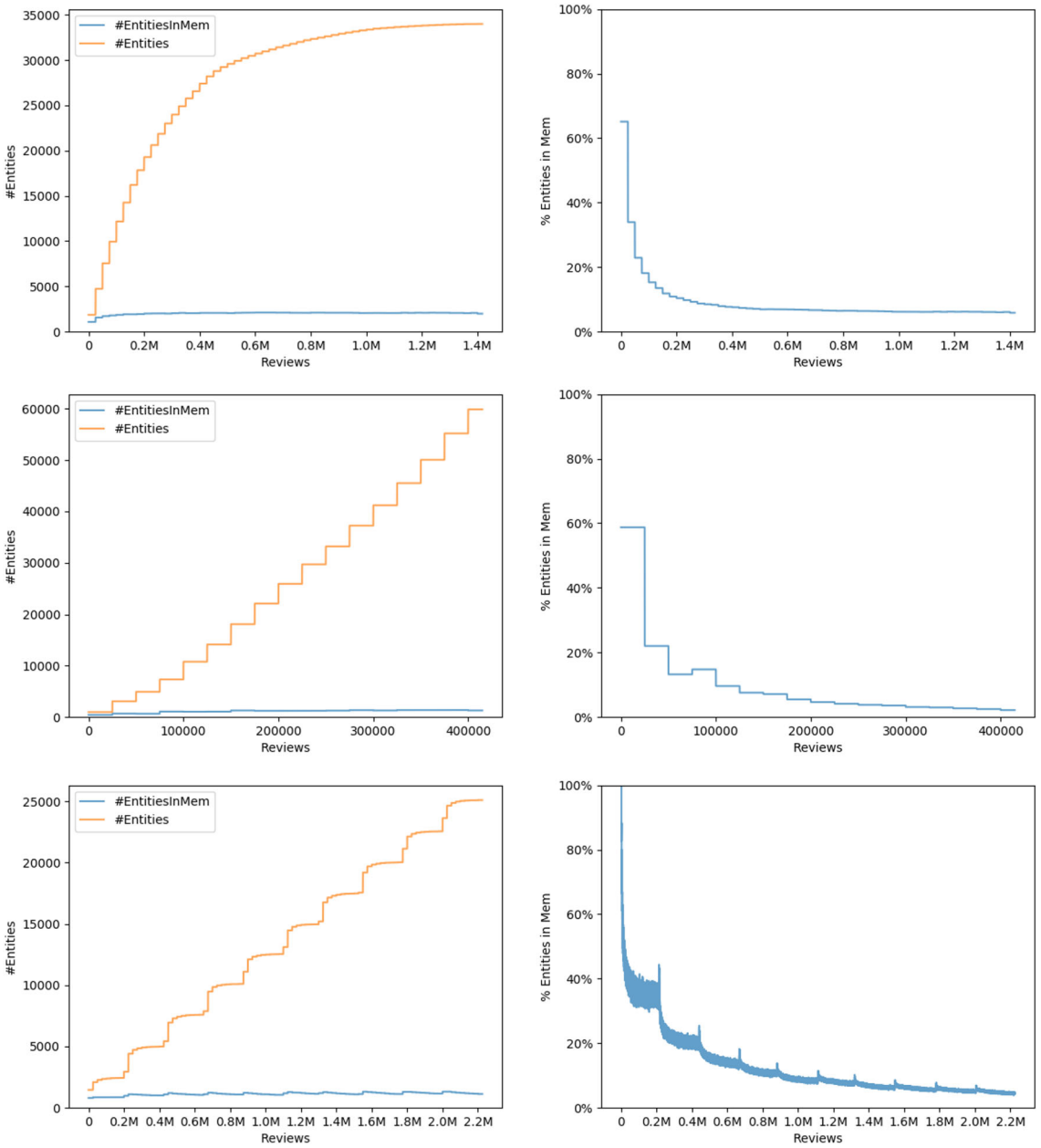

Fig. 8 For the lossy counting approach, we compare the number of entities seen so far against the number of entities in primary memory as well as the percentage of entities seen so far which are stored in primary memory, *tools* at the top, *watches* in the middle, and *bars*5 at the bottom. We can see all data sets that while the number of seen entities rises steeply the number of entities in primary memory remains almost constant and is much lower. Furthermore, the percentage of entities in primary memory shows a remarkable drop at the beginning and a constant decline in percentage afterwards

no additional memory cost. On the *tools* and *barsFull* data sets, this approach leads to a small drop in predictive performance compared to the entity-centric MNBF, but the drop was almost negligible in our case and the roles were even reversed on the *watches* data set, where the majority-label classifier is slightly better. It is impressive to see that combining a complex entity-ignorant model which receives a lot of training data with very simple entity-centric models that receive only little training data can increase prediction quality. We speculate that this is due to the nature of our data set where most entities are very short and therefore contain very few observations that can be used for training an MNBF. Such data sets, where the majority of entities are very short, are very common which makes this finding interesting for future work. This also explains why the drop in performance is higher on the *fullBars* data set as it contains fewer short entities and therefore benefits more from using MNBFs on an entity level. Based on our results,

both approaches are promising steps for making entity-centric learning viable in real-world applications under memory constraints.

6.2 Future work

Firstly, these experiments need to be repeated on more data sets from different domains and especially with different entity length distributions. The second step is to further investigate the trade-off between model complexity, predictive quality, and memory requirements. Additionally, we plan to investigate different models apart from MNBFs and majority label classifiers to see if these effects also hold true for other models.

Funding Open Access funding provided by Projekt DEAL. This work was partially funded by the German Research Foundation, project OSCAR "Opinion Stream Classification with Ensembles and Active Learners." Additionally, the first author is also partially funded by a PhD grant from the federal state of Saxony-Anhalt.

References

1. Spitz A, Almasian S, Gertz M (2019) Topexnet: entity-centric network topic exploration in news streams. In: Proceedings of the twelfth ACM international conference on web search and data mining, pp 798–801
2. Spitz A, Gertz M (2018) Exploring entity-centric networks in entangled news streams. In: Companion proceedings of the the web conference 2018, pp 555–563
3. Bahri M, Pfahringer B, Bifet A, Maniu S (2020) Efficient batch-incremental classification using umap for evolving data streams. In: Advances in intelligent data analysis XVIII. Springer International Publishing, pp 40–53
4. Barddal JP, Gomes HM, Enembreck F, Pfahringer B (2017) A survey on feature drift adaptation: definition, benchmark, challenges and future directions. J Syst Softw 127:278–294
5. Beyer C, Niemann U, Unnikrishnan V, Ntoutsi E, Spiliopoulou M (2018) Predicting polarities of entity-centered documents without reading their contents. In: Proceedings of the 33rd annual ACM symposium on applied computing. ACM, pp 525–528
6. Beyer C, Unnikrishnan V, Niemann U, Matuszyk P, Ntoutsi E, Spiliopoulou M (2019) Exploiting entity information for stream classification over a stream of reviews. In: Proceedings of the 34th ACM/SIGAPP symposium on applied computing. ACM, pp 564–573
7. Dai B-R, Huang J-W, Yeh M-Y, Chen M-S (2006) Adaptive clustering for multiple evolving streams. IEEE Trans Knowl Data Eng 18(9):1166–1180
8. Bifet A, Read J, Žliobaitė I, Pfahringer B, Holmes G (2013) Pitfalls in benchmarking data stream classification and how to avoid them. In: Joint european conference on machine learning and knowledge discovery in databases. Springer, pp 465–479
9. Erdogan BE, Akyüz SÖ, Atas PK (2019) A novel approach for panel data: an ensemble of weighted functional margin svm models. Information Sciences
10. He R, McAuley J (2016) Ups and downs: modeling the visual evolution of fashion trends with one-class collaborative filtering. In: Proceedings of the 25th international conference on world wide web, pp 507–517
11. Liao J, Dai B (2014) An ensemble learning approach for concept drift. In: 2014 international conference on information science applications (ICISA), pp 1–4
12. Liu Z, Hauskrecht M (2016) Learning adaptive forecasting models from irregularly sampled multivariate clinical data. In: Thirtieth AAAI conference on artificial intelligence
13. Lu H, Huang S (2011) Clustering panel data. In: SIAM international workshop on data mining held in conjunction with the 2011 SIAM international conference on data mining, pp 1–10
14. Manku GS, Motwani R (2002) Approximate frequency counts over data streams. In: VLDB'02: proceedings of the 28th international conference on very large databases. Elsevier, pp 346–357
15. Melidis DP, Spiliopoulou M, Ntoutsi E (2018) Learning under feature drifts in textual streams. In: Proceedings of the 27th ACM international conference on information and knowledge management. ACM, pp 527–536
16. Rudovic O, Utsumi Y, Guerrero R, Peterson K, Rueckert D, Picard RW (2019) Meta-weighted gaussian process experts for personalized forecasting of ad cognitive changes. In: Machine learning for healthcare conference, pp 181–196
17. Saadallah A, Priebe F, Morik K (2020) A drift-based dynamic ensemble members selection using clustering for time series forecasting. In: Brefeld U, Fromont E, Hotho A, Knobbe A, Maathuis M, Robardet C (eds) Machine learning and knowledge discovery in databases. Springer International Publishing, Cham, pp 678–694
18. Unnikrishnan V, Beyer C, Matuszyk P, Niemann U, Pryss R, Schlee W, Ntoutsi E, Spiliopoulou M (2019) Entity-level stream classification: exploiting entity similarity to label the future observations referring to an entity. International Journal of Data Science and Analytics
19. Wagner S, Zimmermann M, Ntoutsi E, Spiliopoulou M (2015) Ageing-based multinomial naive bayes classifiers over opinionated data streams. In: European conference on machine learning and principles and practice of knowledge discovery in databases, ECMLPKDD'15, vol 9284, pp 401–416

Affiliations

Christian Beyer[1] · Vishnu Unnikrishnan[1] · Robert Brüggemann[1] · Vincent Toulouse[1] · Hafez Kader Omar[1] · Eirini Ntoutsi[2] · Myra Spiliopoulou[1]

Vishnu Unnikrishnan
vishnu.unnikrishnan@ovgu.de

Robert Brüggemann
robert.brueggemann@st.ovgu.de

Vincent Toulouse
vincent.toulouse@st.ovgu.de

Hafez Kader Omar
hafez.kader@st.ovgu.de

Eirini Ntoutsi
ntoutsi@kbs.uni-hannover.de

Myra Spiliopoulou
myra@ovgu.de

1 Otto-von-Guericke University, Magdeburg, Germany

2 Leibniz University, Hannover, Germany